

AFIPS

CONFERENCE PROCEEDINGS

RICHARD E. MERWIN
Editor and Program Chairman

JACQUELINE T. ZANCA
Managing Editor

MERLIN SMITH
Conference Chairman

1979

NATIONAL COMPUTER CONFERENCE

AFIPS PRESS
210 SUMMIT AVENUE
MONTVALE, NEW JERSEY 07645

June 4-7, 1979
New York, New York

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1979 National Computer Conference or the American Federation of Information Processing Societies, Inc.

Library of Congress Catalog Card Number 55-44701
AFIPS PRESS
210 Summit Avenue
Montvale, New Jersey 07645

© 1979 by AFIPS Press. Copying is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) reference to the AFIPS Proceedings and notice of copyright are included on the first page. The title and abstract may be used without further permission in computer-based and other information-service systems. Permission to republish other excerpts should be obtained from AFIPS Press.

Printed in the United States of America

Preface



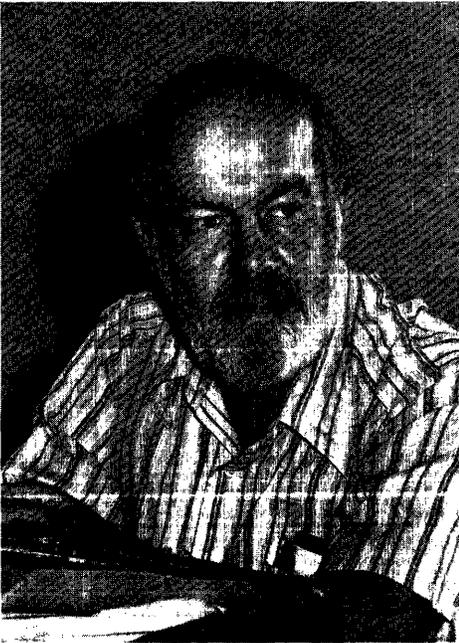
MERLIN SMITH
Conference Chairman

The Proceedings of the 1979 National Computer Conference represents the most comprehensive, in-depth treatment of computing developments available today. It stands as a lasting credit to Richard Merwin and his program committee, and to the many authors, session chairpersons, reviewers and other contributors recognized on these pages.

Data processing has rapidly become one of the more vital factors in the economic and personal well being of all. The NCC program was directed to these many new interests of participants. Such an objective required many panel and discussion sessions beyond the formal papers, and the limits of this Proceedings. We owe a special debt to these participants who helped make our conference a success. Their names are recorded herein.

We appreciate this opportunity to be a part of recorded computer history.

Introduction



RICHARD E. MERWIN
NCC '79 Program Chairman

The NCC '79 technical program was planned to be a learning experience for all attendees. A broad range of paper and panel sessions was selected to emphasize social implications of computers, management issues, technical developments and applications. Each of these areas is represented by both paper and panel sessions which are designed to bring the attendees of this "biggest of all" computer conference to the forefront of knowledge of each specialty.

A major attempt has been made to broaden the scope of the NCC '79 technical program by including three mini-conferences covering the application of computers to financial transactions, law and health services. Thirty-two sessions dealing with these topics will expand the coverage of NCC '79 to an audience of specialists in fields which are increasingly becoming allied with data processing techniques.

The urge to participate in the NCC '79 technical program was overwhelming. Because of a limitation on the size of this Proceedings, the number of papers that could be accepted for publication was curtailed. A large number of proposals were received for panel sessions and a selection of the best of these was made. This trend for more and more participation in both the technical program and the exhibition of the latest computer products mirrors the tremendous growth of the computer industry, especially in the areas of micro-processors.

In response to the wide interest in the use of computers by the non-professional, a special set of sessions and a separate publication devoted to personal computing has been organized to augment the technical program and regular exhibits. The interest in this aspect of the computer industry has increased rapidly and represents a major factor in this industry.

This Proceedings is organized by specialized areas including Applications, Social Implications, Architecture, Data Base Management, Computer Technology, Networking and Software Techniques, in that order. Unfortunately, we had to eliminate overview statements by topical area organizers along with descriptions of panel sessions to maximize the number of technical papers that could be published. I regret these omissions but feel that our policy of publishing only technical papers best serves the technical goals of the conference.

The planning and organization of the NCC '79 technical program involved a number of area coordinators, session organizers and leaders and the panelists and presenters of technical papers. I want to extend my sincere appreciation to all those who supported the organization of this outstanding technical program. Special thanks is due to the hundreds of referees who helped us select the best papers. Finally, I want to thank the program committee staff who tirelessly worked with all participants to make this conference a success.

CONTENTS

Preface	iii
Merlin Smith	
Introduction	iv
Richard E. Merwin	
APPLICATIONS	
Computer technology in the movie industry	1
Suzanne Landa	
The system architecture evaluation facility—An emulation facility at Rome Air Development Center	7
N. Bruce Clark and Michael A. Troutman	
Teaching and research experiences with an emulation laboratory	13
Steven F. Sutphen	
Simulating the delay in logic networks for large, high-speed computers	19
E. A. Wilson	
Languages for operating systems description, design and implementation	29
Philip H. Enslow, Jr.	
The simulation language SIML/I	39
M. H. MacDougall	
Mix-dependent job scheduling—An application of hybrid simulation	45
Steve Tolopka and Herb Schwetman	
Parametric instabilities in computer system performance prediction	51
Lawrence W. Dowdy and Ashok K. Agrawala	
Aids to the development of network simulators	57
Imrich Chlamtac and W. R. Franta	
A stochastic state space model for prediction of product demand	67
William C. Cave and Evelyn Rosenkranz	
The Bus Link—A microprogrammed development tool for the CMOS/SOS processor system	73
Avner Ben-Dor, Paul Baker and Jon Selden	
A computer analysis tool for structural decomposition using entropy metrics	83
Aaron N. Silver	
Interactive modeling systems for managers—Semantic models should underlie quantitative models	89
Rand B. Krumland	
Modeling regular, process-structured networks	95
Bruce W. Arden and Hikyu Lee	
The City of New York's integrated financial management system—From mandate to working system in 18 months	103
Sally J. Rupert	
Recurrent dilemmas of computer use in complex organizations	107
Rob Kling and Walt Scacchi	
Project management through the Accomplishment Value Procedure (AVP)	117
Donald J. Aharonian	

Textfax—Principle for new tools in the office of the future	125
Wolfgang Horak and Walter Woborschil	
Microcomputer programming skills	135
C. Wrandle Barth	
Program conversion—One successful paradigm.....	139
Charles Lynn, Jr., Jean Risley and Robert Wells	
A generalized zooming technique for pictorial database systems	147
S. K. Chang, B. S. Lin and R. Walser	
An approach to real-time scan conversion	157
Franklin C. Crow	
The evolution and architecture of a high-speed workstation for interactive graphics	165
William L. Paisner	
A mathematical model for distributed free space	175
Y. H. Chin and S. H. Yu	
Forecasting computer resource utilization using key volume indicators	185
David E. Y. Sarna	
Workflow—A technique for analyzing JES systems.....	193
H. Pat Artis	
MMPS—A reconfigurable multi-microprocessor simulator system	199
Daniel Klein	
A (31,15) Reed-Solomon code for large memory systems	205
Raymond S. Lim	
English dictionary searching with little extra space	209
Douglas Comer	
New indices for bibliographic data and their applications	217
Yahiko Kambayashi, Shuzo Yajima, Osamu Konishi and Takaki Hayashi	
Visual inspection of metal surfaces	227
J. L. Mundy	
Monitoring the earth's resources from space—Can you really identify crops by satellite?	233
David Landgrebe	
Digital image shape detection	243
R. Michael Hord	
PM ⁴ —A reconfigurable multiprocessor system for pattern recognition and image processing	255
Fayé A. Briggs, King-Sun Fu, Kai Hwang and Janak H. Patel	
Transportable image-processing software	267
R. G. Hamlet and A. Rosenfeld	
A data-handling mechanics of on-line text editing system with efficient secondary storage access	273
Sakti Pramanik and Edgar T. Irons	
SOCIAL IMPLICATIONS	
How do we best control the flow of electronic information across sovereign borders?	279
Peter Safirstein	
Privacy and security in transnational data processing systems	283
Rein Turn	
A modular approach to computer security risk management	293
Robert P. Campbell and Gerald A. Sands	
The design and operation of public-key cryptosystems	305
Eric H. Michelman	

Safeguarding cryptographic keys	313
G. R. Blakley	
Applications for multilevel secure operating systems	319
John P. L. Woodward	
The foundations of a provably secure operating system (PSOS)	329
Richard J. Feiertag and Peter G. Neumann	
A security retrofit of VM/370	335
B. D. Gold, R. R. Linde, R. J. Peeler, M. Schaefer, J. F. Scheid and P. D. Ward	
KSOS—The design of a secure operating system	345
E. J. McCauley and P. J. Drongowski	
UCLA Secure UNIX	355
Gerald J. Popek, Mark Kampe, Charles S. Kline, Allen Stoughton, Michael Urban and Evelyn J. Walton	
KSOS—Development methodology for a secure operating system	365
T. A. Berson and G. L. Barksdale, Jr.	
KSOS—Computer network applications	373
M. A. Padlipsky, K. J. Biba and R. B. Neely	
Considerations in the employment of blind computer professionals	383
James A. Kutsch, Jr., and Kimberly B. Kutsch	
Hiring a deaf computer professional	385
Karen K. Anderson and Philip W. Bravin	
MIS effects on managers' task scope and satisfaction	391
Daniel Robey	
Some neglected outcomes of organizational use of computing technology—And their implications for systems designers	397
M. Lynne Markus	
An academic meets industry—Rethinking computer-based education and personalized systems of instruction ..	403
Kenneth L. Modesitt	
Recent developments in computers and society research and education	407
Richard H. Austing and Gerald L. Engel	
Interactive monitoring of computer-based group communication	411
Kathleen Spangler, Hubert Lipinski and Robert Plummer	
The status of women in health science computing	415
Lynn L. Peterson	
Women and minorities in the computer professions	419
Helen M. Wood	
Computers in judicial administration	425
Charles L. Aird and Barbara H. Todd	
Police and computer technology—The expectations and the results	443
Kent W. Colton	
 ARCHITECTURE	
Distributed algorithms for global structuring	455
Raphael A. Finkel, Marvin Solomon and Michael L. Horowitz	
The tree-structured distributed network front-end processor architecture	461
Robert M. Monroe, Ronald J. Srodawa and Franklin H. Westervelt	
Analysis of real-time control systems by the model of packet nets	469
Mohamed Gawdat Gouda	

Performance and economy of a fault-tolerant multiprocessor	481
Jaynarayan H. Lala and Charles J. Smith	
Serviceability features of the HP 300 small business computer	493
Curtis R. Gowan	
Automatic tuning of computer architectures	499
Ken Sakamura, Tatsushi Morokuma, Hideo Aiso and Hajime Iizuka	
The BTI 8000—Homogeneous, general-purpose multiprocessing	513
George R. Lewis, J. Shirley Henry and Brian P. McCune	
A survey of interconnection methods for reconfigurable parallel processing systems	529
Howard Jay Siegel, Robert J. McMillen and Philip T. Mueller, Jr.	
Adaptation properties for dynamic architectures	543
Steven I. Kartashev, Svetlana P. Kartashev and C. V. Ramamoorthy	
Architectural considerations of the NEC mass data file subsystem	557
Akira Sekino and Takuo Kitamura	
Error-oriented architecture testing	565
Larry Kwok-Woon Lai	
A survey of methods for intermittent fault analysis	577
Yashwant K. Malaiya and Stephen Y. H. Su	
Architectural and design perspectives in a modular multi-microprocessor, the DPS-1	587
Kells A. Elmquist	
Work flow view of a distributed application	595
J. R. Hamstra	
The use of self-inverse program primitives in system evaluation	605
John E. MacDonald, Jr.	
A loosely-coupled applicative multi-processing system	613
Robert M. Keller, Gary Lindstrom and Suhas Patil	
A prototype data flow computer with token labelling	623
Ian Watson and John Gurd	
A view of dataflow	629
Kim P. Gostelow and Robert E. Thomas	
A hardware-independent virtual architecture for PASCAL	637
Viswanathan Santhanam	
Design of a high-level language machine	649
G. J. Battarel and R. J. Chevance	
A programming language for high-level architecture	657
Yaohan Chu and Edward Ray Cannon	
 DATA BASE MANAGEMENT	
Data management in distributed data bases	667
C. V. Ramamoorthy and Benjamin W. Wah	
A unified architecture for data and message management	681
Georges Gardarin	
Design of a prototype ANSI/SPARC three-schema data base system	689
Eric K. Clemons	
On the implementation of a conceptual schema model within a three-level DBMS architecture	697
Shamkant B. Navathe and Johann Lemke	

The practice of data base administration	709
Jay-Louise Weldon	
An approach to automatic maintenance of semantic integrity in large design data bases	713
Gilles M. E. Lafue	
On query-answering in relational data bases	717
E. L. Lozinskii	
ASTROL—An associative structure-oriented language	721
James F. Wirth	
An associative search language for data management	727
Amar Mukhopadhyay and Alireza Hurson	
Updating defined relations	733
I. M. Osman	
Performance enhancement for relational systems through query compilation	741
Randy H. Katz	
 COMPUTER TECHNOLOGY	
System considerations for predicting mass storage subsystem behavior	749
E. J. McBride, A. B. Tonik and G. R. Finnin	
A software reliability study using a complexity measure	761
Thomas J. Walsh	
A Markovian model for reliability and other performance measures of software systems	769
Amrit L. Goel and Kazu Okumoto	
Partial match retrieval for non-uniform query distributions	775
V. S. Alagar and C. Soochan	
 NETWORKING	
Comparing interactive computer services—Theoretical, technical and economic feasibility	781
S. A. Mamrak and P. D. Amer	
Characterizing a workload for the comparison of interactive services	789
Domenico Ferrari	
Control of computing funds and resources in a networking environment	797
Beverly O'Neal and Ronald Segal	
The economic impact of network affiliation upon institutions of higher learning	805
Norman R. Nielsen	
Approaches to concurrency control in distributed data base systems	813
Philip A. Bernstein and Nathan Goodman	
Access control mechanisms for a network operating system	821
Helen M. Wood and Stephen R. Kimbleton	
Public key vs. conventional key encryption	831
Charles S. Kline and Gerald J. Popek	
SIGMA—An interactive message service for the Military Message Experiment	839
Robert Stotz, Ronald Tugender, David Wilczynski and Donald Oestreicher	
The SIGMA experience—A study in the evolutionary design of a large software system	847
David Wilczynski, Ronald Tugender and Donald Oestreicher	
The terminal for the Military Message Experiment	855
Robert Stotz, Paul Raveling and Jeff Rothenberg	

On-line tutorials and documentation for the SIGMA message service	863
Jeff Rothenberg	
Maintaining order and consistency in multi-access data	869
Ronald Tugender	
Exact solution for the initialization time of packet radio networks with two station buffers	875
Daniel Minoli	
Fixing timeout intervals for lost packet detection in computer communication networks	887
Robert J. T. Morris	
Comparison of some end-to-end flow control policies in a packet-switching network	893
G. Pujolle	
Alternatives for providing highly reliable access to X.25 networks	905
Richard J. Chung and A. M. Rybczynski	
A fail-safe distributed local network for data communication	917
Jane W. S. Liu, Izumi Suwa, Robert Stepp, Sergio M. Hinojosa and Tsutoma Utsuqi	
An analysis of a distributed switching network with integrated voice and data in support of command and control	927
Daniel Schutzer	
The exploratory system control model multi-loop network	935
Daniel J. Paulish	
 SOFTWARE TECHNIQUES	
Software reliability measures applied to systems engineering	941
John D. Musa	
Verification procedures supporting software systems development	947
Gruia-Catalin Roman	
A language for distributed processing	957
Ronald J. Price	
Automatic program transformations for virtual memory computers	969
W. Abu-Sufah, D. Kuck and D. Lawrie	
Analysis of data flow models using the SARA graph model of behavior	975
W. Ruggiero, G. Estrin, R. Fenchel, R. Razouk, D. Schwabe and M. Vernon	
Software metrics for aiding program development and debugging	989
N. F. Schneidewind	
A measure of software complexity	995
Ned Chapin	
Relating computer program maintainability to software measures	1003
A. R. Feuer and E. B. Fowlkes	
Program forms and program form analysers for high-level structured design	1013
Jayashree Ramanathan and Meera Blattner	
First-year results from a research program on human factors in software engineering	1021
Sylvia B. Sheppard, Bill Curtis, Phil Milliman, M. A. Borst and Tom Love	
The use and abuse of a software engineering system	1029
D. J. Pearson	
The integrated control/distributed power software development shop	1037
Jean-Paul Renault	
On the fate of software enhancements	1043
Norman K. Sondheimer	

Experiences in building and using compiler validation systems	1051
Paul Oliver	
Automatic program synthesis via synthesis of loop-free segments	1059
Joe W. Duran	
Semantic similarity analysis—A computer-based study of meaning in noun phrases	1063
Lynn L. Peterson	
Heuristic control of design-directed program transformations	1071
Christina L. Jette	
A data flow evaluation system-based on the concept of recursive locality	1079
A. L. Davis	
Data flow languages	1087
William B. Ackerman	

Computer technology in the movie industry

by SUZANNE LANDA

The Rand Corporation
Santa Monica, California

INTRODUCTION

The movie industry uniquely provides the opportunity to combine the creativity of the artist with the technology of science. It was in fact the marriage of art and science that gave birth to filmmaking. While many advances and discoveries have been made in the tools used to make movies, support their production and distribution, and enhance their exhibition, perhaps none since the camera will have the pervasive effect of the computer. Both behind the scenes and on the screen the ubiquitous computer is beginning to have an impact on the movie industry.

This paper follows a movie from its initial conception through production, distribution, exhibition, preservation and redistribution, surveying current and planned applications of computer technology and identifying areas requiring further research. It purposely focuses on the problems of motion picture production amenable to computer application and not on specific technical solutions. The latter will be provided in the session by guest speakers from the movie industry. With the emphasis on movies for this session, computer applications unique to television and other related fields have been excluded.

CONCEPT

A movie begins with an idea. The source of the idea may be an individual's fantasy, an article or book, a newspaper story, or even the preferences of thousands of people compiled and analyzed by computer. Sunn Classics Productions, Inc. has successfully applied the latter approach to come up with the idea for "The Lincoln Conspiracy," "In Search of Noah's Ark," and other box office successes.¹ Their extensive computer analysis approach, which involves not only idea but also story generation, has only been applied to movie making for special audiences (family entertainment). Successful application for general audiences has not yet been ascertained.

However, once an idea exists, studios do use market research and computer analysis to estimate its potential for success. For example, after producing several successful disaster films, Twentieth-Century Fox relied on market research to indicate when audiences had reached a saturation point for that genre.² Market research with computer anal-

ysis for this type of general information is expected to increase.

A movie idea is given life by the writer who turns it into a screenplay. Script writing remains primarily an individual art form centered around the typewriter with occasional forays to the library or other information sources. While the task of typing dialogue lends itself to automated text editing, the author is aware of only one screenwriter who has invested in such a system. Within several years, as the costs of personal computer systems (particularly peripherals) drop, repair support increases, and computerized library and periodical services become more accessible over communications networks, the personal computer will undoubtedly become a valuable aid in screenwriting.

Starting with the purchase of a script and continuing through the distribution and exhibition of a movie, payment to employees is accomplished through a payroll system more complex than any in other industries. The continually changing rules and regulations of over 65 unions and guilds must be handled. Many workers must be paid within 24 hours of the time labor was terminated. If a worker's job is upgraded during the day, his pay for the entire day may have to be adjusted and also the payments to those who worked with him. Depending on when, what, and where he is working, he may earn up to eight times his regular pay. Each union's definition of a work week also varies. Not only must union regulations be tracked, but also the tax structure of every state since the studio must provide a W2 form for every state in which an employee has worked. Another factor contributing to the complexity of the payroll system is that the size of the work force is constantly changing. While a studio may employ 3500-7000 people permanently, total annual employment may easily exceed 50,000 with the total number of checks issued ten times greater. Predominately COBOL written, batch-oriented systems provide payroll support for producers. These services are available from the major studios, e.g. Universal and Warner Bros., and from service bureaus.

In addition to payroll, contracts are issued and modified during all stages of production. This task is handled in Disney's and Fox's legal departments through the use of word processing systems. Interconnectivity of these systems with other departments and those of external concerns, e.g. law firms, has been limited to homogenous systems because of problems with nonstandard communication protocols.

PRE-PRODUCTION

Once a shooting script has been prepared, the pre-production activities of budget and schedule planning commence. These tasks are compounded by the problem that scripts are not shot chronologically. A shooting schedule depends on the availability of actors, sound stages, locations, props, etc. It also depends on economics. For example, since an actor filmed on Monday and Friday must be paid for the entire week, economics dictate that his scenes be shot at closer intervals. Outdoor scenes are usually filmed before interiors because uncontrollable environmental factors reflected in outdoor scenes may impact the indoor scenes. In addition to schedule planning, scene requirements for sets, props, technical equipment, etc. must be estimated before a budget can be set. It is not unusual for a feature film budget to consist of many thousands of separate items. Both scheduling and budgeting are basically manual processes today with some automated support through data entry systems using formatted displays. However, at the University of New South Wales, an interactive system is being designed for film budgeting, the generation of an economic shooting schedule and the breakdown of individual scene requirements. During the pre-production phase of scheduling and costing, the system will accept as data the script breakdown and all relevant costs. Output will be an initial draft schedule and a total cost estimate. When cast, locations, and budgets have been determined, a detailed shooting schedule is then generated through a tree search. Such an approach does not produce the optimum schedule, but experience with other industrial scheduling situations have indicated to the developers that schedules at least as good as those generated by experienced people could be expected.³ While production people have shown interest in this type of total system approach, computer aided budgeting and scheduling will probably expand first through subtask application.

One of the requirements determined for each scene is the number and types of extras. The casting of extras presents a particularly formidable problem. At Universal, for example, between 50 and 2000 extras are required daily to appear as background and atmosphere people in productions. Requests are usually very specific: five men with black beards between 20-30 years old, 5'10"-6'2", who can ride horses and duel with swords. It is even better if they own their own horses and swords. Universal uses an interactive system which accesses a data base containing the names of available extras and information about their skills, attributes, costumes, props, etc. When the next day's casting requirements are released, potential extras who best fit the part can be selected online. A similar system for creative talent, i.e. producers, writers, directors, and actors, will be available at Universal in 1979. A producer may then ask to see, for example, a list of directors who specialize in feature westerns and whose credits have grossed over \$30,000,000.

Once the budget and schedule have been determined and actors, locations, equipment, and crews selected, the director, art director and cameraman must design the sets. Sets are usually overbuilt because they are designed for all con-

tingencies. For example, only two-thirds of a \$3,000,000 set may appear in the final print. In this case, \$1,000,000 was spent on a set that will never be seen by the audience. To avoid this waste, those at Robert Abel & Associates involved in full-scale spaceship set designs for the movie "Star Trek" (to be released December, 1979) are using computer graphic aids to determine the parts of each set which must be built. Line drawn versions of sets and people are entered into an Evans & Sutherland Picture System 2 through a tablet. For each set, camera angles and moves are executed using the System 2 controls. In this way, those portions of a set that need not be built because they will never be visible can be determined. It is also possible to identify areas of a set that may be visible but are amenable to matte effect in place of construction.

For movies which include animation or special effects sequences, storyboards outlining the action are developed during pre-production. At Universal's new special effects facility (Universal Hartland), a computer graphics system on a stand-alone microcomputer is being used to create the storyboard for "Buck Rogers." Since storyboards only include sketches of key actions, during the actual filming it may be discovered that the pacing required to move from one sketch to the next varies from that planned. To avoid this problem in the making of "Star Trek," Robert Abel's is using the Evans & Sutherland to preview action sequences in real-time before filming begins.

PRODUCTION

The actual shooting of a movie may occur at the studio, on location, or a combination of both. Through computer support, producers at the major studios get daily reports on the previous day's expenditures for a particular feature. Overruns are immediately visible so that modifications can be made in the remaining stages of production to absorb or minimize the extra costs. In some cases, early cost excesses result in a project's termination.

Location shooting presents severe cost control and payroll problems. At Paramount, timely and accurate cost information and local payroll capabilities are provided on location by a minicomputer-driven terminal system. Universal is currently implementing a similar minicomputer-based system. At Disney, a microcomputer system with dual-diskettes and printer will be tested on location in Hawaii in early 1979. Disney also expects to use the system on stage at the studio for backlot production sequences. These reporting systems are used during the day on location to record transactions. At night the daily records are transmitted to the central processor. Reports, updated master files, and data discrepancies are then returned to the location for next-day availability. The decrease in reporting time through use of on-location computer support is as much as ten to one. A capability to be added in the future will allow the location auditor to explore the costs of various courses of action when an unforeseen event occurs. For example, should a storm break, with an expected duration of two weeks, the location auditor would like to determine the costs of keeping

everyone on location versus sending them home, paying required penalties and bringing them back later.

Computer technology is also used during production to assist in the generation of the animated images seen on the screen. Animation techniques can be divided into two categories: 2-D animation, involving the use of hand-drawn images, and 3-D animation, involving the manipulation of models and puppets. Both techniques make use of storyboards which are subject to computer application as described under *Pre-Production*. 3-D animation using models will be discussed under *Post-Production*, as it is traditionally associated with the post-production area of special effects.

For 2-D animation, like those of Disney and Hanna Barbera, the first production step requires an artist's rendition of key frames in each scene. The next step calls for an assistant animator or "inbetweener" to fill in the action by providing transition frames between the key frames. Each of these drawings is then photographed, shot onto celluloid, and painted. Finally, each cel or the required combination of cels is placed on an animation stand for filming. For feature films, the only step involving a computer today is the last: Camera settings required to simulate movement are computed and provided to the cameraman filming off the animation stand. However, by mid-1979, several research efforts will have systems commercially available to aid in all these steps of animated feature film production. The systems allow for input of key frames by an artist using a light pen and tablet with the computer performing inbetweening. The artist then "paints" the stored frames interactively with light pen and color selectors. To obtain consistency in scene and character colorization, the systems will allow for the storing and retrieving of colors by picture elements. The need for celluloids is eliminated since frames will be filmed directly off a CRT.⁴ A major difficulty in providing computer aids to the animator has been to provide him with input tools with which he feels as artistically free as with conventional methods. The designers of these systems feel they have overcome the problem by providing paintbrush, pencil, and spraygun options to the artist through software. The other area still open to question for commercial application is whether these systems will produce the high-quality, high-resolution animation required for feature films. An answer to this should be forthcoming in 1979 when at least one production company plans to make a full-length animated movie using this type of computer system.

The use of computers to aid live-action filming premieres this year with the release of "The China Syndrome" (Michael Douglas Productions). For story realism and for legal protection, it was necessary in this movie to duplicate precisely the interior of a nuclear power plant during the various stages of an alert. This required the operation of 131 circuits controlling 2500 instrument panel lights in differing sequences and in differing states (off, slow-flash, fast-flash, solid-on) for each stage of the alert, synchronized with live-action performances. The task was compounded by the need to restart the sequences at any point for retakes and for daily continuity. A combination of manual and electronic methods to handle this type of operation has proved in the past to be costly and unpredictable. To avoid these problems

for "The China Syndrome," Eyewitness, Ltd. programmed a microcomputer in assembly language to allow accurate, flexible, and reliable operation of the panel lights in coordination with the actors' performances.

Computers, of course, have been known to appear or even star in a movie. Usually, however, what is seen are whirling tape drives and a card sorter or maybe a terminal flashing Christmas tree lights. Universal has taken steps to remedy the situation by creating realistic computer environments and systems for production shots. For example, simulated interactive hospital and law enforcement systems are available for use as dictated by a script.

A print of the original camera footage must be made each day for viewing the following day. The automation used to print dailies is part of the systems used for post-production processing in film laboratories which is discussed in the next section.

POST-PRODUCTION

The post-production phase of movie making consists of creating and adding special visual effects and titles, adding music and sound effects, and, finally, processing, editing, and printing the finished product in the motion picture laboratory.

Special visual effects using models have become well known through such movies as "2001: A Space Odyssey" and "Star Wars." Contrary to popular opinion and some press reports, special-purpose, hard-wired machines, not computers, were used to control cameras and models in these and other recent movies. Not until 1979 with the release of "Buck Rogers" (Universal), "The Black Hole" (Disney), and "Star Trek" (Paramount) will the public view special effects created with the aid of computer-controlled cameras and models.⁵⁻⁷ Computer-control is a solution to the problem of repeatability of camera movements for long, intricate shots and movements of the model or objects being photographed. In addition, the automated camera is expected to make some effects possible which were not either physically possible or economically feasible before. Input to the microcomputer-based system may be from a walk-through with the camera or from stored data previously entered via keyboard. At Disney, a cameraman will either manually or electronically operate the camera through the initial shot using a hand-held or small console control unit. Subsequent shots will be repeated automatically from the stored data. At Universal Hartland, designers are using their stand-alone microcomputer system to graphically plan the shots within a scene, calling up stored images of the models, setting model size, roll, pitch, and yaw and grid location together with lens size. At Robert Abel's, with the Evans & Sutherland system, the process is carried one step further: The shots may be played beforehand in real time. For both these systems, the stored data is used to control the microcomputer-driven camera system.

An alternative approach to 3-D animation is computer-generated imagery which eliminates the need to build and manipulate models. This approach was used for a 40-second

sequence in the movie "Futureworld" in which a mask-like image of Peter Fonda's head is seen spinning through space.⁸ While 3-D graphics have been successfully employed in television commercials, the level of complexity and detail required for high-quality, high-resolution feature films currently limits its cost-effective application.

In addition to images, a movie almost always has a musical score and special sound effects. While computer-generated music has not yet been used in a theatrical release, proponents feel that the computer will enable the musician to create scores not otherwise obtainable and that these, like computer synthesized images, will expand the medium of filmmaking. For the time being, however, musical scores for movies are still totally created by composers and arrangers. The use of original music always introduces the possibility of copyright infringement. To minimize the problem at Universal, new scores are translated by an operator into codes which are matched against a stored database of copyright music. Matches exceeding the legally acceptable number of bars are flagged.

Sound editing, like film editing, is a particularly tedious, time-consuming and therefore costly task. The sound editor views a reel of film, noting the sounds and footage required. From a library index, he selects a tentative list of sounds. A technician retrieves the sounds and transfers them to tape. The editor then begins the cutting and modifying process. If the sound he needs is too short he must create a physical loop of the tape so the sound repeats without obvious repetitious characteristics. Synchronizing the sound to the film is literally a cut-and-try process. The assembled edited cuts are mixed down onto a final track and then mixed with music and dialogue. Sound quality is degraded with each transfer from library master to work copy to final mix. The Automated Computer Controlled Editing Sound System (ACCESS) developed by Mini-Micro Systems, Inc. for Neiman-Tillar Associates eliminates manual handling of tape and allows electronic synchronization. It provides immediate availability of sound effects which have been digitized and stored on magnetic disk packs. Sounds may also be modified via computer-assisted controls. While cutting editing time by 80 percent, use of ACCESS has also improved the quality of sound produced. The microcomputer-based system was used for the sound editing of "I Want To Hold Your Hand," "Sorcerer," "The Island of Dr. Moreau" and other feature films.⁹

Final print production involves cutting the original negative, adding special optical effects, and performing color correction. Computers probably first entered the motion picture production cycle in the film processing laboratories which perform these functions. Academy Awards for contributions to movie making that involved the use of computers were first earned by these labs. In 1972 DeLuxe General, Inc. received a Class III (Technical Achievement) Academy Award for a computer system that performs color positive process analysis. Using photographic test results and considering interlayer effects, the system compares sample densities to the laboratory reference densities. In the same year, Consolidated Film Industries received a Class II (Science and Engineering) Academy Award for the devel-

opment of an on-line computerized light valve monitor system.¹⁰ While these systems used minicomputers, MGM Labs has recently implemented a microcomputer system to operate the optical printers and control the firing of the light valves.¹¹

Also at MGM Labs, a system is under development to automatically track and retrieve the myriad of film pieces with which the negative cutter must work. Many hours are spent searching through thousands of feet of film for just the right spot to cut and splice together other cut pieces in building up scenes. Each piece must be carefully labelled and stored for possible later use. A major cost in this operation is the time it takes to search and keep track of all the heads and tails for possible later trimming. The new system will use codes on film to allow automated tracking and retrieval of film segments.

DISTRIBUTION

Long before prints become available, an analysis of where and when to release the film is conducted and advertising campaigns are organized. Computer analysis of revenue and advertising expenditures for previous, similar films by geographical area is used by several studios to help formulate the distribution and advertising plans for new films. Revenue reporting on distributed prints is supported at most studios by online systems. A more comprehensive approach has been taken by Buena Vista Distributors in implementing a microcomputer-based system to automate the following functions: bidding, print control, booking, grosses, box office reports, cash reporting, advertising, and messages. Near the release date of a film, standard letters with specific film details will be produced by the central computer and communicated to the branches for issuance to local exhibitors. Bids received will be entered into the system at the branch offices, and prints assigned based on availability. Previously, branch offices have been limited to the print inventory initially assigned to them. With the automated system, the nearest available print may be located. Revenue reports will be entered daily, providing timely information needed to direct exhibition and advertising. An electronic bulletin board and memo system will aid communication among branch offices and the studio.

EXHIBITION

While theatres make use of data processing for normal business applications, computer technology is not yet used for the actual control of movie theatre operations and equipment. Rather, lights, drapes and projectors operate electronically. Within a year, however, manufacturers like RCA expect to incorporate microprocessors into their advanced projector systems. Eventually we may see computer technology used to provide operational and environmental control in movie theatres as in other buildings and businesses. But even beyond the common applications, the decreasing cost and increasing capability of computers may enable

movie theatres to create total visual and audio environments similar to those available today at special-purpose theatres such as the Space Theatre in San Diego. At this theatre, over 60 pieces of equipment are operated and controlled by microcomputer to create special effects for up to five different shows daily. As a first step towards the expanded theatre concept, but not yet using computer control, Universal is installing special equipment to produce lightning, thunder, and other natural sounds and effects in theatres which will be showing "Weather Wars."⁵

PRESERVATION AND RESTORATION

Eventually (or sometimes very soon) a movie is removed from distribution and stored in a film library. Since film degenerates, there is interest in storing movies digitally to preserve them until actively destroyed. To store a 90-minute, high-quality, color film digitally would require tens of trillions of bits of storage. Data compression techniques exist that might reduce this amount 20-30%, but the storage requirement still remains excessively large for today's technology. At the current rate of advancement, digital storage of films may be feasible within five years.

A film may become damaged during any of the steps described, including storage. Methods of restoration are currently being explored and there is interest in using the computer to analyze previous and successive good frames in order to reconstruct the in-between damaged frames. Similarly, the analysis of good areas within a frame may be used to reconstruct damaged or missing parts. However, computer-aided film restoration must await the availability of digital storage of films or other methods for handling the high-resolution requirement.

Computer-aided restoration has been successfully applied to films transferred to tape. For example, "Gone With the Wind" was reconstructed on videotape from a 1956 Technicolor dye transfer print by Image Transform, Inc. The minicomputer-based system resolved outlines, restored color intensities, and reduced noise.¹²

REDISTRIBUTION

A film never dies—it is just recycled to foreign markets and television. The recycling process takes the film back to the post-production process where the original parts are re-edited to meet television and foreign time, censorship and film size requirements. Residuals must be paid to writers, actors, etc. whenever a film is recycled and this is handled automatically at most studios. Once a film enters the realm of television, another story of automation begins which is beyond the scope of this paper.

SUMMARY

This survey, while not exhaustive, does identify the major areas of current computer usage and the key areas for future

applications in the movie industry. Until recently, computer applications primarily focused on:

1. Batch-oriented accounting support for payroll, costing, residuals, and statistical support for market research;
2. Minicomputer systems for process control;
3. Very limited application of computer graphics for special effects.

Current and planned applications include, in addition:

1. Broader computer use for market research and corporate information systems;
2. Word processing support for script preparation and contracts;
3. Interactive system support for budgeting and scheduling subtasks, for resource information retrieval, for sound editing, and for film processing;
4. Computer graphic aids for set design, storyboarding, and animation with increased use for special effects;
5. Expanded use of on-location reporting systems;
6. Functional expansion of automated distribution systems to include print control, bidding and booking, and electronic mail;
7. Computer control of cameras, projectors, and lab processing equipment;
8. Computer control of set elements for live action filming.

In fact, in 1979 several movies will be released whose creation will have involved the first uses of computers in camera-control, set design, storyboarding, animation, and live action filming.

The one development most responsible for the current growth in computer applications in the movie industry is the microcomputer. For business data processing, it is appearing on stage, on location, and in distribution offices. As part of text editing systems, the microcomputer is now in legal departments and will soon enter the script preparation stage. For equipment control, microcomputers are being used in film processing labs, to operate special effects cameras, and will be used in the near future in projectors. As an aid in scene design, stand-alone microcomputer-based graphics systems are now in use. For live-action filming, microcomputers are controlling parts of sets in synchronization with live performances. The high processing power required to generate images by computer may soon be provided through arrays of microprocessors.

Automated techniques for film editing, storage, and restoration still require further research and development in mass storage and image processing.

In any discussion of computer technology and movie making, the question arises as to the possibility that someday movies will be made without actors or cameras but rather totally by computer. The answer is, I think, an undeniable "yes," but whether movies produced by computer will be competitive in cost and quality to those produced by the traditional process with computer aids remains highly questionable.

CREDITS

As with a movie, this paper is the result of contributions by many people. In particular, the following individuals are acknowledged for their valuable inputs: William Eberly, Walt Disney Productions; Al Jerumanis and Paul McManus, MCA Inc.; Charles C. Tucker, Twentieth-Century Fox; Harold Steintrager, Warner Bros., Inc.; Bob Johnson, Hanna Barbera; Jerry Jeffress and Colin Cantwell, Universal Hartland; Don Miskowich, Robert Abel & Associates; Richard Hollander, Eyewitness, Ltd.; William Dietrich, Mini-Micro Systems, Inc.; Michael Chewey, MGM Labs; Mike Scully, IBM; Toni Shetler, Xerox; and Gary Martins and David Leinweber, The Rand Corporation.

REFERENCES

1. Simpson, Janice, "Studio Cleans Up By Marketing Films. Like Selling Soap," *Wall Street Journal*, June 6, 1978, pp. 1+.
2. Kinney, Harrison, "BOFFO: That's Hollywood for big at the box office. And now it's the computer that's boffo in movieland," *Think*, May/June 1977, pp. 4-9.
3. McMahon, Graham, "Report on Film Scheduling and Costing by Computer," unpublished, May, 1977. For information, write Dr. McMahon, Computer Science Dept., The University of New South Wales, P.O. Box 1, Kensington, New South Wales, Australia, 2033.
4. Crow, Franklin, "Shaded Computer Graphics in the Entertainment Industry," *Computer*, March 1978, pp. 12-22.
5. Barron, Frank, "Universal Building Hartland as Special Effects Shop-Supreme," *The Hollywood Reporter*, October 30, 1978, pp. 1+.
6. "New Camera Designed for Walt Disney Pic," *Variety*, Nov. 17, 1978.
7. Purvis, John, "Levi's to Star Trek: Special Effects Star at Robert Abel & Associates," *Millimeter*, September, 1978, pp. 38+.
8. Sutherland, Don, "How 'Futureworld' Movie Technicians Use A Computer To Recreate 'Life' On Film," *Popular Photography*, December 1976, pp. 106+.
9. Deitrick, William R., "Automated Computer Controlled Editing Sound System," to be published in the *Journal of the Society of Motion Picture and Television Engineers*, 1979.
10. Solow, Sidney, "The History of the Motion-Picture Film Laboratory," *Journal of the Society for Motion Picture and Television Engineers*, July 1976, p. 513.
11. Chewey, Michael, Walter Eggers and Allen Hecht, "Controlling Optical Printers by Microprocessor," to be published in the *Journal of the Society for Motion Picture and Television Engineers*, 1979.
12. Kuttna, Mari, "Computers Behind the Screen," *Sight and Sound*, Spring 1977, p. 85.

The System Architecture Evaluation Facility—An emulation facility at Rome Air Development Center

by N. BRUCE CLARK and MICHAEL A. TROUTMAN

Rome Air Development Center
Griffiss AFB, New York

INTRODUCTION

Military requirements for data processing systems with unusual characteristics to perform specialized jobs have led to research into advanced architectures by the Department of Defense (DoD). Some of the requirements for systems have no counterpart in the civilian industry. Command, control and communications systems are typically complex and must be reliable and available with a high degree of certainty. This places great stress on the development of new data processing systems. The architecture, as the bedrock of all systems, must be continually improved in order to accomplish the increasingly complex software functions now demanded. Spaceborne automated systems simply cannot have an onboard team of vendor maintenance engineers to diagnose problems and replace components; a fault tolerant architecture is needed. Advanced radar surveillance systems provide a tremendous potential for information gathering but must be supported by parallel architectures which are still in the research phase. The DoD is actively involved in research and development of advanced architectures for tomorrow's data processing needs and the System Architecture Evaluation Facility (SAEF) is an example of the use of microprogrammable (and other special purpose) computers to reduce the cost and improve the efficiency of this research.

Direct experimentation with unique hardware architectures is extremely expensive and time-consuming. It is also wasteful of resources, as the prototypes are rarely usable systems and must be discarded. Rather than actually build hardware components, they can be emulated by microprogrammable computers. Emulation is similar to a simulation of hardware, but with an important difference. Software simulation of hardware has existed for years, but traditionally has been limited in its use because of the time versus detail tradeoff. If the architecture is modeled at a very high (or gross) level then that simulation executes very fast. As more and more detail is included down towards the register or gate level of machine design, the simulations become excruciatingly slow, running tens of thousands of times slower than the proposed design will actually execute. The development of computers which are microprogrammable allows a "simulation" to be written in a different way.

Instead of the traditional software programs, the microprograms which determine the actual control signals generated for machine language instructions are modified (or rewritten) to execute a different instruction set, the one for the "simulated" machine. In effect, the microprogrammable computer is molded to look and act like the proposed design at the instruction set (machine language) level. Thus, machine level programs written for the proposed design will execute on the microprogrammable machine.

It is an arguable position that this is still a software simulation, since microprogramming is just a lower level of programming. The difference is that the level of detail being used to describe the target machine is lower than the level it is describing. This is in contrast to using assembly language or a higher order language to simulate the instruction set of a computer. This gives a tremendous advantage in the time versus detail tradeoff, and thus this type of simulation is usually referred to by the special designation of emulation. Well written emulations of most architectures can execute within one order of magnitude of "real-time" for the proposed design. Thus a software function which takes one minute of execution time in the target machine might take 10,000 to 100,000 minutes (one to ten weeks of 24-hour days) on a detailed simulation, but only ten minutes on an emulation. Obviously these figures vary widely depending on the architecture being emulated and the computer being used to emulate, but are representative of the speed advantages gained with emulation over simulation.

SAEF ELEMENTS

To provide the emulation capabilities described, the core of SAEF consists of two microprogrammable computers, the Nanodata QM-1 and the Multiple Microprocessor System (MMS) (Figure 1). Also included is a Goodyear Staran Associative Processor which will aid in evaluating single-instruction stream-multiple-data stream (SIMD) architectures. A larger general purpose computer will be used for the hosting of software tools to be used in connection with SAEF. Finally, all of these elements will be connected via the ARPAnet to facilitate intercommunication.

SYSTEM ARCHITECTURE EVALUATION FACILITY (SAEF)

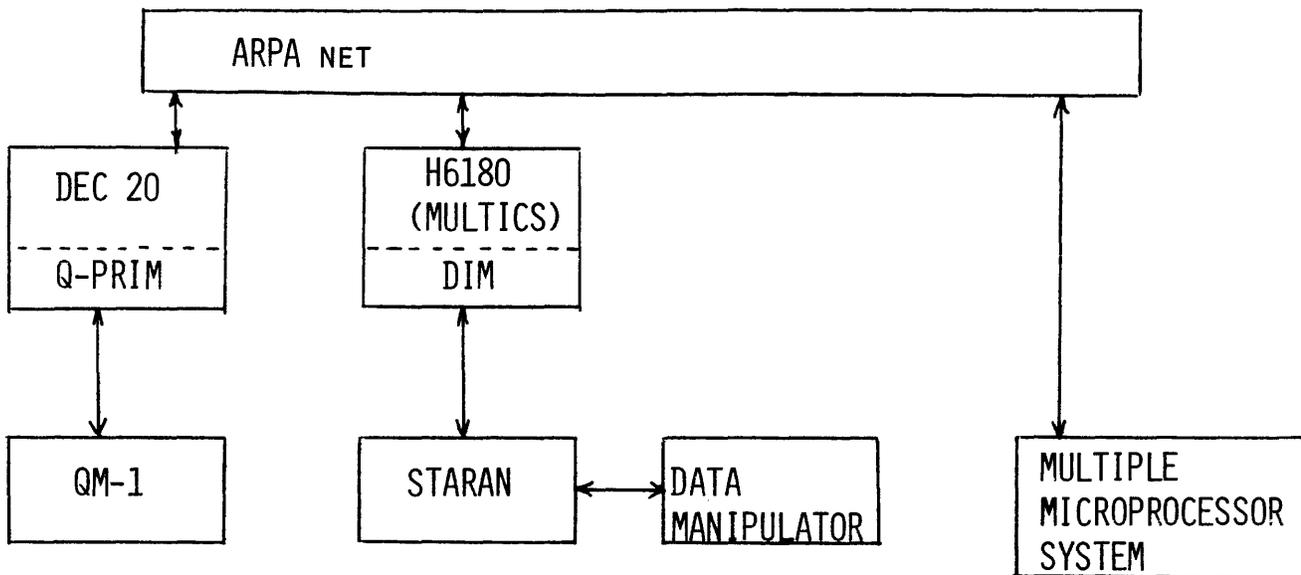


Figure 1—Projected facility.

QM-1

The QM-1 is a high-speed general purpose digital computer that operates under two levels of microprogram control.¹ These two levels provide extreme flexibility in machine definition and allow the advantages of both vertical and horizontal control. Machine instructions resident in Main Store are executed and defined by microprograms in Control Store. Control Store is a vertical level which is in turn implemented by Nanoprograms in Nanostore. The Nano level is a true horizontal architecture which has ultimate control over the total resources of the machine.

The QM-1 is primarily composed of a hierarchy of stores. At the lowest level is Nanostore consisting of 1K of 360 bit words with an access time of 75 ns. One nanoword is made up of a 72-bit K vector and four 72-bit T vectors (only one T vector is in control at any one time). Nanostore is a true read/write memory giving the programmer the ability to dynamically change its contents. Sharing control of the QM-1 with nanostore is F Store. F store consists of 32 six-bit registers which are used for residual control purposes. These registers determine the bus connections between the various units of the QM-1 as well as maintain the state of the machine. Moving away from the low level control of the QM-1, local store consists of 32 18-bit registers. The majority of these registers are general purpose but several of them have specific functions such as microinstruction registers and microprogram counters. External store is a group of 32 18-bit registers which provides specific functions including I/O interfacing, special main store addressing and generation of addresses for interrupts. Control store is a 16K by 18-bit read/write memory having an access time of 75 ns. This memory can be used for data storage and target register

storage in addition to the vertical microinstructions. At the next level up is main store consisting of a maximum 256K 18-bit words. This is a read/write core memory having an access time of 750 ns.

The QM-1 contains several other functional units which are not considered part of the store hierarchy. These include a full 16-function 18-bit ALU, a 36-bit double shifter/shifter extension, an Index ALU for fast indexing and logical operations on local store, an RMI unit for rotating/masking/indexing the output of main store, and an ALU for operating on the six-bit F store.

The QM-1 is operable in both a stand-alone mode and in a time share mode connected to a DECsystem-20. In stand-alone, the QM-1 supports a full complement of peripherals. An operating system is available which maintains control over these devices as well as providing editor, assemblers and other useful routines. Also included are complete emulation debug and support packages which are independent of the operating system. These packages provide simple interfaces between an emulation and QM-1 resources and allow highly interactive sessions between an emulation and its user/developer. While in a stand alone mode, the QM-1 is directly usable by a single user thru the system console. When several users wish simultaneous access to the QM-1 it can be operated in a time-share mode.

In time-sharing the QM-1, it is connected to a DECsystem-20 via a common main store and the DECsystem-20 I/O bus. This system, which is known as Q-PRIM, provides an interactive microprogrammable environment in much the same way as when the QM-1 is stand-alone.² However, in this case the QM-1 is treated as an I/O device by the DECsystem-20 operating system, TOPS-20. In this mode the QM-1 will have no peripherals of its own but will rely on TOPS-

20 to provide all its I/O capabilities. This resource will also be available to remote users because of the ARPAnet connection to the DECsystem-20.

The Q-PRIM software consists of four major modules. These are the QM-1 supervisor or "microvisor," the TOPS-20 QM-1 driver, Q-PRIM Exec and Q-PRIM debugger. The QM-1 microvisor interacts directly with the user's emulation and the TOPS-20 QM-1 driver. It is a small module which communicates between the QM-1 and the DECsystem-20. The microvisor provides context switching capabilities and handles the virtual memory addressing and paging from the QM-1 side. Accessing the QM-1 from TOPS-20 processes is done through the QM-1 driver. The driver communicates with the microvisor and TOPS-20 system calls. It is responsible for initializing the microvisor, controlling, scheduling, and swapping users, accumulating accounting data, and passing along I/O requests. The Q-PRIM Exec provides an environment on the DECsystem-20 which supports each of the emulations executing on the QM-1. The Exec provides a variety of commands that allow a user to build and interact with his emulated system. The Q-PRIM debugger is a table-driven, interactive, symbolic debugger that permits a user to debug target-machine programs in terms of symbols defined for the target machine. Data representations are controllable, thus allowing the user to tailor the emulation interface to more closely match the target machine.

MMS

Another key component of SAEF will be the MMS which is currently in the design phase. The MMS will consist of 64 microprogrammable microprocessors each operating autonomously or connected as part of an SIMD or MIMD architecture. It will contain a highly flexible and versatile interconnection system under software control which facilitates communication between MMS processors. The MMS will be able to effectively emulate shared memory, bus oriented, and crossbar switch interconnection schemes used in distributed multiprocessor systems. Control over the MMS will be accomplished by a Facility Control Processor (FCP) which is expected to be a minicomputer. This system will be usable in both a stand alone mode with the user communicating directly with the FCP, and in a remote mode via its connection to the ARPAnet. For the purpose of this discussion the MMS can be broken down into four sections—(1) FCP, (2) Emulation Engine Support, (3) Processing Elements and (4) Memory Subsystem.

The primary function of the FCP is to maintain control over the operation of the MMS. The FCP will provide both user and ARPAnet interfaces to the MMS. It will contain a host of run time tools which will allow the loading, modification, and control of individual PEs. Other support tools will include microassemblers, assemblers, compilers and software packages for processing of performance data. In its job of control over the MMS, the FCP is aided by the emulation engine support.

Emulation engine support is broken into four areas. The Time Align Controller maintains master pseudo time for the MMS. The Emulated Local I/O Processor will create an I/O

environment for each individual PE. The job of the Shared Resource Controller is to manage memory and communication paths. Last, the function of the Performance Monitor Processor is to collect all Performance Monitor System (PMS) event data from the PEs and emulation support and store this data on mass storage for processing by the FCP. Each of these four devices communicates with controllers which are distributed among the PEs.

Each of the 64 PEs in the MMS will consist of a microprogrammable microprocessor and control hardware for I/O, memory, pseudotime, and messages. The microprocessors will be composed of microstore and an RALU based on bit slice architecture. A 16-bit RALU is the most likely size, with hardware aid for more efficient emulation of smaller word size architectures. Emulation of larger machines will be done with multiple instruction cycles. The I/O and memory controllers work in unison to provide an environment with memory mapped and I/O space I/O, local memory and shared memory. The pseudotime controller coordinates with the master time align controller for keeping PEs in step and the message controller handles communication between the local I/O memory unit and the appropriate emulation support processor.

The memory subsystem is partitioned into 64 each 32K word blocks each associated with a particular processing element. Individual blocks may be subpartitioned in any manner desired between local and global memory. Arbitration for the memory is handled by a portion of the shared resource controller and a local arbitration unit. The memory was partitioned in this way so as to give each PE fast access to 32K local words. Nonlocal accesses will be slower, because they take place through a shared bus.

The MMS as described allows for very detailed system emulations. In addition to emulating the computer architecture and peripherals as usual, one also has the capability to emulate the exact protocols of interprocessor communication and memory accessing. This is made possible by the programmable nature of many of the controllers located throughout the MMS. These features also enable the efficient emulation of I/O devices and virtual memory because the microprogrammable microprocessors are not burdened with these tasks, they can actually be done in parallel by the programmable controllers.

STARAN

Although not an emulation machine, a Goodyear Aerospace Corporation STARAN S-1000 associative processor interfaced to the HIS 6180 Multics system is included in SAEF as an aid in evaluating SIMD architectures. The associative processor can be operated in two modes, a stand-alone mode and an on-line mode to the Multics time-sharing system. In the latter mode, a Multics user is able to control the STARAN from his terminal as he would if he were using the STARAN in stand-alone mode. He can create program and data files using the capabilities of Multics and transmit them to STARAN. Currently the associative processor cannot be time-shared; that is, only one user at a time may utilize the STARAN. All communications between

STARAN and Multics are via a 12-bit parallel buffered I/O channel.

The STARAN basically consists of a conventionally addressed control memory for program store and data buffering, four associative memory arrays, a control logic unit for sequencing and decoding instructions from control memory, and a control logic unit associated with a special parallel I/O (PIO) capability. The associative array memories are the heart of the STARAN system. The array memories provide content-addressability and parallel processing capabilities. Each array consists of 65,536 bits of multi-dimensional access (MDA) memory organized as a matrix of 256 words by 256 bits with parallel access to up to 256 bits at a time in either word (horizontal) direction, bit-slice (vertical) mode or mixed mode (combination of the two). In addition to the MDA memory, each array contains 256 bit-serial processing elements. These processing elements provide the parallel processing capabilities for each array. Processing in the STARAN system can be overlapped with some arrays performing I/O while others are executing arithmetic and logic instructions.

The sequential control portion of STARAN consists of a PDP-11/20 minicomputer with 8K of memory and associated peripherals. The sequential processor also contains logic to interface with other STARAN elements. It runs system software programs such as the assembler and macro preprocessor, operating system, file handling programs, diagnostic programs and debugging routines.

Data manipulator

Another element of SAEF is the Data Manipulator which provides a flexible bit manipulation capability. The basic approach follows that described by Dr. Tse-Yun Feng of Wayne State University. Currently the Data Manipulator is attached to STARAN and allows the programmer to establish a relationship between input and output words such that, for each of the bit positions in the output word, any bit location in the input word may be specified as its data source. In addition, both input and output data can be masked.

Host computers

In order to provide many of the support tools required by SAEF a larger host computer must be included. At the present we will be using the Honeywell 6180 Multics and DECsystem-20 time share systems located at RADC. These two computers will provide capabilities otherwise unattainable on the other elements of SAEF, either because of their small size or specialized nature. Hosting tools on a common computer also has the added benefit of reducing the number of operating systems the user has to learn. This is a primary concern as ease of use is the most important factor for SAEF. Because these hosts provide multiprogramming environments, several users of SAEF may be working on some aspect of a system emulation concurrently. Other obvious advantages are access to the ARPAnet and the amount of

mass storage available on these computers. The host computers will communicate with the remainder of SAEF through the local ARPAnet connections.

Progression of SAEF

SAEF as described above will be developed over the next several years. At the present, SAEF consists of the DEC-system-20 and HIS 6180 both connected to the ARPAnet, the STARAN and Data Manipulator with their connection to Multics, and the QM-1 in a stand-alone mode (Figure 2). Multics is the primary support host with its Meta Assembler, compilers, and editors. A 1200-baud serial line exists from Multics to the QM-1 for downloading purposes. The DEC-system-20 currently supports a preliminary PRIM system utilizing a resident simulation environment instead of emulation by the QM-1. The Q-PRIM system is expected to be operational near the end of 1979. The MMS is currently in the design phase and is projected to be built by the end of 1981.

SUPPORT TOOLS

The hardware elements and software directly supplementing those elements are the core of SAEF. Several additional software support tools are in being or currently under development for use in SAEF, but are not exclusively limited to the facility and may, in fact, be most beneficial in contexts other than SAEF. Specifically, this section discusses the development of a hardware description language called SMITE for writing emulations, and study on the concepts of an automatically retargetable compiler which will accept machine descriptions written in a hardware description language like SMITE and produce an emulation of the machine.

Inherent in the design requirements for any usable item, be it a facility such as SAEF or any of its individual support tools, is its ease of use. No tool, no matter how vital, will be consistently and easily used if it is poorly interfaced with

SYSTEM ARCHITECTURE EVALUATION FACILITY (SAEF)

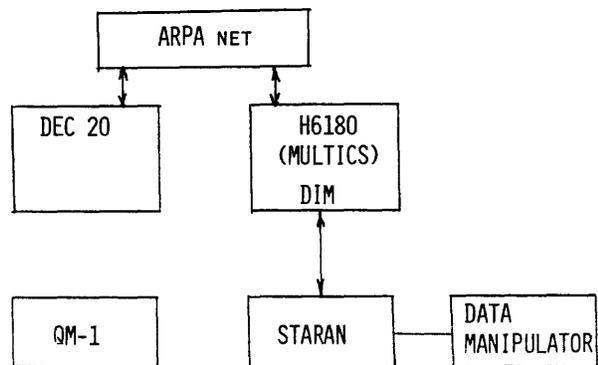


Figure 2—Current facility.

the human link. The necessity to describe machine architectures is a valid research requirement, but the reality of writing an emulation in microcode, where typical productivity is a fraction that of assembly language programming, seriously impairs the utility of the facility. In order to overcome this problem, the development of a "high-order" language for machine description has been undertaken at TRW under contract from RADC. The result is Advanced SMITE (Software Machine Implementation Tool for Emulation), an ISPS based language which allows machine descriptions at the register transfer level to be compiled into microcode for the QM-1.⁶ To support SMITE, a software system resident on the QM-1 is necessary. This system, called SASS (SMITE Applications Support Software), uses an augmented instruction set which is a superset of MULTI, the vertical level instruction set of the QM-1 normally considered to be a "native" instruction set, although it is in itself defined by the nanostore instruction set. SASS is a modification of the vendor supplied run-time package called TASK/PROD. SASS is resident on the QM-1, while the SMITE compiler is written in Fortran and resides on a CDC-6600 located at the Air Force Weapons Laboratory in New Mexico. Compilation of source code is accomplished remotely through the ARPAnet, after which the object code is transferred to the Honeywell 6180 (via file transfer protocol on the ARPAnet) and then to the QM-1 for execution. By the summer of 1979, a new version of SMITE with language enhancements will be delivered written in PL/I and installed on the Honeywell 6180 (Multics) system at RADC. The advanced version of SASS will provide interactive debugging and performance monitoring capabilities not now possible.

To illustrate the basic syntax of the SMITE language, the following example describes a trivial eight-bit-wide machine consisting of three registers, 64 words of memory, and four instructions:

```
EXAMPLE: PROCESSOR;
DECLARE MEM(0:63)(0:7) MEMORY,
        ACC(0:7) REGISTER,
        PC(0:7) REGISTER,
        IR(0:7) REGISTER;
DO FOREVER;
  BEGIN;
    IR←MEM(PC);
    PC←PC+1;
    CASE IR(0:1);
      ACC←MEM(IR(2:7));      "LOAD  ACC"
      MEM(IR(2:7))←ACC;     "STORE  ACC"
      ACC←ACC+MEM(IR(2:7));  "ADD   ACC"
      ACC←ACC+1;            "INCR  ACC"
    END CASE;
  END;
EXAMPLE: END;
```

Once the description of an architecture has been implemented on a microprogrammable computer, there will be a need to write software for that emulated machine. In the same manner that support tools are necessary for writing machine descriptions, tools are necessary for applications

software (the word applications is used to distinguish from the emulation software, or machine description, even though the "application" may be an operating system for the emulated machine). If no support tools are available, the system designer is thrown back to the dark ages of writing machine code for an emulated machine! Cross assemblers make this situation slightly more bearable, but a need exists for a compiler which would automatically compile to the object code of the machine described to it. It has been feasible to rewrite the back end of a compiler for new models of machines as they evolve, but the use of emulation and a high-level language like SMITE means that "new" machines are available from perturbations of the emulations due to fine tuning a design. The several man-months' worth of effort required to modify a compiler is clearly unacceptable.

Recognizing the need for such a compiler, RADC has contracted for the development of a support tool to be called the Retargetable Compiler. As an interim, users of SAEF are writing applications software in the assembly language of the emulated machine and using the Meta-Assembler, developed by McDonnell-Douglas, to create the object code.

RESEARCH AREAS

SAEF is currently projected for use in two distinct areas of research. The most obvious research is into unique machine architectures for special purpose data processing systems and requires no further elaboration. The existence of SAEF also provides for a different type of research which may best be described as the implementation of the "Software First" concept. In the development of computer systems, it has traditionally been necessary and expedient to separate hardware and software functions early in order to define the "machine" and begin work building it. Since a major portion of the cost of systems involved hardware, the software was considered of secondary importance, and was developed to fit the machine. The advent of microprogramming and the tremendous decrease in the percentage of cost devoted to hardware implies that the software now can (and should) be designed to fit the problem being solved and the hardware is then molded to fit the software necessary for that problem. The concept of "Software First" has now been made possible through emulation of hardware and systems development can now be accomplished in a much more orderly and logical fashion. Research into systems development is being conducted at RADC under the name of Total Systems Design (TSD) Methodology.⁷

The TSD Methodology represents a departure from the traditional concepts of computer systems development. Instead of initially dividing the system into hardware and software subsystems and developing each independently until the integration phase, TSD encourages design of the system independent of the ultimate realization of individual functional elements.

The flow of the TSD Methodology breaks into three major divisions. The first portion addresses the general area termed requirements definition. Next is an area of detailed analysis which takes the requirements definition and results in allocated functions implementable in hardware and soft-

ware. Finally, there is an expression of the design process which uses emulation as a substitute for actual hardware until the system is validated to an extent justifying hardware acquisition.

REFERENCES

1. *QM-1 Hardware Level User's Manual*, Nanodata Corporation, Mar 1976.
2. Britt, B., A. Cooperband, L. Gallenson and J. Goldberg, *PRIM SYSTEM: Overview*, ISI/RR-77-58, University of Southern California, Information Sciences Institute, March 1977.
3. *STARAN Users GUIDE*, GER-15644, Goodyear Aerospace Corporation, Aug. 1973.
4. *RDT&E System/Equipment Manual (Handbook) for Installation, Operation and Maintenance of Data Manipulator*, W. W. Gaertner Research, Inc., June 1975.
5. Feng, Tse-Yun, *The Design of a Versatile Line Manipulator*, RADC-TR-73-292, June 1973.
6. *SMITE Reference Manual*, RADC-TR-77-364, Nov. 1977.
7. Clark, N. B., "The Use of Emulation for Total System Design," *Proceedings of 1978 Summer Computer Simulation Conference*, July 1978, pp. 812-814.

Teaching and research experiences with an emulation laboratory

by STEVEN F. SUTPHEN

University of Alberta
Edmonton, Alberta, Canada

INTRODUCTION

User-microprogrammable computers have been generally available since the early 1970s, although in the past few years they have become quite popular. The primary reason for the increased popularity is the decrease in price made possible by technological advances in high-speed memories. Also, the computer manufacturing industry is looking towards microprogramming for increasing throughput of large operating systems, which cannot be replaced because of the large investment for them in software.

Our experiences with a microprogramming or emulation laboratory have come from two points of view—research and instruction. The research has explored several areas including emulation of computers, microprogramming languages and emulation of language machines. The main emphasis in instructional aspects is to illustrate the problems of microprogramming and the difference between it and conventional programming by practical examples programmed in our emulation laboratory.

To provide some insight into our use of the laboratory, a short history of its development will be given. This is followed by our experience with the laboratory in both instructional and research applications.

DEVELOPMENT OF THE EMULATION LABORATORY

In 1971 the Department of Computing Science laid the groundwork for acquiring a minicomputer laboratory for undergraduate instruction.¹ This laboratory was specified to have many diverse computers including one with microprogramming capability. The Microdata 1600 was selected, since it was small and cheap and fit in well with the rest of the laboratory. The machine was purchased in 1972 with 8K bytes of mainstore, 512 16-bit words of writable control store (WCS), and an ASR Teletype as an I/O device. Subsequently, another 512 words of WCS have been added, and an interface to a digital (eight-bit parallel) cassette has been built.

When the department was formulating the mini-lab it was envisaged that a medium sized machine microprogrammed

to emulate all of the minicomputers would be acquired. A special grant was obtained in the spring of 1973, and the QM-1 was selected in the fall of that year; the only other choice was the Burroughs B1700 which was rejected for reasons of cost, restrictions on usage and the fact that it did not support interrupt-driven I/O. The department ordered the QM-1 initially with 32K of mainstore, 3K of control store, 256 words of nanostore, and support peripherals including tapes, disks, a CRT console terminal, a line printer and a card reader. Over the last five years the system has grown to that illustrated in Figure 1.

In late 1973 a Varian V73 was acquired. Although that computer was intended mainly for use in the mini-lab it was purchased with 512 words of WCS, 8K 16-bit words of mainstore and an ASR 33. Since then a dual floppy disk unit has been added and software support developed in-house. The V73 has been used in the minicomputer class with good success and, since the floppies were added, has been utilized in the microprogramming course.

TEACHING WITH THE EMULATION LABORATORY

The Computing Science Department offers a graduate-level course (CMPUT 512—Advanced Minicomputer Systems) which deals primarily with microprogramming. Typically, 10 to 15 students enroll in the course, which covers the structure of emulators and the topics outlined in the text,² along with a brief introduction to the current microprogramming research being done in this department as well as at other sites. In the following discussion the method of using the machines will be mentioned, as well as the learning aspects gained from programming each machine.

Several assignments to teach the fundamental theories and practices of microprogramming have been developed. The main criterion for microprogramming assignments is an algorithm which references mainstore, employs several conditional branches and is familiar, repetitive and computationally simple with easy to check results. Several examples of basic assignments are producing a count of the number of one bits from locations 'A' to 'B' in mainstore; generating a parity bit for a word; and the assignment referred to in the following discussion, sorting into ascending order mainstore

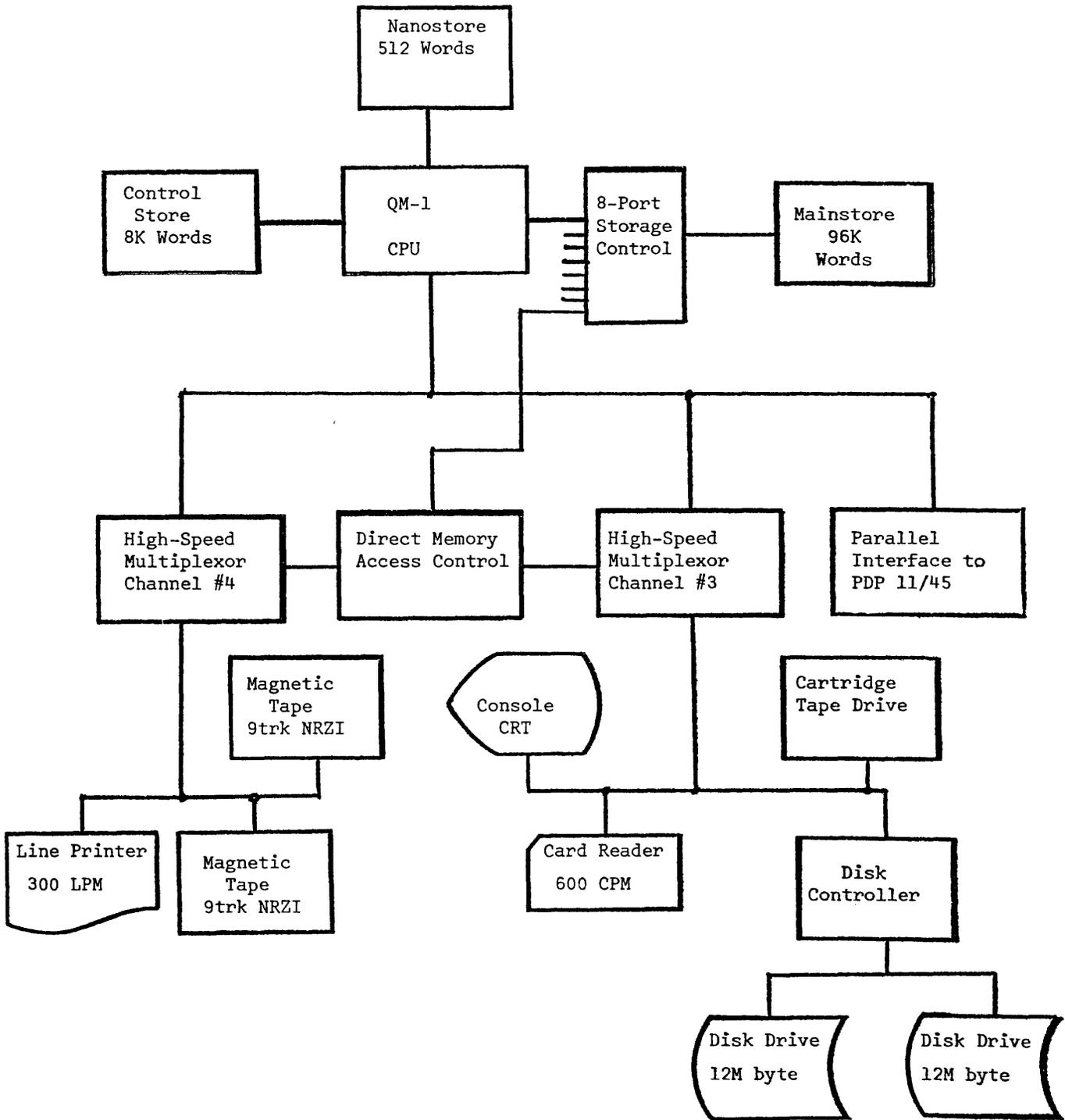


Figure 1—QM-1 configuration.

from 'A' to 'B.' The parameters 'A' and 'B' were to be entered at run-time, as operands for a 'sort' opcode, or in the case of the QM-1, read in from the console terminal. The assignment illustrates the following concepts associated with microprogramming—mainstore accessing, parameter passing, and complex condition testing.

Microdata 1600

The Microdata 1600³ has a vertical micro-instruction word of width 16. Internally the busses are eight bits wide, and there are 16 internal file registers. To minimize the complexity (workload) of the assignment, the students were not

required to program the Microdata to print the results, but rather a system utility was invoked to verify the correctness of the results produced by their sorter.

The scenario, used by students to accomplish the assignment, was to edit and cross-assemble the source on the university's service computer, punch the binary (usually quite small) onto paper tape and load it into the Microdata using the utility program AROS. They would then proceed to debug the program by inserting halts (or using the address-compare stop feature of the 1600 front panel) at critical points, single-stepping through small sections, correcting the results and re-editing the source. There are a few important points in the previous technique which should be noticed. A service computer was used to generate the binaries, even though an assembler exists on the Microdata. This is because it utilizes the various components as they were intended, the Microdata to microprogram, and the service computer for software development. Others⁴ have also found a service computer helpful in increasing throughput on a microprogrammed computer. The second point concerns the usage of the front panel for debugging microcode. Even though a simulator exists it is not only awkward to use, as are most simulators, but also slow to load from paper tape—which can be circumvented through new cassette support. With the exceptionally good front panel support given to micro-debugging, the students did not find the simulator worthwhile.

Besides the basic microprogramming concepts just discussed, the assignment illustrated the following problems—multi-precision arithmetic as 16-bit words were sorted; the largest negative number causes complex condition testing in the sort; simple parallelism as mainstore fetches/stores can be overlapped with other processing; and simple timing problems (the 'U' register is finicky).

In addition to describing the general architectural features of the Microdata, the lectures included a discussion of I/O and used a teletype echo microprogram as an example. Concurrent I/O (a poor man's DMA) was discussed in relation to the small amount of hardware required to implement this relatively powerful concept (similar to the IBM 360/50 channel implementation).

As a machine for teaching elementary microprogramming the Microdata 1600 is very good. The format of its instruction set is quite close to that of conventional machines (as are most vertical microprogrammable machines), it is a general purpose machine and yet it illustrates some of the elementary problems in microprogramming.

Varian V73

The Varian V73⁵ has a horizontal micro-instruction word of width 64. The fields which make up this word have up to four levels of encoding. The branching capability is very general, although ordering control store words to take good advantage of the branching is quite complex. The assignment was, once again, to sort 16-bit words of memory between specified (variable) limits.

Most students prepared source tapes offline on a service

computer with an editor, file system, and other useful utilities such as cross-reference programs. The sources were then transferred to the Varian where they were read from paper tape with the micro-assembler, MIDAS. A modified version of Micro-util was then used to load, execute and debug the binaries. As there is very little front panel support for microprogram debugging, the debug portion was quite trying for the students. The front panel is an I/O device which must be supported in microcode before any internal registers may be displayed. On the Microdata quite the opposite is true; the front panel is an intelligent piece of hardware requiring no software support. Varian intended microprograms to be debugged from an attached processor which could step, trace, etc. the V73 micro machine.

Notice that once again the students used service machines to develop the source code. Most likely the reason for not using the program preparation facilities on the Varian is because they are awkward to use (paper-tape-based). This could have been rectified by adding several thousand dollars worth of main store, disks, tapes and printers to the V73 and using Varian's operating system (VORTEX or MOS), but then the system would be so large that it would not be cost-effective to allow individual students hands-on experience. The assembler supplied by Varian is quite primitive (the service computer's editor helped the students use mnemonics), and a more powerful one has been developed elsewhere.⁶

The Varian illustrates the multi-way branch (for opcode decoding), horizontal microprogramming, and the difficulties of using conventional program design methods for designing microprograms. Also the independent I/O control store and processor are a unique feature discussed in the lectures.

Nanodata QM-1

The QM-1⁷ offers both vertical (control store) and horizontal (nanostore) microprogramming. The higher-level control store is a fully readable and writable general-purpose memory whose word width is 18 bits. Eighteen bits is also the width of mainstore, the internal registers, and the ALU and shifter (although there is also a 16-bit mode). The lower level nanostore is a fully writable, executable memory whose word length is 360 bits. Two 72-bit fields are activated (from the five possible) during each 75 nsec machine cycle. The architectural organization is very parallel, allowing several functional units to perform simultaneously. The QM-1 was designed as an emulation tool, and as such has no native (or most efficient) instruction set. Support software executes on a NOVA emulation, and although reasonable peripherals are attached to the QM-1, the operating system is quite primitive by today's standards.

Since NCS (Nanodata Control System) only supports one terminal it is not time/cost-effective to allow online entry of student programs. Therefore, once again, a service computer was used to produce the source statements for the assembler, read them onto disk, assemble, load and debug them on the QM-1. We have found offline source preparation

so useful that a high-speed interface to a time-sharing PDP 11/45 is being built.

In the class assignment on the QM-1, the students were to program the sort algorithm in nanocode, and output the resulting sorted array onto the console terminal. The I/O portion was microcoded in the MULTI⁸ micro-instruction set. The combination of these elements taught the students the interfaces between the three major stores in the QM-1, and the parallelism and flexibility of the QM-1.

The QM-1 performs very well as a teaching tool. The very horizontal architecture (and parallelism) of the nanomachine illustrate two major theoretical research areas in microprogramming—optimization theory and high-level microprogramming languages. Also, the two-level micro-instruction structure provides a methodology for generating efficient emulators (nanocoding the instruction set for run-time efficiency and microcoding the I/O and console functions for programmer efficiency).

EMULATION RESEARCH

Although three microprogrammable computers are available for emulation research, the QM-1 has been used for most research projects. Quite likely the preference for the Nanodata machine is a reflection of its flexibility as a universal host and the fact that it has a disk file system. The emulation research performed at the University of Alberta can be subdivided into three major areas—emulation of hardware computers (conventional instruction sets), tools for developing emulations (microprogramming languages) and language machine emulations. The current status or results of these projects are summarized in the following sections: the reader is referred to the papers referenced for details.

Conventional machine emulations

A PDP 11/10 emulator has been constructed and evaluated for the QM-1.⁹ This emulator would be a class 'A' emulator, as defined by Flynn,¹⁰ except that it does not check for odd PC values, or handle the trace trap bit in the PS. These features were not included in the emulation only because the implementor did not feel that their usefulness outweighed the overhead involved in the nanoprograms. The PDP-11 emulator successfully executes standard instruction diagnostics, memory, tape and disk exercisers, and also the DOS-11 and MINI-UNIX operating systems. No changes were made to these programs; the MINI-UNIX operating system ran successfully the day it arrived.

After the PDP-11 emulation was thoroughly debugged and MINI-UNIX was obtained, a profiling feature was added to instrument operating system efficiency studies. This program counter-sampling allows one to find where a system is spending the majority of its time. By nanocoding a few identified functions, we have reduced the execution time of a benchmark from over 16 minutes to under six minutes. Further studies with the profiling mechanism need to be done to tune the emulation to an operating system.

Multiple concurrent emulations, sharing a common micro-coded I/O section, have been investigated¹¹ on the QM-1. The QM-1 has been found suitable as a host for multiple emulations, but several problems are yet to be resolved. Device-sharing is the major problem, especially with devices such as magnetic tapes—how does the emulator control program determine when an emulator is finished with the tape? The micro-operations passed to the emulated controller are too small to determine the intentions of the emulated system. (Open and close calls would be required to give the ECP enough information to know when it may allocate the drive to another emulator.) Successful experiments have been performed with dual Nova emulators with a manual task switch facility, sharing the console terminal, disk and clock.

Microprogramming languages

Research has been done on the design and implementation of high-level microprogramming languages for both the QM-1 micro-instruction set (MULTI) and the lower-level nanoprograms. A nano-level language presents difficult problems to the language designer, as illustrated in the Lizard language.¹² After examining the problems of building efficient nanocode, the researcher concluded that an automatic translator would not be cost-effective compared to human nanocoders. Although this result is somewhat discouraging and shows that theory and practice are sometimes disjoint, we have not terminated our research in this area and have achieved better results at the micro-level.

CQ,¹³ a high-level microprogramming language based on the programming language C (produced at Bell Telephone Laboratories), produces code in a slightly extended MULTI instruction set. The compiler (including a MULTI assembler and linker) for CQ is being developed on a time-sharing PDP-11 using the UNIX operating system. It is believed that when this system is operational it will provide much better software development tools than the current QM-1 system has to offer.

Language machines

Language machine research within the department is being done at two levels—high-level language machines, and intermediate-level language machines. APL is the target language for the high-level language machines. A general plan for implementation has been drawn up,¹⁴ and the indexing portion has been coded¹⁵ on the QM-1. Also, research is being performed on the multi-user scheduling portion of the system.

Intermediate-level languages are those languages which fall between high-level languages (FORTRAN, Pascal, APL, COBOL, . . .) and conventional assembler language. These languages are intermediate in both syntax and semantics. An example would be the Pascal 'P' machine for which a microcoded interpreter has been written on the QM-1 by UCSD. Our research group has formulated a methodology

for evaluating intermediate language machines (machines which interpret/emulate intermediate-level languages).¹⁶ To date the methodology has only been shown feasible by modeling on a service computer; actual experiments will be performed on the QM-1.

CONCLUSION

Throughout the previous discussion the notion of using a service computer to support an emulation machine frequently appears. This is quite reasonable when the computers are thought of as tools used to build an emulator. The two functions, development and execution, could be combined on one computer (as Nanodata has done on the QM-1), but this leads to inefficient or primitive development tools, or to host machines which are not universal (for example the Varian).

Having surveyed instructional and research usage of an emulation laboratory our experience indicates that a suitably supported universal host provides an excellent vehicle for exploring many areas. An important area is the relationship between high-level languages (or algorithms written in them) and the instruction sets (architectural machines) which these languages are translated into (or interpreted by).

REFERENCES

1. Marsland, T. A., and J. Tartar, "A Course in Minicomputer Systems," *ACM Sigscce Bulletin*, Vol. 5, No. 1, February 1973, pp. 153-156.
2. Agrawala, A., and T. Rauscher, *Foundations of Microprogramming*, ACM Monograph Series, Academic Press Inc., New York, 1976.
3. Microdata Corporation, *Microprogramming Handbook*, Second Edition, Santa Ana, California, 1972.
4. Vickery, C., "A Microprogramming Design Laboratory," *ACM Sigmicro Newsletter*, Vol. 7, No. 1, March 1976, pp. 34-49.
5. Varian Data Machines, *Varian Microprogramming Guide*, P/N 98 A 9906 072, Irvine, California, 1973.
6. Persson, M., "Design of a Microprogram Generator for the Varian V73," *ACM Sigmicro Newsletter*, Vol. 8, No. 4, December 1977, pp. 14-20.
7. Nanodata Corporation, *QM-1 Hardware Level User's Manual*, Second Edition, Williamsville, New York, March 1976.
8. Nanodata Corporation, *MULTI Micromachine Description*, Williamsville, New York, 1976.
9. Marsland, T. A., and J. C. Demco, "A Case Study of Computer Emulation," *INFOR*, Vol. 16, No. 2, June 1978, pp. 112-131.
10. Flynn, M. J., "Classes of Emulators," *ACM Sigmicro Newsletter*, Vol. 8, No. 4, December, 1977, pp. 34-35.
11. Demco, J. C., *Principles of Multiple Concurrent Computer Emulations*, M.Sc. Thesis, Dept. of Computing Science, University of Alberta, Edmonton, August 1975.
12. Salomon, D. J., *A Sequential Language for Nanoprogramming the QM-1*, M.Sc. Thesis, Dept. of Computing Science, University of Alberta, Edmonton, Sept. 1976.
13. Demco, J. C., "CQ—A High-Level Microprogramming Language," Interim Report, Dept. Computing Science, University of Alberta, Edmonton, 1977.
14. Adams, W. S., "Implementation of APL on the QM-1," personal correspondence.
15. Nielson, W., "APL Indexing Mechanism for the QM-1," M.Sc. Project Report, Dept. Computing Science, University of Alberta, Edmonton, Sept. 1976.
16. Adams, W. S., J. C. Demco and S. F. Sutphen, "A Methodology for Intermediate Language Machine Comparison," Tech. Rep. 78-4, Dept. of Computing Science, University of Alberta, Edmonton, Oct. 1978.



Simulating the delay in logic networks for large, high-speed computers

by E. A. WILSON

Honeywell Information Systems
Phoenix, Arizona

When designing a computer with TTL logic circuits, the delays of logic paths have been estimated by considering the number of gate delays and adding in load and media factors. Such a simplistic approach is not accurate enough for calculating delays when designing high-performance large systems using high-speed, non-saturating circuits such as HCML (Honeywell's Current Mode Logic). There are several reasons:

- The clock (cycle) time is considerably faster for a high speed machine, hence the calculations must be very accurate in order to meet performance goals.
- The loading on the driving gate varies with the number of driven gates, hence affecting the rise time of the line (interconnect) voltage.
- The geometry of the interconnect (branch points, connectors, various media impedances) has an effect on signal propagation with high-speed edges.
- Media delay is a significant percentage of path delay as ICs become faster.

All of the above leads to a need for an ability to simulate the delay for proposed interconnects which do not meet simple driver-line-load configurations without intermediate branch points and/or media changes.

The large package programs which are available on the open market are well suited for circuit design work when developing the circuit set, but they are unsuitable for the multitude of cases which must be run when simulating the full design. They require too much memory, take too long to execute, and are too general when the same gate can be used for every case simulated.

This paper presents a simulation program and a design methodology which have proven successful in our actual large-computer design environment. The simulation program has been tailored to the HCML circuit set and optimized for small storage, very fast execution and minimal data input. Also, the program has been written so that automatic or manual modes of operation are available.

In the automatic mode, the network checking program (which checks the logic designer's data base for correct pin assignments, wiring rule violations, etc., but is not a part of this paper) feeds the data into the delay simulation program

and receives back the interconnect portion of the delay for each load gate. The delay data are added to the logic designer's data file and reported to him when he accesses the file for the results of the network checking run.

In the manual mode, the designer selects an interconnect which has a problem (such as an unexpected long delay) or inputs a proposed interconnect for which he wants to calculate the delay before continuing his design. In this mode, he has several options for the output. He may select just the individual interconnect delays for the loads; a printout of driver and load voltages with time; or a plot of the waveforms of the driver and load voltages. In addition, the load voltages may be printed/plotted as either the input voltage to the load gate, or the output voltage from the load gate. By using one or more of these options wave reflections, effect of input rise on load delay, turn-on/turn-off/turn-on, etc. may be observed and frequently the problems corrected by interconnect modification before the design is released. If interconnect modifications do not correct the problem, then either an alternate logic implementation may be used, or the problem may be compensated for in the rest of the logic chain. In any case, the simulation provides the designer with the information before the design is released and built as hardware.

The simulation program uses a two-model approach with an interaction between the models similar to substructuring in finite element programs, except in this case the interaction is a function of time.

INTERCONNECT (LINE) MODEL

This model is based on a finite element rather than a mesh or loop current formulation which actually only affects the terms in the resulting capacitance matrix, as will be seen. A basic goal was to minimize the execution time and past experience has shown that if this goal is kept in mind from the beginning, a more efficient program can be written than would be if the theory were developed and then the programming tacked on as an independent activity. Therefore, a method (finite elements) was chosen for the theoretical model which was known to lead to a straightforward matrix model.

The basic current voltage relations for a line element are:

$$\frac{\partial e}{\partial x} = -iR, \quad i = -k \frac{\partial e}{\partial x} \quad (1)$$

$$\frac{\partial e}{\partial x} = -L \frac{di}{dt}, \quad i = -m \int_0^T \frac{\partial e}{\partial x} dt \quad (2)$$

$$e = S \int_0^T i dt, \quad i = c \frac{\partial e}{\partial t} \quad (3)$$

where in equation (3), the voltage, e , is relative to ground (or some fixed reference).

In vector form (although this is only a one dimensional problem, it does not hurt to be mathematically correct), the divergence of the current at any point along the line element yields:

$$\nabla \cdot i = c \frac{\partial e}{\partial t} \quad (4)$$

and using the second relations of (1) and (2),

$$k \nabla^2 e + m \int_0^T \nabla^2 e dt = c \frac{\partial e}{\partial t} = c \dot{e} \quad (5)$$

with the boundary conditions

$$i \cdot n = -(k \nabla e + m \int_0^T \nabla e dt) \cdot n \quad (6)$$

where n is an outward vector from each end of the line element.

Equations (5) and (6) may be combined in a variational equation as:

$$\begin{aligned} \delta(\text{Integral}) = & \int (k \nabla^2 e + m \int_0^T \nabla^2 e) dt - c \dot{e} \phi dx \\ & - \int (i \cdot n + (k \nabla e + m \int_0^T \nabla e dt) \cdot n) \phi db \end{aligned} \quad (7)$$

where ϕ is an approximation function for e and (*Integral*) will have to be minimized. The term db is an increment of the boundary, which in this case consists of the two ends of the line element.

Using Green's theorem,

$$\begin{aligned} \delta(\text{Integral}) = & \int (-k \nabla e \cdot \nabla \phi - m \int_0^T \nabla e \cdot \nabla \phi dt - c \dot{e} \phi) dx \\ & - \int (i \cdot n) \phi db \end{aligned} \quad (8)$$

where e has been assumed to be piecewise continuous in time. Since ϕ is a variation of e ($\phi = \delta e$), the integral becomes

$$\begin{aligned} (\text{Integral}) = & -\frac{1}{2} k \int (\nabla e)^2 dx - \\ & \frac{1}{2} m \int \left[\int_0^T (\nabla e)^2 dt \right] dx - \frac{1}{2} c \int \dot{e} dx + \\ & \int (\text{input current}) e db \end{aligned} \quad (9)$$

Now assume a function for e of the form

$$e = a_1 + a_2 x = [1 \ x] \{a\} = [E] \{a\} \quad (10)$$

In the terms of the end (nodal values),

$$e = [E][A^{-1}]\{V\} \quad (11)$$

where

$$[A] = \begin{bmatrix} 1 & 0 \\ 1 & l \end{bmatrix}, \quad l = \text{line element length}$$

$$\{V\} = \text{vector of nodal voltages}$$

then,

$$\nabla e = \frac{\partial e}{\partial x} = [0 \ 1][A^{-1}]\{V\} \quad (12)$$

Placing the above into equation (9) and taking the variation with respect to the nodal voltages yields;

$$\begin{aligned} \delta(\text{Integral}) = & 0 = \\ & -k[A^{-1}]^T \int (\{\nabla\}[E])^T (\{\nabla\}[E]) dx [A^{-1}]\{V\} \\ & - m \int_0^T [A^{-1}]^T \int (\{\nabla\}[E])^T (\{\nabla\}[E]) dx \\ & [A^{-1}]\{V\} dt \\ & - c[A^{-1}]^T \int [E]^T [E] dx [A^{-1}]\{\dot{V}\} \\ & + \{\text{input current at nodes} = I\} \end{aligned} \quad (13)$$

where a subscript T means the transpose of a matrix, which is of the form

$$[K]\{V\} + \int_0^T [M]\{V\} dt + [C] \frac{\partial}{\partial t} \{V\} = \{I\} \quad (14)$$

and the matrices are given by

$$[K] = kl \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (k = \text{mm/ohm}) \quad (15)$$

$$[M] = ml \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (m = \text{mm/nanohenry}) \quad (16)$$

$$[C] = cl/6 \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad (c = \text{nanofarads/mm}) \quad (17)$$

If a mesh current formulation had been used, the capacitance matrix (17) would have had only diagonal terms instead of the true distributed line capacitance which is inherent in the finite element approach.

LOAD (GATE) MODEL

The particular gate used in this paper is a CML circuit; however, the load model does not have to be confined to any single type of circuit set since the model in the preceding section can be used with any load which uses a voltage input as a boundary condition. This will be more fully explained in the "Substructure" section.

The seven-node model which uses an Ebberts-Moll model for the transistors is shown in Figure 1. The voltages V_1, \dots, V_7 are unknown and vary as V_m varies. The node (subscript) numbers are specifically chosen to reduce the

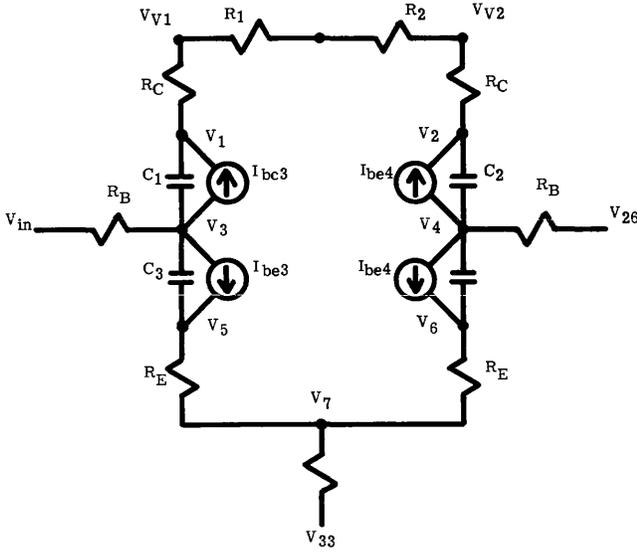


Figure 1—Model of Honeywell Current Mode Logic gate.

number of arithmetic operations required for solving the simultaneous equations (a seven-by-seven matrix will be obtained from Figure 1).

The basic equations for the transistor model are:

$$B_N = \frac{1}{B_N} + 1 \quad (18)$$

$$B_I = \frac{1}{B_I} + 1 \quad (19)$$

$$I_I = I_{cs} \exp(\theta_I V_{bc}) \quad (20)$$

$$I_N = I_{es} \exp(\theta_N V_{be}) \quad (21)$$

$$C_{be} = a_1 N_1 / (\phi_1 - V_{be}) + \theta_N T_{CN} I_N \quad (22)$$

$$C_{bc} = a_2 N_2 / (\phi - V_{bc}) + \theta_I T_{CI} I_I \quad (23)$$

$$I_{be} = (I_N - I_{es}) B_N - I_I + I_{cs} \quad (24)$$

$$I_{bc} = (I_I - I_{cs}) B_I - I_N + I_{es} \quad (25)$$

The values for a_1 , a_2 , ϕ_1 , ϕ_2 , I_{cs} , I_{es} , θ_I , θ_N , T_{CN} , T_{CI} , B_N , and B_I are experimentally determined from actual circuits.

In relation to Figure 1,

$$C_{be} = C_3, C_4 \quad (26)$$

$$C_{bc} = C_1, C_2 \quad (27)$$

$$V_{be} = (V_3 - V_5), (V_4 - V_6) \quad (28)$$

$$V_{bc} = (V_3 - V_1), (V_4 - V_2) \quad (29)$$

$$I_{be} = I_{be3}, I_{be4} \quad (30)$$

$$I_{bc} = I_{bc3}, I_{bc4} \quad (31)$$

Writing the nodal equations will lead to an equation of the form

$$[A]\{V\} = \{r\} \quad (32)$$

where the right hand side ($\{r\}$) will be composed of a current vector, a vector containing capacitance terms times nodal voltages, and a vector of known voltages (V_{in} , V_{26} , V_G , V_{33}). The approximation of constant capacitance during each time step (which is due to the finite difference approximation which will be imposed in the next section on equation (14)) is consistent with the interconnect model. However, such an approximation for the current generation is not made since a better approximation is easily achieved.

For example, in terms of the voltage at the beginning of the time step, the current at the end of the time step may be written as

$$I_{bc} \approx -I_{cs} + I_{cs} (e^{\theta_I V'_{bc}} + e^{\theta_I V_{bc}} (V_{bc} - V'_{bc}) \theta_I) \quad (33)$$

where V'_{bc} is the voltage at the beginning of the time step. A similar expression exists for I_{be} . This expression is obtained from the first two terms of a series expansion of the exponential factor.

These expressions for current will contribute terms to the matrix $[A]$ in (32), and also destroy its symmetry. However, since the matrix is sparse, it can be easily solved by hand and the solution can be coded directly into the program. The method of solution follows.

The matrix $[A]$ is of the form

$$\begin{bmatrix} A_{11} & A_{13} & A_{15} & & & & \\ & A_{22} & A_{24} & A_{26} & & & \\ A_{31} & A_{33} & A_{35} & A_{36} & & & \\ & A_{42} & A_{44} & A_{46} & A_{47} & & \\ A_{51} & A_{53} & A_{55} & A_{56} & A_{57} & & \\ & A_{62} & A_{64} & A_{66} & A_{67} & & \\ & & & A_{75} & A_{76} & A_{77} & \end{bmatrix} \quad (34)$$

The vector $\{r\}$ is given by

$$\{r\} = \begin{Bmatrix} I_1 \\ I_2 \\ -I_1 - I_5 \\ -I_2 - I_6 \\ I_5 \\ I_6 \\ 0 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 0 \\ V_{in} K_B \\ -0.26 K_B \\ 0 \\ 0 \\ -3.3 K_3 \end{Bmatrix} +$$

$$\begin{bmatrix} C_1 & & -C_1 & & & & \\ & C_2 & & -C_2 & & & \\ -C_1 & & C_1 + C_3 & & -C_3 & & \\ & -C_2 & & C_2 + C_4 & & -C_4 & \\ & & -C_3 & & -C_3 & C_4 & \\ & & & -C_4 & & & 0 \end{bmatrix} \begin{Bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \end{Bmatrix} \quad (35)$$

where

$$I_1 = -I_{cs} B_I + I_{es} + I_I B_I (1 - \theta_I (V_3' - V_1')) \quad (36)$$

$$-I_N (1 - \theta_N (V_3' - V_5'))$$

$$I_2 = -I_{cs} B_I + I_{es} + I_I B_I (1 - \theta_I (V_4' - V_2')) \quad (37)$$

$$-I_N (1 - \theta_N (V_4' - V_6'))$$

$$I_5 = -I_{es} B_N + I_{cs} + I_N B_N (1 - \theta_N (V_3' - V_5')) \quad (38)$$

$$-I_I (1 - \theta_I (V_3' - V_1'))$$

$$I_6 = -I_{es}B_N + I_{cs} + I_N B_N (1 - \theta_N (V_4' - V_6')) \quad (39)$$

$$- I_f (1 - \theta_f (V_4' - V_2'))$$

and the values of I_N and I_f are functions of the voltages in each current equation.

By using the Crout algorithm, the matrix $[A]$ may be easily modified into a new matrix $[\tilde{A}]$ which permits a simple solution to the seven simultaneous equations.

$$\tilde{A}_{ij} = A_{ij} - \sum_{k=1}^{j-1} \tilde{A}_{ik} \tilde{A}_{kj} \quad i \geq j, j \geq 2 \quad (40)$$

$$\tilde{A}_{ij} = (A_{ij} - \sum_{k=1}^{j-1} \tilde{A}_{ik} \tilde{A}_{kj}) / \tilde{A}_{ii} \quad i < j, i \geq 2 \quad (41)$$

$$\tilde{A}_{1j} = A_{1j} / A_{11} \quad j \geq 2 \quad (42)$$

$$\tilde{A}_{i1} = A_{i1} \quad (43)$$

Then the vector $\{r\}$ is modified by

$$\tilde{r}_i = (r_i - \sum_{k=1}^{i-1} \tilde{A}_{ik} \tilde{r}_k) / \tilde{A}_{ii} \quad i \geq 2 \quad (44)$$

$$\tilde{r}_1 = r_1 / A_{11} \quad (45)$$

and the voltage vector is obtained by

$$V_7 = \tilde{r}_7 \quad (46)$$

$$V_i = \tilde{r}_i - \sum_{k=i+1}^7 \tilde{A}_{ik} V_k \quad i \leq 6 \quad (47)$$

The above solution is applied for each time step of the interconnect model solution and the capacitance values are updated for each time step.

SUBSTRUCTURE ASSEMBLY

Equation (14) must be represented in a finite difference form for calculation purposes. This will be of an implicit, trapezoidal integration form to obtain a better approximation than a simpler explicit, rectangular integration form would provide. The result is that larger time steps may be used, hence faster execution.

This is easily achieved by writing the discreteized form of equation (14) for the voltage at time $T - \frac{1}{2}\Delta T$ instead of time T (the end of the present time step for which the nodal voltages are unknown). This means

$$\{V\}_{i-1/2} = \frac{1}{2}(\{V\}_i + \{V\}_{i-1}) \quad (48)$$

in terms of subscripts (i at T and $i-1$ at $T - \Delta t$). Then, for the D.C. terms,

$$\{I\}_{i-1/2} = \frac{1}{2}[k](\{V\}_i + \{V\}_{i-1}) \quad (49)$$

for the capacitive terms,

$$\{I\}_{i-1/2} = \frac{1}{\Delta t} C(\{V\}_i - \{V\}_{i-1}) \quad (50)$$

and for the inductive terms,

$$\{I\}_{i-1/2} = (\Delta t)[M](\frac{1}{2}\{V\}_i + \frac{1}{2}\{V\}_{i-1} + \sum_{j=1}^{i-2} \{V\}_j) \quad (51)$$

where the factors $\frac{1}{2}$ and $\frac{1}{2}$ come from using

$$\{V\}_{i-3/4} = \frac{1}{4}\{V\}_i + \frac{3}{4}\{V\}_{i-1} \quad (52)$$

for the integration in the half time step from $T - \Delta t$ to $T - \frac{1}{2}\Delta t$.

The resulting finite difference equation is (let $(\Delta t)[M]$ become just $[M]$ and $[C]/(\Delta t)$ become just $[C]$):

$$(\frac{1}{2}[K] + \frac{1}{2}[M] + [C])\{V\}_i = \{I\}_{i-1/2} - \frac{1}{2}[K]\{V\}_{i-1} \quad (53)$$

$$+ [C]\{V\}_{i-1} - \frac{1}{2}[M]\{V\}_{i-1} - \sum_{j=1}^{i-2} ([M]\{V\}_j)$$

Given the above equation, the two models may now become interactive in time.

For the interconnect model, $1/R_B$ from the load model may be included in the conduction matrix, $[K]$, with the gate voltage V_3 as a known boundary condition for the interconnect model. At any given time step, the gate voltage is known for $T - \Delta t$, $T - 2\Delta t$, etc., but not $T - \frac{1}{2}\Delta t$. This is easily resolved by the finite difference extrapolation formula

$$(V_3)_{i-1/2} = \frac{1}{2}(3(V_3)_{i-1} - (V_3)_{i-2}) \quad (54)$$

Defining the subscript R as identifying quantities associated with the gate (i.e. $-R_B$ and V_3) model, equation (53) may now be written as:

$$(\frac{1}{2}[K] + \frac{1}{2}[K]_R + \frac{1}{2}[M] + [C])\{V\}_i = \{I\}_{i-1/2}$$

$$- \frac{1}{2}[K]\{V\}_{i-1} - \frac{1}{2}[K]_R(\{V\}_{i-1} - 3\{V_R\}_{i-1} + \{V_R\}_{i-2})$$

$$+ [C]\{V\}_{i-1} - \frac{1}{2}[M]\{V\}_{i-1} - \sum_{j=1}^{i-2} ([M]\{V\}_j) \quad (55)$$

where $[K]_R$ is null except for diagonal elements to which loads are attached (or terminating resistors if the need arises).

The final boundary condition is the input or driver. This is simply an input current to the first node (I_1 in the vector $\{I\}$) which matches the wave form for actual experimental data.

Once the interconnect model has been solved for time T , the nodal voltage at each load location becomes V_{in} for the load model in Figure 1. The same basic model is used for each load, but the voltages (V_1 , V_2 , etc.) are stored in arrays so that the loads at different positions in the network can be simulated autonomously.

The substructure interaction scheme follows two simple repetitive steps:

1. Impose boundary conditions for time $T - \frac{1}{2}\Delta t$ from current source (driver) and extrapolated load voltages ($V_{R's}$) on interconnect model and solve for nodal voltages at time T .
2. Impose V_{in} boundary conditions from interconnect nodes on load model and solve for gate voltages at time T for all load locations. Then repeat Step 1 for next time increment.

This process is repeated until all of the loads have reached a stable on (or off) condition. The stable condition is defined by a prescribed voltage above the threshold switching volt-

age below which V_{v2} (Figure 1) does not fall once it has been attained.

IMPACT OF MATRIX STORAGE ON EXECUTION TIME

The line model developed in the earlier section cannot be used for unlimited lengths as a single element. For example, a 75 ohm transmission line with a propagation delay of 5 nanoseconds per meter should be limited to a 50 or 60 mm long element. Therefore, long line segments must be broken into shorter elements. This can result in large matrices (200 by 200 for example) if the lines are long and there are many branch points. Such large matrices not only require much storage, but also much computation time. This can be reduced by considering the nature of the interconnect problem. An example is shown in Figure 2. Each line segment would have one or usually more elements. The resultant matrix would be tridiagonal except at the branch points. Since all arithmetic operations outside of the banded portion of the matrix will result in zero, there is no need to either store or operate on these outside elements.

Advantage was taken of the matrix type and a one-dimensional, variable bandwidth storage scheme was developed which stores, hence, operates on, only the affected elements in the matrix. This results in a reduction of storage of almost one order of magnitude and a similar reduction of execution time. This makes the manual version of the program practical since the user gets the results from the terminal within seconds after he types in the data.

SAMPLE PROBLEM

Figure 2 is typical of those types of logic networks which can be simulated with this approach. The four basic media

are the multichip package, the board, the cable between boards, and the connectors. The type of each line segment is given in Table I.

For the sake of simplicity, the plot in Figure 3 shows only the voltage at the source and the output of the gates at loads 3 and 5. Notice the reflections, turn off of load 5, etc., which affect the delay of the network. Such information is extremely important in the early stages of design in order to assure meeting performance goals.

As a contrast, a simple hand calculation based on propagation time would have predicted turn-ons for load 3 at 13 ns and load 5 at 13.3 ns. The difference between the simple assumption and the full simulation (17 ns for load 3 and 35 ns for load 5) is obvious from Figure 3. This illustrates the importance of a full simulation instead of estimates based on run lengths and loading factors which can not take into account reflections caused by mismatched impedances.

STAR CONFIGURATION SAMPLE PROBLEM

While the preceding example demonstrated the complex logic interconnection which can be modeled, a simple problem will help to demonstrate further the need for full simulation.

This problem will be developed from a simple, impractical (in the sense that it would not appear in a real design) star to a more realistic interconnect which may loosely be described as a star.

The simple star is shown in Figure 4 and for the first simulation, all of the lines were treated as board lines instead of using the connector or micropackage parameters. Then the micropackage and connector parameters were used for the appropriate segments and the load output plots are shown in Figure 4 along with the driver output voltage for the case of all three media in the problem. Because all signal

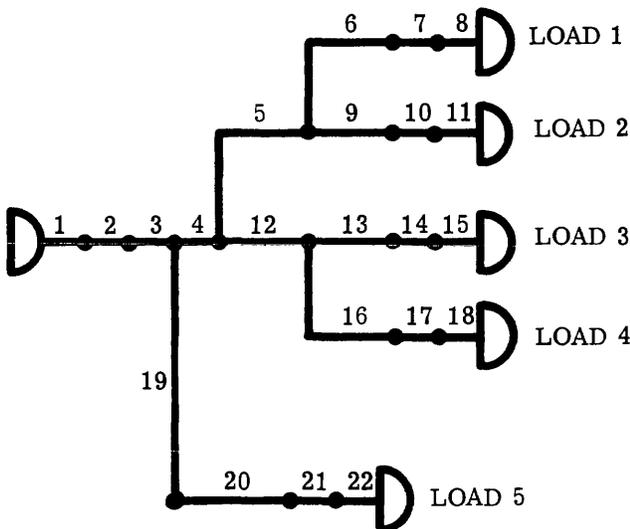


Figure 2—First sample problem interconnect configuration. Numbered line segment types are given in Table I.

TABLE I

LINE	LENGTH (MM)	MEDIUM
1	25	MICROPACKAGE
2	25	CONNECTOR
3	250	BOARD
4	750	RIBBON CABLE
5	375	BOARD
6	250	BOARD
7	25	CONNECTOR
8	25	MICROPACKAGE
9	250	BOARD
10	25	CONNECTOR
11	25	MICROPACKAGE
12	125	BOARD
13	375	BOARD
14	25	CONNECTOR
15	25	MICROPACKAGE
16	50	BOARD
17	25	CONNECTOR
18	25	MICROPACKAGE
19	625	RIBBON CABLE
20	625	BOARD
21	25	CONNECTOR
22	25	MICROPACKAGE

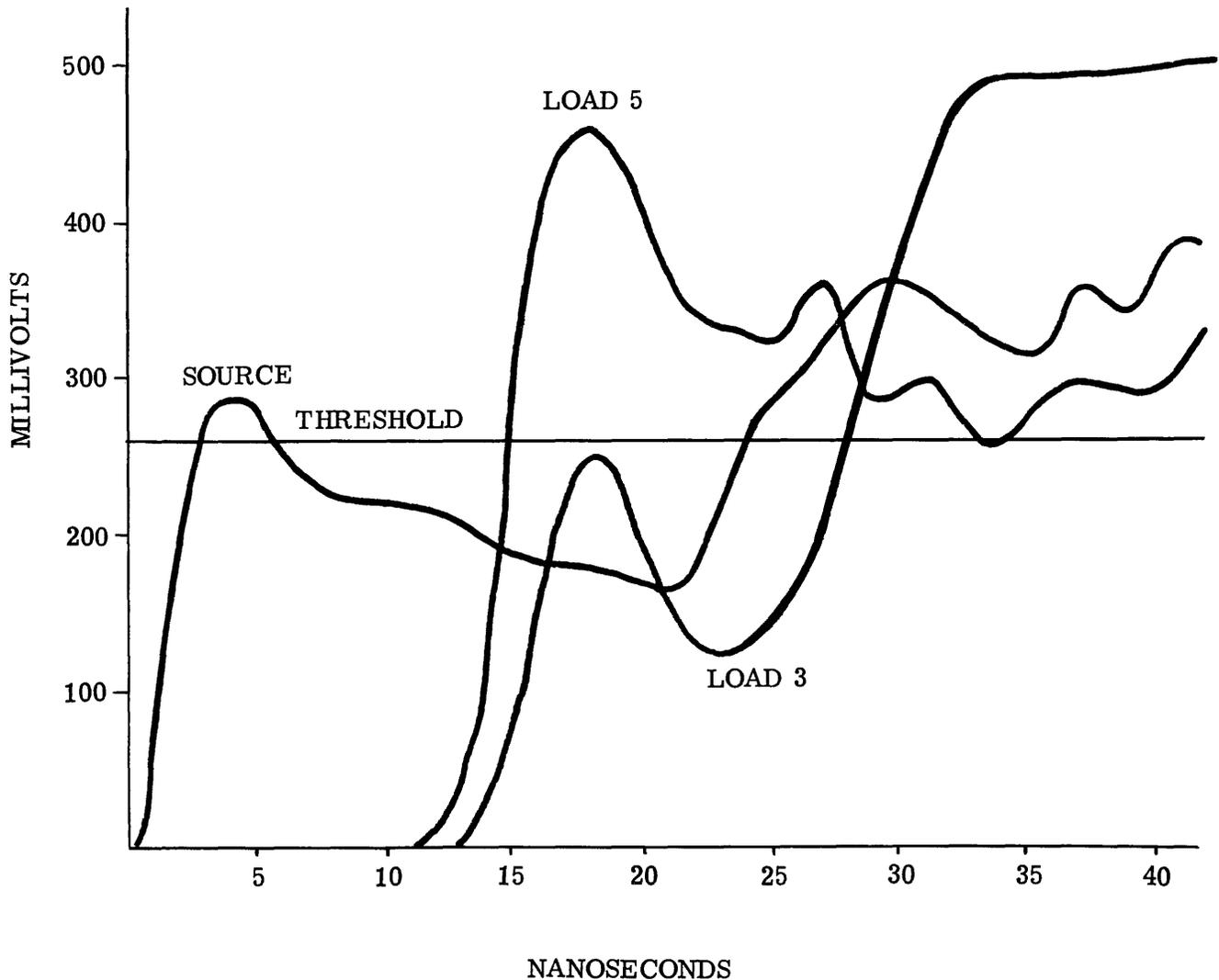


Figure 3—Voltage waveform, plots for the sample problems shown in Figure 2.

paths are the same length and all load factors equal, only one load output plot exists for each case. Of note is the fact that when all media are considered, the delay is three nanoseconds longer than for the all board lines case. This is important because if the difference in propagation speed between board lines and connector or micropackage lines is multiplied by the appropriate line lengths, only .48 nanoseconds can be accounted for in the three-nanosecond difference. The bulk of the difference is therefore due to the variation of line characteristics in the signal paths. This can be noted by the dip in the source (driver) voltage.

The next case also uses the star except that the top load has a load factor of 2.5 (for example, two "high" current gates and a "low" current gate driven in parallel on the same chip or adjacent chips) and the other three loads only have a load factor of 0.5. Note that the total load seen by the source is still four, the same as the previous case. The output plots are in Figure 5 and show a 1.5-nanosecond

difference in delay time. However, the possibly surprising result is that the gates on the more heavily loaded line turn on before the half loads. This is due to the capacitance of the larger loading making the top line less sensitive to the dip in the source voltage which the lightly loaded lines track closer.

A true equal line length star would not likely be found in a real design since board routing and micropackage placement would preclude such an ideal case. The quasi star in Figure 6 is more representative of a real interconnect. The board line lengths are chosen to relate to the previous star configuration. The average distance to the four loads is the same as the previous equal board line lengths. Likewise, the total loading (four) is the same. In Figure 6, only the output of loads one and two and the source are shown for the sake of clarity. Load one turns on 10.5 nanoseconds sooner than for the equal line length case yet the difference in distance only accounts for the signal reaching the load two nanose-

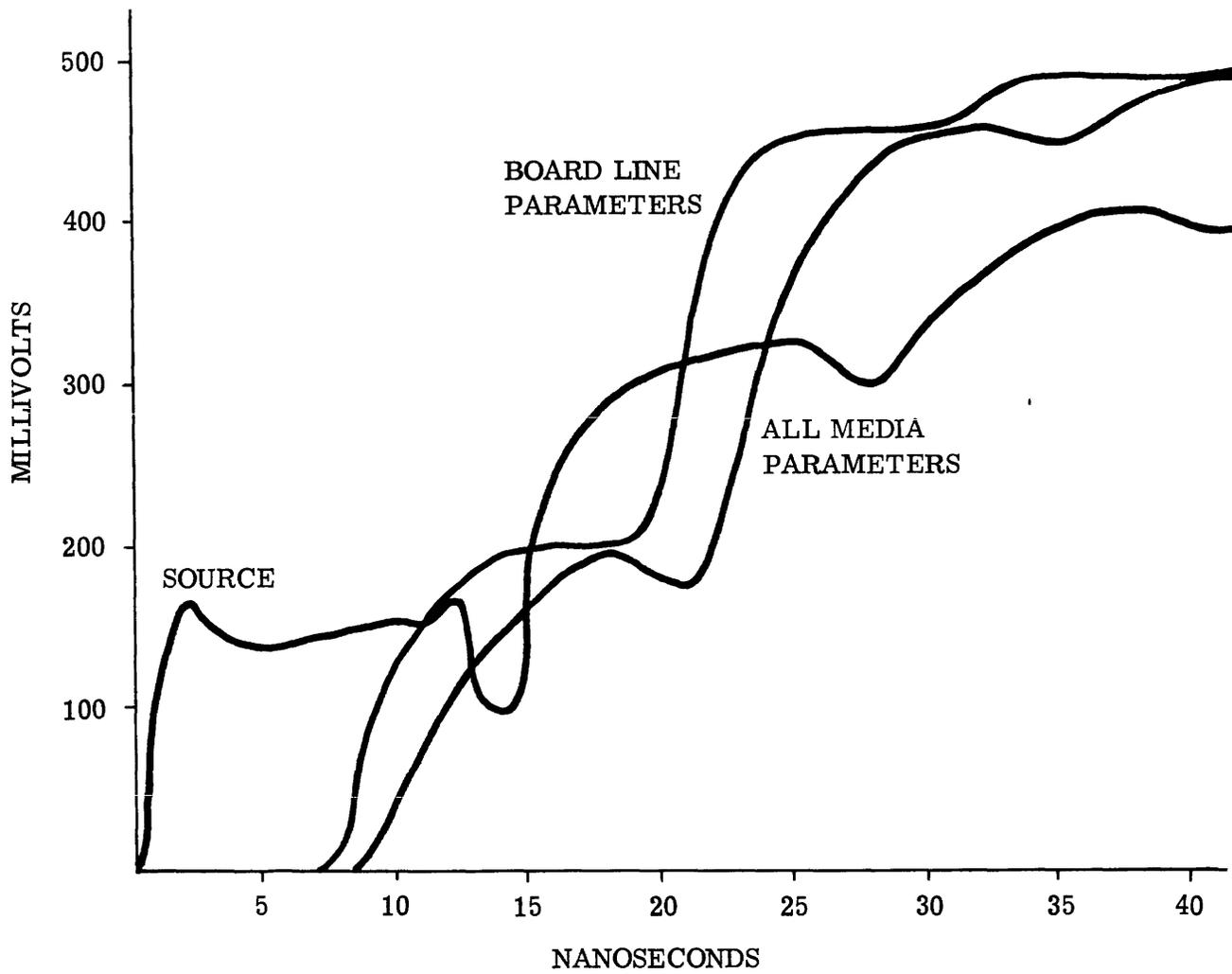
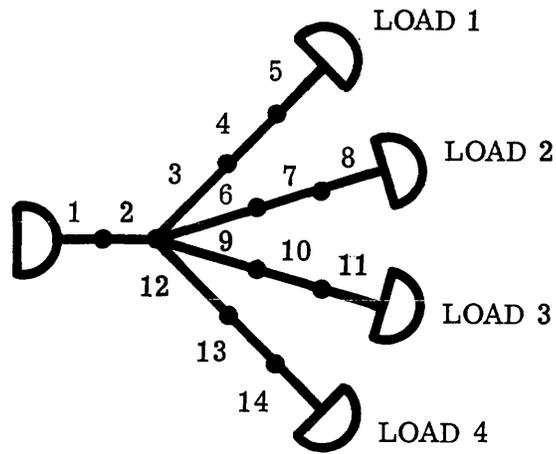


Figure 4—Simple star sample problem. The SOURCE waveform is for the ALL MEDIA PARAMETERS case.

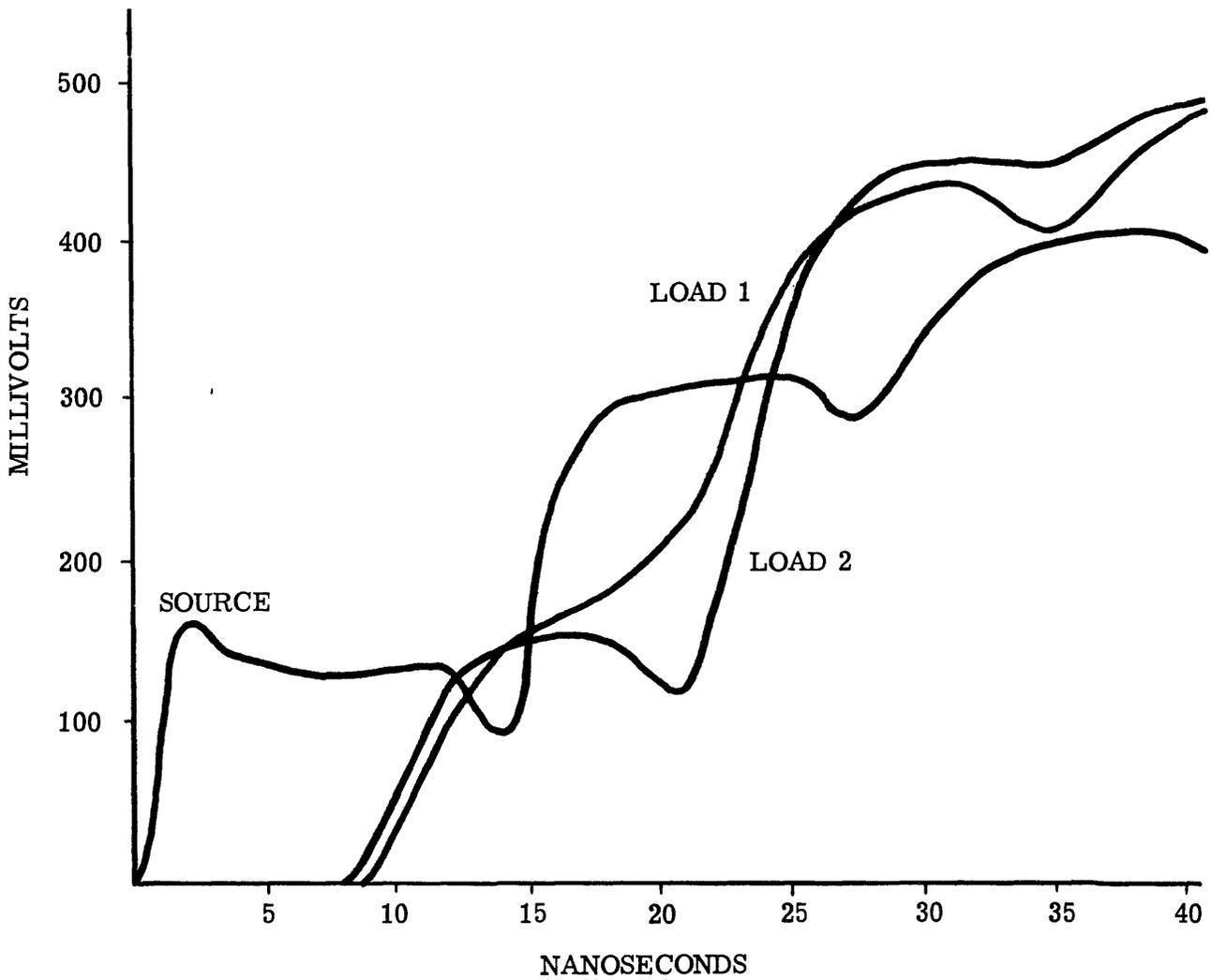
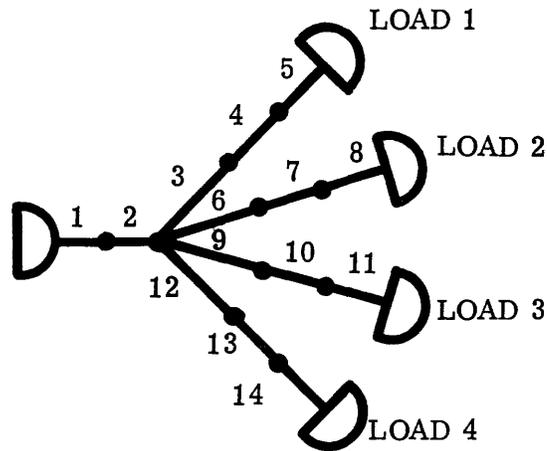


Figure 5—Simple star configuration except LOAD 1 has a load factor of 2.5 while the other loads are only 0.5.

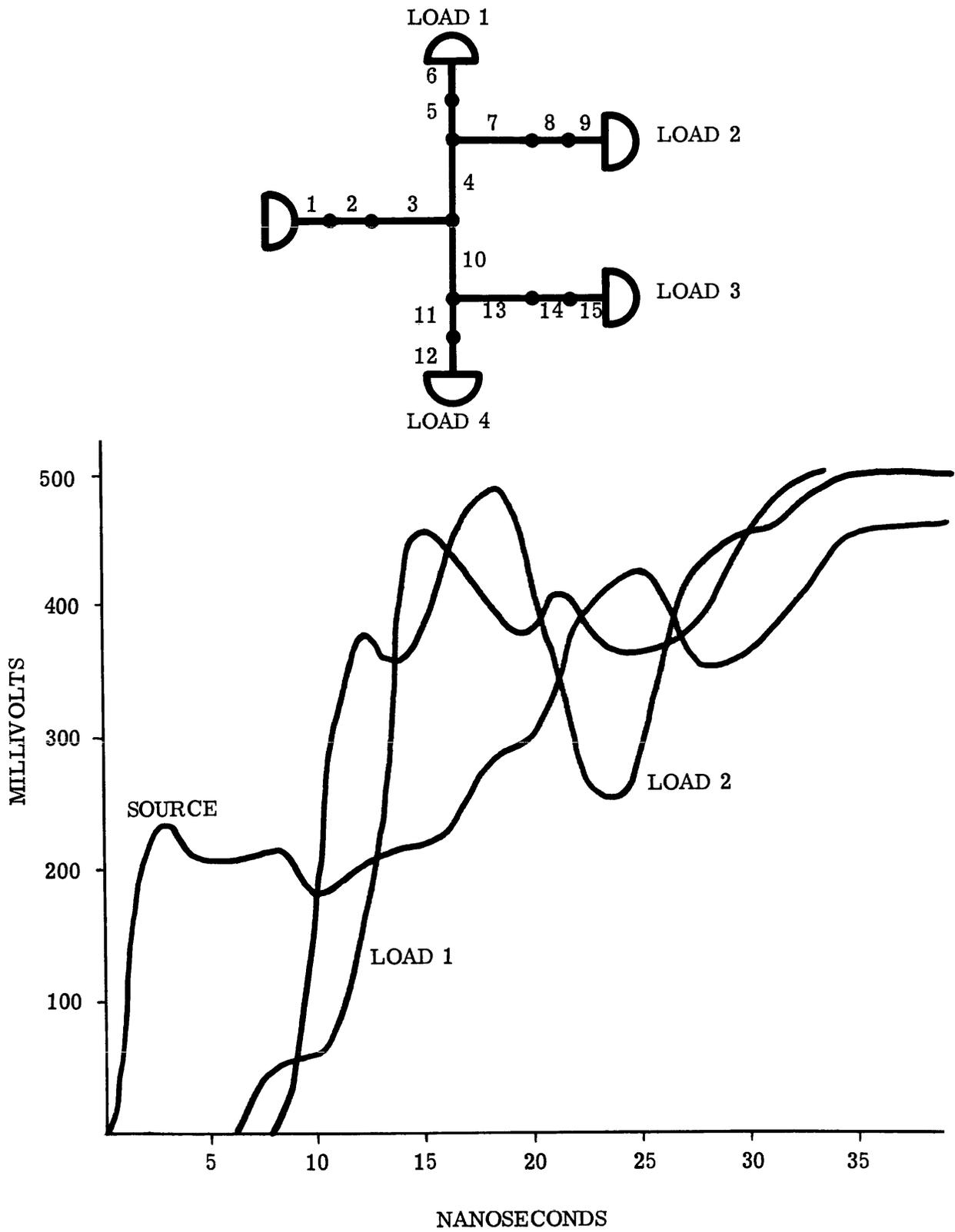


Figure 6—Quasi star configuration. The average distance to each load is the same as for the simple star and all loads factors are 1.

conds sooner. The reason for the faster turn-on can be seen in the source voltage. It reaches a higher initial plateau which is due to the first branch point having only two instead of four branches. The waveform of load two is also noteworthy. Although it turns on before load one, it turns off at 23 nanoseconds and on again at 24 nanoseconds. Hence, 24 nanoseconds must be taken as the true turn-on time since the load is itself a driver for the next step in the logic path and must be stable before turn-on of its loads can be assured.

SIMULATION PROGRAM CHARACTERISTICS

Once the program had been developed, it was checked for accuracy by running several sample problems which were also modeled on a popular circuit analysis program, SCEPTRE. The voltage plots were identical and the only variations were in the third significant digit when the voltage values were printed instead of plotted. An important difference between the two programs is that the version of SCEPTRE used required a minimum of 25k words of storage and approximately one minute of execution time (plot suppressed to reduce I/O time), while the program used for this paper took less than 12k of storage and ran the comparison problems in less than a second (also, plot suppressed). Both programs were run on the same Honeywell Level 66 Computer.

The program can handle up to 100 nodes and line elements

(each line segment may be one or more line elements which are 125 mm. or less) but that may be increased by just a dimension statement change. The first sample problem required only about 50 nodes. The program is written in FORTRAN and may run on any computer with adequate storage.

CONCLUSION

The simulation method presented in this paper is proving to be a useful tool in the early design phases of logic networks for two reasons. First, it provides essentially automatic timing analysis of all designs without any extra work on the part of the designer when the design data base is used for input. The typical execution time of less than one second per simulation run allows the computer-aided design automation system to include a full set of timing analyses without any adverse effect on turn-around time or competition for resources. Second, the designer is free to use more complex branch schemes than could be used if the delay predictions had to be done according to simplified wiring rules. This freedom allows more efficient design and routing while assuring that the result will be predictable. Again, the fast execution and the small storage makes desk-top terminal operation practical. The designer may obtain voltage waveform plots on the terminal as fast as it will type, and then check out design modifications as fast as he can type.

Languages for operating systems description, design and implementation

by PHILIP H. ENSLOW, JR.

Georgia Institute of Technology
Atlanta, Georgia

INTRODUCTION

The transfer of information about operating systems is severely hampered by the lack of suitable means for precise communication. If major progress is to be made in reducing the high cost of developing and maintaining system software, new communication techniques must be fashioned to improve both vertical and horizontal transfer of information about operating systems.

“Vertical information transfer” refers to communication between individuals or groups working at the various levels on the development or use of some specific operating system. Designations of these levels might be specification, description, design, experimentation, implementation, evaluation, use and maintenance. However, these classifications are not all mutually exclusive, nor is this the only taxonomy possible. In contrast, “horizontal information transfer” refers to communication between different projects or between individuals working on an operating system and outsiders (i.e., a new member of the team, another operating system development project, students, etc.). The essential factor characterizing the horizontal flow of information is that the recipient is not already intimately familiar with the total environment of the system under study, and does not want to be forced to learn more about that environment than necessary.

Although some work has been done in support of information transfers along both axes, most of the effort has been focused on vertical flow, with only a small amount of effort devoted to horizontal transfer to a different environment. In only a very few instances has the combined problem been considered. (Examples of the combined approach are References 17, 58 and 59.) Previous work was performed primarily within the framework of problem-solving or systems development and focused on the development of formalized approaches involving a statement of the functional requirements of the system, specifying these requirements in a formal language, similarly defining process interactions, and progressing through various other steps such as the use of a high-level programming language, assembly language, etc. Unfortunately, such work results in the development of different languages for specific purposes without regard to their position in the hierarchy.

From a vertical point of view, the concept of structured programs suggests the possibility of utilizing the same language at different levels of abstraction. This possibility may prove workable when the requirements of only two levels are considered; however, the requirements for efficient horizontal transfer of information may well make such a solution a poor one.

In the vertical direction, the value of a complete operating system language(s) system meeting the characteristics to be described will be primarily economic. The total time, and hence the cost, required to design, implement, test and maintain an operating system will be most directly affected. In the horizontal direction, the resulting effect will be primarily an improved transfer of technology and general knowledge about operating systems. This will of course have an ultimate effect on the economics of operating system development, but the emphasis here is on knowledge transfer. The question has often been raised as to why operating system developments from universities are not utilized better by industry. The availability of a suitable means of accurate, succinct and informative communication would certainly aid this transfer process.

The emphasis of this paper is on the role of the languages as effective information transfer mechanisms, but this orientation is not meant to deny the value of complete software development systems which not only require “good” languages but also the ability to handle the design data bases. (Good examples of discussions of this type of system are References 3 and 50.) The primary thesis of this paper is that the horizontal transfer of information, though certainly as important as the vertical one, has been seriously neglected in most of the work thus far.

LEVELS OF USAGE

The language available at each “level” must provide the ability to “describe” the operating system; however, the purpose of each of these is quite different:

- Specification*—Functional requirements of the system and the criteria of performance in providing those functions;
- Design*—Proposed solutions to meet the specifications;
- Simulation*—A functional model of the proposed system

that can be utilized for experimenting with various designs;

Implementation—The detailed solution of the problem in a form that can be directly translated to a machine-executable form;

Description—A description of the functionality, implementation and performance of the operating system usable for instruction and training, maintenance, demonstration or study.

The relationships of these activities one to another and to the other activities involved are shown in Figure 1.

It appears that none of these is driven totally by either the vertical or horizontal transfer requirements. For example, instruction about an operating system is performed horizontally for general students; it is also an extremely important, but expensive and time-consuming, factor in the training of new individuals on a specific operating system project. Similarly, design and experimentation may be carried on horizontally for instructional or research projects on a specific aspect of operating systems in general or vertically in the analysis of the performance of a specific system. The one level that does seem to have almost total vertical orientation is check-out and test support; however, even here the horizontal influences and features of the system implementation, such as good programming practices, will certainly be felt.

Specification

A specification language should be problem-oriented. It is a statement of the problems to be solved, the functionality

of the system, and the criteria to be met in solving these problems. The specification may also include restrictions on system behavior. What must not be included in the specification are any *unnecessary restrictions on how* the problem will be solved. The specification for an operating system should be organized on the basis of functions provided (i.e., a problem orientation) and should be as non-prescriptive as possible. A specification is also a statement of the *policies* to be implemented in the system but not the mechanisms that will be implemented.

Design

A system design prescribes the general mechanisms that will be utilized to implement the policies and functions set forth in the specification. The major difference between a design and an implementation is in the level of abstraction. The design will not include all the details of the mechanisms implementations. There is, in fact, a large degree of machine-independence present in a design.

Simulation

One well demonstrated requirement of the total design methodology is "the need to test proposed modifications to a complex system before they are made, regardless of how beneficial these changes might appear. The number of times that the system . . . did not react as expected in response to adjustments should serve as a warning."⁴⁵ Although much work has been done on the simulation of operating systems

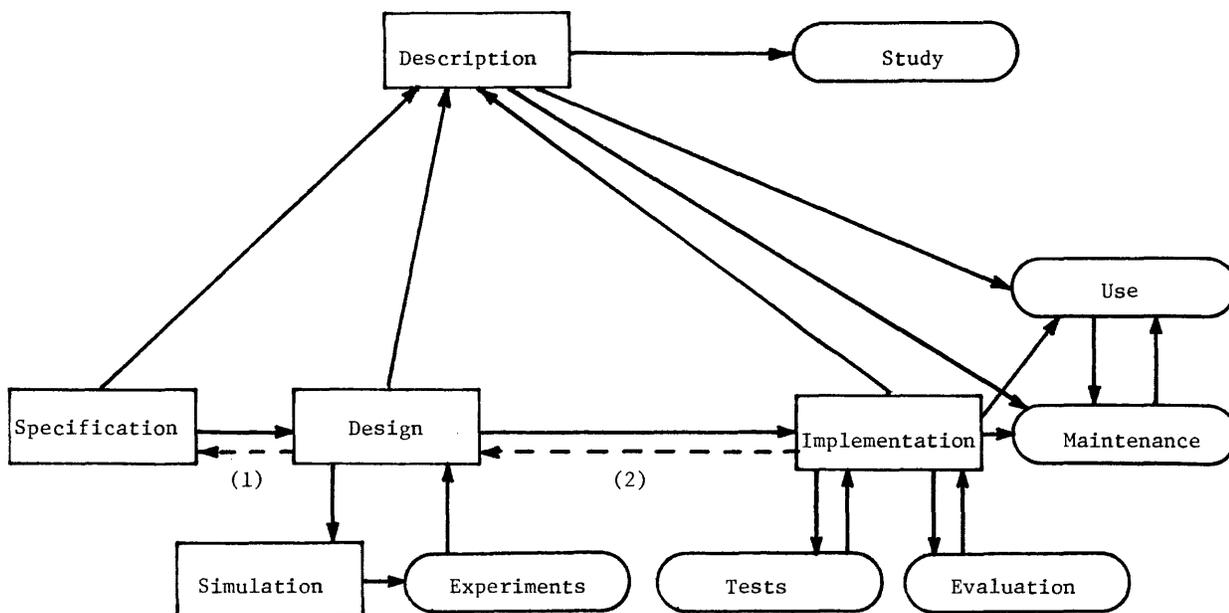


Figure 1—Operation system activities.

1) Changes in the specifications as a result of changes in the design based on simulation experiment result and/or evaluation tests of the implemented system.

2) Changes in the design based on the results of evaluation tests of the implemented system.

and on special languages to facilitate the construction of simulation models, the fact is that the cost of simulation studies of operating systems remains almost as high as the cost of experimenting with the actual system. The reason for this is that there is an almost total lack of continuity or direct transferability between the design description of the system and its simulation description. Although it is shown in Figure 1 as an adjunct activity, the language support for simulation must be unified with the "main-line" activities.

Implementation

A statement of the implementation of the system must be a description containing sufficient detail that it can be executed after appropriate processing by a language translator. The description of mechanisms in the implementation are machine-dependent except to the degree that there may exist some capability for portability of the programming language utilized for the implementation description.

Description

Whereas the problems associated with specifying, designing and implementing operating systems are well recognized, the difficulties encountered in *describing* such a system are *not* so well recognized—nor, as any student of the subject is well aware, have effective techniques been developed to accomplish such a task. What is commonly seen is a description of only one portion of the system, such as the kernel (e.g., References 53 and 72); a specific feature, such as protection (e.g., Reference 7); or the design/implementation policy (e.g., Reference 34). The weakness of our present capabilities to describe operating systems succinctly and completely is evidenced by the few overview papers that have been produced (e.g., References 14, 54, 73). Although each of these papers is informative, none provides a large amount of insight nor the detail necessary for understanding the design and implementation of the complete system. At the other end of the spectrum of descriptions of a single system is the complete coverage of MULTICS.⁴⁸ It is interesting to note that even in non-system specific discussions of a topic such as security, there is no common basis for discussion and presentation. The presentation techniques encountered in open publication range from totally verbal (e.g., References 15 and 44) to theorem-based discussions (e.g., Reference 22) and a combination of theorem and code (e.g., Reference 52).

DESIRABLE CHARACTERISTICS AND CAPABILITIES

A single unified system

It is important that the "language" support provided be a single unified system, not a large collection of separate design and programming aids each applicable to merely one

level. When the various levels are considered, this unification may initially be only a philosophical one; however, the ultimate goal should be physical unification through capabilities to automatically translate the representation at one level into that applicable at the next.

Graphical techniques

Graphical or pictorial techniques have always been extremely useful and powerful tools for explaining the operation of relatively complex systems. In computer systems, such techniques have usually taken the form of flowcharts. Though one experimental study has indicated that flowcharting has little value in *increasing* programmer productivity,⁶⁰ it is nonetheless clear that flowcharts do have a positive value in facilitating information transfer about the logic and sequencing in an existing program. On the other hand, the extreme complexity of operating systems and the inability of flowcharts to portray concurrency work together to greatly diminish the value of this technique for documentation and information transfer with respect to operating systems. A modification of flowcharting called flowgraphs has been presented in Reference 32 as a technique for detailed analysis to include timing studies of real-time systems. However, since the level of detail presented in flowgraphs is quite high, its value for the analysis and description of a complete operating system is also questionable.

Language-based

Formal languages have been studied extensively and have a strong theoretical base for their design and the automation of checks for internal consistency and completeness as well as translation. These characteristics, plus the ability to easily manipulate a collection of statements in a formal language system, are strong arguments for having the support for each level be based on a formal language.

It is possible to modify a formal language definition tool such as the Vienna Definition Language⁷⁰ to support "the formal description of operating system features, like the concepts of parallelism and information sharing."⁴³ However, based on the example applications of this technique, it does not appear that the results are very easy to understand. The problems here are primarily the language notation and semantics.

Ability to express the same abstraction or control structure at all levels

The capability of the language system to perform this task is central to the entire argument for its development. However, basic research is still needed to provide full "abstraction expression" capabilities at even one level. Work is presently being performed on languages that may provide the facilities necessary to fully describe (define) the types of structures used by Dijkstra in the T.H.E. operating system,

but it does not appear that current projects will adequately support the two dimensions of information transfer presented here.

Separation of policies and mechanisms

It is on this point that there might be divergence of goals in considering both horizontal and vertical information flows. From a vertical point of view, an objective might be that "We would like to develop a language mechanism which allows us to capture an *abstraction* and its *implementation* in the program text."⁷¹ However, from the point of view of horizontal flow of information, if the abstraction of the policy and its implementation mechanism are inextricably combined, there is probably too much information being passed, since the boundary between the levels of design and implementation has been removed. There is nothing sacred about these boundaries, nor has it been proved that they are essential; however, the desirability of this characteristic should be examined. At the descriptive level there is no question that one would wish to be able to describe various policies without considering their implementations.

There are several readily-apparent reasons for the separation recommended above. The study of operating systems can be divided into two general classifications; (1) The investigation of the logic and operation of algorithms that control specific functions of the operating system; and (2) The consideration of the various implementations (mechanisms) possible for the logic underlying these policies. In addition to clearly identifying just what it is that is being examined, this fundamental separation directly supports two other objectives of any design and implementation system—a systematic approach to performance analysis (Which is the cause and which is the effect, the policy or the mechanism?) and the identification and cataloging of reusable policy modules even though their implementations might be unique (i.e., target system dependent).

Separation of policy and mechanism should greatly assist in the development of automatic checks of the program developed. The methodology employed for such checks will be different at each level, but all methodologies will have the goals of ensuring consistency and facilitating verification. A full consideration of this requirement is present in the work of several workers at the implementation level.⁶³ However, all of the problems have not been solved at even that single level.

Understandability and intelligibility

William Wulf has commented that "Whatever it is that makes a program understandable *must* be a property of the program text." The problem is one of attaining and retaining understandability for both vertical and horizontal information transfer. A descent of one vertical level might be characterized as the removal of one level of abstraction whereas a horizontal transfer implies a complete understanding of the abstraction at that level. Entering into the discussion

here is the old argument about the use of comments vs. a verbose language (e.g., COBOL). It appears at this time that the total "understanding" required for either direction of transfer should be provided by the program statements themselves; and, since the language itself must support horizontal transfers, the demands made on the information content of the language exceed those made in the past. There is a grave danger that, in attempting to increase the information content of the basic language statements so as to improve understanding, the result will be the development of an extremely complex or artificial notation system that greatly diminishes intelligibility.

Promote (enforce) good programming practices

The promotion—or better, the enforcement—of good programming practices is one of the most desirable characteristics of the complete system from the point of view of actual usage in a vertical manner. Much has been said about the desirability of omitting several types of constructs from the language; unfortunately, much less has been said about what should be included to provide the capability to do "good systems design and implementation;" and research on the enforcement of procedures that will result in quality work is almost non-existent. It appears that the problem is divisible into two general subjects—the organization of the control structures used in the program and the physical organization of the program.

SURVEY OF PREVIOUS WORK

In the past, the primary motivation for language work in this area has been the need to develop tools that will facilitate the evaluation of various features included in an operating system, the selection of parameter values, etc. This is obvious from the number of citations given below in the section on simulation. The area with the next higher level of activity has probably been implementation languages, although only a small selection of references is cited here. There has been some work on design languages, but descriptive systems have received very little attention. The work on languages or programming systems that addresses more than one level of usage has been very small.

Hardware languages

It is not advisable to attempt too close an analogy between software systems design languages and those developed for hardware; however, a few comments on the latter appear to be in order. One is inclined to think of the problem of "common" notation as being much simpler for hardware, but such a situation is not the case. A 1967 review of documentation in use at the lowest (logic gate) design level showed nine techniques then in use.¹¹ The problem was

actually even worse than this diversity of systems would seem to indicate. The motivations for the use of higher-level languages or machine manipulatable notations for hardware are almost the same as they are for software—vertical and horizontal information transfer plus automatic translation and expansion to take care of the simple repetitive tasks, error and consistency checks, and interfaces with automated systems that carry the design process through the next logical step.

An example of a notational system that was originally designed to support horizontal information transfer is ISP.² Of particular interest is the later work on ISP that focused on expanding its use along the vertical dimension with complete, automatic translation support down through simulation and implementation. ISP corresponds to a lower-level of language hierarchy discussed above; however higher-level languages have not been neglected. Some early work was on the development of an Algol-type language,⁵ and today there is much activity within a group known as the Conference on Hardware Design Language (CHDL). Of particular note is the sentiment that several members of the CHDL group have expressed on the importance of the use of the output of this group as a teaching or descriptive language.

Descriptive languages

The first language developed specifically for operating system description was a notation introduced by Leo Cohen in his preliminary text, *Theory of the Operating System*.⁸ Cohen was one of the first individuals presenting introductory courses on operating systems and his notation reflects an appreciation of the importance of understanding the concurrent operation of the operating system with the user program and concurrency within the operating system itself as well as recognition of the fact that it is a “transaction” that is being processed. The technique, which involved identifying the “Current Operating Transaction” and the “Available Transaction” as the operating system crossed logical boundaries was described fully and used extensively in Cohen’s book.⁹ Although the COT-AT technique was quite limited in its capabilities, it did provide the basis for a rather concise description of an existing operating system, OS/360 MVT, in 18 small flowcharts⁴⁰ and a conceptual design for a proposed operating system.⁴¹ The early work with the COT-AT notation system also greatly influenced the design of the first operating system simulation language, S3, to be discussed.

Another notational technique applied to the descriptions of both software and hardware is the contour model.²⁷ This technique, which can illustrate the nesting of control and concurrent execution environments, was used extensively in a complete treatise on the Burroughs 5700 and 6700 systems.⁴⁹ This technique may not have been available for use by the same author in an earlier text on the Multics system⁴⁸ which also has a block-type structure; however, there also appears to be some question of its applicability to this situation. The contour model also formed part of the basis of the picture-system.^{24–26}

Experimentation and simulation languages

Simulation has long been recognized as a very useful tool in the study and evaluation of operating systems. As can be seen from the references to be cited, there has been much work done in this area—far more than that done at any other level.

Almost any general-purpose discrete event simulation language, e.g. GPSS^{39,55} and SIMSCRIPT,³⁰ or even a general-purpose language such as FORTRAN could be used to model an operating system,^{35,45} and many such models have been constructed. However, most of these models cover only a small portion of the complete system such as the scheduler/dispatcher subsystem, the memory allocator/manager, etc. GPSS has been popular for this purpose. Although a model programmed in a general-purpose language can provide an accurate description of the logic of the target system, the source code is not easy to read, and the depiction of many of the operations found in operating systems requires some rather involved coding that obscures the basic logic. Some work of note in this area has been the use of SIMULA as the simulation language. A specific example is a complete model of the CDC 6600/SCOPE.⁵¹ SIMULA has several features, particularly the CLASS concept, which made it extremely attractive for use in this application and its applicability has been considered by several individuals at the Norwegian Computer Center.^{13,47,42}

The earliest known example of a programming system developed expressly for simulation studies of computer systems is the Computer System Simulation (CSS)⁶² which was utilized by IBM in three ways: “First, in the development of a programming system . . . so that proposed changes can be evaluated . . . second, in establishing a system configuration for a given workload . . . third, after a system is operational . . . to predict the effect of expected or proposed changes to the actual system.”⁶² The CSS model consisted of (1) a description of the system, (2) description of the operating system, and (3) description of the workload. The CSS language provided a means to describe the model of the operating system so that each block in the flowchart was represented by a single statement. What was lacking was a clear picture of the concurrency present. (See Reference 39 for an application of CSS.)

The earliest known example of a simulation language developed expressly for operating systems is S3, the Systems and Software Simulator.⁶ This system was developed under contract to the U.S. Army which was exploring alternatives to benchmarking as a system evaluation procedure for procurement purposes. (Although the review of the S3 System indicated that it would provide sufficient accuracy for this purpose, its use was not adopted because of the high cost of maintaining a data base containing up-to-date descriptions of all of the various hardware and software systems that might be proposed by suppliers. Benchmarking was retained as the most cost-effective technique that would satisfy both the vendors and the Army.) The S3 simulation language provided full capabilities to describe both the hardware and the software involved to include both user programs and the operating system. The simulator provided a “true discrete

event simulation" utilizing a complete future events chain. The language provided statements to accurately model the logic of the software as well as statements to maintain the proper timing with respect to the hardware and the system actually being modeled. The power and usefulness of the language is best attested to by the fact that a completely validated and time-adjusted simulation of the GE 635 computer and the GECOS II operating system required *only 211 S3 statements*. (The instructional value of such a language as a descriptive tool was first appreciated by this author during his review and evaluation of the S3 contract.¹² S3 was used internally by Cohen Associates for verifying the performance of the operating system for an airborne computer which had extremely stringent timing requirements, e.g., proper recovery and handling of six different interrupts that could occur during one microsecond. The work on S3 obviously influenced Cohen in the development of the commercially available simulation system SAM.¹

OSSL was another language developed primarily for generalized simulation studies of computer systems.¹⁰ The OSSL model had three components: "(a) The hardware characteristics and system configuration, (b) the operational philosophy of the system [the manner in which user jobs flow through the system], and (c) the environment in which the system is to function [the services that the system is called upon to perform]." Simulation languages and systems developed expressly for operating system study do provide models with a rich content of system description. Their major weakness in application has been the cost in processor time of executing the model. An example is the GECOS II/635 model referred to above which produced validated results within five percent of actual system runs but required 10-15 times as much time even when executing on a UNIVAC 1108. The high time requirement for executing the simulator is not totally the result of using a "high-level" simulation language. Even for models written in assembly language "the simulated-time/real-time ratio is much greater than one."⁴⁶

Analytic models can provide the same degree of accuracy with execution costs much less than actual runs. Although the state-of-the-art in analytic models has progressed to very high levels of both completeness and detail (a comprehensive survey is presented by Reference 65), the description of the model, in whatever language is utilized, provides little, if any, intuitive understanding or insight into the nature of the system being modelled.⁶⁴

A compromise between the two forms of modelling or simulation has resulted in the development of hybrid techniques in which both discrete event and analytic procedures are utilized.^{16,31,61} "In computer system models, it is convenient to partition the resources of the system into two mutually exclusive sets, denoted long-term resources and short-term resources respectively, creating a two-phase model. In a hybrid model, discrete-event simulation is used to model the first phase, which is the arrival of tasks and the allocation of long-term resources. This second phase—use of short-term resources by active tasks—is then modeled by any technique which produces an expected residency time (active time) for each active task."⁶¹ The intuitive un-

derstanding obtainable from the "language" statement of a hybrid model does exceed that present in pure analytic models, but the logic of the system being simulated is still often submerged in programming details of the model.

Instructional laboratory project languages

It is not clear exactly where to place those languages developed expressly to support operating system laboratory courses. To support the general teaching objectives, they certainly should be highly descriptive in nature, but it appears that this aspect of the language has often been sacrificed for the sake of expediency in getting the laboratory exercise completed in a limited amount of time. However, this latter comment certainly does not apply to all of the work in this area. Several of these instruction-oriented systems provide very usable simulation capabilities, e.g., OASIS^{66,68} and ITS,^{36,37} while others provide a language intended for the actual implementation of a complete operating system or portions of a system, e.g., Concurrent Pascal⁴ and Concurrent SP/k.²³

Certainly, a primary objective of any instruction language is the transfer of information about concepts as well as implementation techniques and examples. Also, the influence of an operating system simulation language on system descriptions is quite beneficial;⁶⁷ however, the instructional environment must be able to accommodate student time limitations, which results in shifts in emphasis on the various capabilities of the language as contrasted to what they might be in a "production" environment.

Design languages

There are two examples of design languages. These are definitely multi-level, and either could *possibly* be made to span the entire spectrum of language levels being discussed here. Both of these projects date from 1972—one as academic research that was utilized for one small job and the other as a totally operational support system. The first phase in the academic project was a study of "A Programming Language for Concurrent Processing."²⁶ The underlying model of this system is based on the contour model discussed earlier.²⁷ Of more interest to this paper is the follow-up work which led to the development of the Picture-System model and the PS notation which supports model development.^{24,25} Picture-system models "are useful in defining, communicating, and simulating computer system designs, especially in the early design stages."²⁴ The PS system generates the models "from a description of a computer system as a structure of finite-state components. The PS system also does an analysis of the state-transition graph of the subject computer system which detects design problems such as deadlocks, looping, and races."²⁴

The other project,¹⁸⁻²¹ known as Higher Order Software (HOS), is "a formal methodology for reliable systems specification and development."²¹ A specific goal of the HOS system is to tightly control those areas causing most of the

design errors, such as interface problems, and to automate that control.

It is now possible within the framework of HOE to develop automatic tools to aid verification as well as design. For example, the interfaces of an HOS system can be exhaustively tested by an automated analyzer without program execution. This is especially significant since interface testing in a large system is known to be a very costly procedure. (73 percent of all problems found during the APOLLO integration effort were interface problems; and verification accounts for 50 percent of the total software development effort.)

Higher Order Software is software expressed in its own meta-language and conforming to a formalized set of laws. The basic components of HOS methodology are: (1) the application of the formal set of laws to the design of a given problem; (2) a meta-language adhering to these laws; (3) the automatic analysis of design interfaces by the design analyzer and the structuring executive analyzer; (4) the architectural virtual layers produced from analyzer output in the form of software, firmware or hardware, and (5) the hardware that is transparent to the user. (Our work, to date, has concentrated on the first three areas of HOS methodology.) In addition, support tools based on axiomatic consistency can enhance a given development process in such areas as: performance analysis, simulation, design automation, definition of subsystem requirements, automatic documentation and automatic management techniques.²¹

There have been several design approaches involving path descriptions. A large amount of it centers around the basic concepts of Petri nets. It is well known that the original work by Petri in this area remained unnoticed and unused for some time after its initial publication. The long-term value of the technique appears to be as difficult to predict as it was to recognize its immediate applicability. There is something inherently attractive about the use of graphical representation to depict parallelism, but this author has difficulty in visualizing clearly how such techniques can be integrated into a unified "vertical" information-transfer system. Activity continues in this area, and it appears quite reasonable to anticipate more progress along these lines.

Implementation languages

Activity in this area has been very high, and work on the topic has been identified by the designation "machine-oriented high-level languages," "systems programming languages" as well as "operating system implementation languages." IFIP has established a technical group on Machine-Oriented Higher-Level Languages and a working conference has been held.⁶⁹ This working conference led to the establishment of a permanent working group, WG 2.4, under the auspices of the IFIP Technical Committee on Programming, TC 2. It is not possible to cover all of the work done in this area; however, a typical example is Reference 38. Markstein developed an operating system programming language, PSETL, which is based on SETL, a set-theoretic programming language. The extensions required for PSETL are capabilities "to allow the description of algorithms involving

interrupts, parallelism, and to some extent, machine dependent features."³⁸ Although it is stated that PSETL is "intended for operating system description," it is much more of an implementation language. The extensive examples given in the report illustrate that the program is not very "descriptive" nor very understandable without the liberal use of comments (perhaps as much as 50 percent).

A general discussion of the desired characteristics of implementation languages including specific comments on Concurrent PASCAL and MODULA is presented in Reference 28. Another derivative of PASCAL is CCNPASCAL.²⁹

Multi-level language systems

As was stated previously, there has been only a limited amount of work done in this area. A useful survey of languages for specification and design was presented in Reference 56. Language classifications that were identified and briefly discussed in that survey are state-based languages such as TOPD and DREAM, event-based languages such as path expression and flow expression systems, and relational language systems such as ISDOS and REVS. All of these systems do provide some formal linkage between the statement of the specification and the design.

An early example of a development system that extends into even the simulation level is DES. A Design and Evaluation System. "A system which integrates performance evaluation with design and implementation" is described in Reference 17. "[T]his system is based on a simple, high level language which is used to describe the evolving system at all stages of its development. The source language description is used as direct input to performance analysis and simulation routines."

A description of a complete program development methodology focused specifically on "the design, implementation, and proof of large systems, based upon a hierarchical decomposition of the system" is given in Reference 57.

SUMMARY

A major factor contributing to the high cost of the development and use of an operating system is the lack of effective means for transferring information vertically within one operating system project and horizontally between projects. Most of the work in the past has addressed only vertical information flow. It is important that the importance of the description of operating systems and the value of horizontal transfers also be recognized and addressed by future work.

ACKNOWLEDGMENT

The ideas expressed in this paper were developed during discussions with several individuals, among whom were E. Dijkstra, C. A. R. Hoare, Gerhard Seegmueller, and W. Wulf. As I look back on these discussions, I think I detect

a definite dichotomy of points of view—I was interested in a balanced solution considering both horizontal and vertical information transfer, whereas these individuals were focusing primarily on vertical transfer between the lower levels (e.g., Reference 63). In attempting to sort out and properly place the various concepts we discussed, I am sure that I have made errors in interpreting their comments. The responsibility for that is totally mine, as is a sincere appreciation for their help.

REFERENCES

1. ADR, "SAM, Systems Analysis Machine," Applied Data Research, Ind.
2. Bell, C. Gordon and Allen Newell, *Computer Structures: Readings and Examples*, McGraw-Hill, New York, 1971, 669 pp.
3. Boehm, Barry W., Robert K. McClean and D. B. Urfrig, "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," *IEEE Trans. S/W Engr.*, Vol. SE-1 No. 1, March, 1975, pp. 125-133.
4. Brinch-Hansen, Per, "The Programming Language Concurrent Pascal," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, June 1975, pp. 199-207.
5. Chu, Yoahan, *Introduction to Computer Organization*, Prentice-Hall, Englewood Cliffs, NJ, 1970.
6. Cohen Associates, "Systems and Software Simulator." A summary description of the S3 system is given in Enslow, P. H., "Systems and Software Simulator," Appendix D (91 pp.) of Ref. 12. Complete documentation of S3 is available from NTIS, AD 679 269/270/271/272.
7. Cohen, Ellis and David Jefferson, "Protection in the Hydra Operating System," *ACM Operating System Review*, Vol. 9, No. 5, *Proc. of the Fifth Symposium on Operating System Principles*, 19-21 November, 1975, pp. 141-160.
8. Cohen, Leo J., *Theory of Operating Systems*, Institute for Automation Research, Inc., copyright by Cohen, January 24, 1968, 209 pp.
9. Cohen, Leo J., "Operating System Analysis and Designs," Spartan Books, New York, 1970.
10. Dewan, Preum Bhushan, C. E. Donaghey and Joe B. Wyatt, "OSSL—A Specialized Language for Simulating Computer Systems," *Proc. SJCC—1972*, pp. 799-814.
11. Enslow, Philip H., Jr., "Documentation Techniques for Digital Computers," Tech. Report No. Elec-67-1, Department of Electricity, United States Military Academy, West Point, NY, January 1967, 44 pp. (AD 650 645).
12. Enslow, Philip H., Jr., "Operating Systems: Supervisors, Monitors, and Executives." Class notes, Fifth Revision, June 1971.
13. Fjellheim, Roar, Petter Handlykken and Kristen Nygaard, eds., "Report from a Seminar on System Description at 'Skogen', Roros," DELTA Report No. 2, Publication No. S-65, Norwegian Computing Center, May 1974, 50 pp.
14. Frailey, Dennis J., "DSOS—A Skeletal, Real-Time, Minicomputer Operating System," *Software-Practice and Experience*, Vol. 5, 1975, pp. 5-18.
15. Gaines, R. Stockton and Norman Z. Shapiro, "Some Security Principles and their Application to Computer Security," *ACM Operating Systems Review*, Vol. 12, No. 3, July, 1978, pp. 19-28.
16. Gomaa, H., "The Calibration and Validation of a Hybrid Simulation/Regression Model of a Batch Computer System," *Software-Practice and Experience*, Vol. 8, 1978, pp. 11-28.
17. Graham, Robert M., Gerald J. Clancy, Jr. and David B. DeVaney, "A Software Design and Evaluation System," *CACM*, Vol. 16, No. 2, February, 1973, pp. 110-116.
18. Hamilton, M., "A Discussion of Higher Order Software Concepts as They Apply to Functional Requirements and Specifications," The Charles Stark Draper Laboratory, MIT, Cambridge, Mass., December 1972.
19. Hamilton, M., and S. Zeldin, "Top-Down, Bottom-Up Structured Programming and Program Structuring," Report No. E-2728 Revision 1, Charles Stark Draper Laboratory, MIT, Cambridge, Mass. December 1972.
20. Hamilton, M., and S. Zeldin, "Higher Order Software Requirements," Report No. E-2793, The Charles Stark Draper Laboratory, Cambridge, Mass., August 1973.
21. Hamilton, M., and S. Zeldin, "Higher Order Software—A Methodology for Defining Software," AIAA Paper 75-593, AIAA Digital Avionics Systems Conference, Boston, Mass., 2-4 April, 1975.
22. Harrison, Michael A., Walter L. Ruzzo and Jeffrey D. Ullman, "Protection in Operating Systems," *CACM*, Vol. 19, No. 8, August, 1976, pp. 461-471.
23. Holt, R. C., G. S. Graham, E. D. Lazowska and M. A. Scott, "Structured Concurrent Programming with Operating System Applications," Addison-Wesley, Reading, MA, 1978.
24. Isaacson, Portia, "Picture-System Models and Computer System Design," Ph.D. dissertation, Southern Methodist University, Dallas, Texas, 11 November, 1974.
25. Isaacson, Portia, "Picture-System, PS, and the Design of a Channel-to-Channel Computer Interface," *Procs 2nd Annual Symposium on Computer Architecture*, Houston, Texas, 20-22 January, 1975, pp. 63-70.
26. Jackson, Portia, "A Programming Language for Concurrent Processing," Master of Science Thesis, North Texas State University, Denton, Texas, August, 1972.
27. Johnston, J. B., "The Contour Model of Block Structured Process," *Procs ACM SIGPLAN Symposium on Data Structure in Programming Languages*, University of Florida, Gainesville, Florida, February, 1971. *SIGPLAN Notices*, Vol. 6, No. 2, February, 1971, pp. 55-82.
28. Joseph, M., "Towards More General Implementation Languages for Operating Systems," *Procs. 2nd International Symposium on Operating Systems*, IRIA, France, 2-5 October, 1978.
29. Joseph, M., V. R. Prasad, K. T. Narayana, I. V. Ramakrishnan and S. Desai, "Language and Structure in an Operating System," *Procs. 2nd International Symposium on Operating Systems*, IRIA, France, 2-5 October, 1978.
30. Katz, Jesse H., "An Experimental Model of System/360," *CACM*, Vol. 10, No. 11, November, 1967, pp. 694-702.
31. Kimbleton, S., "A Heuristic Approach to Computer Systems Performance Improvement—A Fast Performance Prediction Tool," *Procs AFIPS 1975 NCC*, Vol. 44, AFIPS Press, Montvale, NJ, 1975, pp. 839-846.
32. Kodres, Uno R., "Analysis of Real-Time Systems By Data Flowgraphs," *IEEE Trans. S/W Engr.*, Vol. SE-4, No. 3, May, 1978, pp. 169-178.
33. Lampson, Butler W. and Howard E. Sturgis, "Reflections on an Operating System Design," *CACM*, Vol. 19, No. 5, May, 1976, pp. 251-265.
34. Levin, R. E. Cohen, W. Corwin, F. Pollack and W. Wulf, "Policy/Mechanism Separation in Hydra," *ACM Operating System Review*, Vol. 9, No. 5, pp. 132-140.
35. MacDougall, M. H., "Computer System Simulation: An Introduction," *ACM Computing Surveys*, Vol. 2, No. 3, September, 1970, pp. 191-209.
36. Madey, Jan, "Users' Language for 'OS Demonstration Kit', Part I: Program Level Language," Report No. 7430, Institute of Informatics, Technical University of Munich, Germany, November, 1974.
37. Madey, Jan, "ITS Project—Toward Specification," Report No. 7501, Institute of Informatics, Technical University of Munich, Germany, February 1975, 33 pp.
38. Markstein, Peter, "Operating System Specification Using Very High Level Dictions," Report No. NSO-6, Computer Science Dept., Courant Institute of Mathematical Sciences, New York University, June 1975.
39. Merikallio, Reino A. and Fred C. Holland, "Simulation Design of a Multiprocessing System," *Procs FJCC-1968*, pp. 1399-1410.
40. MetaSystems, "Descriptions of a Complete Operating System: Flow Charts of OS/360 MVT," prepared by MetaSystems Corp., Trenton, NJ.
41. Meta Systems, "Proposal for a B6500/ILLIAC IV Software Design Study," Meta Systems Corp., Trenton, NJ., January 16, 1970.
42. Myrhaugh, Bjorn, "Theoretical Description of an Operating System," Draft, November 1971.
43. Neuhold, E. J., "The Formal Semantics of Operating Systems," *Procs of the International Computing Symposium, 1973*, North-Holland, pp. 127-133.
44. Neumann, Peter G., "Computer System Security Evaluation," *Procs 1978 NCC*, AFIPS Press, Montvale, NJ, 1978, pp. 1087-1095.
45. Nielsen, N. R., "The Simulation of Time Sharing Systems," *CACM*, Vol. 10, No. 7, July, 1967, pp. 397-412.

46. Nutt, Gary J., "A Case Study of Simulation as a Computer System Design Tool," *IEEE/CS COMPUTER Magazine*, October, 1978, pp. 31-36.
47. Nygaard, Kristen, "System Description by SIMULA—An Introduction," Norwegian Computing Center, Publication No. S-35, 1970, 42 pp.
48. Organick, Elliott I., "The Multics System: An Examination of Its Structure," The MIT Press, Cambridge, Mass., 1972.
49. Organick, Elliott I., "Computer System Organization: The B5700/6700 Series," Academic Press, New York, 1973.
50. Pearson, D. J., "A Study in the Pragmatics of Operating Systems Development," *Procs 2nd International Symposium on Operating Systems*, IRIA, France, 2-5 October, 1978.
51. Piene, Jo., "Simulation of Computer Systems—A Case Study on the CDC 6000/SCOPE System," Publication No. S-46, Norwegian Computing Center, April 1973, 115 pp.
52. Popek, Gerald J. and David A. Farber, "A Model for Verification of Data Security in Operating Systems," *CACM*, Vol. 21, No. 9, Sept., 1978, pp. 737-749.
53. Prasad, K.V.S., N. Natarajan, and Mukul K. Sinha, "Physical and Logical Abstractions in a Kernel," *Procs 2nd International Symposium on Operating Systems*, IRIA, France, 2-5 October, 1978.
54. Pruitt, J. L. and W. W. Case, "Architecture of a Real Time Operating System," *ACM Operating System Review*, Vol. 9, No. 5 *Procs of the Fifth Symposium on Operating System Principles*, 19-21 November, 1975, pp. 51-59.
55. Rehmann, Sandra L. and Sherbie G. Gangwere, Jr., "A Simulation Study of Resource Management in a Time-Sharing System," *Procs FJCC-1968*, pp. 1411-1430.
56. Riddle, William E. and Jack C. Wileden, "Languages for Representing Software Specifications and Designs," *ACM SIGSOFT Software Engineering Notes*, Vol. 3, No. 4, October, 1978, pp. 7-11.
57. Robinson, Lawrence, Karl N. Levitt, Peter G. Neumann, and Ashok R. Saxena, "A Formal Methodology for the Design of Operating System Software," in R. T. Yeh, ed., *Current Trends in Programming Methodology: Vol. I—Software Specification and Design*, Prentice-Hall, 1977, pp. 61-110.
58. Ross, Douglas T., "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Trans S/W Engr.*, Vol. SE-3, No. 1, January, 1977, pp. 16-34.
59. Ross, Douglas T., and Kenneth E. Schoman, Jr., "Structured Analysis for Requirement Definition," *IEEE Trans S/W Engr.*, Vol. SE-3, No. 1, January, 1977, pp. 6-15.
60. Schneiderman, Ben, Richard Mayer, Don McKay and Peter Heller, "Experimental Investigations of the Utility of Detailed Flowcharts in Programming," *CACM*, Vol. 20, No. 6, June, 1977, pp. 373-381.
61. Schwetman, H. D., "Hybrid Simulation Models of Computer Systems," *CACM*, Vol. 21, No. 9, September, 1978, pp. 718-723.
62. Seaman, P. H. and R. C. Soucy, "Simulating Operating Systems," *IBM Systems Journal*, Vol. 4, 1969, pp. 264-279.
63. Seegmueller, Gerhard, (Technical University of Munich and Leibniz Computing Center) "Some Desirable Properties of Languages for Writing Operating Systems," Position paper prepared for NSF Workshop on New Directions in Operating Systems, Austin, Texas, 17-18 November, 1975.
64. Towsley, D., K. M. Chandy, and J. C. Browne, "Models for Parallel Processing Within Programs: Applications to CPU:I/O and I/O:I/O Overlap," *CACM*, Vol. 21, No. 10, October, 1978, pp. 821-831.
65. Trivedi, Kishor S., "Analytic Modeling of Computer Systems," *IEEE/CS COMPUTER Magazine*, October, 1978, pp. 38-56.
66. Unger, Brian W., "Programming Languages for Computer System Simulation," *Simulation*, April 1978, pp. 101-110.
67. Unger, Brian W., and Donald S. Bidulock, "A Modular Operating System for a Network Message Processor," Research Report No. 78/22/1, Dept. of Computer Science, University of Calgary, Alberta, Canada, February, 1978. (Presented at CIPS '78, Edmonton, Canada, May, 1978.)
68. Unger, Brian W., Donald S. Bidulock, J. Gosling, and D. Robinson, "OASIS Reference Manual," Report 77/15/4, Dept. of Computer Science, University of Calgary, Alberta, Canada, February, 1977.
69. Van Der Poel, W. L. and L. A. Maarsen, eds., "Machine Oriented Higher Level Languages," Proceedings of an IFIP TC2 working conference, North Holland, 1974, 475 pp.
70. Wegner, Peter, "The Vienna Definition Language," *ACM Computing Surveys*, Vol. 4, No. 1, March, 1972, pp. 5-63.
71. Wulf, W., Personal Communication, August, 1975.
72. Wulf, W. Cohen, A. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "HYDRA: The Kernel of a Multiprocessor Operating System," *CACM*, Vol. 17, No. 6, June, 1974, pp. 337-345.
73. Wulf, W., R. Levin, and C. Pierson, "Overview of the Hydra Operating System Development," *ACM Operating System Review*, Vol. 9, No. 5, pp. 122-131.

The simulation language SIML/I

by M. H. MACDOUGALL

Amdahl Corporation
Sunnyvale, California

INTRODUCTION

The conceptual advantages of process-oriented simulation languages have become generally recognized. SIMULA^{1,4} is the best known and most widely available language of this type; others include ASPOL,^{3,12} SIMPL/I,⁹ and SOL,^{10,11} and processes have been retrofitted into SIMSCRIPT.¹⁴ SOL was the genesis of the process view of system behavior. SIMULA is an elegant general-purpose programming language; its particularization to system simulation is nicely described by Franta.⁷

The process view introduced by SOL evolved more rapidly in the area of operating systems than in system simulation, with consequent influence on the development of process synchronization constructs. In SOL, processes could control their activities in accordance with the values of expressions involving global variables. In operating systems, considerations including efficiency and the need for hierarchical structures resulted in the development of more specialized constructs for process synchronization; these included events,^{2,5} semaphores,⁶ and monitors.⁸ Recent simulation languages often incorporate similar constructs. Among the advantages of this approach is the extent to which similitude—the resemblance of the model to the design—can be realized in system-level models. (While it is possible that a simulation model can be a valid representation of a system in a behavioral sense and, at the same time, bear little resemblance to the system, it usually is difficult to extend such a model to represent increased detail or to reflect design changes. At the very least, a lack of similitude can hinder communication between designer and modeler.) This trend ultimately may result in the merger of simumodel and design specification, as proposed by Randell.¹³

The improvements such languages bring to system-level modeling, and the relative ease with which they permit expansion of the level of detail of models of software elements, may not be realized when the modeling objective is oriented more toward the hardware elements of the system. For example, ASPOL is a simulation language whose process synchronization facilities, based on entities called "events," derive from operating system constructs. It serves quite well for the development of various kinds of system-level models (perhaps its main shortcomings at this level are in the area of facility preemption and the associated process interruption and queuing). For models requiring a more detailed

representation of hardware elements (instruction pipeline simulations, bus conflict models, etc.), the operating-system-oriented event facilities are much less satisfactory. For the sake of both simplicity and similitude in such models, it often is desirable to represent important control functions essentially at the logic level. Modeling these functions with events is awkward at best; less specialized constructs are needed at this level of detail.

SIML/I is designed to provide a simulation capability which extends into the current gap between system-level and register-transfer-level simulation languages. Its process synchronization facilities are logic-oriented; they permit straightforward representation of the logical expressions involved in modeling hardware structures, as well as meeting the generally simpler needs of system models. SIML/I takes the form of an extension of PL/I, so the general programming facilities of PL/I are available to the modeler. The basic simulation constructs of SIML/I—models, processes, and signals—are described in the following sections.

MODELS AND SUBMODELS

The procedural forms of SIML/I include model, submodel, and process descriptions, together with the procedure descriptions of PL/I. A SIML/I simulation program may comprise a single model description, or may comprise a model description together with one or more submodel descriptions. A model description begins with a MODEL declaration, which is analogous to the PL/I declaration PROCEDURE OPTIONS(MAIN), and ends with the delimiter END MODEL. Model execution is initiated by the system as directed upon completion of loading. Submodel descriptions are separately compiled components of a simulation model and constitute separate load modules. A submodel description begins with the declaration SUBMODEL, which may be accompanied by a formal parameter list, and ends with the delimiter END SUBMODEL. Execution of a submodel is initiated by a model or submodel via an INITIATE statement, which may include an actual parameter list. Once initiated, a submodel executes independently of its initiator unless their activities are explicitly coordinated. Parameters may be transmitted to submodels at the time of their initiation. Thereafter, a model and its submodels may communicate with one another only through variables and signals declared external in each description.

The bodies of model and submodel descriptions essentially are identical in form. All process descriptions appear directly in a model or submodel description; a process description may not itself contain process descriptions. Model, submodel, and process descriptions may contain PL/I procedures, but these may not contain simulation declarations or statements. Rules for the scope of names of simulation entities are similar to those defining the scope of PL/I names. Signals declared in a model or submodel description, as well as PL/I variables declared in that description, are global to all contained process descriptions. Only a single instance of execution of a model description occurs in a simulation, so only a single instance of variables and signals declared in a model description is created. Multiple instances of submodel description execution may be initiated; for each, a unique instance of the signals and variables declared in that description (except for those declared external) is created.

Once initiated, a model or submodel executes like and exists in the same states as any process. A model or submodel may initiate submodels and processes, wait for and set signals, etc. The current simulation time is maintained as the global variable TIME. A model, submodel, or process may suspend its execution for an interval T via the statement HOLD(T); it will be returned to execution at time TIME+T. Execution of a TERMINATE statement in a model terminates the simulation; execution of a TERMINATE statement in a submodel terminates only that particular instance of submodel description execution. The delimiters END MODEL and END SUBMODEL are implicit TERMINATE statements.

Submodels are important in developing large simulation models (since model components can be separately compiled), in multi-level modeling (where submodels of different levels of detail can be constructed and combined as needed for a particular simulation), and in modeling parallel systems (since multiple instances of execution of submodel descriptions can be initiated). As an example of the last, suppose a submodel of a disk subsystem is developed as part of a computer system simulation model. This submodel might represent a data channel, the controllers connected to that channel, and the devices connected to each controller. To simulate a configuration containing several similar (but not necessarily identical) disk subsystems, several instances of execution of this submodel description could be initiated.

PROCESSES

A process description begins with the declaration PROCESS (which may have a formal parameter list) and ends with the delimiter END PROCESS. A process is a particular instance of execution of a process description; a number of such instances may simultaneously exist at any point in a simulation. A process may be initiated by the model or submodel in which it is contained, or by some other process in that model or submodel, including a process of the same description. Upon initiation, a unique set of the variables and signals declared in the process description is created. After initiation, a process executes independently of its initiator and of processes of the same or different descriptions

unless explicitly synchronized. Process execution is terminated via execution of a TERMINATE statement or the delimiter END PROCESS; upon termination, variables and signals local to the process are destroyed.

A process (or model or submodel) exists in one of four states: execute, ready, hold, or wait (queue). A process in execute or ready state is active; a process in hold or wait state is suspended. Since the simulation of a system of concurrently-executing processes is carried out sequentially, only one simulation process is in execute state at any instant; any others able to execute at that instant are in ready state. For example, when a process is initiated, it is placed in ready state; its initiator continues in execute state.

A process may suspend its execution until a signal changes state via a wait (or queue) statement, in which case it is placed in wait state. When the selected signal changes state, the process is placed in ready state (activated). A signal's change of state may activate several processes, so that a number of processes may be in ready state at any instant; these processes are placed on the ready list. When the currently-executing process suspends its execution, the process at the head of the ready list is removed and placed into execution. The ready list is ordered on the basis of priority; equal-priority entries are ordered first-in, first-out. Priority is an attribute of a process (or model or submodel) which may be changed at any time via an assignment statement of the form PRIORITY = expression. Submodels and processes inherit the priority of their initiators.

A process may suspend execution until a specified reactivation time is reached via a HOLD statement, in which case it is placed in hold state and some other process selected from the ready list and placed into execution. When a process suspends execution and the ready list is found empty, the process with the earliest-occurring reactivation time is selected from the set of processes in hold state, the simulation time TIME is advanced to that time, and the process is placed on the ready list. If several processes have the same reactivation time, all are placed on the ready list in priority order. The process at the head of the ready list then is placed in execution.

Parameters may be transmitted to a process or submodel by its initiator. Parameter transmission is uni-directional; parameters are passed by value at initiation time, and modification of a variable which is a formal parameter does not result in modification of the corresponding actual parameter. An actual parameter may be a signal local to (declared and created in) the initiator. The corresponding formal parameter is identified as a signal name by its appearance in a SIGNAL declaration in the process description of the initiated process.

SIGNALS

Processes (and models and submodels) in SIML/I coordinate their activities via operations on signal elements. A signal element is a two-state variable which may be assigned either the value '1' ('set') or '0' ('reset') via SET or RESET statements. A process may suspend execution until a signal element S becomes set or reset via the statements

WAIT(S) or WAIT(\neg S), or QUEUE(S) or QUEUE(\neg S). Throughout this discussion, signal elements are, for the sake of brevity, referred to simply as signals. However, a signal actually should be viewed as a dynamic entity—a particular occurrence of a signal element's change of state; the medium should be distinguished from the message. Signal elements and their associated operations provide a general mechanism for process coordination; a signal element may represent a gate or a latch in a hardware-oriented model, or a table lock or semaphore in a software-oriented model.

Signals are defined, named, and created via SIGNAL declarations. Both simple (single) signals and sets (one-dimensional arrays) of signals may be defined, and signals may be defined in terms of logical combinations of other signals. A signal which is defined in terms of other signals is called a *derived* signal; a signal which is not defined in terms of other signals is a *basic* signal. In the declaration

```
SIGNAL A, B, C, D(6), E INITIALLY SET,
      F=A|B, G= $\neg$ B &  $\neg$ C;
```

A, B, C, D, and E are basic signals; F and G are derived signals. Basic signals are placed in the reset state when created (unless placed in the set state via an initial clause, as in the previous case of signal E). The initial state of derived signals is determined from the states of the signals on which they are defined. In the preceding declaration, F will be initially placed in the reset state and G will be placed in the set state. Only basic signals can be operated on by SET and RESET statements.

Signals declared in a model or submodel description are global to all process descriptions contained in the model or submodel description. Several instances of execution of a submodel description may be initiated; a unique set of the signals declared in that description is created for each instance (except for signals declared external). Only processes initiated directly or indirectly by a particular submodel have access to the signals created in that submodel. Signals declared in a process description are created whenever a process of that description is initiated. Local signals may be passed from one process to another as process initiation parameters. A typical application of local signals arises when one process initiates another and then suspends execution until the initiated process reaches a particular point in its execution; the initiator passes a local signal to the process being initiated and then waits for that signal to be set (or reset).

Signals are created at execution time, not at compile time. Thus, the dimension of a signal set may be determined by computations within the model, and signal creation can be made dependent on model input parameters.

SIGNAL OPERATIONS

A simple basic signal S is set or reset by the statements SET(S) or RESET(S); this may cause a change of state of a derived signal directly or indirectly defined on the basic signal. The Ith element of a basic signal set S can be set or reset by the statements SET(S(I)) or RESET(S(I)). The

statement SET(S) or SET(S,I), where S is a basic signal set, causes the set to be searched for an element in the reset state; if one is found, it is placed in the set state and, in the second form, the index of the element is assigned as the value of the variable I. (If no element is found in the reset state, the statement is ignored.) The RESET statement functions similarly.

A process may suspend execution until a basic or derived signal is set or reset via WAIT or QUEUE statements. Execution of one of these statements is called signal selection; processes suspended while waiting for a signal to change state are called selectors of that signal. Those processes which selected the signal via wait statements all are activated (placed in ready state) when the signal changes state. Of those processes which selected the signal via queue statements, only one is activated when the signal state change occurs. This one is chosen on the basis of priority; among equal priority selectors, the first to enter the queue is chosen. Signal selections may be a mixture of wait and queue selections; when the signal changes to the selected state, all waiting selectors and one queued selector are activated.

When a process executes a set or reset statement, it is placed on the ready list; any wait selectors of the signal are placed on the ready list next, followed by the queued selector (if one was activated). The next process to execute is then selected from the ready list on the basis of priority, as described earlier. If a process executes a wait statement and the selected signal already is in the specified state, it continues in execution. If it executes a queue statement and the selected signal already is in the specified state, it continues in execution only if there are no higher-priority processes queued on that state of the signal; otherwise, it is suspended and enqueued.

A process may select the Ith element of a basic or derived signal set S by selection statements such as WAIT(S(I)) or QUEUE(\neg S(I)); such signal set element selections are processed identically to simple signal selections. It is also possible to select the set itself. Associated with each signal set is a signal of the same name representing the state of the set; by definition, the state of a set is the logical sum of the states of its elements. Thus,

$$S=S(1)|S(2)|\dots|S(n)$$

and

$$\neg S=\neg S(1)\&\neg S(2)\&\dots\&\neg S(n)$$

For a signal set S, the statement WAIT(S) causes a selector to be suspended until some element of S becomes set; the statement WAIT(\neg S) causes a selector to be suspended until all elements of S are reset. Queue selections function similarly. Selection statements of the form WAIT(S,I) can be used to obtain the index of the element whose state change caused the selector to be returned to execution.

SIGNAL EXPRESSIONS

A derived signal definition defines a signal (simple, set, or set element) in terms of a signal expression. The signals

in this expression may themselves be derived, so signals of arbitrary complexity may be constructed. Signal expressions take the logical sum-of-products form; parenthetical sub-expressions are not permitted, and operators are restricted to '&' ("and") and '|' ("or"). This choice of signal expression form was based on two considerations; representational efficiency and simulation efficiency. The applications of SIML/I in hardware-oriented modeling often include control mechanisms, but rarely go beyond that. For example, it may be desired to model the ingating to an adder, but simulation of the adder itself is not likely to be required. A study of control logic in various systems of interest indicated that the need for parenthetical sub-expressions and other operators (even exclusive or) arose infrequently. Restricting signal expressions to the sum-of-products form greatly expedited SIML/I implementation and permitted a very efficient structure for propogating signal state changes.

A derived signal is called a sink signal, and the signals on which it is defined are called source signals. Both sink and source signals may be either simple signals or signal sets: the various combinations govern the mapping between source and sink. Some examples of source/sink mappings are diagrammed in Figure 1; these diagrams show the mappings resulting from the following signal declarations:

SIGNAL A,B,C(2),D(3);

(a) SIGNAL W(3): W(1)=A, W(2)=C(1), W(3)=D(1);

(b) SIGNAL X=D|B&C;

(c) SIGNAL Y(3)=C&D;

When the sink signal is a single element (either a simple signal or a signal set element) and a source signal also is a single element, sink and source signals are directly mapped, as shown in Figure 1a. When the sink signal is a single

element and a source signal is a set, the source signal (by definition) is taken to be the signal representing the set, and the mapping illustrated in Figure 1b results. When sink and source signals both are sets, they are mapped element-by-element up to the dimension of each set, as shown in Figure 1c. When the sink signal is a set and the source signal is a single element, the latter is mapped to each element of the set.

Sink/source mappings for single elements are determined by the axioms of logical arithmetic. Mappings involving sets were, to some extent, chosen on consideration of the functions needed to simply construct hardware, firmware, and software control structures for various systems. One missing capability which appears desirable is source signal set concatenation (currently, this can be effected only by writing a term-by-term definition for the sink signal).

APPLICATIONS

While the signal facilities of SIML/I were designed to extend system simulation capabilities nearer to the hardware realm, they provide a general means of process coordination, and support models over a range of levels of abstraction. In current applications, signal representations range from logic gates to complete CPUs. The following declarations represent the control logic shown in Figure 2.

SIGNAL TR_ENABLE

=ENABLE&RTR|ENABLE&WTR|ENABLE&HTR;

SIGNAL TR_GATE=TR_ENABLE&CLOCK;

Figure 3 shows a block diagram of a queueing network model of a computer system; a SIML/I simulation program

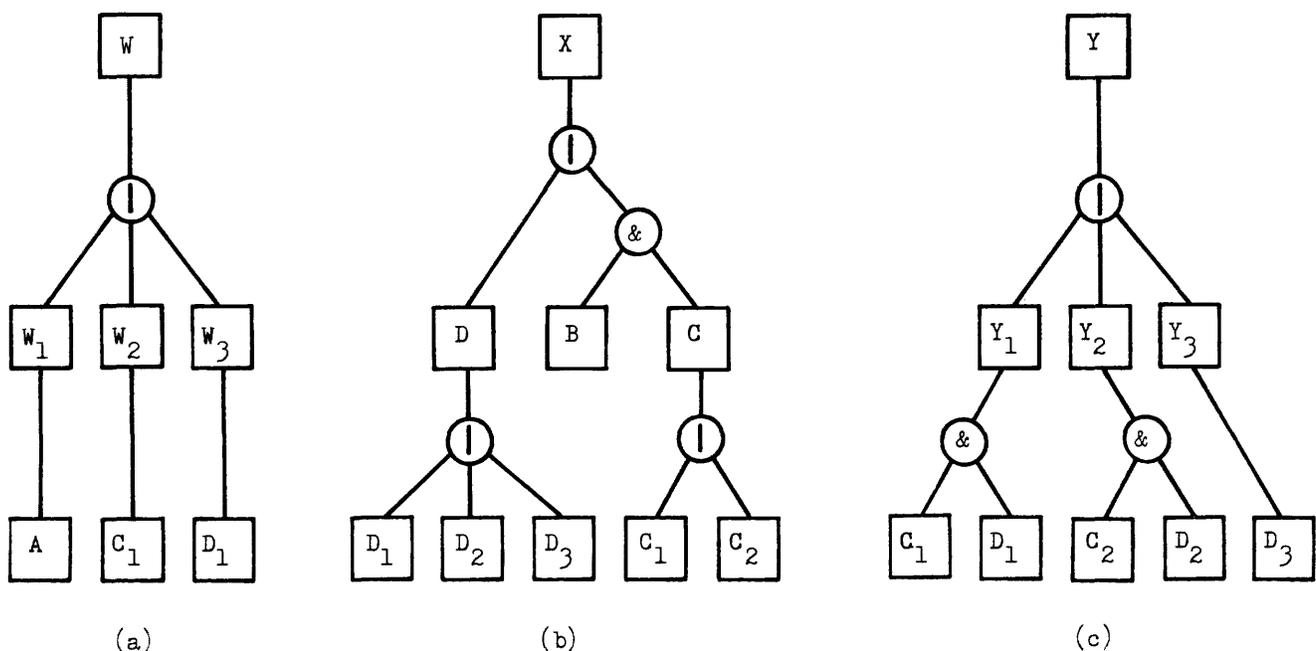


Figure 1—Sink/source signal mapping examples.

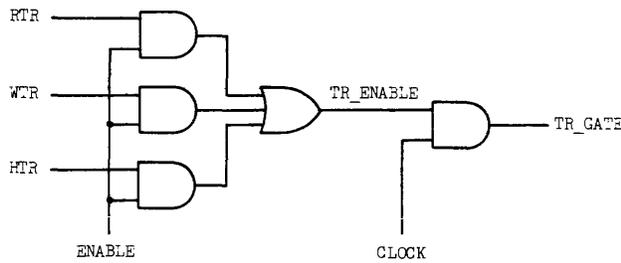


Figure 2—Control logic modeling with signals.

for this network is as follows:

```

MODEL QNM;
  SIGNAL CPU(2) INITIALLY SET, IO(8) INITIALLY
  SET;
  DO I=1,32;
    INITIATE JOB;
  END;
  HOLD(5000.);
  TERMINATE;
  PROCESS JOB;
  DO WHILE(TIME<5000.);
    QUEUE(CPU,I); RESET(CPU(I));
    HOLD(1.);
    SET(CPU(I));
    J=IRANDOM(1.8);
    QUEUE(IO(J)); RESET(IO(J));
    HOLD(EXPNTL(5.));
    SET(IO(J));
  END;
  END PROCESS JOB;
END MODEL QNM;

```

In the foregoing example, there are 32 jobs circulating in the

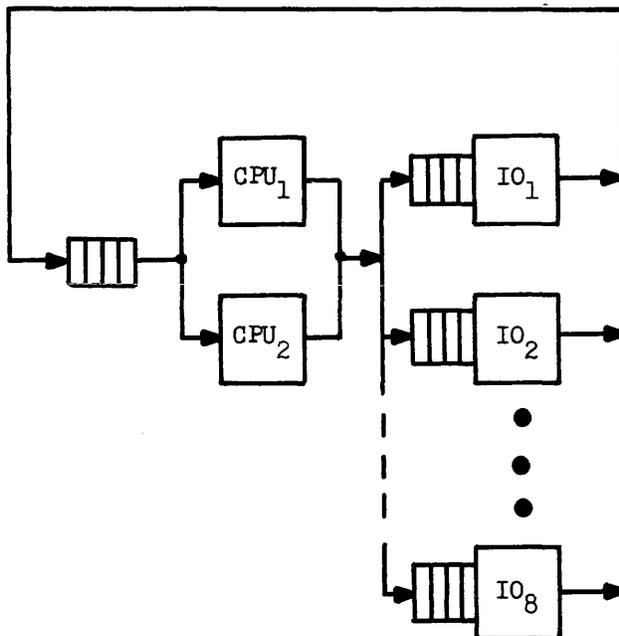


Figure 3—Queuing network model.

system. Job CPU times are constant and equal to one time unit, while I/O service times are exponentially distributed with a mean of 5 time units. When a job completes CPU service, it selects an I/O device and queues for it; when it completes I/O service, it queues for the set of CPUs. The set state of the CPU and I/O signals in this model corresponds to the nonbusy state.

Signal definitions are used to vary the level of detail of model components. The declarations below illustrate a signal defined at three different levels of detail.

- (1) SIGNAL INTERRUPT;
- (2) SIGNAL IO, FAULT, CHECK, TIMER;
SIGNAL
INTERRUPT=IO|FAULT|CHECK|TIMER;
- (3) SIGNAL END_RD(8), END_WR(8),
RD_FAULT, WR_FAULT;
SIGNAL IO=END_RD|END_WR,
FAULT=RD_FAULT|WR_FAULT;
SIGNAL CHECK, TIMER;
SIGNAL
INTERRUPT=IO|FAULT|CHECK|TIMER;

Definition expansions of this form provide part of the mechanism for the vertical communication between components of different levels of detail in a multi-level model.

As an incidental note, SIML/I does not use the PL/I multi-tasking and event facilities for process and signal control: these functions are performed by the SIML/I run-time systems.

ACKNOWLEDGMENTS

The author would like to acknowledge the fine work of Carol Realini and Allan Rhodes in developing the SIML/I language processor, and of Scott Spadafore in developing the interface between the SIML/I and PL/I run-time systems.

REFERENCES

1. Birtwhistle, G. M., O.-J. Dahl, B. Myrhaug and K. Nygaard, *SIMULA Begin*, Auerbach, 1973.
2. Cleary, J. G., "Process Handling on the Burroughs B6500," *Proc. Fourth Australian Computing Conf.*, 1969, pp. 321-329.
3. Control Data Corp., *ASPOL Reference Manual*, Pub. No. 17314200-B, 1975.
4. Dahl, O.-J., B. Myrhaug and K. Nygaard, *Common Base Language*, Pub. No. 5-22, Norwegian Computing Center, 1970.
5. Dahm, D. M., F. H. Gerbstadt and M. M. Pacelli, "A System for Resource Allocation," *Comm. ACM*, Vol. 10, No. 12, Dec. 1967, pp. 772-779.
6. Dijkstra, E. W., "Cooperating Sequential Processes," *Programming Languages*, F. Genuys (ed.), Academic Press, 1968, pp. 43-112.
7. Franta, W. R., *The Process View of Simulation*, Elsevier North-Holland Inc., 1977.
8. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," *Comm. ACM* Vol. 17, No. 10, Oct. 1974, pp. 549-557.

9. IBM Corp., *SIMPLII Program Reference Manual*, Pub. No. SH19-5060-0, 1972.
10. Knuth, D. E., and J. L. McNeley, "SOL—A Symbolic Language for General Purpose Systems Simulation," *IEEE Trans. on Computers*, Vol. C-13, Aug. 1964, pp. 401-408.
11. Knuth, D. E., and J. L. McNeley, "A Formal Definition of SOL," *IEEE Trans. on Computers*, Vol. C-13, Aug. 1964, pp. 409-414.
12. MacDougall, M. H., "Process and Event Control in ASPOL," *Proc. Symposium on the Simulation of Computer Systems III*, 1975, pp. 39-51.
13. Randell, B., "Towards a Methodology of Computing System Design," *Proc. 1968 NATO Conf. on Software Engineering*, pp. 204-208.
14. Russell, E. C., *Simulating with Processes and Resources in SIMSCRIPT II.5*, CACI Inc., 1975.

Mix-dependent job scheduling— An application of hybrid simulation

by STEVE TOLOPKA and HERB SCHWETMAN

Purdue University
West Lafayette, Indiana

JOB SCHEDULING

In a computer system, scheduling occurs whenever the next-task-to-receive-service is selected from a queue of waiting tasks.⁵ This paper considers the scheduling of jobs which will become active; that is, which will begin to compete for use of the system processors. Jobs waiting to be activated may be scheduled by referring to properties of the job such as priority or total resource usage.^{4,7} It is also possible to examine some feature of the jobs already activated (i.e., the jobs in the multiprogramming mix) in order to determine a suitable candidate for admission to the mix. This is called mix-dependent scheduling. Here, we prefer to examine some intrinsic property of the jobs rather than an external one. The property we have chosen to investigate is the rate of processor (CPU or I/O) usage.

In order to quantify what we mean by rate of processor usage, we define a *CPU usage index*, K , for each job as

$$K = \frac{C_t}{C_t + I_t} * 6 + 1$$

where

C_t = CPU time used by the job, and

I_t = I/O time used by the job.

Notice that K ranges between 1 and 7, with the lower value denoting a totally I/O-bound job, and the upper value denoting a totally CPU-bound job. The scaling for K is arbitrary; the range 1 to 7 was chosen because our operating system at Purdue uses a similar index.

Assuming that this usage index is computed for each job (active or waiting) in a system, it is then necessary to devise scheduling algorithms which utilize this index and to evaluate the effects. The classical approach is to admit equal numbers of CPU- and I/O-bound jobs in the hope that all resources in the system will be fully utilized. However, other algorithms can be devised which may show better performance. The goal of this paper is to use a system model to evaluate job schedulers and to verify the merits of mix-dependent scheduling algorithms.

HYBRID MODEL

An experimental technique is used in this study. A realistic system model is constructed so that different job scheduling strategies can be evaluated. Figure 1 is a schematic of this model. As can be seen, there is a queue for arriving jobs which are waiting to enter the active phase. The active phase consists of the processors (CPU and I/O) which are required by each job. As is typical in real systems, the number of available positions for jobs in the active phase is limited. Thus, the job scheduler is responsible for filling empty positions as they occur. For purposes of simplification, the number of positions (also called the maximum level of multiprogramming) is eight. Also, memory constraints are omitted, since the emphasis of these tests is to evaluate scheduling techniques for improving processor usage, not memory scheduling. In the model, every job arrives with a class designator and the amount of processing it will require. As previously indicated, an arriving job joins a queue of jobs waiting to be scheduled. There can be a separate queue for each class. The job scheduler, using one of several selectable strategies, then activates jobs as positions in the mix of active jobs become available.

A hybrid simulation technique has been developed which appears to satisfy the needs of this study. While the technique has been described in detail before,^{8,6} it is briefly presented here.

In a hybrid model, discrete-event simulation models the arrival of jobs and job scheduling. Then an analytic modeling technique is used to estimate the processing time for each job in the active phase. The current experiment uses the central server model^{2,3} to model resource consumption by jobs in the active phase. This part of the model consists of a CPU station and three disk stations, as shown in Figure 2. There are three classes of jobs in the system: $P_i(r)$ represents the probability that a class r job leaving the CPU will go next to station i . The mean service times, S_i , at all devices are the same for all job classes, and the queueing discipline is FCFS. Table I displays the parameters of the model.

A *cycle* through this network begins when a job first enters the CPU queue and ends after it has passed through one of the disk stations and reappeared at the CPU queue.

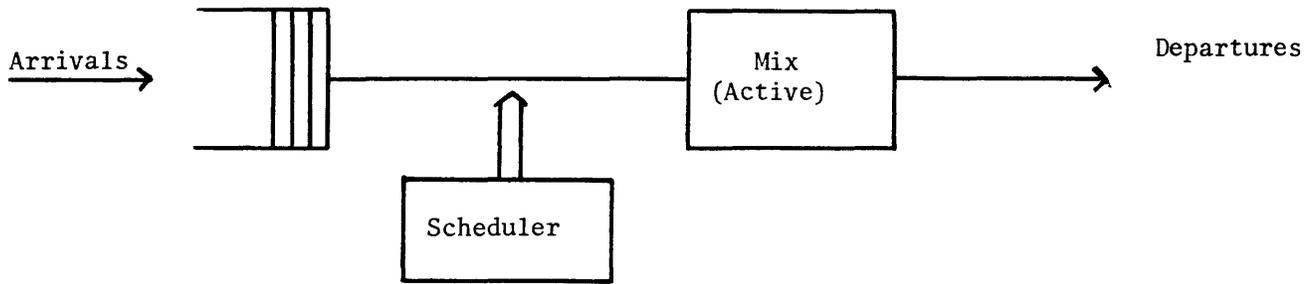


Figure 1—Schematic of system model.

Notice that a job will make one or more trips through the CPU station, and one trip through a disk station per cycle. The central server model can be solved to obtain the mean job waiting times at each device. These waiting times, together with the branching probabilities for each job, can be used to calculate the mean cycle times for each job.

In the system model, as a job joins the mix of active jobs, a new configuration of active jobs is created. The mean cycle times and the number of cycles remaining are multiplied to produce estimated completion times for each active job. If no other jobs are added to the mix, then simulated time can be advanced to the minimum of these times. In this case, the job with the shortest remaining time departs, the cycles remaining for each job are updated and the process is repeated. If another job is added to the mix before the shortest departs, then the cycles remaining for each active job are updated, the new job is added, and processing continues as just described.

This hybrid simulation has been shown to produce models which are as accurate as equivalent simulation-only models while reducing CPU processing time requirements by factors of up to 200 in some test cases.⁸ A key feature of this model is that as more jobs are added to the mix, the time to complete each job may increase. Furthermore, the type of load imposed by jobs in each class depends on the resource usage patterns of the job. For example, a CPU-bound job can have a large impact on the active time of another CPU-

bound job, but perhaps a negligible impact on the active time of an I/O-bound job. This type of behavior coincides with commonly held views of the behavior of actual systems.

JOB CLASSES

As indicated in an earlier section, the CPU usage index, K , is a guide to the proportion of CPU time a job uses per cycle. The system model was parameterized so as to allow for three classes of jobs, with Class 1 representing I/O-bound jobs, Class 2 "balanced" jobs, and Class 3 CPU-bound jobs. This was done by adjusting the probabilities of returning to the CPU ($P_1(r)$) for each class to obtain the desired behavior. Since the expected number of trips that a class r job will make through the CPU before moving to a disk is $1/(1-P_1(r))$, we can easily calculate the mean CPU time per cycle, and so can compute a CPU usage index, K , for each class of jobs. (See Table II.) The resulting mixture of jobs have CPU usage indices corresponding to jobs of the required types.

To help eliminate bias caused by long and short jobs, the number of cycles that a job of each class would have to complete was chosen so that the total processing time of each job was approximately constant (Table III). Prior to using the system model, the central server model was solved¹ for all possible triples (n_1, n_2, n_3) , where n_i represents the number of class i jobs in the system and $n_1 + n_2 + n_3 \leq 8$ (since the level of multiprogramming will not exceed 8). There are 165 such triples for the current model. These solutions yielded the necessary mean job cycle times.

In order to evaluate the accuracy of the hybrid model, a discrete event simulation of the system model was written. Both the hybrid model and the discrete event simulation were executed using the same set of 100 jobs, equally distributed among the three classes. The mean job interarrival

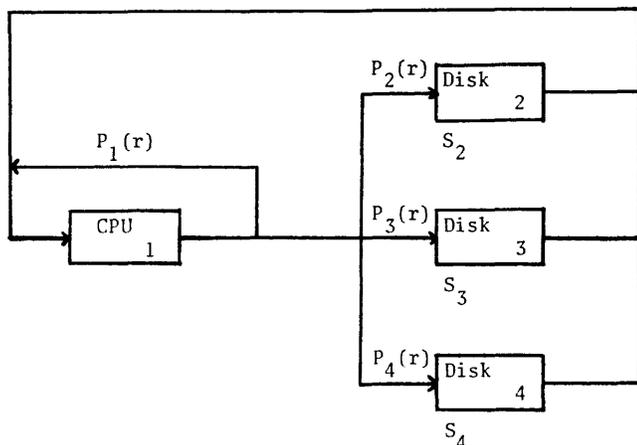


Figure 2—Central server model.

TABLE I.—Central Server Parameters

	i	S _i	P _i (r)		
			r=1	r=2	r=3
CPU	1	.010 sec.	.5	.9	.98
Disk	2	.100	.167	.033	.007
Disk	3	.100	.167	.033	.007
Disk	4	.100	.166	.034	.006

TABLE II.—CPU Usage Indices

r	CPU trips	CPU time per cycle	I/O time per cycle	K
1	2	.020	.100	2
2	10	.100	.100	4
3	50	.500	.100	6

time was 0.0 seconds; i.e., all jobs were in the queue at the start of the experiment. The level of multiprogramming was limited to eight, although no other memory constraints were imposed. The results of the two runs are compared in Table IV. Utilizations differed by at most seven percent while elapsed time and response time (whose corroboration is notoriously difficult⁹) disagree by only 2.5 percent. The hybrid model reduced the CPU time needed to run the model by a factor of 50, with very little (if any) resulting loss in accuracy. While these exact figures are certainly dependent upon the model parameters, other studies⁸ have also shown that the hybrid technique works well for similar models. Thus, for a reasonable range of parameters, the hybrid model seems to provide an accurate, inexpensive way to evaluate scheduling strategies.

MIX-DEPENDENT SCHEDULING

The initial series of tests used a version of the system model with the same set of 100 jobs, all of which were present at the beginning of each test. The number of processing cycles for each job was uniformly distributed over the ranges for each class, as shown in Table V. As discussed in the previous section, these ranges yield processing times which are approximately equal for all jobs. The actual set of 100 jobs generated for each test has the properties shown in Table VI.

The basic thrust of the tests was to examine different techniques for selecting collections of jobs to be simultaneously active. The schemes that were evaluated were all variations of the following general strategies:

1. Schedule jobs from one class prior to scheduling jobs from another class (designated class-first-strategies).
2. Schedule jobs from all classes subject to class constraints (class-constrained-strategies).
3. Schedule jobs without regard for classes (ignore-class-strategies).

Within each of these general strategies there were many possible variations. For example, in the class-first-strate-

TABLE III.—Mean Processing Time Per Job

r	Cycles per job	Proc. time per cycle	Proc. time per job
1	100	.120	12.0
2	60	.200	12.0
3	20	.600	12.0

TABLE IV.—Comparing Simulation and Hybrid Models

	Hybrid	D. E. Simulation
Elapsed Time	673 sec	659 sec
Proc. time per job	11.865	11.839
Active time per job	52.466	51.284
Response time per job	298.735	297.887
Utilization		
CPU	88.9%	88.4%
Disk 1	29.1	29.4
Disk 2	29.1	31.2
Disk 3	29.1	30.5
CPU time required to run the model	1.6 sec	89.2 sec

gies, the order in which the job classes were processed could be varied. In the class-constrained-strategies, both the order of classes and the class constraints were changed.

In these tests, a strategy was judged primarily on the total elapsed time required to process the 100 test jobs—the shorter this time, the better the strategy. The elapsed times are reported without confidence intervals because the different strategies are all being evaluated on the same set of jobs. Once the set of jobs and the scheduling strategy have been chosen, the simulation is totally deterministic. Hence, the elapsed time gives us a fair comparison of the relative merits of the various scheduling strategies.

Class-first-scheduling

In class-first-scheduling, all jobs from one class are started; then as positions become available, jobs from another class are started; finally jobs from the remaining class are started. A major consequence of this strategy is that several jobs from the same class tend to be active at the same time. As will be seen, this turns out to be the worst possible configuration of active jobs.

Two tests were made with this strategy; in one, the classes were processed in the order 1,2,3; in the other, the order 3,2,1 was used. Table VII summarizes the results of these two tests.

Class-constrained-scheduling

The unbalanced scheduling should be compared with a balanced approach to see if improvements are obtainable. In a balanced strategy, the scheduler attempts to keep approximately the same number of I/O-bound and CPU-bound

TABLE V.—Ranges for Cycles per Job

Class	Minimum	Maximum
1	90	110
2	50	70
3	10	30

TABLE VI.—Summary of 100 Simulated Jobs

	Class 1	Class 2	Class 3	Overall
No. of jobs	33	31	36	100
Cycles/job	100	60	19	58
Processing time/job	12.058 sec	12.123 sec	11.467 sec	11.865 sec

TABLE VII.—Class-First-Strategies

	Elapsed Time
Order 1,2,3	673.2 seconds
Order 3,2,1	673.1

TABLE VIII.—Cyclic Strategies

	Elapsed Time
Order 1,2,3	600.2 seconds
Order 3,2,1	599.7

TABLE IX.—Class-Constrained Strategies

Desired Configuration	Elapsed Time
4,2,2	601.1 seconds
2,4,2	599.9
2,2,4	600.5
2,3,3	599.5
3,2,3	599.2
3,3,2	600.4
8,0,0	672.5
0,8,0	600.0
0,0,8	616.8

TABLE X.—Mean Job Active Times

	Class 1	Class 2	Class 3	Overall
Class-first	34.3 sec	48.3 sec	72.7 sec	52.5 sec
Cyclic	27.7	48.1	63.5	46.9

TABLE XI.—Common Job Configurations

Strategy	Configuration	Percent of Time
Class-first	0,0,8	42.6%
	0,8,0	22.6
	8,0,0	20.2
Cyclic	3,3,2	52.3
	0,0,8	18.4
	0,4,4	16.8

jobs active. The simplest such strategy starts up jobs in a cyclic order. The results obtained using cyclic scheduling are shown in Table VIII.

Another approach to balanced scheduling is based on selecting a desirable job configuration (n_1, n_2, n_3) and then causing the scheduler to attempt to keep the current configuration (m_1, m_2, m_3) as "close" to this desired configuration as possible. This strategy was implemented by calculating the vector distance between the (m_1, m_2, m_3) obtained by tentatively adding one job to each class in succession and then selecting as the new configuration the one which was the minimum distance from (n_1, n_2, n_3) . The results of using this strategy for several different desired configurations are summarized in Table IX. In this test, the order in which jobs were considered had almost no effect on the results; hence, these cases are not included in Table IX.

While we know of no efficient technique for obtaining an optimal schedule for our model, we can point out that CPU utilizations seen in these tests were over 99 percent. This observation leads us to believe that 600 seconds is about the minimal elapsed time which can be achieved. In Table IX, it can be seen that those desired configurations having approximately equal numbers of I/O-bound and CPU-bound jobs produce near optimal results.

In trying to understand why these "balanced" configurations exhibit good performance, we can look at two additional kinds of information. One output of the model is the mean active time per job in each class. Another output is the percent of time the system was in each possible job configuration. Tables X and XI summarize these items for a 'good' schedule and a 'bad' schedule.

These data suggest that when a class is 'overloaded,' the jobs tend to interfere with each other to the point that the entire schedule can be significantly lengthened.

Ignore classes

There are scheduling strategies based on job attributes other than processor usage which can be used. Four such strategies were evaluated as part of this test. In these tests, the length of a job is estimated by the processing time requirement (the expected cycle time with one job of the specified class active multiplied by the number of processing cycles required). The results obtained using these ignore-class strategies are summarized in Table XII.

Notice that the random and first-come, first-served strategies actually perform at a near-optimal level. We speculate that this is because of the test configuration in which the jobs are all about the same length and are equally distributed over the three classes. In this case, any strategy which tends

TABLE XII.—Ignore-Class Strategies

Strategy	Elapsed Time
Shortest job first	608.9 seconds
Longest job first	608.2
First-come, first-served	599.4
Random	600.5

TABLE XIII.—Results for 200 Jobs

	Elapsed Time
Class-first	1168.6 seconds
Cyclic	944.4
Shortest Job First	1046.9
Longest Job First	1040.9
First-come, First-served	969.4
Random	973.4

to schedule jobs from all classes simultaneously should do well. A final test used a different set of 200 jobs in the ratio of three Class 1 jobs to two Class 2 jobs to one Class 3 job. The best (cyclic) and the worst (class first) strategies as well as the four ignore-class strategies were used to verify conclusions reached in earlier tests. Table XIII summarizes these results.

SUMMARY

Mix-dependent job scheduling has been proposed as a technique which can improve system performance. The hybrid simulation model has been used to evaluate a large number of scheduling strategies based on this technique. These tests showed that such scheduling can lead to near-optimal as well as very bad system performance. The variation between good and bad exceeded 10 percent in the test cases. These strategies all used a processor usage index to partition the jobs into classes. When the scheduler kept the number of I/O-bound and CPU-bound jobs approximately balanced, mix-dependent scheduling produced near-optimal schedules. When jobs from one class predominated, they seemed to interfere with each other, causing each job to be active for an excessively long interval, lengthening the entire schedule.

The results obtained so far are all based on the system model. A natural question is whether these results can be extended to actual computer systems. In systems, the collection of jobs is dynamic, rather than static as in the model. In the course of this project, we did use dynamic job arrivals, but saw no appreciable differences in the results. A more

fundamental question involves the accuracy of the hybrid system model—Does it accurately portray the behavior of an actual system? Some preliminary work at Purdue University⁹ has shown that at the level of job completion times, the performance predicted by the hybrid model is within 10-15 percent of the actual system performance.

Future work for the project includes implementing some mix-dependent schedulers on an actual system to verify the performance predictions of the hybrid model. Use of the hybrid model has been crucial to the work to date, because the low operating costs allowed us to evaluate several (in excess of 20) strategies. The results for balanced class-constrained schedules and cyclic schedules suggest that near-optimal performance can be achieved. This type of job scheduling should probably be viewed as "icing on the cake." In other words, after other scheduling goals are being met, then mix-dependent scheduling can be used to improve performance a little more. However, care must be used, as some mix-dependent strategies can degrade performance significantly.

REFERENCES

1. Balbo, G., S. C. Bruell, and H. D. Schwetman, "Customer classes and closed network models—a solution technique," *Proc. IFIP Congress 77*, North-Holland Publ. Co., Amsterdam, The Netherlands, pp. 559-564.
2. Buzen, J. P., "Computational algorithms for closed queueing networks with exponential servers," *Comm. ACM*, Vol. 16, No. 9, Sept. 1973, pp. 527-531.
3. Denning, P. J., and J. P. Buzen, "The operational analysis of queueing network models," *Comput. Surv.*, Vol. 10, No. 3, Sept. 1978, pp. 225-261.
4. Forbes, K., and A. W. Goldworthy, "A prescheduling algorithm—scheduling a suitable mix prior to processing," *The Computer Journal*, Vol. 20, No. 1, Feb. 1977, pp. 27-29.
5. Ruschitzka, M., and R. S. Fabry, "A unifying approach to scheduling," *Comm. ACM*, Vol. 20, No. 7, July 1977, pp. 469-476.
6. Schwetman, H. D., "Hybrid simulation models—a speed-up technique combining analytic and discrete-event modeling," *Modelle für Rechensysteme* (P. Spies, Ed.), Springer-Verlag, New York, 1977, pp. 226-236.
7. Schwetman, H. D., "Job scheduling in multiprogrammed computer systems," *Software-Practice and Experience*, Vol. 8, No. 3, 1978, pp. 241-255.
8. Schwetman, H. D., "Hybrid simulation models of computer systems," *Comm. ACM*, Vol. 21, No. 9, Sept. 1978, pp. 718-723.
9. Schwetman, H. D., "Validating system models: a case study," Submitted for publication, 1978.



Parametric instabilities in computer system performance prediction*

by LAWRENCE W. DOWDY and ASHOK K. AGRAWALA

University of Maryland
College Park, Maryland

PROBLEM OVERVIEW

A typical computer system undergoes continual "upgrading." Unfortunately, this "upgrading" is not always synonymous with system improvement. For example, allowing more users to simultaneously access the system (i.e., increasing the degree of multiprogramming) increases the utilization and throughput of the computer system. However, extra system overhead (e.g., swapping) is required to properly manage the extra load. In some cases,⁴ this overhead offsets the potential throughput improvement. Because interactions between the components of a modern computer system are so complex, there is a critical need for system models which accurately predict performance.

Consider a proposed alteration to a computer system (e.g., adding a new disc storage unit). Current predictive models of computer systems are typically constructed using the following scenario.³ First, a model is constructed of the current system. Second, this model is subjected to numerous tests, and the model results are compared against the observed results obtained from the current system. If the two results are in good agreement, the model is termed "validated." Third, certain parameters of the validated model are altered to represent the proposed alteration to the real system. All unaltered parameters are typically assumed to remain constant. Fourthly, this altered (predictive) model is then subjected to tests. The test results predict what the performance of the real system would be if the proposed alteration were implemented.

This paper presents results found by applying the above scenario to a particular system, the University of Maryland Computer Center's Univac 1100/42. The results concerning the first and second scenario steps have been excellent. Models were constructed which can accurately describe the current system's behavior. However, results from applying the third and fourth scenario steps have been very disappointing. When a system change is made, the predicted

performance has, in many instances, been in error. Upon closer analysis of these instances, the scenario assumption that all unaltered parameters remain constant is found to be false.

MODEL CONSTRUCTION

Three separate models (Models 1, 2, and 3) reflecting three successive configurations of the same basic system are constructed. The initial modeling effort is to construct a model of the Univac 1100/41 (Model 1) which consists of a single processor. Based upon this effort, performance is predicted when another processor (1100/42) is added. Our secondary modeling effort is to construct a model of the 1100/42 (Model 2) and predict performance when the system is upgraded by adding a new drum and new disc units (Model 3). The results between the predicted and actual performance, when the proposed changes are configured, are compared and analyzed.

Model 1 of the 1100/41 is the classical central server model.¹ The topology configuration is given in Figure 1. The assumptions of the model are 1) a fixed degree of multiprogramming (DMP), 2) exponentially distributed holding times at each queuing station, i , with parameter $\mu(i)$, 3) fixed branching probabilities, $p(i)$, which reflect the stationary probability of requiring channel service from channel i , given that some channel service is needed, and 4) first-come-first-serve and processor sharing queuing disciplines for the channel devices and the CPU, respectively. The validation criterion for the model is the utilization of all devices.

Model 2 of the 1100/42 is identical to Model 1 of the 1100/41 with the exception of the central server (CPU) complex. The 1100/41 CPU complex in Figure 1 is replaced by the CPU complex of Figure 2. Because both CPU servers are identical, they share identical service rates. The complex is modeled as a single, load-dependent server with service rate $\mu(\text{CPU})$ when one customer is at the complex and $2\mu(\text{CPU})$ when two or more customers are at the complex.

Model 3 of the updated 1100/42, with additional channels for a new disc subsystem and a new drum, is illustrated in Figure 3. The assumptions for all three models are identical and are as specified in the description of the 1100/41 model.

* This research was supported in part by the National Aeronautics and Space Administration, Goddard Space Flight Center, under Grant NASA #NAS 5-24407, and in part by the Environmental Protection Agency under Grant R805478-01-0, to the Department of Computer Science, University of Maryland, College Park, Maryland.

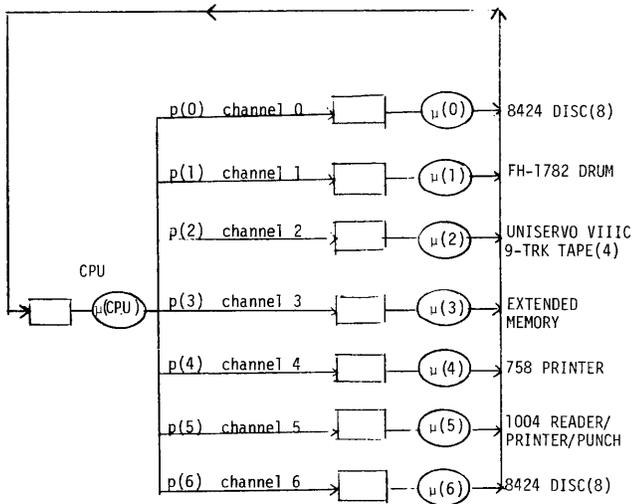


Figure 1—1100/41 Model 1 topology.

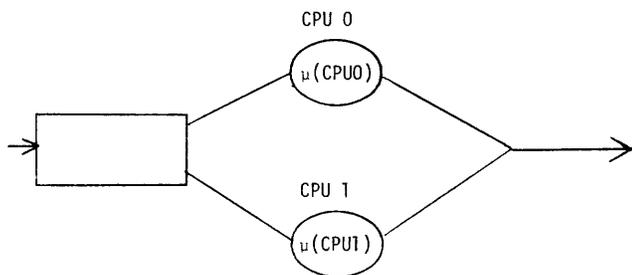


Figure 2—1100/42 CPU complex for Model 2.

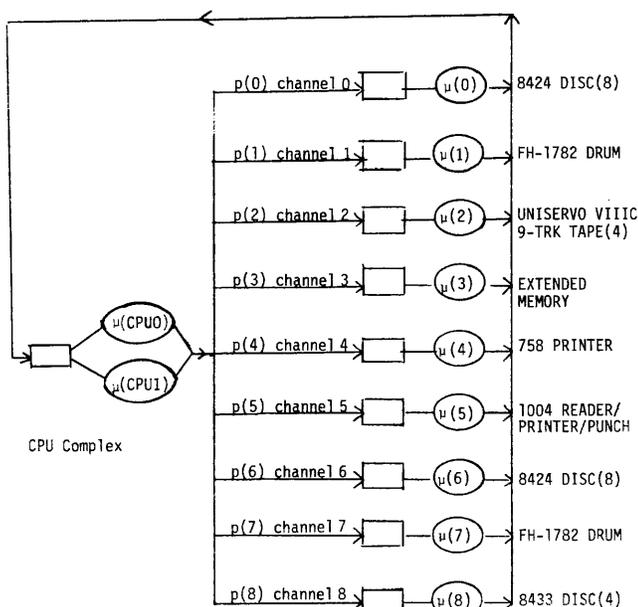


Figure 3—Upgraded 1100/42 Model 3.

PARAMETERIZATION

Employing this level of queuing network modeling, the utilization of any device can be specified as a function of four parameters: the number of devices, the μ s, the p s and the DMP. The number of devices is set by the model topology (eight for the first two models—Figure 1; ten for the third model—Figure 3). The other three parameters are calculated using the Univac Software Instrumentation Package, SIP.

The μ s, the service rates, are calculated as follows. The basic work unit (job, customer, request, run) is called a transaction. SIP records the number of transactions that each channel services in a unit of time, typically one hour. Dividing this number by the active time of the channel (also collected by SIP) yields an estimate of the channel service rate. Since it is assumed by the topology that all transactions flow through the CPU complex, the service rate of the CPU is computed by dividing the total number of transfers by the CPU's active time.

The p s, the branching probabilities, are similarly calculated. The number of transfers serviced by channel i divided by the total number of transactions serviced by all channels is taken as $p(i)$.

The degree of multiprogramming, DMP, cannot be calculated directly from SIP. SIP does record the average number of jobs in core memory. However, this number is an unpredictable overestimate of the DMP. Jobs may be resident in core which are waiting for activity from devices not explicitly modeled (e.g., interactive terminals). This violates a model assumption that all jobs must be either executing at, or queued for, a modeled device. The average number of jobs in core, as will become evident later, is not an accurate estimate for the DMP, and alternate methods become necessary for determining the DMP.

EXPERIMENTAL RESULTS

Once all the model parameters are obtained, standard queuing theoretical solution techniques are applied.² These techniques assume that DMP is integral. The SIP estimated DMP (based upon the number of jobs in core) is an average and is typically non-integral. Algorithms exist⁵ which allow non-integral values of DMP, but at the expense of disallowing load dependent servers in the network. The topology assumed in Figure 1 is not disallowed, but the models in Figures 2 and 3 are disallowed. In this case, interpolation around integral valued degrees of multiprogramming is performed.

As an example, the parameters for the 1100/41 model (Figure 1) based upon a typical hour of SIP data are summarized in Table I. (The μ s are measured in transactions per second.)

Using the parametric values of Table I, the modeled device utilizations are compared against the SIP observed utilizations. Table II presents the comparison results. Comparing SIP utilizations with the model-derived utilizations using the SIP estimated DMP (13.4), the error involved in

TABLE I.—Typical 1100/41 Parametric Measurement

Number of devices = 8	
SIP estimated DMP= 13.4	
u(0) = 27.00	p(0) = .247
u(1) = 40.42	p(1) = .452
u(2) = 50.10	p(2) = .005
u(3) = 600.88	p(3) = .057
u(4) = 1730.12	p(4) = .080
u(5) = 10.20	p(5) = .014
u(6) = 33.71	p(6) = .145
u(CPU) 68.17	

each component is about seven percent. However, it is noticed that all predicted utilizations are overestimates, indicating that DMP is incorrectly calculated. Changing the DMP to 7.5 produces a model which correctly matches the utilizations of all devices within 0.3 percent. The value, 7.5, is termed the effective DMP and is found by calibration,⁶ by allowing the DMP to vary to a point where a modeled parameter (e.g., CPU throughput) matches the observed parameter. The effective DMP was verified by independent techniques to be the actual multiprogramming level. The system was randomly halted and the number of jobs in execution was noted. This number is less than the SIP reported number of runs in core memory because of the problems mentioned in the third section.

The three models were tested on 37 separate time periods, and the model utilizations matched the observed utilizations within one percent in all cases on all devices when the effective DMP was used. The implication is that observed performance can be accurately modeled for any given time period, provided the correct parameters are supplied. It is further indicated that the assumptions (e.g., exponentially-distributed service times) do not seriously affect the model.

We now consider the prediction of the 1100/42 performance based upon the 1100/41 Model 1, and the prediction of the upgraded 1100/42 performance based upon the 1100/42 Model 2 without the new devices. We have already established the fact that in all the various models, the actual SIP measured performance is very accurately predicted by the models, if the correct parameters are provided. It is theoretically possible to have models with differing, but offsetting, parametric values which yield a similar performance measure, such as CPU throughput. However, it is our ex-

TABLE II.—Utilization Comparisons

Channel	SIP Observed Utilization	DMP 13.4	Modeled Utilization		
			Percent Error	DMP 7.5	Percent Error
0	.574	.613	6.8	.575	0.2
1	.703	.752	7.0	.705	0.3
2	.006	.006	0.0	.006	0.0
3	.006	.006	0.0	.006	0.0
4	.003	.003	0.0	.003	0.0
5	.086	.091	5.8	.086	0.0
6	.270	.288	6.7	.270	0.0
CPU	.922	.985	6.8	.923	0.1

TABLE III.—1100/42 Predicted Parametric Results

Parameter	Model Predicted	Actual	Percent Error
DMP	9.2	8.0	15.0
u(0)	30.95	29.72	4.1
u(1)	40.58	41.07	1.2
u(2)	48.84	13.71	256.2
u(3)	871.96	735.66	18.5
u(4)	1727.66	1722.62	0.2
u(5)	10.44	8.17	27.8
u(6)	37.03	34.89	6.2
u(CPU0)	72.89	65.05	12.0
u(CPU1)	72.89	65.06	12.0
p(0)	.251	.235	6.8
p(1)	.440	.373	18.0
p(2)	.005	.019	73.7
p(3)	.065	.071	8.5
p(4)	.072	.090	20.0
p(5)	.027	.043	27.0
p(6)	.140	.169	17.2

perience that parametric prediction errors do not offset each other, but compound performance prediction errors. Therefore, it suffices to present the results only for the parametric prediction.

Table III gives the results of predicting the 1100/42 parameters from 1100/41 observed data. The parameters used for the prediction were averages from several hours of observed data of the 1100/41. All parameters were assumed to remain the same, except for the service rate of CPU 1 which was predicted to equal the service rate of CPU 0. The actual parameters are likewise averages from several hours of observed data of the 1100/42.

Table IV gives the corresponding results of predicting the parameters of the 1100/42 with the new drum and disc units (Model 3) from the observed data of the 1100/42 without the new units (Model 2). The parameters are predicted to be

TABLE IV.—Upgraded 1100/42 Predicted Parametric Results

Parameter	Model Predicted	Actual	Percent Error
DMP	8.0	3.9	105.1
u(0)	29.72	42.49	30.1
u(1)	41.07	46.30	11.3
u(2)	13.71	41.77	67.2
u(3)	735.66	1093.80	32.7
u(4)	1722.62	1726.42	0.2
u(5)	8.17	9.73	16.0
u(6)	34.89	45.23	22.9
u(7)	41.07	41.30	0.5
u(8)	76.92	75.12	2.4
u(CPU0)	65.05	86.15	24.5
u(CPU1)	65.06	86.15	24.5
p(0)	.118	.092	28.3
p(1)	.187	.267	30.0
p(2)	.019	.039	51.3
p(3)	.071	.092	22.8
p(4)	.090	.065	38.5
p(5)	.043	.036	19.4
p(6)	.169	.012	1308.3
p(7)	.186	.293	36.5
p(8)	.117	.104	12.5

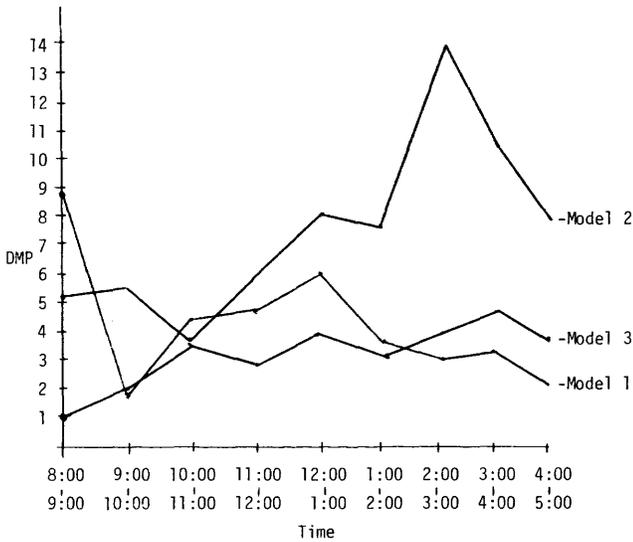


Figure 4—Effective DMP hourly variability.

identical to those observed in Model 2 (Table III, Column 3) with the following additions. The predicted service rate of the new drum is set equal to the observed service rate of the existing drum unit. The predicted service rate for the new disc channel is 76.92 transfers per second. This value is based upon the hardware specifications for the new disc units and upon the predicted number of words per transfer. The number of words per transfer is predicted to be identical to that observed from the existing disc units. The motivation behind acquiring the new drum and the new disc systems is to lighten the loads on channels 1 and 0, respectively. Therefore, the existing load placed on the drum is predicted to be evenly divided between the two drums in the new system,

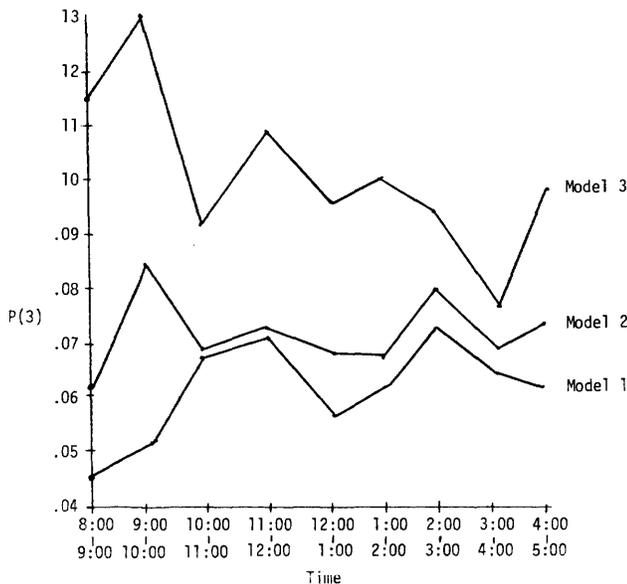


Figure 5—P(3) hourly variability.

i.e., $p(1)$ is predicted to equal $p(7)$ where the sum of $p(1)$ and $p(7)$ in Model 3 equals the observed $p(1)$ in Model 2. Likewise for the disc systems, $p(0)$ is predicted to equal $p(8)$ where the sum of $p(0)$ and $p(8)$ in Model 3 equals the observed $p(0)$ in Model 2.

To investigate the source of the poor results of predicting the model parameters (Tables III and IV), we analyze the degree of parametric change observed on an hour-by-hour basis for the three system models. Figures 4 to 7 give the SIP observed results for four of the parameters. The remaining parameters possess similar results.

CONCLUSIONS AND IMPLICATIONS

We have been successful in constructing accurate models of specific computer systems. Given the correct model parameters, the system performance from the model closely matches the actual system observed performance. However, the major application of system modeling is the prediction of performance when the system configuration is altered. The prediction results presented are disappointing but useful. We have been unable to satisfactorily predict system performance even on an hour-by-hour basis, where parameters from one hour are used to predict performance of the following hour. Using larger time intervals for prediction yields poorer results. The major problem is traced to para-

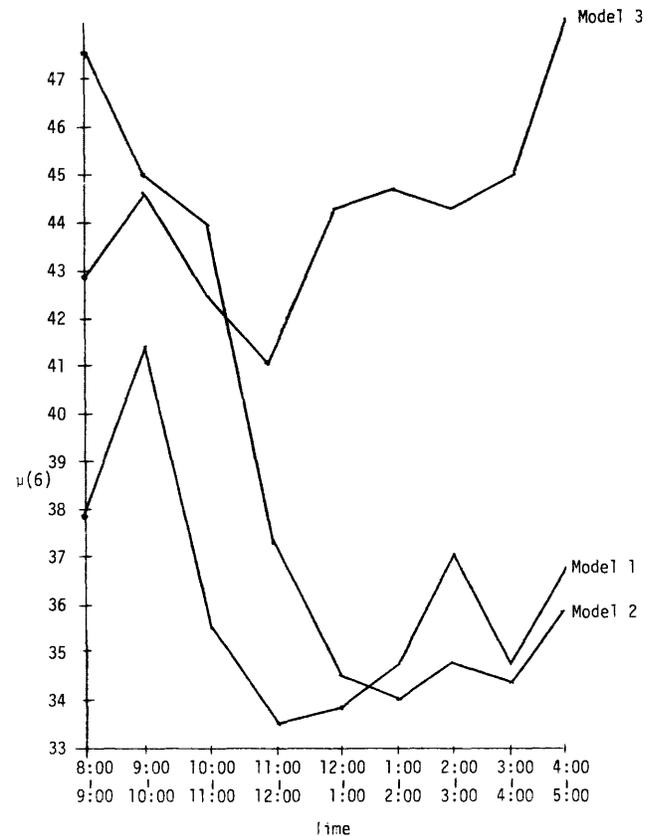


Figure 6— $\mu(6)$ hourly variability.

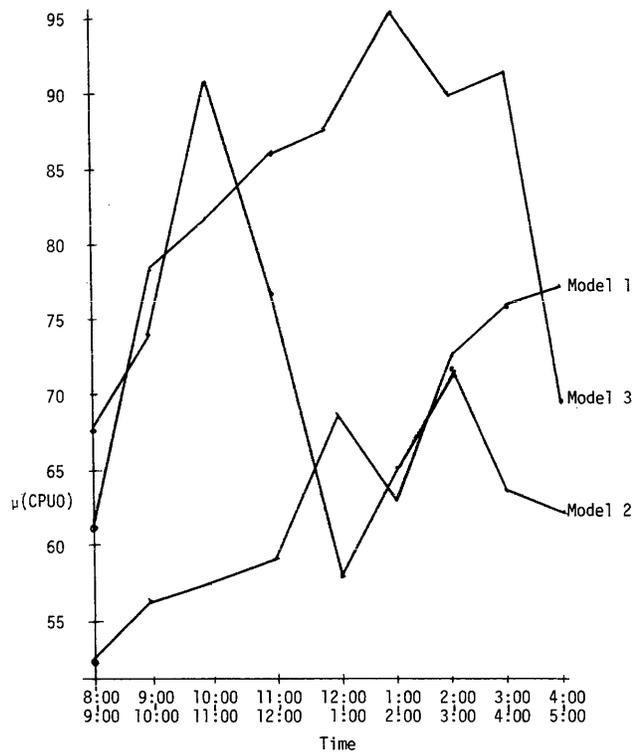


Figure 7— $\mu(\text{CPU0})$ hourly variability.

metric instabilities. As evidenced by Figures 4-7, parametric variations of 20 to 30 percent on an hourly basis are not uncommon. The usefulness of these results is to alert system analysts to these instabilities, even when the model structure is correct.

The model parameters are a function of the total demands placed on the system resources. Clearly these demands in-

clude both the user processing requests and the system overhead activities. In addition, the parametric values depend upon the device hardware characteristics. The interactions between the user demands, overhead activities and device characteristics must be considered in developing useful system models. The results presented in this paper represent some of our initial findings from applying traditional modeling techniques to real systems.

ACKNOWLEDGMENTS

We are grateful for the suggestions and contributions the following people have made: the University of Maryland Computer Center's System staff, H. S. Kresin, S. K. Tripathi, K. D. Gordon, J. R. Agre, J. M. Mohr, and R. F. Sparks.

REFERENCES

1. Buzen, J. P., "Queueing Network Models of Multiprogramming," Ph.D. Dissertation, Div. of Engineering and Applied Physics, Harvard University, Cambridge, Mass., May 1971.
2. Buzen, J. P., "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *CACM*, Vol. 16, No. 9, September 1973, pp. 527-531.
3. Denning, P. J., and J. P. Buzen, "The Operational Analysis of Queueing Network Models," *Computing Surveys*, Vol. 10, No. 3, September 1978, pp. 225-261.
4. Denning, P. J., "Thrashing: Its Causes and Prevention," *AFIPS Conf. Proc.*, Vol. 33, Part I, 1968, pp. 915-922.
5. Dowdy, L. W., and D. V. Foster, "Alternate Algorithms for the Computation of the Normalization Constant for Certain Closed Queueing Networks," Dept. of Computer Science Technical Report TR-649, University of Maryland, College Park, Maryland, 1978.
6. Rose, C. A., "A 'Calibration-Prediction' Technique for Estimating Computer Performance," *AFIPS Conf. Proc.*, Vol. 46, 1977, pp. 813-818.

Aids to the development of network simulators

by IMRICH CHLAMTAC and W. R. FRANTA

University of Minnesota
Minneapolis, Minnesota

INTRODUCTION

In this paper we describe the features of a program designed to simulate computer networks. The networks are assumed to consist of hosts attached to one or more network nodes (communication units) which in turn communicate with one another via one or more communication channels. The program accounts for user-host, host-node, node-node, node-host, and host-user protocols, as well as network topology and hardware characteristics. As reported in a later section, the program has been primarily used to conduct simulations to ascertain the performance (in terms of message or packet throughput and delay) for a variety of node-node channel access protocols, for nodes connected to a single common communication medium (e.g. a multi-drop cable or radio channel). It is, however, possible to accommodate networks in which nodes may be connected to several separate communication channels. Further, it is possible to simulate the behavior of store-end-forward networks as they can be viewed as a network in which the nodes are connected to multiple communication links. For either class of network the program can be used to verify protocol correctness or to assess network performance.

MODEL PROGRAM FEATURES

The model program is designed (layered) to account for interactions at the user-host, host-node, node-node interface levels. The model program is constructed in a highly modular manner, so that alterations at one interaction level can easily be made without affecting the other layers. That is, the program is designed to accommodate alterations for the inclusion of additional or altered features. The program was developed along modular lines so as to make it more receptive to the inclusion of new protocols, buffering strategies, etc. Its basic organization accommodates the following characteristics.

Host-User Characteristics—Including message inter-generation and length modules, specification of the node or nodes to which a host is attached, and modules to handle the host-node message transfer mechanisms, including considerations of available buffer storage in both the host and node, and message addressing.

Node-Characteristics—Including queues and buffers for outgoing messages and message reassembly (from packets), buffers for retransmission of packets (messages) and acknowledgments.

- Additionally, node modules characterize the channel access protocol, i.e., node-node interactions (discussed later) and node-host interfaces including message addressing and reassembly issues. Finally, modules are included to accommodate channel-error characteristics (due to noise) and error-handling procedures (for errors resulting from noise or message collision, as applicable).

Measures—The model program modules currently collect data to provide estimates, where applicable, on:

1. Channel throughput.
2. Average node dependent transmission delays (message or packet).
3. The total offered channel traffic.
4. The expected length of the various node queues.
5. The number of doubly-received packets (messages).
6. The number of packet collisions.
7. Graphs for delay/throughput and delay/offered traffic.

Additionally, the modules will produce a full trace for each message that enters the network, from which additional information can be abstracted.

NODE-NODE CHANNEL ACCESS PROTOCOLS

We have stated that, on the one hand, the model program is constructed in a highly modular fashion to mitigate the process of altering one or more levels of protocol. We have, on the other hand, also attempted to make the extant modules general so that changes can be accomplished by varying module parameters rather than module structure. It is possible, for example, to characterize seemingly unrelated channel access protocols via *scheduling functions* and the notion of *node-groupings*, and to capture the essence of network topology in the *connectivity* or *distance-control* matrix.

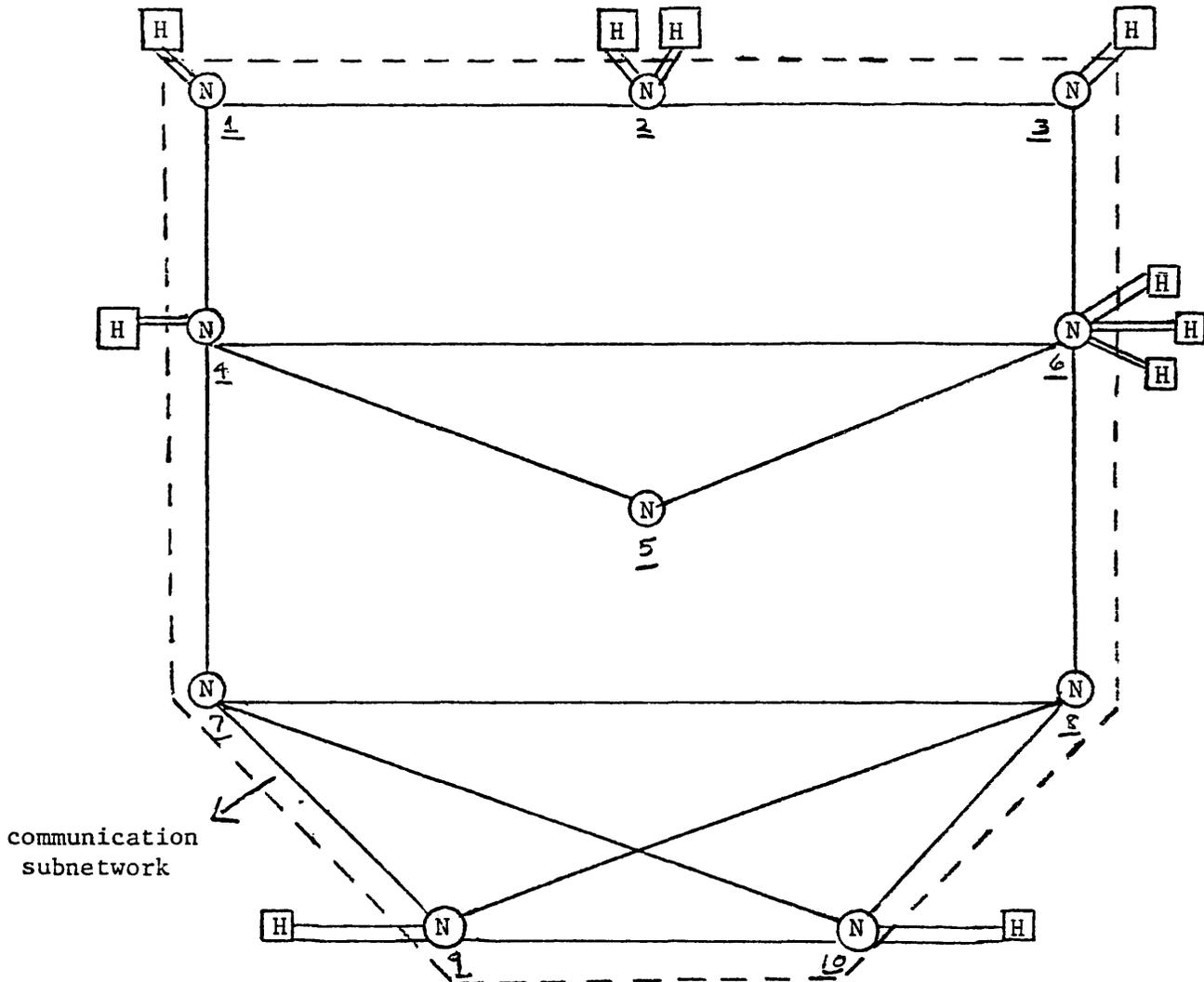


Figure 1a—Sample network, where H=host: N=node: The set of *host+communication subnetwork=computer communication network* solid lines between a pair of nodes implies they are within range and in LOS of each other. Subscripts identify the corresponding node sites.

To explain these notions we must imagine the K nodes to be mapped by a grouping function $g(i)$, into m , $1 \leq m \leq k$, groups such that each node with index- i is a member of one and only one group. For purposes of scheduling transmission times a node assumes the group identity, i.e., its identifier becomes the index of the group of which it is a member. The LOS (line-of-sight) matrix, X , for a network is $K \times K$ and has entries x_{ij} , such that:

$$x_{ij} = \begin{cases} 1 & \text{node-}i \leftrightarrow \text{node-}j^* \\ 0 & \text{node-}i \nleftrightarrow \text{node-}j \end{cases}$$

where \leftrightarrow denotes the presence of a direct point-to-point channel (not necessarily dedicated) between *node- i* and *node- j* , and \nleftrightarrow its absence. Using X , the connectivity matrix DCM (distance control matrix) can be constructed.³ The

DCM matrix has entries d_{ij} such that:

$|d_{ij}|$ = minimum number of node-node transmissions required for a packet (message) to propagate from *node- i* to *node- j* .

The value of $\text{sign}(d_{ij})$ can be used for secondary purposes and an example will be given below. Thus a DCM can be established to represent, for example, a star or loop network, a network organized around a multiple drop cable, a network based upon a radio channel with an arbitrary dispersion of hidden nodes, or even a store and forward network. A sample network is shown in Figure 1a, and its corresponding DCM matrix is given in Figure 1b.

Using the grouping function, g , the connectivity matrix, DCM, and the scheduling function a wide variety of network configurations and transmission protocols are easily represented. To so demonstrate, we must first define the sched-

* Note that this definition reverses a definition given in Reference 9.

Node j \ Node i	1	2	3	4	5	6	7	8	9	10
1	-0	+1	-2	+1	+2	-2	+2	-3	-3	-3
2	+1	-0	+1	-2	-3	-2	-3	-3	-4	-4
3	-2	+1	-0	-2	-2	+1	-3	+2	+3	+3
4	+1	+2	-2	-0	+1	+1	+1	+2	+2	+2
5	-2	-3	-2	-1	-0	-1	-2	-2	-3	-3
6	-2	+2	+1	+1	+1	-0	+2	+1	+2	+2
7	+2	-3	-3	+1	-2	-2	-0	+1	+1	+1
8	-3	+3	+2	-2	+2	+1	-1	-0	+1	+1
9	-3	-4	-3	-2	-3	-2	-1	-1	-0	-1
10	-3	-4	-3	-2	-3	-2	-1	-1	-1	-0

Figure 1b—DCM matrix for network of Figure 1a.

uling function. To determine the next potential time at which it may transmit, a ready node with *index-k*, uses the scheduling function $SF(i, j)$, where $g(k)=i$ and $j=g(1)$ if *node-1* transmitted last, in the algorithm which appears below. That is each ready node can be thought to execute Algorithm A.

Algorithm A

1. Determine, T , the next *scheduling period* leading edge.
2. Compute the next potential transmission time (for the ready node with *index-i*) by $t_i = T + SF(g(i), j)$, with j the group identification of the node which successfully transmitted last.
3. At time, t_i , if the channel is sensed idle (as determined by the LOS relationships given by the DCM) then transmit; otherwise return to Step 1.

The value of the scheduling period leading edge, T , is usually given by the trailing edge of a transmission (successful or not), or by the value of a counter which is zeroed and begins to increment at the termination of every transmission (used in the case where nodes approach an idle channel). Details on the determination of T can be found in References 1 and 3.

The expressive power of Algorithm A together with $g(\cdot)$, $SF(\cdot)$ and the DCM is demonstrated by Figure 2, wherein networks with a wide variety of channel access protocols are characterized by various configurations of g , SF , and DCM values. Descriptions of the protocols mentioned in Figure 2 can be found in Reference 7 (ALOHA, CSMA variants, TDMA), Reference 8 (MSAP), Reference 1 (BRAM), Reference 2 (parametric BRAM), and Reference 3 (SUPBRAM).

Some are applicable to a cable or radio channel (e.g., CSMA, BRAM) or to radio channels with a network containing hidden nodes (e.g., BRAM, SUPBRAM, etc). In the case of BRAM or SUPBRAM operating in a radio network with hidden nodes, the DCM sign is used to convey additional message routing information. A sample DCM with

routine information is given in Figure 1b, and Reference 3 should be consulted for details.

Other protocols can also be captured by the structures, and to imagine doing so the reader must bear in mind that while certain protocols may not be realized by these constructs (in an actual system) they can be represented by them for simulation.

FORMAL VALIDATION OF MODELS

Typically a simulation model program is validated employing pilot runs and statistical tests. It is possible, however, to formally validate the model program's input/output behavior against the actual system. Such validation can thus serve as a kind of "proof of correctness" of the model program. The methodology employed is described in Reference 10 and in this section, for demonstration, we apply it to a simple network model which employs a slotted ALOHA transmission protocol. The reader should be assured, however, that the technique is applicable to any model program (configuration, protocol, etc.) constructed.

The methodology is based on the notions of base model, an experimental frame and a lumped model. In summary (consult Reference 10 for details), a *base model* is a model capable of accounting for all the input/output behavior of the actual system. As a consequence it must be faithful, detailed and consequentially complex representation of the system.

An *experimental frame* characterizes a limited set of circumstances under which the actual system is to be observed. Thus the experimental frame serves to define the subset of allowable input/output behaviors which are of interest.

The *lumped model* results from a process of abstraction by taking the base model and simplifying it (by grouping or lumping components, etc.) so that it accounts for the input/output behaviors specified by the given experimental frame, but not necessarily others.

The validation involves establishing a structural homomorphism between the base model and the lumped model as determined by the well defined experimental frame. In the network context, the base model is required to account for all network nodes, queues, protocols, etc. as encountered in the actual system.

To establish a homomorphism between the base and lumped models a formal machinery is necessary. The methodology of Reference 10 proceeds by first generating an informal description of the models in terms of system *components*, *descriptive variables* (descriptive of the components) and *component interaction* rules. From the informal description a formal discrete event system specification (DEVS) springs. The formal description is given in terms of:

$$\begin{aligned} \alpha_1, \alpha_2, \dots, \alpha_n & \text{—The input variables,} \\ \beta_1, \beta_2, \dots, \beta_m & \text{—The state variables, and} \\ \delta_1, \delta_2, \dots, \delta_n & \text{—The output variables,} \end{aligned}$$

with the cross product of the ranges of these variables giving the set of INPUTS, STATES, and OUTPUTS of the system.

Protocol name	SF(i,j)=	g(i)=	LOS/ DCM
ALOHA [7,9]	0	1	$d_{ij} = y \neg \text{LOS}$ $x_{ij} = 1 \text{ LOS}$
l - persistent CSMA p - persistent CSMA [7]	0	1	$x_{ij} = Y$
non-persistent case CSMA [7]	0	1	$x_{ij} = y$
prioritized CSMA see [4]	i	i	$x_{ij} = 0$
TDMA (time division multiple access) [7,9]	i packet transmission time	i	$x_{ij} = 1$
MSAP see [8]	$(i-j+k) \bmod k$ if $i \neq j$ 0 if $i = j$	i	$x_{ij} = 0$
BRAM or centralized polling [1]	$(i-j+m) \bmod m$ if $i \neq j$ m if $i = j$	$1 \leq g(i) \leq m$ $1 \leq m \leq k$	$x_{ij} = 0, \text{LOS case}$ $d_{ij} = y, \neg \text{LOS case}$
SUPBRAM [3]	See [3]	$1 \leq g(i) \leq m$ $1 \leq m \leq k$	arbitrary, d_{ij} see [3]

Figure 2—Protocols currently captured by $g(\cdot)$, SF(\cdot) and DCM configurations. k =number of nodes in network; y =topology-dependent value; LOS \Rightarrow within line of sight and range.

To account for the discrete time points at which events (inputs, outputs, state changes) occur, the notion of a *hatching time* is introduced, to specify the times at which events occur. The next hatching time is specified by examination of a non-ordered set of linearly *decreasing* variables, T_1, T_2, \dots, T_k , (a kind of sequencing set) such that at time t_i , the *next* event time (hatching time) is given by $\dagger(s) = t_i + \min\{T_1, T_2, \dots, T_k\}$ where s refers to the state variables. The formalism is completed by specifying first:

$$\delta\phi: \text{STATES} \rightarrow \text{STATES}$$

which describes the internal or endogenous *state transition* that occurs in traversing from time t_i to time $t_i + \dagger(s)$, specifying second:

$$\delta ex: Q \times \text{INPUTS} \rightarrow \text{STATES}$$

which characterizes the transitions brought about by externally-generated events, with Q denoting the set of pairs (s, e) where s denotes system state (as before) and $e \in R [0, \dagger(s)]$ (i.e., a real number in the range $(0, \dagger(s, e))$), and finally specifying:

$$\lambda: Q \rightarrow \text{OUTPUT}$$

the *output function*. Taken together, the set $\langle \text{INPUTS}, \text{STATES}, \text{OUTPUTS}, \delta\phi, \delta ex, \lambda, \dagger \rangle$ constitutes the DEVS for the model (lumped or basic).**

To apply the methodology we must first develop a DEVS for the base model, then determine the experimental frame from which the lumped model and its DEVS can be derived.

Owing to limitations of space, we will not present the

** Additional discussion can be found in Reference 10.

DEVS for the base model, but instead, we will first informally describe how our sample lumped model results from the base model; second, we give the DEVS for the lumped model; and finally informally establish the *required* homomorphism, and hence the validation of the lumped model.

To proceed, an experimental frame must be specified, and for simplicity we assume the experimental frame as given by the single variable which gives channel utilization (under the assumption of a slotted ALOHA protocol as assumed earlier). The lumped model then results in:

1. Omitting in the lumped model variables, components, and interactions which account for user \leftrightarrow host \leftrightarrow node relationships.
2. Accounting for the omissions of (1) by replacing certain deterministic variables having to do with packet length, etc. by variates drawn from distributions determined appropriate by examination of the omissions.
3. Coarsening in the lumped model the range of certain descriptive variables in the base model (e.g., packet identification in the lumped model need only specify the associated source node, while in the base model packet identification must include source node identification, destination, node identification, sequence number within a message, message number, etc.) and eliminating several queues.

The network lumped model informal description

Under the assumptions given above the lumped model consists of components, descriptive variables and interactions as given next.

Components—Source, packet-queue, retransmission-queue, channel, with their obvious interpretations.

Descriptive Variables—For the source we have (the variable) NEW PACKET which can assume values X , with

Further, using the state variables given in the informal description, we obtain:

$$s \in \text{STATES} = \left\{ \begin{array}{l} \text{RETR. TIME. SEED, PACKET. QUEUE}_1, \text{RETR. QUEUE}_1, \text{CHANNEL}_1 \\ \text{PACKET. QUEUE}_2, \text{RETR. QUEUE}_2, \text{CHANNEL}_2 \\ \vdots \\ \text{PACKET. QUEUE}_K, \text{RETR. QUEUE}_K, \text{CHANNEL}_K \end{array} \right\}$$

$$= \left\{ \begin{array}{l} r_1, y_1, (Z_1, T_1), \sigma_1 \\ y_2, (Z_2, T_2), \sigma_2 \\ \vdots \\ y_K, (Z_K, T_K), \sigma_K \end{array} \right\}$$

The time advance function becomes:

$$t(s) = \min\{\sigma_1, \sigma_2, \dots, \sigma_K, T_1, \dots, T_K\}$$

and for simultaneous events we choose the tie-breaking rule function:

$$\begin{aligned} &\text{SELECT} (\{\text{TRANSM. TIME. LEFT, SCHEDULED. TRANS. TIME}\}) \\ &= \text{TRANSM. TIME. LEFT} \\ &\text{otherwise: SELECT} (\{X\}) = X \text{ (in the absence of simultaneous} \\ &\text{events)} \end{aligned}$$

range $\chi \in \{0, 1, 2, \dots, K\}$. This specification is compactly written as:

$$\begin{aligned} &\text{NEW.PACKET: } \chi \in \{0, 1, 2, \dots, K\} \\ &\text{for a } K \text{ node network. Additionally we have,} \\ &\text{PACKET.QUEUE: } y_i \in \mathcal{Z}^+ \end{aligned}$$

where y_i gives the length of the queue at node i , and

$$\text{RETR.QUEUE: } (Z_i, T_i) \in \mathcal{Z}^+ \times \mathcal{R}$$

with Z_i denoting the length of the retransmission queue at node i and T_i giving the scheduled retransmission time with T_i a variable drawn from a retransmission time generator with seed $r_i \in [0, 1]$. For the channel we have $\text{TRANS. TIME. LEFT } i = \sigma_i \in [0, \text{Packet transmission time}]$ with K , packet transmission time (PTT), and the retransmission time ($\text{SCHEDULED. TRANS. TIME}$) distribution established parameters.

Component Interactions—Component interactions are specified by noting that a newly-arrived packet joins PACKET.QUEUE , according to X , the value assigned NEW.PACKET , causing Z_i to be incremented. If $Y_i = 1$ the node proceeds to transmit by increasing σ_i to the value packet transmission time. If $Y_i > 1$, a transmission is scheduled for the time when σ_i becomes zero. If σ_i is reset while another $\sigma_j, j \neq i$, is non-zero a retransmission time is drawn and Z_i , and T_i are adjusted.

Formal lumped model description

The formal description constitutes the DEVS for the lumped model and contains the input variable NEW.PACKET such that:

$$\text{INPUTS} = \{\phi, 1, 2, \dots, K\}$$

The next hatching time will therefore be given by:
 $t_{i+1} = t_i + \min(\sigma_i, T_i)$.

If we let $T_l = \min T_i$, and $\sigma_l = \min \sigma_i$, then $\delta\phi$ is given by:

if $T_l < \sigma_l$

$$\text{and if } \sigma_l = 0 \quad S = \begin{pmatrix} \Gamma(r_1), y_1, (\overline{\overline{Z_1, T_1 - T_l}}), \sigma_1 - T_l \\ \vdots \\ y_l, (\overline{\overline{Z_l, -1, 0}}), \sigma_l + PTT \\ \vdots \\ y_\kappa, (\overline{\overline{Z_\kappa, T_\kappa - T_l}}), \sigma_\kappa - T_l \end{pmatrix}$$

$$\text{and if } \sigma_l > 0 \quad S = \begin{pmatrix} r_1, y_1, (\overline{\overline{Z_1, T_1 + \sigma_l - T_l}}), \sigma_1 - T_l \\ \vdots \\ y_\kappa, (\overline{\overline{Z_\kappa, T_\kappa + \sigma_l - T_l}}), \sigma_\kappa - T_l \end{pmatrix}$$

if $T_l > \sigma_l$

$$\text{and if } y_l \neq 0 \quad S = \begin{pmatrix} r_1, \overline{y_1}, (\overline{\overline{Z_1, T_1 - \sigma_l}}), \sigma_1 - \sigma_l \\ \vdots \\ y_l - 1, \quad \vdots, \quad 0 \\ \vdots \\ y_\kappa, (\overline{\overline{Z_\kappa, T_\kappa - \sigma_l}}), \sigma_\kappa - \sigma_l \end{pmatrix}$$

$$\text{and if } y_l = 0 \quad S = \begin{pmatrix} r_1, y_1, (\overline{\overline{Z_1, T_1 - \sigma_l}}), \sigma_1 - \sigma_l \\ \vdots \\ y_l, (\overline{\overline{Z_l + 1, T_l - \sigma_l}}), 0 \\ \vdots \\ y_\kappa, (\overline{\overline{Z_\kappa, T_\kappa - \sigma_l}}), \sigma_\kappa - \sigma_l \end{pmatrix}$$

where $X = \begin{cases} -\infty & x < 0 \\ x & x \geq 0 \end{cases}$, $X = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$, and $\Gamma(r_i)$ denotes a drawing from a stream with seed r_i , and initially $\sigma_i = T_i = 0$, and $T_i = -\infty$ if $t_i > T_i$. In the absence of external events, i.e., when SOURCE = Φ , the next hatching time, t_{i+1} , is given by $t_{i+1} = t_i + \min(\sigma_i, T_i)$. For times, t , at which SOURCE $\leftarrow m$, $l \leq m \leq \kappa$, the state at time $t_i + t$ is given by:

$$\delta_{ex}(s, t, m) = \begin{cases} r_1, y_1, (Z_1, T_1 - t), \sigma_1 - t \\ y_m + 1, \quad \vdots, \quad \vdots \\ y_\kappa, (Z_\kappa, T_\kappa - t), \sigma_\kappa - t \end{cases}$$

Finally using the experimental frame which characterizes channel utilization, the output function becomes:

$$\lambda(S) = \begin{cases} \text{YES} & \text{if } \exists m \ni \sigma_m \neq 0 \wedge \sum_{\substack{i=0 \\ i \neq m}}^{\kappa} \sigma_i = 0 \\ \text{NO} & \text{otherwise,} \end{cases}$$

as for $\sigma_i \neq 0$ for multiple i values collision results. From $\lambda(s)$ channel utilization is trivially obtained.

Lumped model validation

Recall that our purpose is to validate the lumped model for the given experimental frame by using the DEVS of the base and lumped models. This validation is done by showing the existence of a homomorphism between the base and lumped models. That is, a valid lumped model has the same input/output behavior as the base model for the given experimental frame.

The homomorphism is established by proving the existence of a mapping h , h : (base model states) onto (lumped model states), which preserves the time advance, transition and output functions. To formally establish the mapping requires a formal DEVS for the base model which we have not supplied. We proceed informally, therefore, by suggesting that h has the form shown in Table I, wherein grouped states of the base model correspond to states of the lumped model (assuming both models initialized identically).

As a result the states have been grouped (or lumped) from many to four. It is important to note that the mapping, h , must be onto so that each component interaction in the base model is a member of only one group in the lumped model. Having suggested a viable homomorphism, h , we can proceed to establish the preservation of the input/output time advance and transition functions.

Time advance preservation

For both base and lumped models the next hatching time for a channel event is determined by the end of a transmission or retransmission. Denoting the columns of the grouped base states by g_1, \dots, g_4 (from Table I) and the columns of the lumped states by S_1, \dots, S_4 , then $\#(s) = \min(\sigma_i, T_i)$ for the lumped model, and $\#'(g) = \min(\sigma_i, T_i)$ for the base model. Since we have assumed identical initial conditions for the two models, we have:

$$\begin{array}{ccc} g = (g_1, \dots, g_4) \#'(g) & & \\ \downarrow h & \searrow & \min(\sigma_i, T_i) \\ s = (s_1, \dots, s_4) \#(s) & \nearrow & \end{array}$$

which guarantees the preservation of the time advance function.

Transition function preservation

To formally prove transition function preservation we have to show that given initial correspondence between the states of the two models, that any (group \rightarrow group or state \rightarrow state) transition brings both models into corresponding states. Specifically, we have to show that: $h(\delta\phi(g)) \rightarrow \delta\phi(s)$, i.e., for the columns g_1, g_2, g_3, g_4 we obtain by the h mapping the corresponding states S_1, \dots, S_4 for the columns of the lumped model.

Additionally, we must show that s_i represents g_i following

TABLE I—Suggested Homomorphism between Grouped Base Model States and Lumped Model States

GROUPED BASE MODEL STATES "BECOME" LUMPED MODEL STATES	
1. User \rightarrow host \rightarrow node \rightarrow packet queue;	packet queue
2. Control unit \rightarrow transmission buffer \rightarrow channel	channel
3. Receiver buffer retransmission queue	retransmission queue
4. Retransmission seed	retransmission seed

all state transitions. Although our ability is limited here due to not having a DEVS for the base model, we can still show the preservation by basic observation. We begin by recalling that the lumped model states are in fact representatives of "groups" of base model states. We further note that we assume all transitions within a group of base model states to be zero time transitions that do not affect σ_i, T_i . Thus only transitions among states from different groups have to be observed—and this is done by observing the state changes in the corresponding states of the base model. Because of this special relationship between base and lumped model structures, the transition function preservation can be justified without use of DEVS.

Output function preservation

In this case preservation dictates that corresponding states (in the two models) have to provide the same output. In the lumped model a YES period is recorded for those cases where a single transmitter has a non zero σ_m value. In the base model a receiving controller records correct reception, i.e., a YES period if and only if the channel is busy with only one transmission. It is easily seen that the two YES periods correspond so that $\lambda(\text{lumped model}) = \lambda(\text{base model})$, since in the base model we record YES if:

$$\sum_{\substack{i=1 \\ i \neq m}}^K \sigma_i = 0 \wedge \exists \sigma_m \neq 0.$$

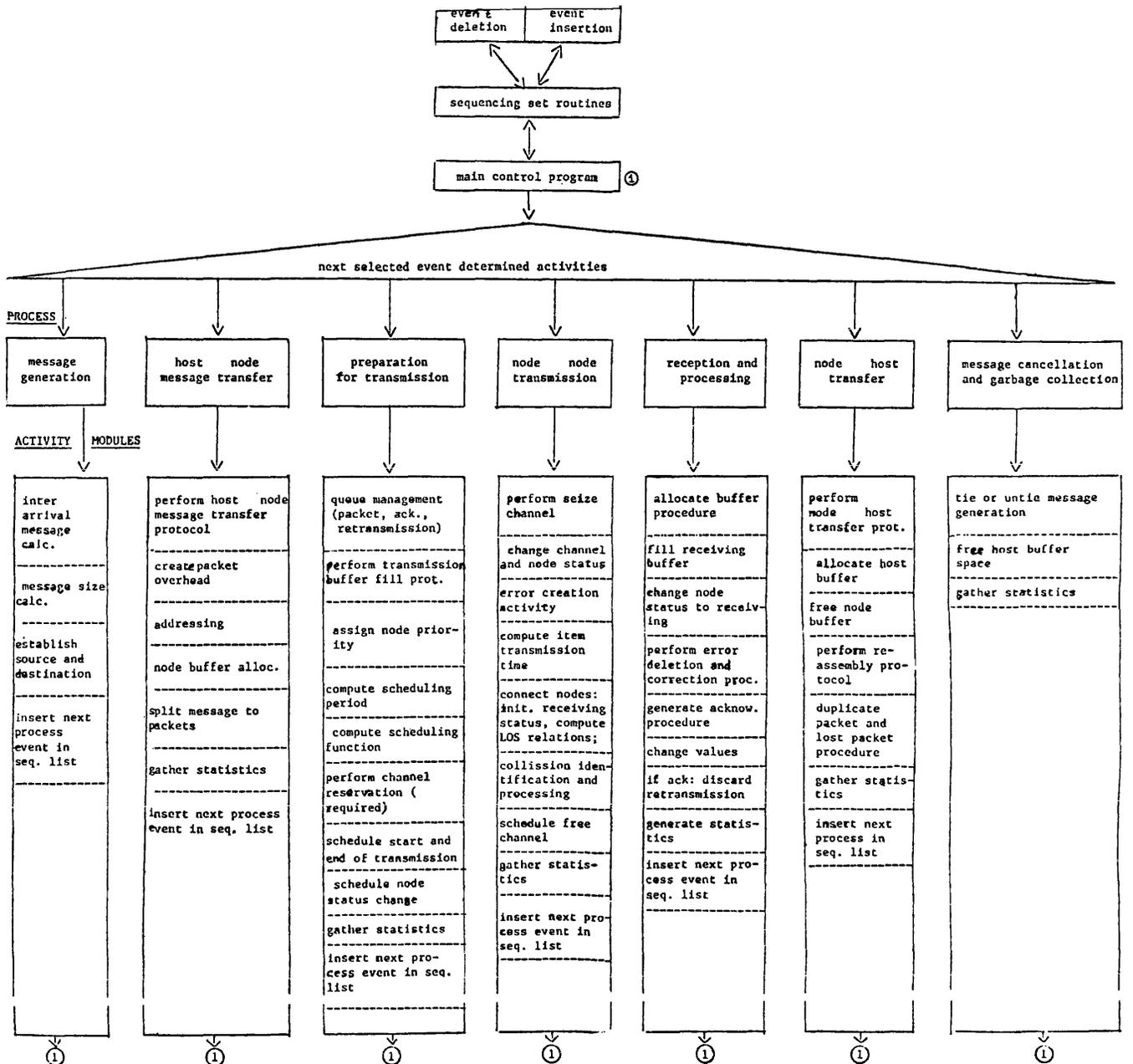


Figure 3—Model program structure.

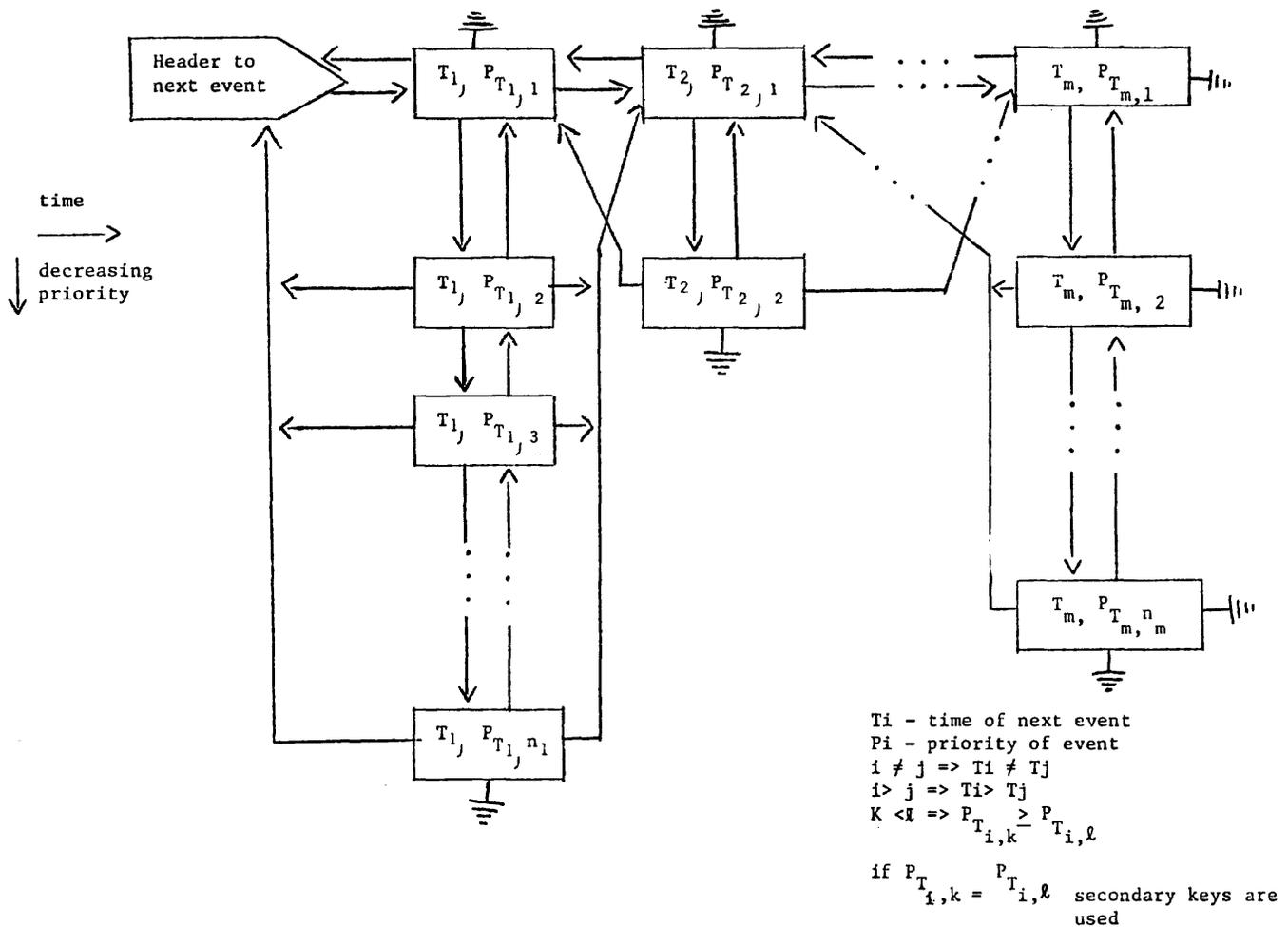


Figure 4—Model program sequencing set structure.

Commentary on validation approach

Our purpose has been to demonstrate a formal approach to the validation of a simulation model. Our example was purposely chosen to be simple, where complexity of the lumped model is dependent upon the experimental frame and actual system. For an experimental frame addressing only channel utilization only four-state descriptions were necessary. Quite obviously, as the experimental frame complexity becomes more comprehensive, the lumped model approaches the base model in complexity, and Reference 10 should be consulted for additional details.

We find the approach helpful in structuring network model programs (configurations, protocols, etc.) even when applied with no more rigor than was used in the example. Specifically, its use promotes the same forethought and structuring exercises for simulation programs as do verification and specification techniques for programming generally.

We believe errors have been avoided and more highly structured lumped models produced in reasonable times as

a result of only informal use of the methodology. In any case it serves as a guide to the abstraction process (always difficult), is consistent with the view of simulation we believe most natural,⁵ and so represents a formal approach to the art of modeling (model development), and several variants of the network model program were informally validated using the approach.

MODEL PROGRAM IMPLEMENTATION CONSIDERATIONS

Guided by the validation procedure of the fourth section, and the scheduling function, $SF(\cdot)$, grouping function, $g(\cdot)$, and distance control matrix, DCM, as given in the third section, the model program is organized according to the modular structure shown in Figure 3. The program is written in FORTRAN (for efficiency, universality, and sequencing set structural reasons) and consists (for most variants) of around 3500 lines, which compile into approximately 62000 octal words of executable code on a CYBER 74.

Sequencing set considerations

The model program structure shown in Figure 3 is designed, of course, to promote the alterability of the program. That is, for example, all issues concerned with addressing at the *host*→*node* interaction level are contained in a single module so that addressing functions can be identified and altered (replaced) without affecting the remaining program. While serving the primary function of flexibility the structure causes certain potential overheads at execution time. Specifically because of the structure, i.e., the columns in Figure 3, it becomes necessary for activities at time t_i to schedule multiple other actions for time t_i . Moreover, to maintain the rectitude of the model the multiplicity of events scheduled for a given t_i must occur in a specified order. The use of conventional sequencing set structuring techniques, e.g., the linear list, (see Reference 6), causes considerable overhead in the insertion or deletion of event notices when there are multiple events scheduled for a given event time. Furthermore, these structures do not give particular attention to our paramount requirements on the ordering of the execution of these simultaneous events. The nature of our model program suggests use of the two-dimensional sequencing structure shown in Figure 4, wherein each activity scheduling notice contains not only a primary key giving time of occurrence, T_i , but also a secondary key $P_{T_i,j}$ which reflects the activities assigned priority (which determines its placement among the collection of events scheduled for T_i). Note further that the priority value assigned a scheduled event cannot be static (i.e., cannot be assigned *a priori* to the

activity) but rather must be assigned *dynamically* when the activity is given an occurrence time, with assignment based on the state of the system (the collection of events scheduled) when the scheduling occurs. For example, the order of handling the simultaneous events "set node busy status," "set channel busy status" occur in different orders dependent upon whether the system state is node transmission or node reception.

The two-dimensional structure, Figure 4, aids the assignment of activity priorities as well as the insertion and deletion of notices by reducing the number of scans (of notices) necessary to a more acceptable level than would be possible with conventionally used sequencing set structures. The structure thus allows use of the modular structure without the penalty in model program execution speed which would be incurred with conventional sequencing sets.

Input traffic generators

A large number of network simulations are conducted assuming that message intergeneration (arrival for transmission) times at *node-i* obey a Poisson process with parameter λ_i .

Typically a simulation program responds to this assumption by scheduling an activity "arrival" for each of the K nodes in the network. For large K this adds significantly to the number of event notices present in the sequencing set, and hence degrades notice insertion deletion operations. A cleaner more efficient approach is given by forming

$$\lambda = \sum_{i=1}^K \lambda_i \text{ and scheduling a single event with inter-event}$$

PROTOCOL	centralized star network	distributed network	node priority assignment	availability of slotted or unslotted model	centralized or distributed control	protocol dependant variants	separate Ack. channel or incorporated ack. channel	LOS/non LOS coverage
ALOHA	✓	✓	✓	Both	no control	existence of special file transfer protocol	both possibilities exist	non-significant
CSMA	✓		✓	Both	Both	l-pers., pers. CSMA p-pers., prioritized CSMA Pers. = Persistent	separate zero time	Both
BRAM	✓	✓	✓	Both	Both	simple, parametric, grouped for non LOS	separate zero time	Both
SUPBRAM	✓	✓	✓	Both	distributed	simple, grouped, l - optimized, l-m optimized	"	both (but specially designed for ¬LOS)
MSAP	✓	---	---	Both	centralized	-----	"	LOS case only
TDMA	✓	✓	✓	non-significant	---	-----	"	non-significant

Figure 5—Networks simulated to date.

times drawn from the exponential distribution with parameter λ . Each occurrence of the event signals "arrival" and the "cumulative distribution" of the λ_i , can be used to determine the node to which the message has arrived. Specifically if a random number U is drawn and $(1/\lambda) \sum_{i=1}^{j-1} \lambda_i <$

$u \leq \frac{1}{\lambda} \sum_{i=1}^j \lambda_i$, the message is declared to have arrived at node-

j . The procedure works due to the ability to split a Poisson stream into multiple streams via a multibranch Bernoulli trial, and serves to insure that a single, rather than $K \gg 1$, notice is in the sequencing set to signal message arrivals.

CONCLUSION

In this paper we have examined a number of techniques designed to aid our development of a modular network simulator designed to serve as a research tool. We believe the aids have contributed to our ability to easily simulate a variety of network topologies and access protocols. In Figure 5 we summarize the uses of the simulator to date. The uses shown by Figure 5 reflect our current interests rather than the limitations of the model.

REFERENCES

1. Chlamtac, Imrich, W. R. Franta and Dan Levin, "BRAM: The Broadcast Recognizing Access Method," submitted to *IEEE Trans. on Communications*.
2. Chlamtac, I., and W. R. Franta, "The operational performance of the Broadcast Recognizing Access Method in a network with hidden nodes," *TR 78-16*, Dept. Computer Science, University of Minnesota, July, 1978.
3. Chlamtac, I., and W. R. Franta, "A Description of the Supervisory Node Broadcast Recognizing Access Method (SUPBRAM)," *TR 78-18*, Dept. Computer Science, University of Minnesota, August, 1978.
4. Christensen, Gary S., and W. R. Franta, "Design and Analyses of the access protocol for Hyper channel networks," *Proc. 3rd USA-JAPAN Conference*, October, 1978.
5. Franta, W. R., *The Process View of Simulation*, Elsevier, North-Holland, 1977.
6. Franta, W. R., and K. Maly, "An Efficient Data Structure for the Simulation Event Set," *CACM*, Vol. 20, No. 8, August, 1977.
7. Kleinrock, L., *Queueing Systems, Volume 2, Computer Applications*, Wiley, Interscience, 1976.
8. Scholl, M., "Multiplexing Techniques for Data Transmission over Packet-switched Radio Systems," Ph.D. Thesis, Dept. of Computer Science, University of California, Los Angeles, 1976.
9. Tobagi, Fouad A., and L. Kleinrock, "Packet-switching in Radio Channels: Part III—Polling and (Dynamic) Split-channel Reservation Multiple Access," *IEEE TCOM* Vol. COM—24, No. 8, August, 1976.
10. Zeigler, Bernard P., *Theory of Modelling and Simulation*, John Wiley and Sons, 1976.

A stochastic state space model for prediction of product demand

by WILLIAM C. CAVE

Prediction Systems, Inc.
Manasquan, New Jersey

and

EVELYN ROSENKRANZ

Western Electric Company
Newark, New Jersey

INTRODUCTION

This paper is concerned with the development of a fixed price, supply/demand market model which can be used to predict demand for customer premises telephone equipment. A state space approach is used to model system dynamics and a Kalman filter is used for estimation. The model is nonlinear, and provides for nonstationary statistical characterization of the elements. The formulation indicates theoretically that, given perfect input (driving force) data, predictions could be highly inaccurate using linear models (even if they are dynamic) or nonlinear models which assume stationary statistics. The conceptual framework afforded by state space provides a vehicle for structuring more accurate models to predict product demand than do conventional approaches. Finally, the general model is suitable for predicting product demand in a wide range of markets.

DESCRIPTION OF THE PROBLEM

Recent legal developments in the telecommunications industry have necessitated the re-evaluation and consequent revision of the existing market philosophy. Emphasis has been on the need for accurate prediction of product demand, as opposed to the naively formulated forecasts of the past. This problem must be faced in total by the manufacturing branch of the industry which must anticipate the demand for the individual telephone companies and long lines division. This demand is a derived version of actual demand generated by the ultimate consumers of the various final products.

The extremely large number of products to be forecast together with a multiplicity of causal relationships on demand necessitate the development of general forecasting models to optimize the accuracy of prediction. Much of the work done in the area of estimating demand functions has been accomplished using statistically stationary processes with estimates provided only in the steady state. In some

instances such methodology may be sufficient in the sense of providing an adequate level of accuracy of prediction. However, if any one of the assumptions (i.e. steady-state *vs* dynamic, linear *vs* nonlinear, stationary *vs* nonstationary statistics) is violated, then these methods can be relatively inaccurate. A method for achieving significant improvement in accuracy encompasses the development of dynamic models in state space.

To obtain a grasp of the many facets of the prediction problem, it is necessary to have an understanding of the overall operating perspective. There are approximately 500,000 manufactured items (of which about 14,000 are actively tracked), divided into nine product lines. Each product line is broken down as indicated in Figure 1.

The MFLs (Master Forecast Lists) are group configurations depicting a structure of individual items, defined as key or non-key, selected for the purpose of capturing a certain percentage of sales. Key items are components that are required solely for a particular product to function. These items are frequently options or ancillary items essential for capturing the target total dollar sales for the MFL. The MFL is the level of aggregation for which accuracy of prediction is not only desired but necessary for future planning.

Current policy dictates that forecasts be made both on a short and long term basis. Short term forecasts are made three times a year projecting six quarters ahead. Long term forecasts involve projections for the next five years, revising figures as additional information becomes available.

The product line under investigation is station equipment, the family is residential systems, the groups are various categories of telephones, i.e., rotary, touch-tone, and the MFLs are a narrow category such as wall rotary. The intention is to develop models that will encompass the items contained within an MFL to forecast the latter.

As an additional consideration, predictions are needed at various levels of aggregation, such as by region (seven regions) or by telephone company (twenty companies), as well

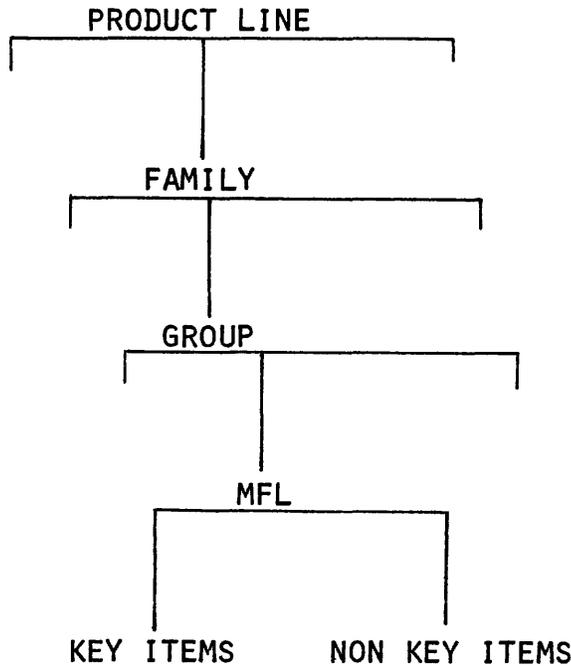


Figure 1—Product line hierarchy.

as a national total. Thus, the model must provide for isolation of regional demand functions and the special driving forces which affect them.

In summary, the basic problem is to predict the volume of demand for a "canonical" set of MFL items T_p periods into the future. The volume of demand for other MFL items can then be related to demand for the canonical items on a linear stationary basis.

DEFINING VOLUME OF DEMAND

In this paper, volume of demand is defined by the area under a "demand function" curve, wherein the "demand function" represents the number of items which buyers (ready, willing, and able) will pay for at the maximum price. This general demand function differs from demand curves commonly used in economics as explained in Appendix A. Referring to Figure 2, $q(p, t)$ is the general demand function, and allows improved conceptual representation of demand under a free or multiple price structure. If the product is supplied at fixed price P_s , then volume of demand is given by the area under the demand curve from P_s to the cutoff price P_c beyond which there is no demand.

$$A(t) = \int_{P_s}^{P_c} q(p, t) dp \quad (1)$$

Supply at a given price is represented by a Dirac delta function,

$$Q_s(t) = \delta(p - p_s)$$

wherein $Q_s(t)$, the area or "height" of the delta function, represents the quantity on hand at time t . Although multiple

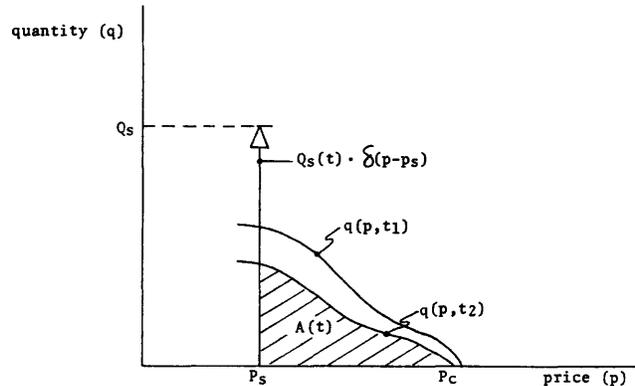


Figure 2—Demand function characterization.

pricing, e.g. discounting, is not treated here, it can be modeled using an extension of the basic approach.

We note that it is not economically feasible to directly measure $q(p, t)$ or $Q_s(t)$. Thus, we look to estimate these quantities based on directly observable quantities, e.g. *orders* and *shipments*. The precision applied to approximating the demand function $q(p, t)$ in the interval of interest $[P_s, P_c]$ can in general affect the accuracy of prediction. However, for predicting volume demand at a fixed price, the quantity of interest is the area, $A(t)$, and any representation which accurately characterizes this area is suitable. In the case that price changes are being considered, a more accurate characterization of the demand function can be accomplished using the previous formulation.

PREDICTION OF DEMAND

Before proceeding, we note the following. The competitive business enterprise is concerned with production scheduling to satisfy demand in a way which maximizes profits, subject to various social and economic constraints. This problem can be treated as an optimal control problem. If one is concerned with maintaining market share, it is not sufficient to "track" demand. Rather, one must also be able to predict changes in demand in order to maximize utilization of resources. This problem is different from that normally found in engineering applications wherein the driving forces are either random or controlled. In the competitive environment, one must determine an optimal trajectory when the driving forces, the model of the system, and the terminal state rapidly become more uncertain as we move into the future.* Control systems for solving this type of problem will only be as good as the imbedded prediction system for estimating trajectories into the future.

The development of an accurate prediction system requires three ingredients. These are an accurate model of the system, an accurate quantification of the forces which drive the system, and an accurate prediction algorithm. These elements are described in the following sections as they relate to predicting product demand.

* Refer to Reference 1 for definitions of "optimal trajectory" and "terminal state."

DRIVING FORCE CHARACTERIZATION

Change in demand over the interval $[T, T+1]$ due to various driving forces observed at T can be expressed as

$$D(T+1) = d_1[u_1(T)] + d_2[u_2(T)] + \dots + d_i[u_i(T)] \quad (2)$$

where the d_i s are functions of the driving forces $u_i(T)$. These functions can represent nonlinear, time-variant transformations. For example, consider a continuous or smoothed state variable $x_1(t)$ which represents change in demand at time t due to u_1 , sampled at time t_0 in the past. If the effect of u_1 on $x_1(t)$ approximates a simple damped response, then it might be characterized by

$$x_1(t) = u_1(t_0) \cdot e^{-(t-t_0)/\tau} \cdot U(t-t_0)$$

where $U(t-t_0)$ is the unit step function,² and τ is the time constant. Converting to discrete form using a Taylor series approximation,

$$x_1(T+1) = K(\tau)x_1(T) + u_1(T+1) = d_1(x_1, T) \quad (3)$$

where

$$K(\tau) = 1 - \frac{1}{\tau} + \frac{1}{2!\tau^2} - \frac{1}{3!\tau^3} \dots$$

is truncated to yield the desired accuracy (refer to Figure 3). If, in addition, the relationship between x_1 and u_1 is nonlinear, this can also be taken into account in d_1 . For example, if the demand, A , saturated as d_1 increased, this could be represented by a describing function of A , (refer to Figure 4). Examples of driving forces which must be characterized as described above are number of new building permits (linear) and advertising budgets (nonlinear).

MODEL OF SYSTEM DYNAMICS

In this section we offer, with rationale, a simplified example of a deterministic model of the dynamics of the system which generates demand for new telephone installations. The effects of uncertainty are added in the next section to produce a stochastic model.

Our objective is to accurately predict the area $A(T+1)$ under the demand curve, Figure 2, T_p time steps into the future. The demand for new installations, A_n , at $T+1$ can be expressed as

$$A_n(T+1) = A_n(T) + D_n(T+1) - Q_n(T+1) \quad (4)$$

where $D_n(T+1)$ is a function of the form $D(T+1)$ given by Equation 1. The individual d_i s can be nonlinear functions of $A_n(T+1)$, as shown in Figure 4. Such effects, commonly due to saturation, can occur with advertising or similar forces driving market demand. $Q_n(T+1)$ is the number of new installations during $[T, T+1]$ and is given by

$$Q_n(T+1) = \alpha(Q) \cdot [O_n(T) + \gamma_n(A_n(T+1) - A_n(T))] \quad (5)$$

where $O_n(T)$ is total outstanding orders for equipment installation, and $\alpha(Q)$ is a nonlinear coefficient depending on $Q_n(T+1)$ characterizing inventory levels. (In a more complete control model, α can depend on the value of inventory.) γ_n is a coefficient representing the efficiency of converting demand to orders. Total outstanding orders at time $T+1$ is given by

$$O_n(T+1) = O_n(T) + \gamma_n[A_n(T+1) - A_n(T)] - Q_n(T+1) \quad (6)$$

Because outstanding orders at the end of a period may be small in relation to orders filled during the period, it is more accurate to use orders filled as an observable. Total orders filled during $[T, T+1]$ is given by

$$O_f(T+1) = O_n(T+1) - O_n(T) \quad (7)$$

To convert this problem to state space notation, we define the state vector x as

$$x(T) = \begin{bmatrix} x_1(T) \\ x_2(T) \\ x_3(T) \end{bmatrix} = \begin{bmatrix} A_n(T) \\ Q_n(T) \\ O_n(T) \end{bmatrix}$$

and the observation vector as

$$z(T) = \begin{bmatrix} z_1(T) \\ z_2(T) \end{bmatrix} = \begin{bmatrix} O_f(T) \\ O_n(T) \end{bmatrix}$$

The previous simplified model typifies more complete

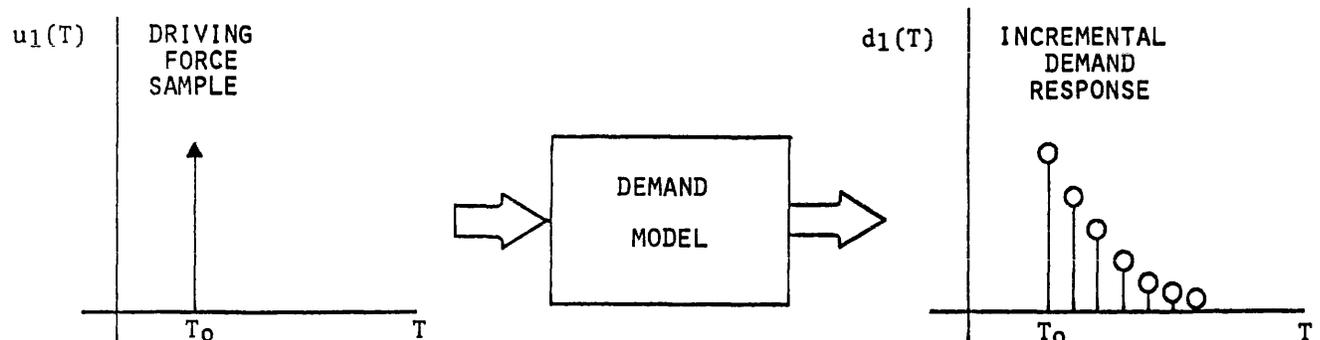


Figure 3—Example of damped response to driving force impulse.

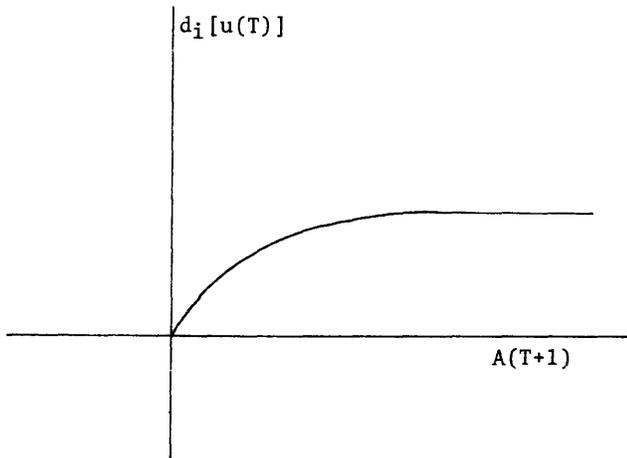


Figure 4—Nonlinear effects, e.g. caused by saturation.

models of the generalized form

$$\underline{x}(T+1) = \underline{f}(\underline{x}(T+1), \underline{x}(T), \underline{u}(T), T) \quad (8)$$

$$\underline{z}(T) = \underline{h}(\underline{x}(T), T) \quad (9)$$

where \underline{x} , \underline{z} , \underline{u} , \underline{f} and \underline{h} are vector quantities.

Through linearization over the interval $[T, T+1]$, these equations can be rewritten as

$$\underline{x}(T+1) = \Phi(T) \cdot \underline{x}(T) + B(T) \cdot \underline{u}(T) \quad (10)$$

$$\underline{z}(T) = H(T) \cdot \underline{x}(T) \quad (11)$$

The transformations of \underline{f} to matrices Φ and B , and \underline{h} to matrix H can be accomplished using describing functions or other linearization methods. Refer to Reference 3.

STOCHASTIC MODEL

We recall that certain unobservable state variables, e.g. demand, were selected to provide a conceptual representation of what are believed to be the dynamics of the system. We therefore look to estimate the state $\hat{\underline{x}}$ from observations of \underline{z} , and to predict $\hat{\underline{z}}$ based on estimated system dynamics.

Due to uncertainty of the model and observations, the stochastic representation of the system is

$$\hat{\underline{x}}(T+1) = \Phi(T)\hat{\underline{x}}(T) + B(T)\underline{u}(T) + \underline{w}(T) \quad (12)$$

$$\hat{\underline{z}}(T) = H(T)\hat{\underline{x}}(T) + \underline{v}(T) \quad (13)$$

where $\underline{w}(t)$ represents uncertainty in the model, and $\underline{v}(t)$ uncertainty in the observations. If we compute the estimate $\hat{\underline{x}}(T+T_p)$ from Equation 12, prior to observing \underline{z} , we can then estimate future values of $\hat{\underline{z}}(T+T_p)$ based on Equation 13.

If a model could be constructed which was precise for all T , i.e., $\underline{w}(T)$ and $\underline{v}(T)=0$ for all T , then estimates would be identical to actual values. Because precise models and measurements are not possible, predictions must be made in terms of probability density functions of the estimated values. Thus, given a model of the system, and a characteri-

zation of the observable driving forces, it remains to produce estimates of the probability density functions which describe the predicted trajectories $\hat{\underline{z}}(T)$ and $\hat{\underline{x}}(T)$ over $[T, T+T_p]$.

ESTIMATION PROCEDURE

If the probability density functions are characterized by means and variances which are derived from the error residuals

$$\delta = \hat{\underline{z}} - \underline{z}$$

then accuracy of prediction can be considered to be inversely proportional to the variance of the propagated density functions. Using this measure of accuracy, we seek minimum variance estimates conditioned on all available information. Kalman⁴ described such an estimator, and many authors, e.g. References 5 and 6, have subsequently expanded the base of knowledge on similar estimation procedures, all convenient to state space modeling.

The Kalman Filter, as it is widely known, provides minimum variance Bayesian estimates of both the state of the dynamic system and the observation vector as described in (12) and (13). The basic algorithms can be modified to estimate nonlinear systems, such as described by (8) and (9), as well as systems whose statistics are non-stationary. To summarize the estimation procedure, one must identify the statistics of the uncertainty elements \underline{w} and \underline{v} . It is assumed that these can be characterized as normally distributed white processes with zero mean, and covariance matrices given by

$$Q = E[\underline{w} \cdot \underline{w}^T]$$

$$R = E[\underline{v} \cdot \underline{v}^T]$$

(Refer to Reference 6 for a discussion of items to be considered when trying to characterize uncertainty.) Our problem is to estimate future states of the system given a starting state estimate and a statistical characterization of the uncertainty. This can be restated as follows. Given an estimate of \underline{x} and a measurement of \underline{z} at time T , we seek to propagate moments which describe the probabilities of the values of \underline{x} and \underline{z} at time $T+T_p$ in the future. This is accomplished by computing the estimates $\hat{\underline{x}}$ and $\hat{\underline{z}}$, and their corresponding error covariance matrices via the recursive Kalman filter algorithm.⁴

NUMERICAL METHODS

Given the model, driving force characterization, and estimation algorithms, we must now come up with maximum accuracy predictions of future demand and corresponding orders. To do this we must determine numerical values for the unknown coefficients in the model, including those in the driving force characterization, which maximize prediction accuracy. From a practical standpoint, this model identification process is the most difficult part of the problem. In general terms, one seeks values of coefficients in the

system equations which will maximize some predetermined measure of accuracy. This model identification process must be accomplished using numerical optimization. If the system equations are nonlinear, then the optimization algorithms must be capable of seeking the global maximum. If the statistics are nonstationary, adaptive algorithms must be devised for updating the covariance matrices, R and Q , based on most recent history. Finally, methods are needed to test for the presence of non-white noise components in the error residuals which can be further characterized as driving forces or model elements.

Two systems have been used which remove most of the burden associated with accomplishing the above. These are the General Stochastic Analysis (GSA) and General Stochastic Modeling (GSM) systems. GSM provides the user with a high level stochastic modeling language which affords a direct description of the problem as described above. It also provides for tabularized input of describing functions, and nonlinear optimization algorithms for model identification. GSA provides for interactive input of vector time-series data, statistical testing, and plotting and report generation.

To describe a state space model in the GSM language, the user writes FORTRAN like expressions for each state equation, e.g. Equations 4 through 7 and observation equation. GSM scans these equations, which can contain $X(T+1)$ terms and describing functions on the right-hand side, and generates optimal sparse matrix solutions. These are then linked to a table look-up method for solving the nonlinear equations at each time-step. The user can also write any valid FORTRAN expression for inequality constraints as well as a function to be maximized (or minimized). For model identification, the user need only specify range limits on the unknown parameters. Starting solutions are unnecessary. Using the GSM language, the person doing the modeling is left to concentrate on the development of intelligent model structures and characterization of driving forces to improve prediction accuracy.

SUMMARY OF RESULTS

Before comparing results of different modeling techniques, it should be noted that, given any modeling technique it is possible to take alternative approaches to constructing the model. Thus, it is possible for two people to obtain different results using the same technique. This is particularly true when the modeling tools being used provide a wide range of capabilities as well as a high degree of flexibility. The authors view the modeling process as a continual refinement process, and therefore consider their results for each technique subject to further improvement.

A brief summary of results using other conventional techniques versus a simplified version of the model presented is as follows. Deviation error is defined as

$$\text{Deviation Error} = \frac{|\text{Actual} - \text{Predicted}|}{\text{Actual}} (\%).$$

For the most accurate conventional approach used to date,

the average deviation error over a 22 time-step trajectory was 18 percent compared to 13 percent for the model presented. Maximum deviation over the trajectory was 60 percent for the conventional approach versus 30 percent for the model presented. As indicated, the authors believe that improvements can be made in all techniques investigated and are presently designing improved experiments to further validate model comparisons.

CONCLUSIONS

A general approach for developing more accurate predictions of telephone product demand has been described. This approach is based on a state space framework which provides for maximum use of human judgment in structuring models to represent market dynamics. The models which have been structured are generally nonlinear, and methods have been devised for identification of statistically nonstationary parameters. When structuring intelligent models of the type presented, it is apparent that numerical solutions are heavily dependent upon the use of highly sophisticated software, and the ease with which it can be used.

APPENDIX A

Consider the demand curve in Figure 5a with Q as total quantity of demand at price p . Except for the interchange of ordinate and abscissa this is the curve normally referenced in economics, with total quantity of demand decreasing as price increases.

Figure 5b represents a general demand function as used in this paper, and described in Figure 2, where q represents

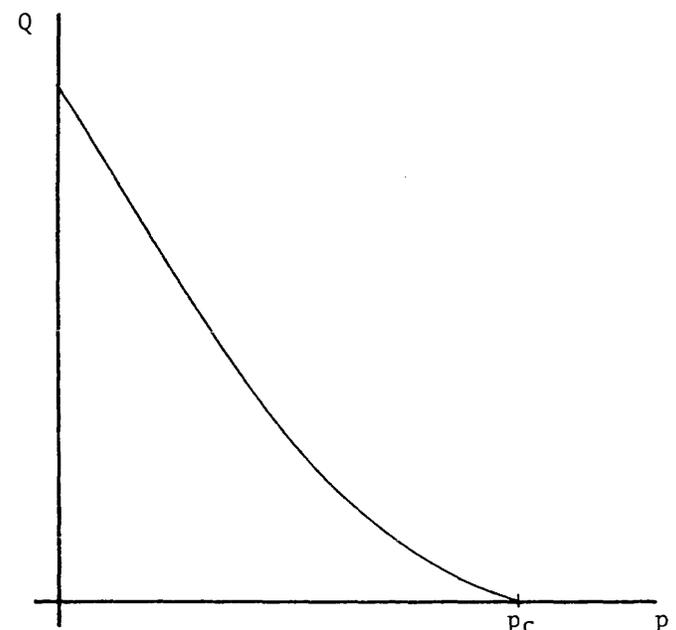


Figure 5a—Normal economic demand curve.

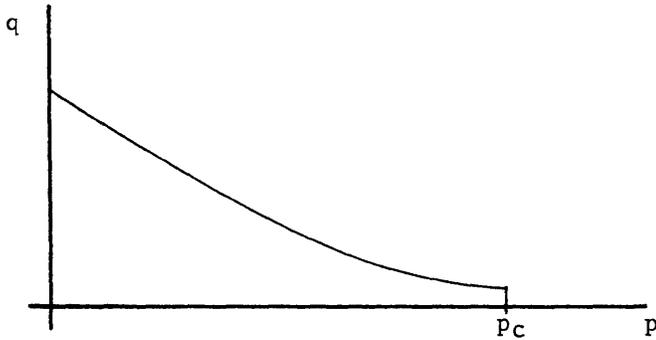


Figure 5b—General demand function.

the demanded quantity at a *maximum* price which buyers will pay. Total quantity of demand using this function is defined by the integral, Equation 1, with P_s replaced by the general price variable p . Since this integral evaluated at the

upper limit is always zero, the general demand function can be related to the demand curve (Figure 5a) as

$$q(p) = -\frac{d}{dp} [Q(p)].$$

REFERENCES

1. Athans, M., and P. L. Falb, *Optimal Control*, McGraw-Hill, New York, N.Y., 1966.
2. Papoulis, A., *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill Book Company, New York, N.Y., 1965, pp. 97.
3. Gelb, A., et al., *Applied Optimal Estimation*, The MIT Press, Cambridge, Mass., 1974.
4. Kalman, R. E., "A New Approach to Linear Filtering and Prediction Problems," *Transactions ASME, Ser D: J. Basic Eng.*, Vol. 82, 1960, pp. 35-45.
5. Jazwinski, A., *Stochastic Processes and Filtering Theory*, Academic Press, New York, N.Y., 1970.
6. Schweppe, F. C., *Uncertain Dynamic Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1973.

The Bus Link—A microprogrammed development tool for the CMOS/SOS processor system*

by AVNER BEN-DOR

Telesensory Systems, Inc.
Palo Alto, California

and

PAUL BAKER and JON SELDEN

Hewlett-Packard Company
Cupertino, California

INTRODUCTION

The widespread use of microprocessor-based systems has made the problems of development time and development cost most urgent. With the increasing complexity of recent systems, there has come a great need for powerful and adaptive development tools.

The Bus Link is a development tool for the MC² processor system, which was developed along with the microprocessor chip. The Bus-Link can be used throughout the development cycle of any MC²-controlled system to solve either hardware or software problems. This tool is not restricted to any particular system configuration and can operate with the maximum allowable processor speed (see Figure 1).

SYSTEM DESCRIPTION

The Bus Link hardware can be partitioned into two parts—the controller and the Unit Under Test (UUT) interface (see Figure 2).

The controller consists of a microprocessor and 8K × 16 bit words of memory. In addition, the controller contains a serial data interface port (UART) and an IEEE standard 488 interface port. Since the only front panel controls are the POWER ON switch and a RESET button, the user interacts exclusively from the CRT terminal connected to the serial data port. The terminal also contains a dual cartridge tape unit which can be used to load programs to or from the UUT memory. Since most of the hardware (including the dual comparators) is software controlled, the user can add and modify the entire system by loading new routines from the cartridge tape unit.

The UUT interface consists of the following parts:

1. The dual comparator unit with an ALU.

2. The trace buffer unit.
3. The UUT bus interface.
4. The bus synchronizer unit.

Each one of the above units is a separate entity, and the connection between them is done under the user's supervision.

MODES OF OPERATION

The Bus Link can operate in two modes:

1. Management Mode.
2. Monitor Mode.

Management Mode

In the Management Mode of operation the user enters commands from the CRT keyboard. The commands can be entered at any time even while the UUT is executing programs. The following capabilities are provided:

- a. Load and Store Programs—User programs are loaded from UUT memory to the cartridge tape unit or vice versa.
- b. Display and Modify UUT memory, registers and I/O.
- c. Display Trace—The specified number of entries in the Trace Buffer is displayed.
- d. Run, Halt and Single Step user programs.
- e. Interrupt UUT—A forced hardware interrupt.
- f. Reset the UUT system.
- g. Force Handshake—The Bus Link records initiation and completion of handshaking activities (either Memory or I/O) on the UUT Bus. When this command is executed, the Bus Link senses the incompleting handshake activity and simulates its completion so that UUT may resume execution.

* This work was compiled while employed by Hewlett-Packard Co. Data System Division, Cupertino, California.

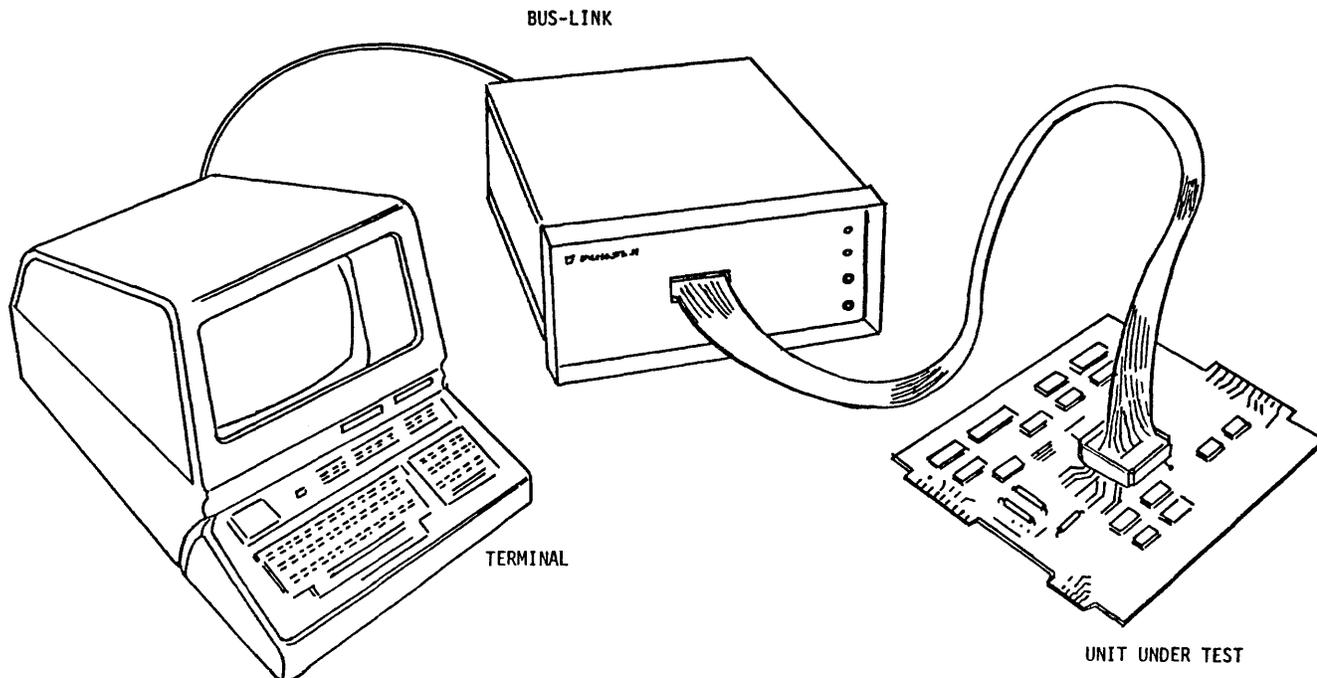


Figure 1—Bus link development system.

- h. Add Command—The user can enhance the capabilities of the Bus Link with additional commands which can be loaded into the Bus Link's memory from the system's cartridge tape unit.

Monitor Mode

Once the user issues the RUN command, control is passed to the UUT and the Bus Link enters the Monitor Mode. In this mode the Bus Link is monitoring a continuous process on the UUT. All 43 UUT bus lines (16 address, 16 data, 11 control) are continuously sampled.

A Bus Event occurs whenever a preprogrammed set of specifications describing conditions on the MC² bus has occurred. The occurrence of an event is a result of two individual comparisons being processed by an ALU unit and a delay counter (see Figure 3). Each comparator samples an actual condition on the UUT bus and compares it with an expected condition. Each one of the two comparators consists of three independent field comparators for the data field, address field and control field with a choice of: >, <, ≥, ≤, =, ≠, and bit mask on each field along with AND operation of all three fields (refer to Figure 3). Altogether we should have six comparator circuits (2 × data, address, control) and their associated masking registers. However, no physical comparators can be found in the Bus Link since the entire task is performed by software, a technique that will be discussed later. Each of the two comparators is connected to the ALU unit and a delay counter to provide the desired event pulse. The event pulse is used to start or stop the trace buffer or to halt the execution of a program. While in Monitor Mode, the static condition on the UUT

may be loaded into a local memory (64 words × 43 bits) called the Trace Buffer. The Trace Buffer may be started with an immediate command or following an occurrence of an event. Similarly, the Trace Buffer may be stopped by a command or following an event. The Trace Buffer may be loaded continuously, may be examined at any time, and may be stopped once the Buffer has been filled up.

The UUT may be programmed to halt under a certain set of conditions. When the condition occurs, the UUT is halted and control is passed to the user at the terminal. The UUT internal register values are updated on the screen and the instruction register is automatically disassembled. A UUT HALT condition can be set to follow an event condition or when the Trace Buffer is Full (see Figure 4).

THEORY OF OPERATION

The main building block of any microprocessor development system is the comparator circuits. There are two popular ways of implementing this block; the first one is by using SSI gates (see Figure 5a) and second one is by using MSI circuits (see Figure 5b). When using only SSI circuits, the expected response is stored in the data register and the DON'T CARE (X) bits are stored in the mask register. The sampled data is stored into the input register and compared with the expected data stored in the data register. The result is then ANDed with the bit pattern stored in the mask register, before it is being ORed to provide a compare signal (see Figure 5a). Another common way of implementing the same block is by using an MSI comparator which can provide not only the compare signal (−), but also greater than (>) or less than (<) signals.

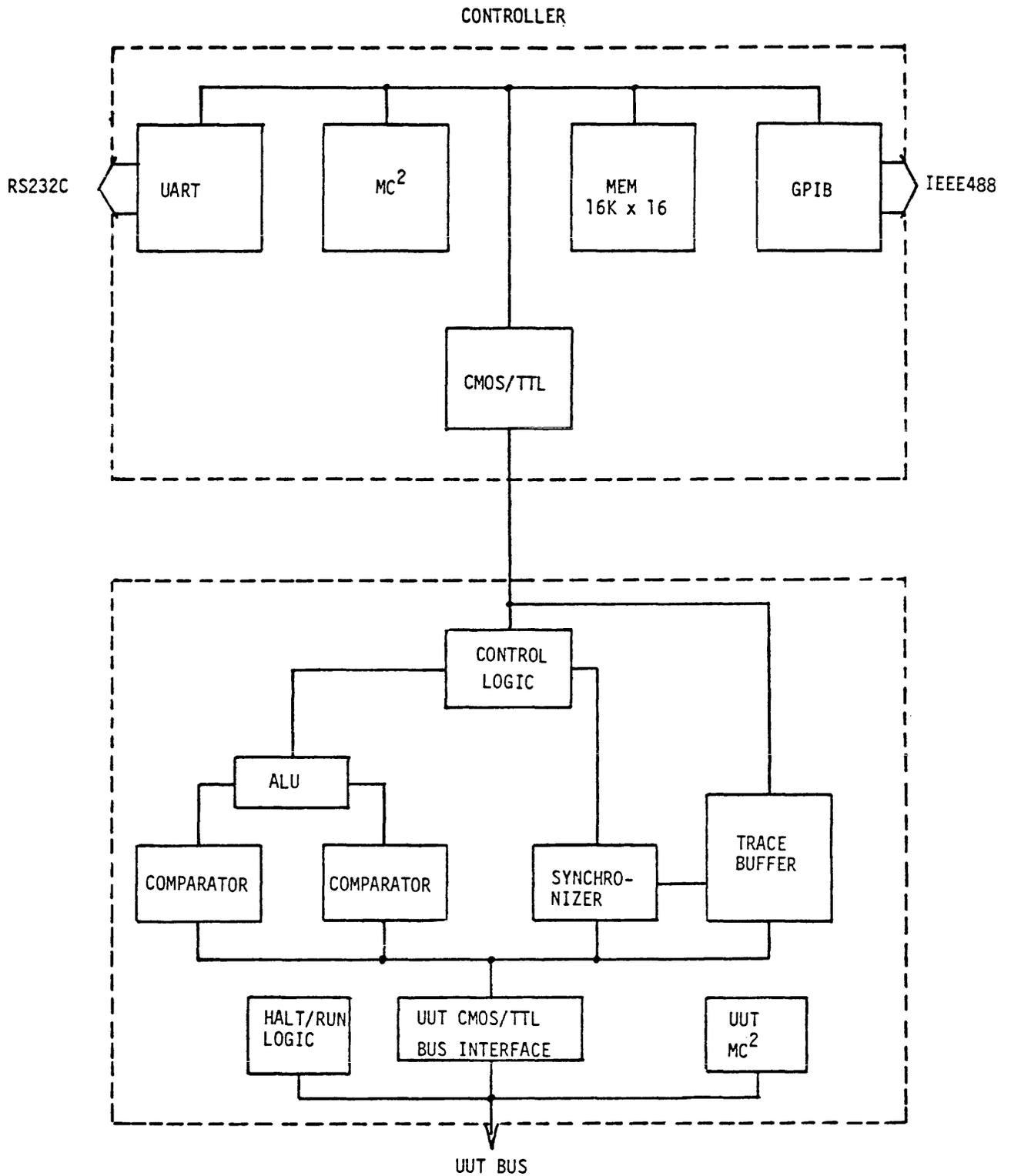


Figure 2—Bus link functional block diagram.

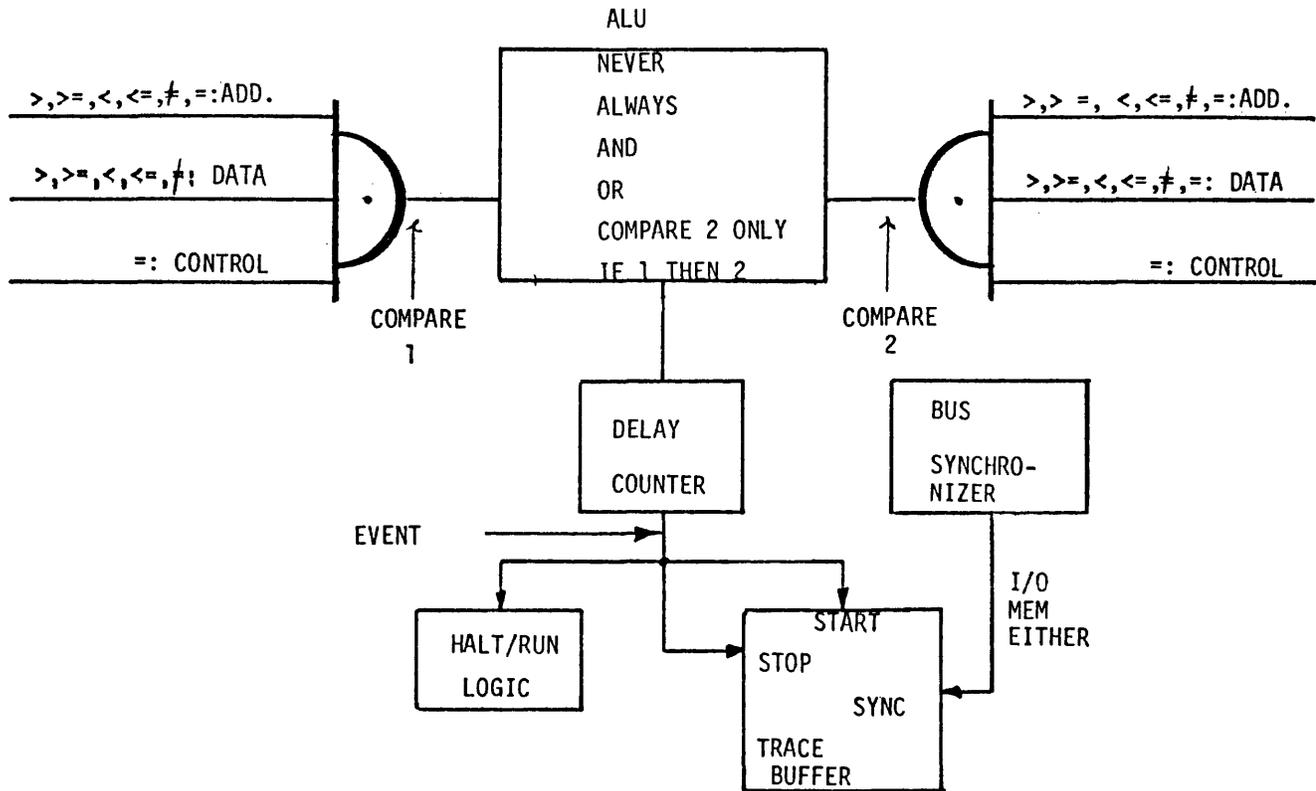


Figure 3—Bus link monitor.

ADDR =	0000	0000.0000.0000.0000	OP	NEVER	ADDR =	FFFF	1111.1111.1111.1111
DATA =	0000	0000.0000.0000.0000	COUNT	001	DATA =	FFFF	1111.1111.1111.1111
CNTL =		XXX.XXXX.XXXX			CNTL =		XXX.XXXX.XXXX

P	S	IOs	KB	IR	GOTO	!00D4	IF<>	HALT	TRACE	MEMORY
00E0	0000(FFFF)	0000	85D4	0010	!00D4	IF<>	0000	00FULL	ENABLE	ALWAYS
R0	R1	R2	R3	R4	R5	R6	R7	DISABLE	NEVER	
FFFF	1365	AAAA	0004	AAAA	0000	8000	9000			

[?]RU CV
OUT HALTED

[?]DT

ADDR	(HEX	DATA	INSTR	ASCII)	I	G	M	M	I	I	F	I	I	P	W
					D	N	E	G	O	O	E	A	N	O	R
					L	1	N	0	E	G	T	K	I	N	I
00C1	8700				0	1	0	0	1	1	0	1	1	1	0
00C2	B803	LLIT	R3=		0	1	0	0	1	1	1	1	1	1	0
00C3	FFFD				0	1	0	0	1	1	0	1	1	1	0
00C4	CB04	R4=(R3)			0	1	0	0	1	1	1	1	1	1	0
FFFD	8700				0	1	0	0	1	1	0	1	1	1	0
00C5	2224	CMP	R4, R2		0	1	0	0	1	1	1	1	1	1	0
00C6	82CB	GOTO	!00CB	IF=	0	1	0	0	1	1	1	1	1	0	
00CB	B806	LLIT	R6=		0	1	0	0	1	1	1	1	1	0	
00CC	8000				0	1	0	0	1	1	0	1	1	0	
00CD	B807	LLIT	R7=		0	1	0	0	1	1	1	1	1	0	
00CE	9000				0	1	0	0	1	1	0	1	1	0	

Figure 4—Bus link screen format.

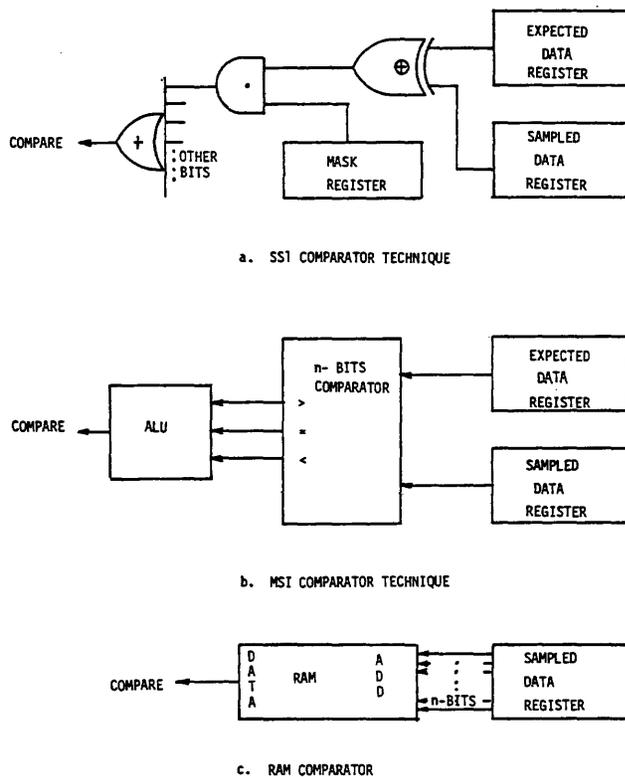


Figure 5—Digital comparison techniques.

The conventional techniques just described have three major disadvantages:

- Expensive—Both circuits require a large amount of integrated circuits and also require a large P.C. Board since many traces must be routed between the components.
- Incomplete—Comparing the sampled data with an expected response to determine a greater than ($>$) or a less than ($<$) equality while some of the bits are masked can't be implemented with the above circuits.
- Inflexible—Changes of existing design are hard to implement.

The comparator circuit can also be implemented with Random Access Memories (RAMs), and such implementation contains none of the disadvantages described earlier (see Figure 5c). The Bus Link is only one of many development tools in which the comparators are designed by utilizing RAMs. This concept will be explained with examples in the next paragraphs.

- Comparing single breakpoint with single RAM*—Let's assume that we want to compare an n -bit word with another n -bit word (expected response vs. actual response). If an n -word \times 1 bit RAM is available, and it is possible to store the data (1-bit word) in each location (n -bits address), the comparison process can be prepared by software and can be exercised by the RAM.

The software routine stores a "1" in the K th word where the address of K is equal to the bit pattern we want to compare with (see Figure 6a).

K th word address = expected response of n -bit word. The software also stores a "0" anywhere else in the memory. Now the RAM is ready to compare any actual data sampled on the UUT bus. The sampled data word is connected to the address field of the RAM. The RAM is read continuously by the processor which can now determine the result of the comparison. If the data word read (one bit) is found to be a "0," it implies that the expected n -bit pattern is not equal to the actual one. If the data word is found to be a "1," a match between the expected data and the actual one exists.

- Comparing Multiple breakpoint with single RAM*—Multiple breakpoints comparison can be performed with the same hardware, only the software routine must be modified. Let's keep the same definition of the K th word. Again— K th word address = expected response of n -bits word; we can deduct the following:
 - To compare \neq not equal, store in the K th word a "0" and store a "1" anywhere else in the memory space. (See Figure 6b.)
 - To compare $<$, less than (or \leq) store a "1" in the memory space from location 0 to (inclusive) the K th address and a "1" from the K th address and on. (See Figure 6c.)
 - To compare $>$, greater than (or \geq), use the 2b algorithm with a complemented data word. (See Figure 6d.)
- Handling masked (Don't Care-X) bits with single RAM*—Masked bits (X) can be easily processed by the same algorithm, with minor modification. A masked bit is essentially a multiple compare situation; each masked bit doubles the number of words to be compared with since $X = 1$ and also $X = 0$ (see Figure 6e). No mask registers or AND gates are needed.
- Comparing multiple breakpoints with more than a single RAM*—The Bus Link was designed as a development tool for the MC² microprocessor. As mentioned earlier the MC² has 16 bits of data word, 16 bits of address word and 11 bits of control word. If a RAM is to be used as a comparator, it must contain at least $2^{16} \times 1$ bit words (64K words). With an additional data bit, every single RAM can be divided down into smaller RAM units which may be cascaded to any arbitrary length. The additional bit is a dependency bit (carry bit) which enables the comparison of lower-order bits. In the example described in Figure 7, the 16-bit word comparator is implemented with two 256×2 bit words RAM. A comparison is always started with the high-order bits (byte) and carried to the lower-order byte only if it is needed. The software processes the expected data word format and determines whether a carry bit should be entered. This example can be carried further if either a larger than 16-bit word is to be compared or if the RAMs are to be partitioned to smaller units (for economic reasons).

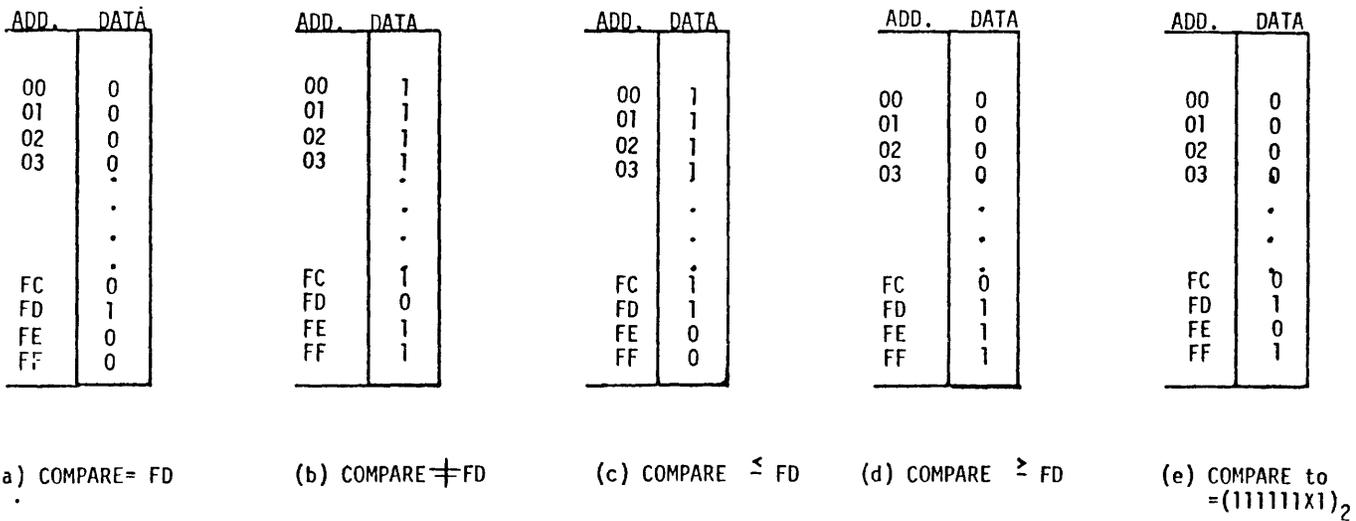


Figure 6—Examples of comparing with a single RAM. Addresses are expressed in hexadecimal notation.

e. *Multiple comparison of multiple breakpoints with more than a single RAM*—Most development tools available today offer only a single comparator, which is adequate for most applications. But a single comparator isn't effective if complicated software routines or complicated I/O ports are to be developed. For example, an

address space can be bounded to be greater than a minimum value AND less than another maximum value—upper bound \geq address \geq lower bound. If a hardware comparator (with SSI or MSI devices) is used, a second comparator almost doubles the amount of hardware. In using RAMs, only the RAM size (and some logic) is doubled which yields much greater price/performance ratio. As shown in Figure 7, the additional comparator function is added with almost no additional cost since the standard 256 word RAMs are four bits wide anyway. More comparators can be added to the circuit without much difficulty. The software prepares the data for each comparator separately and then concatenates the corresponding data words to form a single block before writing the entire block into each RAM.

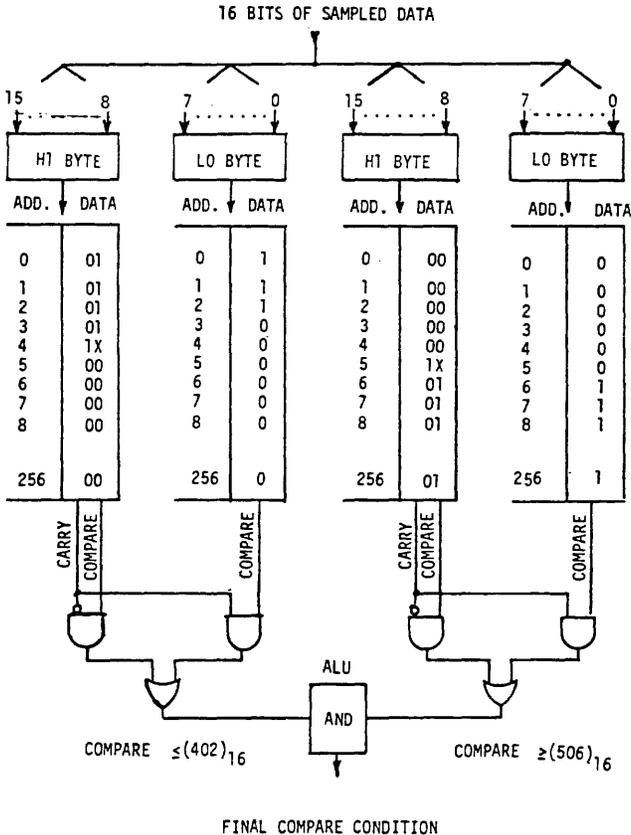
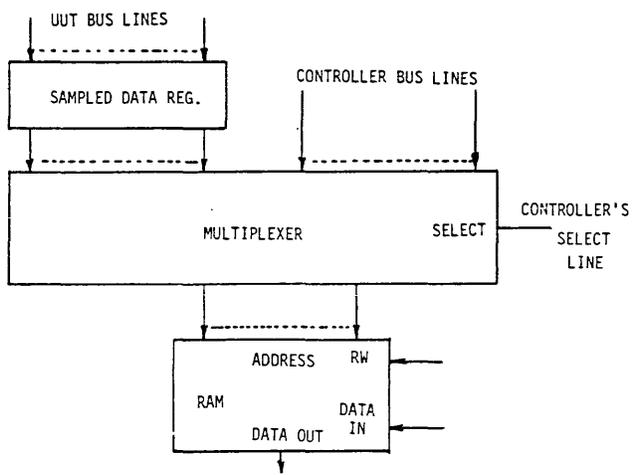


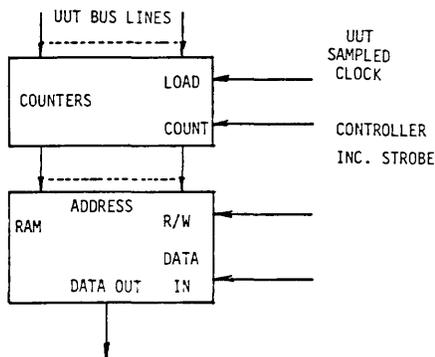
Figure 7—Dual comparison of multiple breakpoint with two 256 x 4 RAMs.

f. *Do without multiplexers*—The address for the RAMs is provided from two sources—the UUT and the controller. When the controller writes the data into the RAMs it must provide the address field, but when the actual comparison is performed, the address field must be connected to the UUT bus. Multiplexing 43 lines between the controller and the UUT bus to the RAMs address field requires many multiplexers and consumes much of the PC Board area due to the large number of traces. Since the software prepares the data in blocks, multiplexing is not required, synchronous counters can be used instead. As shown in Figure 8, when the controller writes the information into the RAMs, the counters are automatically incremented and the processor only provides the data word to be written. When the actual comparison is to be made, the counters are used as registers where the count pulse is disabled and the load pulse is connected to the UUT clock.

g. *Single multitask sequential control circuit—the Hardware Subroutine*—In the management mode of operation, the user can enter one of many optional functions. The common method of designing the control logic is to implement each sequential logic (in a minimized



(a) CONVENTIONAL MULTIPLEXING SCHEME



(b) COUNTERS- MULTIPLEXING SCHEME

Figure 8—Conventional vs. counter multiplexing scheme.

form) of every function separately, and to attach a selector unit to activate each one of the functions separately (as shown in Figure 9). Further study of all the control functions revealed three important facts:

1. The functions are similar to each other.
2. Only one function can be active at any single time.
3. The functions are not time-sensitive.

Taking the previous facts into consideration and deviating from the standard approach, a new design method has resulted:

1. Find the largest common denominator among the functions. Simple functions can be made to act like complicated ones by adding redundant states.
2. Use a selector device to demultiplex the selected set of inputs (qualifiers) into the general sequential circuit block.
3. Use a demultiplexer to select one set of outputs from the sequential circuit block.

Since this method resembles the activity of writing a general-

purpose subroutine, it is called The Hardware Subroutine. A minimum of 3:1 reduction in components count was achieved utilizing this approach.

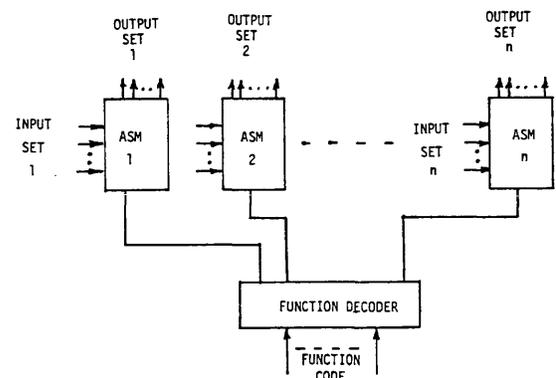
SOFTWARE DESCRIPTION

The operation of the Bus Link is controlled by the software, which resides in 8K words of RAM. The software can be partitioned into three main blocks:

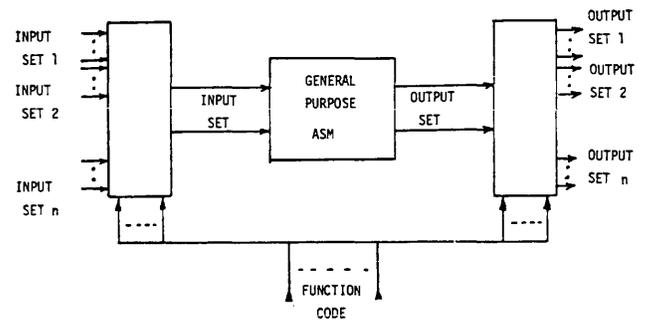
- a. Terminal handler.
- b. Controller.
- c. Comparator processor.

Terminal Handler

The terminal screen is partitioned into two areas (see Figure 4)—the dynamic and the static area. The static area is used to display the status of the processor and the status of the Bus Link. Commands are entered by the user in a form-fill-out fashion and interpreted only when the user exits this area. The dynamic area can be used for all control



(a) CONVENTIONAL IMPLEMENTATION OF MULTIFUNCTION SEQUENTIAL LOGIC



(b) THE HARDWARE SUBROUTINE

Figure 9—The hardware subroutine vs. conventional implementation. "ASM" stands for, Algorithmic State Machine.³

commands, for displaying the status of memory and registers, and for the content of the trace buffer. The terminal handler routine also loads and stores programs in the cartridge tape unit.

Controller

The controller part is the supervisor of the entire system. It is composed of interrupt-driven routines which answer demands from either the terminal or the various hardware blocks in their assigned priorities. The controller is also capable of self-diagnosing the system once a failure is suspected.

Comparator Processor

As described earlier, the preparation of the data to be stored in the comparator RAMs is done by the software. Since setting a breakpoint could be done while the UUT processor is running, a powerful algorithm was developed to minimize the required time to process the breakpoint data. The algorithm can be described as follows:

1. Retrieve the breakpoint word.
2. Form a mask word which is a copy of the breakpoint word after all the don't care bits (x) are marked as "0"s, and all the actual bits ("1" or "0") are marked as "1"s.
3. Form a compare word which is a copy of the breakpoint word when all the don't care bits are modified to "0"s and the actual bits are left unchanged.
4. Prepare a table in which the address of the table corresponds to an identical breakpoint word (256 entries for eight-bit word) and the content of each address carries the following data—0: =(equal), 1: >(greater than), 2: <(less than).
5. Each address of the table must be ANDed with the mask word and compared with the compare word. The data of each address reflects the result of the comparison 1, 0 or 2 (stands for =, >, <).
6. For the high byte RAM refer to Table I. Select the desired row for the condition specified (=, ≠, <, >, <=, >=) and change each entry of the table prepared in (5) to the corresponding bit format described in Table I (word-by-word table lookup).

TABLE I.—Lookup Table for the Comparator's RAM Data

Condition	Hi Byte			Low Byte		
	=, 0	<, 2	>, 1	=, 0	<, 2	>, 1
1. =	1	0	0	2	0	0
2. ≠	1	2	2	0	2	2
3. <	1	2	0	0	2	0
4. < =	1	2	0	2	2	0
5. >	1	0	2	0	0	2
6. > =	1	0	2	2	0	2

2 = compare; 1 = carry; 0 = doesn't compare

7. Store the new table in the high byte comparator RAM.
8. Repeat Steps 2 through 6 for the low byte RAM and use Table I for the same task.

9. Store the new table in the low byte comparator RAM.

The following is an example of using the previous algorithm for preparing the data for a four-bit compare word (see also Table I).

Step 1—Compare to $\geq 10X0$

Step 2—Mask word: 1101

Step 3—Compare word: 1000

Steps 4,5—

table word address	masked mask word address	masked word address	compare word	comparison result	table data
0000	. 1101	0000	1000	2	0
0001	. 1101	0001	1000	2	0
0010	. 1101	0000	1000	2	0
0011	. 1101	0001	1000	2	0
0100	. 1101	0100	1000	2	0
0101	. 1101	0100	1000	2	0
0110	. 1101	0100	1000	2	0
0111	. 1101	0101	1000	2	0
1000	. 1101	1000	1000	0	2
1001	. 1101	1001	1000	1	2
1010	. 1101	1000	1000	0	2
1011	. 1101	1001	1000	1	2
1100	. 1101	1100	1000	1	2
1101	. 1101	1101	1000	1	2
1110	. 1101	1100	1000	1	2
1111	. 1101	1101	1000	1	2

Step #6—Compare with low byte of Table I, condition 6. Table data is shown on last column of previous step.

CONCLUSIONS

Four unique design contributions in the Bus Link have been presented in detail—RAM comparator unit, counter multiplexing logic, multitask sequential control circuit and a powerful algorithm to compute breakpoints. The above features when used together enable the designers to construct a very powerful development tool which is useful throughout the development cycle of any microprocessor-based system. The Bus Link is adaptive and can be easily modified to accommodate different systems. Each one of the previous features can also be used separately for different design applications to improve the cost-performance ratio.

The Bus Link and the associated products are currently being used in many applications with great success.

ACKNOWLEDGMENTS

The authors would like to thank Janelle Bedke for her help in defining the project and Larry Lopp for his support throughout the development phase. Special thanks are due to Kim Leeper who wrote many of the software routines and Helen Azadkhanian for typing the paper.

REFERENCES

1. Forbes, B. E., "Silicon-on-Sapphire Technology Produces High-Speed Single-Chip Processor," *Hewlett-Packard Journal*, April 1977.
2. Bell, C. G., and A. Newell, "The PMS and ISP descriptive systems," Chapter 2 of "Computer Structures: Reading and Examples," McGraw-Hill Book Company, Inc., New York, 1971.
3. Clare, C. R., "Logic Design of Algorithmic State Machines," Hewlett-Packard Laboratories, November 1970.
4. Dang, L. G., Ashkin, P., O'Brien, M., and R. Yee, "A CMOS/SOS 16-Bit Parallel Micro-CPU," *IEEE International Solid-State Circuits Conference*, 1977.

A computer analysis tool for structural decomposition using entropy metrics

by AARON N. SILVER

Martin-Marietta Corporation
Denver, Colorado

INTRODUCTION AND BACKGROUND

The decomposition of a metric space into successive subregions exhibiting distinctive characteristics is a problem of broad application. In pattern classification, the object is to partition the space such that pattern classes are easily separable; that is, so that each subregion of the partition contains predominantly samples of only one class. In piece-wise-constant approximation the decompositions produced contain samples whose values are sufficiently close to allow approximation with a specified degree of accuracy. In defining software it is quite often necessary to derive a structural model of a computer program which contains modules, i.e., partitions exhibiting the flow relations or connectivities among the elements (statements) in a program. The subsequent analysis and manipulation of the structural model produces useful design alternatives that enhance the operational qualities of the software generated in terms of program control, logic paths, data transfer and other relevant software issues. The basic feasibility of this approach has been demonstrated by numerous investigators.¹⁻⁵ However, the analytical and diagnostic tools for performing structural decompositions require further refinement and development. For example, the metrics usually used^{6,7} for defining the topology of a given software structure are primarily single attribute measures. Although the entropy metric proposed in this paper is metrizable in terms of its hypergraph representation,⁸ the extension to a multi-attribute unique formulation is, as yet, elusive. This is because an all-purpose problem-independent metric space places unrealizable constraints on the structure it proposes to define. Thus, as Koontz et al.⁹ point out, even when a metric is given and a structure well known, the notion of neighboring points can not be rigorously defined for finite point sets from a computational point of view, since the simplest Euclidean distance measure must be scaled by a factor indicating its own respective distance to the nearest neighbor in order to avoid overlapping and ambiguous regions. Although conceptually, the construction of a neighborhood and the determination of the limit point of a sequence of real numbers is a widely used idea, a more fundamental requirement for metrizable hyper-spaces is that of specifying the existence of a limit point of a set. The resultant necessary and sufficient conditions for identifying

metrizable spaces is given by Hausdorff.¹⁰ However, equivalent normalizations and the use of discrete semi-metrics over a restricted space have precluded some of these inherent problems in the quest for such a unique, multi-attribute metric. Thus, the primary emphasis is to obtain realizable decompositions using readily-implementable metrics, as well as focus upon suitable partitioning alternatives in terms of identifying mathematically consistent criteria for structural decompositions.

Figure 1 indicates the application of several useful metrics to well defined problem areas. In this respect, it should be observed that the determination of the "correct" metric properties to be abstracted is largely an experimental process. This process involves two broad and interrelated questions. The first of these concerns the investigation and classification of the various concrete realizations, or models, which one may encounter. This entails the recognition of equivalent models, as is done for isomorphic groups, graphs, or congruent geometric figures, for example. In turn, this equivalence of models is usually defined in terms of a one-to-one reversible transformation of one model onto a metric space. This equivalence transformation is so chosen as to leave invariant the fundamental properties of the models. Typical examples are the rigid motions in geometry, the isomorphisms in group theory, etc. At the extreme end of the spectrum, Figure 1 includes the non-metric measures of a Calhoun distance as discussed by Bartels et al.¹¹ This distance measure utilizes only the ordering of points along each dimension of the space. The basic concept for measuring the distance between two points is to imagine them as opposite vertices of a hypervolume whose sides are parallel to the axes of the space. The distance is basically the fraction of all points which fall into this hypervolume and its extensions. The Calhoun distance is invariant to transformations which preserve the order of the points along the measurement axes. One kind of transformation to which the Calhoun distance is not invariant is an orthogonal rotation. Still another non-metric measure is the Lance and Williams¹² ratio, which was developed as a generalization of the Czekanowski or Dice coefficient measure.

The second broad question in studying structural decom-

APPLICATION AREA	SUITABLE METRIC
I. STRUCTURAL NETWORK PROBLEMS 1. Non-Directed Graphs 2. Directed Graphs 3. Structural Connectivity 4. Tree Structures 5. Geometric Graphs 6. Hamiltonian Cycles	a) THE MINKOWSKI METRIC: $d_p(x, y) = \left\{ \sum x_i - y_i ^p \right\}^{1/p}$ b) L_1 METRIC ("TAXICAB", $p = 1$): $d_1(x, y) = \sum x_i - y_i $ c) L_2 METRIC (EUCLIDEAN): $d_2(x, y) = \left\{ \sum (x_i - y_i)^2 \right\}^{1/2}$ d) CHEBYCHEV METRIC: $d_\infty(x, y) = \text{Max} \{ x_i - y_i \}$
II. FREE FORM STRUCTURES 1. Vertex Connectivities 2. Edge Progressions	GENERALIZED ALEXANDROFF METRIC: $Q(x, y) = \phi(x, y) + \sum \psi_n(x, y)$ where $\psi_n(x, y) = \left\{ \frac{ g(x) - g(y) }{1 + g(x) - g(y) } \right\}$
III. OTHER SPECIAL PROBLEMS 1. Steiner Problem 2. Parallel Configurations	RECTILINEAR METRIC: $d(x, y) = \{ x_i - x_j + y_i - y_j \}$ Marked graphs (M_0, \dots, M_{1c})
IV. NON-METRICS 1. Correlations 2. Scaling	$d(x, y) = \left\{ \frac{\sum x_i - y_i }{\sum (x_i + y_i)} \right\}$

Figure 1—Application of metric spaces to graphs.

position as exemplified by topologized sets involves consideration of transformations more general than one-to-one equivalence transformation. The condition that the transformation be one-to-one and reversible is dropped and one retains only the requirement that the basic structure is to be preserved. The homomorphisms in group theory illustrate this situation. In topology, the corresponding transformations are those that preserve limit points.

METRIC SPACES, ENTROPY FUNCTIONS AND GRAPHS

It is convenient to establish a mathematical basis for mapping a structure, i.e., graph representation onto a metric space (particularly an entropy metric space), so that subsequent decompositions may be rigorously analyzed. The following concept of a metric space and its associated topology enables the investigator to identify a potential metric-based decomposition strategy, as well as formulate effective

quality measures for the overall structure based upon a metrizable function; i.e., in this case the entropy function.

Metric spaces

A set X of elements x, y, z, \dots is called a metric space if each pair x, y in X is assigned a non-negative number $d(x, y)$ (called the distance from x to y), satisfying the following conditions:

1. (identity axiom) $d(x, y) = 0$, if $x = y$; otherwise $d(x, y) > 0$
2. (symmetry axiom) $d(x, y) = d(y, x)$
3. (triangle axiom) $d(x, y) + d(y, z) \geq d(x, z)$

It is clear that $d(x, y)$ is a function of s, y ; it is called the metric in X . A function having Properties 1 and 2 but not Property 3 is called a "semi-metric." If Property 3 is replaced by

- 3'. $\text{Max}\{d(s, y), d(y, z)\} \geq d(x, z)$

then a function with Properties 1, 2 and 3 and 3' is called an "ultra-metric," since 3' is considerably stronger than 3.

A topology may be introduced in the metric space X by taking the neighborhood basis of the point $x_0 \in X$ to be the set of all open spheres with center x_0 . It is easily verified that Axioms 1-3, for a neighborhood basis and the separation axiom (i.e., every pair of distinct points in X have disjoint neighborhoods) are satisfied so that X is indeed a Hausdorff topological space. In this case, the topology in X is defined by the metric $d(x, y)$. Thus, a topological space is said to be "metrizable" if a metric can be introduced in X which defines a topology in X, coinciding with the initial topology. Furthermore, a mapping of a metric space X into the metric space X' is said to be isometric if it preserves distances, i.e., if the distance $d(x, y)$ between any two points $x, y \in X$ equals the distance $d(x', y')$ between their images in X'. Clearly, an isometric mapping is a homeomorphism. Two metric spaces X, X' are said to be isometric if there exists an isometric mapping of X onto X'. Obviously, isometric spaces are homeomorphic. From the viewpoint of metric space theory, two isometric spaces are considered to be essentially the same.

Entropy functions

Consider the sequence $Z = \{Z_1, Z_2, \dots, Z_n\}$ of random variables defined on some metric space, i.e., a probability metric space with measure P. Let the entropy function $H(Z)$ be denoted by

$$H(Z_1, Z_2, \dots, Z_n) = \sum P(Z) \log_2 P(Z)$$

and let p_i equal the values assumed by the random variables. Thus,

$$H(p_1, p_2, \dots, p_n) = - \sum_{i=1}^n p_i \log_2 p_i$$

Let the set $z = \{1, \dots, n\}$ be partitioned into two disjoint subsets $\{i_1, i_2, \dots, i_k\} = \{\zeta_n\}$ and $\{j_1, j_2, \dots, j_{n-k}\} = \{\theta_n\}$

For simplicity

let $X = X\{\zeta_n\} = \{X_{i_1}, X_{i_2}, \dots, X_{i_k}\}$
 and $Y = Y\{\theta_n\} = \{Y_{j_1}, Y_{j_2}, \dots, Y_{j_{n-k}}\}$

also let x and y be the values assumed by the random variables X and Y.

Thus, X and Y are finite non-empty sets such that $\{S\} \subseteq X \times Y$, and

$$S_x: \{y | (x, y) \in S\} \text{ for } x \in X$$

and

$$S_y: \{x | (x, y) \in S\} \text{ for } y \in Y$$

Property 1—The conditional entropy is non-negative

$$H(x | y) \geq 0, \text{ or } H(y | x) \geq 0$$

Property 2—The joint entropy is given by

$$H(x, y) = H(x) + H(y)$$

Property 3—Combining 1 and 2 above

$$\begin{aligned} H(x, y) &= H(x) + H(y | x) = H(x) + H_x(y) \\ &= H(y) + H(x | y) = H(y) + H_y(x) \end{aligned}$$

where

$$H(x | y) = H_y(x) \text{ and } H(y | x) = H_x(y)$$

These three properties satisfy the conditions for a metric stated in the previous section.

Graph structures

A graph $G = (V, E)$ consists of finite set of vertices V, and a finite set of edges E (adjacency) which is symmetric and irreflexive.

$G = (V_1, E_1)$ is a subgraph of G if $V_1 \subseteq V$ and E_1 is the restriction of E to E_1 (some authors call G_1 a vertex-generated subgraph). The complement of G is a graph $G^c = (V, E^c)$ with the same vertex and adjacency defined: for $x, y \in V$ and $x \neq y$, then $x \in E^c \iff \sim x \in E \iff y$ (any pair of distinct vertices are adjacent in exactly one of G and G^c).

A path v_1 to v_n is a sequence of edges $(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$. If all vertices $v_i, 1 \leq i \leq n$ are distinct, the path is simple. A graph is connected if there is a path between any two distinct vertices. A bridge is an edge (v, e) that is in every path for v to e .

A tree is a connected graph, where any two distinct vertices are joined by a unique simple path. A spanning tree of G is a subgraph containing all vertices of G.

In a weighted graph, every edge e has a number $W(e)$ called its weight. If T is a set of edges such as a spanning tree, the weight of T is $\sum_{e \in T} W(e)$.

Using the preceding three sections dealing with metric spaces, entropy functions and graph structures, a hypergraph¹³ may be defined on an entropy metric as follows:

Denote the hypergraph by $G_H = (X, \Theta)$ such that

$$\Theta = \{E_y | y \in Y, Y \text{ is a finite set } S_y\}$$

and

$$\cup_{y \in Y} \{E_y\} = X, X \text{ is a finite set } S_x$$

Then for G_H , $f(H)$ is given by

$$f(H) = \sum_{i, j \in Y} \left\{ |E_i \cap E_j| \log_2 \frac{|X|}{|E_i \cap E_j|} \right\}$$

where $E_i \cap E_j \neq \emptyset$

or

$$H(X, Y) = - \sum_{i \in Y} \left\{ \frac{E_y}{X} \log_2 \frac{E_y}{X} \right\}$$

Here $|X|$ denotes the number of vertices in the metric space, while the connectivities (if they exist) are given by $|E_i \cap E_j|$. Thus the hypergraph G_H is defined on the metric H.

DERIVATION OF THE DECOMPOSITION CRITERION

Consider an arbitrary partition of the space {S} into subsets {S₁, S₂, . . . , S_n} such that {S_i} ∩ {S_j} = 0, and ∪_n {S_i} = {S}. Let the information contained in {S} be represented by the entropy function H{S} as previously defined. Thus, the information or entropy contained in {S_i} taken separately is Σ {H{S_i}}. Except in the case where there is no interaction at all between the subregions (i.e., statistical independence) the expression Σ H{S_i} will be larger than H{S} only because some information (entropies) will be counted more than once. This difference is commonly called the "redundancy" and is given by the difference between the two expressions: Σ {H{S_i}} - H{S}. It is a measure of the *strength* of the connectivities. Thus, one may write for two subregions x and y

$$H(x:y) = H_{\max}(x, y) - H(x, y)$$

As the space is partitioned into another subregion denoted by w, expressions for the redundancies may be easily obtained as follows:

$$\begin{aligned} H(y:w, x) &= H(y:x) + H_x(y:w) \\ \{H(y:w, x) &= H(y:w) + H_w(y:x)\} \end{aligned}$$

By adding ±H(y:x) and regrouping terms

$$H(y:w, x) = H(y:w) + H(y:w) + H(wxy)$$

where H(wxy) = H_w(y:x) - H(y:x) is a composite entropy term. As the space is partitioned further into n regions, the

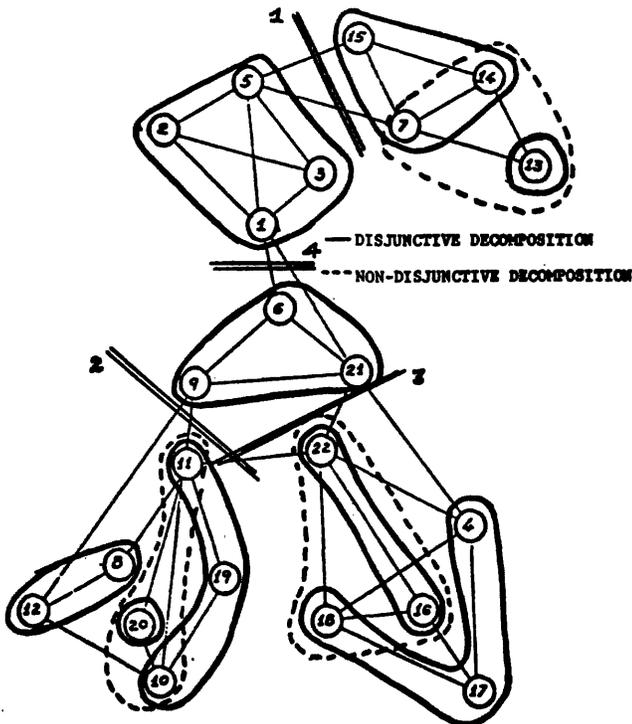


Figure 2—Decomposition using entropy metrics.

general expression for the redundancy criterion becomes

$$\begin{aligned} &H(y_1, y_2, \dots, Y_n; X_1, x_2, \dots, x_n) \\ &- H(y_1: y_2: \dots, y_n) \\ &- H(x_1: x_2: \dots, x_n) \end{aligned}$$

The minimization of this expression constitutes the decomposition criterion. Of course, each of the terms in the previous equation can be expanded to examine individually all of the simple redundancy terms, plus the various composite terms. To extremize these equations constitutes a formidable task mathematically, i.e., analytically, since the functional forms must be explicitly stated and may involve complex expressions. However, some heuristics concerning feasible alternative metrics are under consideration, utilizing

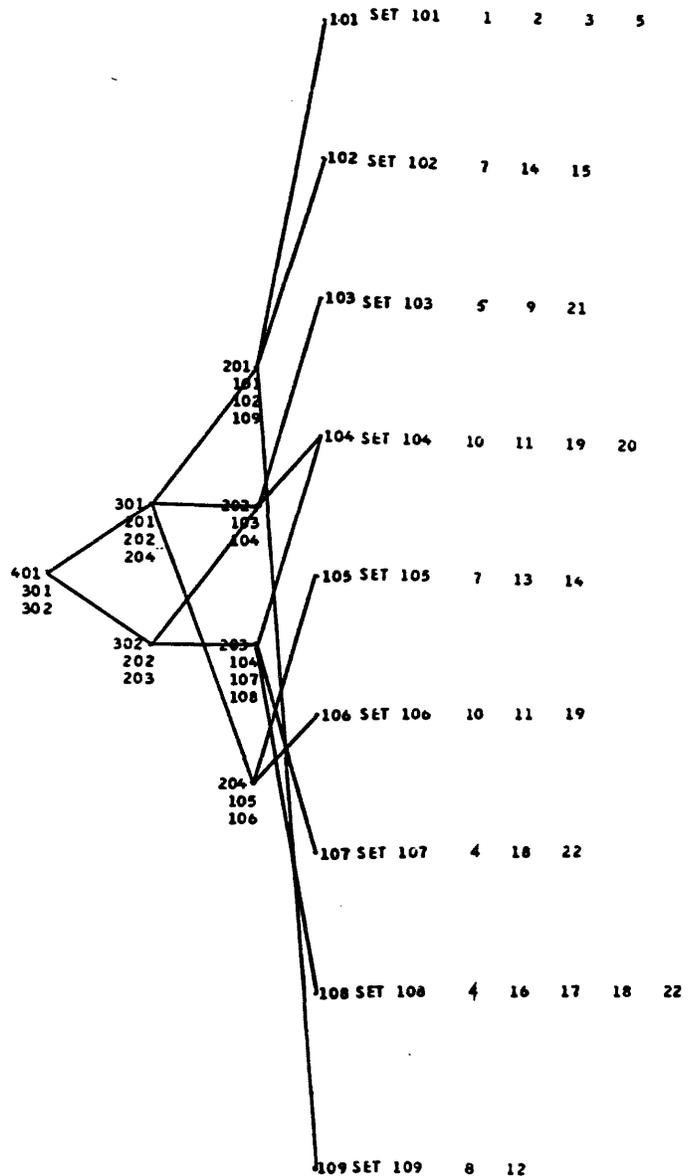


Figure 3—Hierarchical recombination.

An independent solution, using agglomerative clustering, is given in Figure 4.¹⁶ The solution was obtained using the matrix of "core sets" given by Andreu.

The use of a multi-attribute semi-metric is illustrated in Figure 5. Here, selected segments of rows and columns in the distance adjacency matrix were constructed and the dendrogram drawn based upon the polythetic clustering algorithms. At this point, no attempt was made to analyze in detail the particular structure of the various solutions obtained.

However, some general observations may be made. For example, weighting the variables by the variance accounted for resulted in a rather sharp delineation of the clusters when the distance function is used as the basis for grouping. The utilization of correlation matrices had the effect of producing larger, but fewer overall clusters. In both cases the varimax rotation produced the "best" parsimonious solution in terms of simple structure taxa. From empirical results the covarimin criterion was found to produce primary factors which were highly correlated; the bi-quartimin is an intermediate position but tending toward the lower correlations. (See References 17-26).

The conclusions emanating from this study will be deferred to a more comprehensive paper²⁷ dealing with the details of the methodology. The intent of this investigation is to lay the foundation for decomposition using entropy metrics, without resorting to excessive rigor, while at the same time report some interesting results.

ACKNOWLEDGMENTS

The author wishes to express his appreciation to Dr. C. E. Velez, Manager, Software Research, the Martin Marietta Corporation, for initially suggesting the problem, as well as for his many helpful insights and discussions concerning the uses and application of metric spaces. A special note of thanks is due to Mrs. Rose Lemieux for her painstaking efforts in the preparation of the manuscript.

REFERENCES

1. Paige, M. R., "Program Graphs, an Algebra, and Their Implication for Programming," *IEEE Trans. Software Eng.*, Vol. SE-1, Sept. 1975, pp. 286-291.
2. Paige, M. R., et al., "Structural Techniques of Program Validation," *Dig. 1974 COMPCON*, Feb. 1974, pp. 161-164.
3. Karp, R. M., "A Note on the Application of Graph Theory to Digital Computer Programming," *Inform. Contr.*, Vol. 3, 1960, pp. 179-190.
4. Ramamoorthy, C. V., et al., "Design and Construction of an Automated Software Evaluation System," *Proc. 1973 IEEE Symp. Computer Software Reliability*, pp. 28-37.
5. Stucki, L. G., "Automatic Generation of Self-Metric Software," *Proc. 1973 IEEE Symp. Computer Software Reliability*, pp. 94-100.
6. Deo, N., *Graph Theory with Applications to Engineering and Computer Science*, Englewood Cliffs, N.J., Prentice-Hall, 1974.
7. Busacker, R. G., and T. L. Saaty, *Finite Graphs and Networks: An Introduction with Applications*, McGraw-Hill, N.Y., 1965.
8. Berge, C., *Graphs and Hypergraphs*, New York, American Elsevier, 1973.
9. Koontz, W. L., P. M. Narendra and K. Fukunaga, "A Graph-Theoretic Approach to Nonparametric Cluster Analysis," *IEEE Trans. Comput.*, Vol. C-25, Sept. 1976, pp. 936-943.
10. Hausdorff, F., *Set Theory*, Chelsea, New York, 1957; GTTI Moscow-Leningrad, 1937 (Russian).
11. Bartels, P. H., G. F. Bahr, D. W. Calhoun and G. L. Wied, "Cell Recognition by Neighborhood Grouping Technique in TICAS," *Acta Cytol.*, Vol. 14, No. 6, AD 710844, 1970, pp. 313-324.
12. Lance, G. N., and W. T. Williams, "Computer Programs for Hierarchical Polythetic Classification ('Similarity Analysis')," *Comput. J.* Vol. 9, No. 1, 1966, pp. 60-64.
13. Schutt, D., "On A Hypergraph Oriented Measure for Applied Computer Science," *IEEE COMPCON*, Sept. 1977.
14. Alexander, C., and M. L. Manheim, "Hidecs 2: A Computer Program for the Hierarchical Decomposition of a Set which has an Associated Linear Graph," MIT. Civil Eng. Labs. Pub. 160, June. 1962. Revised. 1968.
15. Andreu, R. C., "Solving Decomposition Problems: Alternative Techniques and Description of Supporting Tools," MIT, Sloan School, June, 1977 (Draft).
16. Parks, J. M., "Q-Mode Cluster Analysis," *Univ. of Kansas Computer Contrib.*, #46, 1970.
17. Harman, H., *Modern Factor Analysis*, Univ. of Chicago Press, 1960.
18. Anderson, T. W., and H. Rubin, "Statistical Inference in Factor Analysis," *Proc. of the Third Berkeley Symp. on Math. Statistics and Probability*, Vol. 5, 1956, pp. 11-50.
19. Baggaley, A., and R. B. Catell, "A Comparison of Exact and Approximate Linear Function Estimates of Oblique Factor Scores," *BJ Stat. Psych.*, Vol. 9, 1956, pp. 83-86.
20. Harris, Chester W., and Dorothy M. Knoell, "The Oblique Solution in Factor Analysis," *JEP*, Vol. 39, 1948, pp. 385-403.
21. Harris, C. W., and J. Schmid, Jr., "Further Application of the Principles of Direct Rotation in Factor Analysis," *J. Exp. Ed.*, Vol. 18, 1950, pp. 175-93.
22. Harsh, Charles M., "Constancy and Variation in Patterns of Factor Loadings," *JEP*, Vol. 31, 1940, pp. 335-59.
23. Bargmann, R., "The Statistical Significance of Simple Structure in Factor Analysis" (Mimeographed), Frankfurt-Main (Germany); Hochschule fuer Internationale Paedagogische Forschung, 1953.
24. Bargmann, R., "A Comparison of New Analytic Methods for the Determination of Simple Structure" (Mimeographed), Frankfurt-Main, Germany, Hochschule fuer Internationale Paedagogische Forschung, 1953.
25. Barlow, J. A., and C. Burt, "The Identification of Factors from Different Experiments," *BJ Stat. Psych.*, Vol. 7, 1954, pp. 52-56.
26. Bartlett, M. S., "The Statistical Conception of Mental Factors," *BJP*, Vol. 28, 1937, pp. 97-104.
27. Silver, A. N., "On the Structural Decomposition and Hierarchical Re-combination of Non-Directed Linear Graphs Using Multi-Attribute Agglomerative Polythetic Clustering Metrics," to be presented at the *Constructive Approaches to Mathematical Models Symposium* to be held at the Mellon Institute of Carnegie-Mellon University on July 10-14, 1978.

Interactive modeling systems for managers—Semantic models should underlie quantitative models

by RAND B. KRUMLAND

Baylor College of Medicine
Houston, Texas

INTRODUCTION

This paper is concerned with helping managers build and use quantitative models in problem-solving, especially models that are built and used on on-line, interactive computer systems. Quantitative models play a large part in many of the activities of managers of a wide variety of organizations. Indeed, several companies currently find it profitable to provide managers with the service of access to models and modeling systems which have increasing levels of sophistication.^{2,8} However, such models are not used as widely as they could be.⁶ Certain concepts are beginning to emerge which will hasten the development and deployment of more sophisticated interactive modeling systems which may improve this situation. Here we define one such concept—that of a *semantic model*—and discuss its place in an advanced interactive modeling system.

THE CONCEPT OF A SEMANTIC MODEL

Neither a semantic model nor the facilities for constructing one exist in any modeling system available today for general use. The proposition that a semantic model would be a useful adjunct to a modeling system rests on new ideas about the process of creating quantitative models.^{1,4,5} Consider the following simple description of a business situation:

U. S. Robot is a corporation which produces and sells robots. Robots are produced out of bodies and central processing units. The bodies are fabricated out of sheet metal and rivets, and the central processing units are purchased from Texas Instrument. In 1977 sales of robots were 17 million dollars. It is expected that sales this year will increase by 12 percent, direct costs will increase by 10 percent, and overhead will increase by six percent.

Suppose that the president of U. S. Robot would like to know what profits will be in 1978 in light of the expectations that are given. The following quantitative model would help

him answer that question:

$$\begin{aligned}\text{PROFIT}(1978) &= \text{SALES}(1978) - \text{COSTS}(1978) \\ \text{SALES}(1978) &= \text{SALES}(1977) * 1.12 \\ \text{COSTS}(1978) &= \text{DIRECT-} \\ &\quad \text{COSTS}(1978) + \text{OVERHEAD}(1978) \\ \text{DIRECT-COSTS}(1978) &= \text{DIRECT-COSTS}(1977) * 1.10 \\ \text{OVERHEAD}(1978) &= \text{OVERHEAD}(1977) * 1.06\end{aligned}$$

Although this is a simple model, an analysis of what underlies its production can yield important insights.

Let us presume that a consultant has produced this model for the president, has implemented it as a computer program and stands ready to use it to answer the president's question. What must be true of the consultant and what must he have done to have produced the model? First, he must obviously know many things—most importantly, he must know something about business in general and about manufacturing firms and processes in particular; he must know how business activities are measured and characterized; and he must know how to structure sets of algebraic equations into a useful model. Second, he must have learned the information that is given in the brief description of U. S. Robot. Third, he must know how to apply elements of the first set of knowledge to what he learned about U.S. Robot. In addition, of course, the consultant must know much more—many “common sense” things, natural language, how to write or type, etc.—but we have enough to work with for the moment.

A diagram of what the consultant knows is shown in Figure 1. Of everything the consultant knows—of his entire *knowledge base*—only the quantitative model is ever made known to the computer. Yet it is those other things that the consultant knows that have allowed him to be successful in formulating the model of his client. This brings us to the first major point of our argument—if the power of the computer is to be brought to bear on the model building task, then it will have to contain more elements analogous to those that make up the consultant's knowledge base than it currently does. If the computer is to act more like the con-

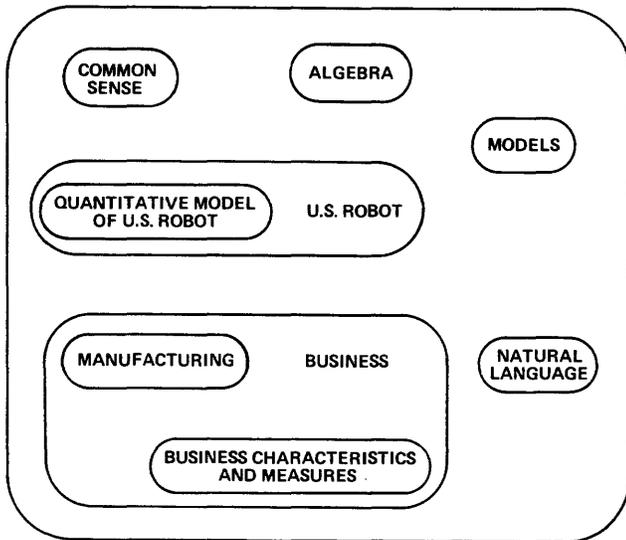


Figure 1—What the consultant must know in order to construct the quantitative model.

sultant, it will have to “know” about more things than simply the quantitative model.

If we want to build such additional knowledge into a computer system, where do we start? More common sense would certainly be a candidate as would a natural language facility. However, the diagram of the consultant’s knowledge points to a different answer—the quantitative model that the consultant built is part of his knowledge about U.S. Robot. That model is obviously linked in many ways with the other knowledge about U.S. Robot; in fact, it is an abstraction of other parts of that knowledge. This brings us to the second major point of our argument—based on our observations, that other knowledge about the current situation—about U.S. Robot—would be extremely useful to have inside the computer and the capability to capture and use it is the best next thing to try to add to a modeling system for a manager. So in terms of the diagram in Figure 1, if the current boundary of a computer’s knowledge is congruent with the circle delineating the quantitative model, then we would like to push that boundary out to encompass more knowledge about the current situation.

A semantic model is just that knowledge of the current situation; it includes the quantitative model plus other quantitative and non-quantitative information. For the consultant it is everything he knows about the current situation; for a computer system then, a semantic model is the quantitative model plus other information that the system obtains and stores that is not necessarily part of the quantitative model but that represents parts of the situation for which that model is built.

We have stressed the static relationships between semantic models and other knowledge; the place of semantic models in the modeling process is important to understand as well. Any model is an abstraction of the things that it models, a representation of them in some other terms. In the process of formulating the quantitative model, the con-

sultant first constructed his semantic representation, and then through abstraction processes he derived the quantitative model from it, as depicted in Figure 2. Thus, if the computer is going to aid substantially in the task of formulating a quantitative model, it, too, will have to have the semantic model as a basis for its actions.

It is not uncommon for users of computer systems to wish that those systems were smarter, more intelligent, or “not so stupid.” Minsky argues that one hallmark of an intelligent system is that it contains “internal models” of its environment and of the various objects, including itself, within that environment.¹³ Our semantic models could serve as more effective internal models for quantitative modeling systems and therefore form the basis for more intelligent action in general part of which could be improved modeling capabilities.

BUILDING A SEMANTIC MODEL

A semantic model is a representation of something in the real world and as such it needs a representational system. Researchers in the fields of artificial intelligence and knowledge-based systems have developed methods to represent knowledge in computer systems—these include production systems, semantic nets, frames, conceptual dependency structures, etc. No single best way to represent knowledge has been found, but there are clearly several facilities that a representation system must provide and many problems that it must deal with. To be a useful basis for a quantitative model for a manager, a semantic model will at least have to contain representations of entities, actions, events, values and relationships in a business environment. Thus we would expect to be able to represent such things as a corporation, company or other business entity, the products or services that a company produces or provides, the different functional and organizational parts of a company such as a division or the marketing department, activities that the organization engages in such as selling, employing, producing and advertising, characteristics that a business entity uses to measure its activities such as sales, costs, prices and wages, etc. In addition, a representational system will need to deal with relationships that link these entities, actions and characteristics together.

A simple example of a representation will illustrate some of these concepts. Figure 3 illustrates part of an encoding of the description of U.S. Robot in a diagram rendering of a frame system, i.e., a system which uses the ideas of frame theory for representation. The diagrams are similar to those used by Winston in explaining frames.¹⁸ In Figure 4 an



Figure 2—The model building process.

encoding is given in a version of a knowledge representation scheme called OWL which is under development at MIT's Laboratory for Computer Science.^{17,10}

In spite of the simplicity of this description, it presents some difficult problems in representation. For instance, there are two activities that U.S. Robot engages in—producing robots and selling robots—and a natural presumption is that the robots they sell are the same ones that they produce. Also, note that the quantity “sales of robots” is a measure of the selling robots activity. Such links must be encoded in the semantic model and the representation scheme must therefore provide ways to handle them.

The nature of the representations given point to many aspects of a semantic modeling system which have not been addressed. First, semantic models are not very useful without programs which “know” how to use them to help in modeling. In fact, the knowledge that such programs effectively contain must be part of any semantic modeling system. Second, a semantic model is realized as a set of links and pointers to a larger structure within the system. For instance, the fact that U.S. Robot is a CORPORATION cannot be represented unless the concept CORPORATION has been previously defined in some way to the system. In addition, the fact is not useful unless some other general facts about CORPORATIONS are known which can be used to help understand and deal with this particular corporation. The concepts and links that make up the semantic model of the current user's situation are really only the tip of an iceberg, or rather the tip of a much larger knowledge base. A semantic modeling system that rests on only a limited knowledge base could be useful, but a larger knowledge base will provide greater power, flexibility and increased usefulness for the quantitative modeling task.

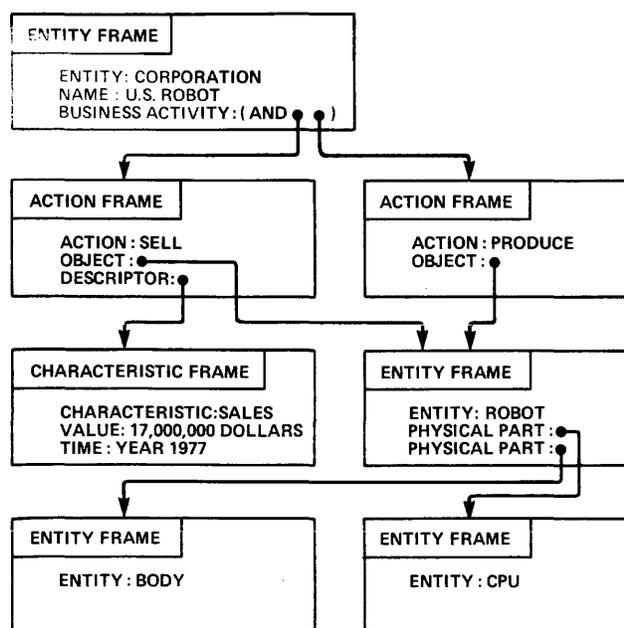


Figure 3—U.S. Robot description in a frame encoding.

```

[ US-ROBOT = (CORPORATION "") ]

[ (PRODUCE ROBOT 1 = (ROBOT 1))
  AGENT: US-ROBOT
  OBJECT: ROBOT 1 ]

[ (SELL ROBOT 1)
  AGENT: US-ROBOT
  OBJECT: ROBOT 1 ]

[ ROBOT 1
  PHYSICAL-PART: [ROBOT-BODY = (BODY ROBOT 1)]
  PHYSICAL-PART: [ROBOT-BRAIN = (CPU ROBOT 1)] ]

[ SALES-ROBOTS-77 = ((SALES ROBOT 1) (TIME (YEAR 1977)))
  (DESCRIPTOR (SELL ROBOT 1))
  VALUE: (DOLLARS 17000000) ]

[ SALES-ROBOTS-78 = ((SALES ROBOT 1) (TIME (YEAR 1978)))
  (DESCRIPTOR (SELL ROBOT 1))
  [ ((BE (GREATER-THAN SALES-ROBOTS-77))
    SALES-ROBOTS-78)
  BY: (PERCENT 12) ] ]
  
```

Figure 4—U.S. Robot description in an OWL encoding.

USING A SEMANTIC MODEL

A semantic model serves as the repository for information that the user has transmitted to the system. Thus, for the quantitative modeling system it provides a focal point for interaction with the user; it provides a means for the system to “remember” what it has learned and it serves as the “raw” data from which the parts of the quantitative model can be constructed or parameterized. In a future modeling system quantitative models could be more automatically constructed from semantic models; however, the current genre of interactive modeling systems could also benefit by the incorporation of semantic models.

Techniques have been developed for automatically formulating appropriate quantitative expressions and relations from semantic structures.^{1,4,9,5} For example, in our sample of a semantic model given previously the English language phrase

U.S. Robot's profit in 1978 will be 12 percent greater than in 1977

if rendered in a semantic model as

```

[[ (BE (GREATER-THAN
  [PROFIT-US-ROBOT-1977
  = ((PROFIT US-ROBOT)
    (TIME (YEAR 1977))))))
  [PROFIT-US-ROBOT-78
  = ((PROFIT US-ROBOT)
    (TIME (YEAR 1978))))]]
  
```

can be translated automatically into an equation such as

$$profit(1978) = profit(1977) * 1.12$$

For a more difficult example, consider generating an equation for the quantity "cost of producing robots" which decomposes that cost into other costs. Cost is a measure of a process here that has an output, two inputs—CPUs and bodies—and undoubtedly involves some direct labor and overhead. Therefore the cost equation must be generated from the semantic descriptions of the process, of the descriptions of the inputs and outputs and other information that must be gleaned about labor quantities and rates as well as overhead and overhead allocation procedures.

Individual terms and equations is the easy part of automatic model generation; producing a model that is complete, well formed and which addresses the manager's problem is significantly harder. General mechanisms have been identified which make up any procedure for producing an entire model.⁵ They include *concept production*, or finding parts of the semantic model to be recast for the quantitative model, *integration* or *fitting*, or completing concepts so that they correctly reflect the user's current situation and *equation generation*, or generating variables and the appropriate set of algebraic connectors and relations for a well formed equation. A modeling system that is at all automatic will incorporate these mechanisms to a degree.

Semantic models would also be useful adjuncts to existing modeling systems. They could be used to enable more natural command languages, to allow more flexible command structures, to provide for more intelligent and intelligible prompts for data acquisition and model parameterization, to satisfy a broader spectrum of naive and sophisticated users, etc. For instance, when a system prompts for and receives the value of a variable, a semantic model would provide a general place to store it allowing it to be more readily accessed by other parts of the system. Or, a semantic model could allow model variables to be more readily related to the manager's meaning for them; if there is a variable "SLS(77)" in the quantitative model and the user refers to it as "our sales in 1977," through the use of a semantic model the system could "understand" the reference. A semantic model is a tool that could facilitate the addition of many important features to a modeling system which would make it much more useable and used.

CONCLUSION

We have discussed the notion of a semantic model as an emerging concept in the design and construction of interactive modeling systems for managers. In fact, a semantic model will underlie any interactive system which is intended to have users who are not obliged to program the computer at a low level. A detailed discussion of how to build a more general semantic modeling facility and how to integrate it with a quantitative modeling system must be deferred until better examples of such systems have been constructed and tested in use. Progress in this area will depend on progress in the relevant areas of AI and on the ability of builders of practical systems to take the work and results from that field and develop them for practical use. In the meantime systems

like Management Decision System's EXPRESS and Meador's PROJECTOR remain the most advanced practical systems in this regard^{8,11} and Malhotra's work on a prototypical system demonstrates many things that can be done.⁷ The primary utility of a semantic model for current system builders is as an organizing concept and a conceptual focal point for design considerations that to date has been lacking.

ACKNOWLEDGMENT

This work is based on thesis research which was supported in part by the Advanced Projects Research Agency of the Department of Defense and was monitored by the Office of Naval Research under contract number N00014-75-C-0661. I want to thank Bill Martin and Tony Gorry for significant contributions to the ideas presented here.

REFERENCES

1. Charniak, Eugene, "CARPS, A Program Which Solves Calculus Word Problems," Technical Report TR-51 (Thesis), Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, July, 1968.
2. Execucom Systems Corporation, *IFPS User's Manual*, Austin, TX, March, 1978.
3. Gorry, G. Anthony, "The Development of Managerial Models," *Sloan Management Review*, Vol. 12, No. 2, Winter, 1971.
4. Heidorn, George A., "English Language as a Very High Level Language for Simulation Programming," in *Proceedings of a Symposium on Very High Level Languages* which is reprinted as *ACM SIGPLAN Notices*, Vol. 9, No. 4, April, 1974.
5. Krumland, Rand B., "Base Concepts and Mechanisms in Knowledge-Based Model Building for Managers," Ph.D. Thesis, Laboratory for Computer Science and Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, June, 1977.
6. Little, John D. C., "Models and Managers: The Concept of a Decision Calculus," *Management Science*, Vol. 16, No. 8, April, 1970.
7. Malhotra, Ashok, "Design Criteria for a Knowledge-Based English Language System for Management: An Experimental Analysis," Technical Report TR-146 (Thesis), Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, February, 1975.
8. Management Decision Systems, Inc., *Express Reference Manual*, Weston, MA, July, 1974.
9. Mark, William S., "The Reformulation Model of Expertise," Technical Report TR-172 (Thesis), Laboratory for Computer Science, Massachusetts Institute of Technology, August, 1976.
10. Martin, William A., "Description and the Specialization of Concepts," Technical Memo TM-101, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, March, 1978.
11. Meador, C. L., "Long Range Planning with an Interactive Computer Based Decision Support System," Thesis, Sloan School of Management, Massachusetts Institute of Technology, August, 1972.
12. Meador, C. L., and D. N. Ness, "Decision Support Systems: An Application to Corporate Planning," *Sloan Management Review*, Vol. 15, No. 12, Winter, 1974.
13. Minsky, Marvin, "Matter, Mind, and Models," in *Semantic Information Processing*, Marvin Minsky (ed.), M.I.T. Press, Cambridge, MA, 1968.
14. Minsky, Marvin, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, Patrick Henry Winston (ed.), McGraw-Hill Book Company, New York, 1975.
15. Newell, Allen, and Herbert A. Simon, *Human Problem Solving*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972.
16. Shank, Roger C., "Identification of Conceptualizations Underlying Natural Language," in *Computer Models of Thought and Language*, Roger

- C. Shank and Kenneth Mark Colby (eds.), W. H. Freeman and Company, San Francisco, CA, 1973.
17. Szolovits, Peter, Lowell B. Hawkinson and William A. Martin, "An Overview of OWL, A Language for Knowledge Representation," Technical Memo TM-86, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, June, 1977.
 18. Winston, Patrick Henry, *Artificial Intelligence*, Addison-Wesley Publishing Company, Reading, MA, 1977.
 19. Woods, William A., "What's in a Link: Foundations for Semantic Networks," in *Representation and Understanding*, Daniel G. Bobrow and Allen Collins (eds.), Academic Press, Inc., New York, 1975.

Modeling regular, process-structured networks*

by BRUCE W. ARDEN and HIKYU LEE

Princeton University
Princeton, New Jersey

INTRODUCTION

An interesting strategy to exploit micro-technology is to interconnect microcomputers (i.e., microprocessor with local memory) in a regular dataflow network of low degree. The network is regular so that each computer is a similar "building block" with regard to the number of connecting data buses. The number of buses is small not only because microcomputers are inherently limited in their bus capacity but also because many incident buses lead to switching and "memory port" complexity, which is difficult to handle by micro-circuits. Such complexity is one of the reasons why shared memory "mainframe" computers are relatively expensive. In essence, the low-degree, regular network approach replaces the hard-wired switching with programmed message-passing. Since the processor node will not be fully utilized by productive computing, some of the capacity can profitably be used for such message-handling.

This paper is concerned with an illustrative comparison of performance of two such data flow networks. The first is not regular and it models a particular process structure in which all messages are passed to the intended destination within a single step. For convenience, we call this network a *process network*. In the second, the same process structure is mapped onto a *regular network* (of degree 3) with the result that a number of multi-step message paths are introduced. The comparison shows the effects on system utilization and response time of the additional message handling overhead introduced in the regular network.

PROCESS STRUCTURE

Current multiprogrammed systems are usually described as *process-structured*. The processes that are enabled for execution are assigned system resources on the basis of some scheduling procedure. An alternative architecture, which arises from the development and low cost of microcomputers, is to assign each system and user process to a microcomputer (which may have attached I/O devices) for the entire life of the process. The use of a process becomes a matter of sending data to the appropriate process node,

and ultimately, receiving the response. This is a data flow network in the sense that the messages enable the resident processes to carry out their specific computation. It is not unreasonable to think about networks of hundreds of microcomputers but, of course, the geometry of a regular network and the mapping of the process structure onto the regular network become important considerations.¹

In a system of processes, some are *autonomous* and some are *subordinate*. That is, some spontaneously generate a message and others are passive in the sense that they only react to the receipt of a message and are otherwise idle. Clearly, *user* processes are in the first category and most, but not all, of *system* processes are in the second. For example, system processes such as archiving, and spooling routines can act autonomously.

Each autonomous process and the subordinate processes with which it communicates and for which it provides a workload comprise a *chain* with one circulating message. It is assumed that chains, even if they are indistinguishable with respect to their workload demands, cannot be combined in the process system, since they cannot be correspondingly combined in the regular system due to the distinct locations of each autonomous processes resulting from the 1-1 mapping of the process structure onto the regular network. Hence, the number of chains is equal to the number of autonomous processes and each chain contains one circulating message. This situation is quite different from the models of conventional, tightly-coupled systems where the number of processes is generally larger than the number of processors.

In the language of queueing theory, the circulating entities are usually called *customers* and a specific *customer class* is identified by the distinguishing workload it presents at each service center. To preserve the data flow orientation, this concept is here called *message class*. Circulating messages are *processed* at each node in the process system. In the regular system, they are, in addition, simply *retransmitted* at some nodes without being processed locally.

UNDERLYING MODEL FOR PROCESS SYSTEM

As the preceding suggests, the underlying model for the process system is a closed network of queues model with multiple chains and different classes of messages. There is

* This work has been supported by NSF Grant DCR 74-18655.

a finite number of processors indexed $1, 2, \dots, M$ and a finite number of circulating messages numbered $1, 2, \dots, N$, where N is equal to the number of autonomous processes in the system. Obviously, we have $N < M$. Since each autonomous process defines a chain, we assume that there is a finite number of different classes of messages $1, 2, \dots, R$, where R is equal to the number of autonomous processes, i.e., $R = N$. Let N_i denote the number of messages in chain i ($i = 1, 2, \dots, R$), which is equal to one for each chain i . Assuming that the messages in chain i are of class i , N_i also denotes the number of class i messages in the system.

The behavior of circulating messages is determined by its service time distribution at each service center and transition pattern through the network. The service time distribution of messages is assumed to be general with rational Laplace transformation⁶ and the queuing discipline at each service center is assumed to be LCFS-PR (Last Come First Served-Preemptive Resume). The transition of circulating messages is described by the transition probability matrix $Q = \{q_{irjs}\}$, where q_{irjs} denotes the probability that a class r message completing its service at service center i will require its next service at service center j in message class s . Since class r messages are confined in chain r , we have

$$q_{irjs} = 0 \quad \text{if } r \neq s (r, s = 1, 2, \dots, R)$$

We define the state of the model for the process system as the number of messages in each class at each service center. Thus, the state S is written as

$$S = (y_1, y_2, \dots, y_M),$$

where

$$y_i = (n_{i1}, n_{i2}, \dots, n_{iR}).$$

Then a feasible state of the model satisfies

$$\tilde{N} = (N_1, N_2, \dots, N_R) = \sum_{i=1}^M y_i,$$

where

$$N_j = \sum_{i=1}^M n_{ij} = 1 \quad (j = 1, 2, \dots, R)$$

and

$$N = \sum_{i=1}^R N_i = \text{constant}.$$

The n_{ir} denotes the number of class r messages at service center i .

UNDERLYING MODEL FOR REGULAR SYSTEM

The regular system is obtained from the process system by mapping the process structure onto a regular network. Hence, as in the process system, there are M processors and N circulating messages in the system. We assume that a specific network geometry and the mapping for the regular system are given.

Since the adjacency relationship between processes in the process system is not, in general, preserved due to the mapping, transition of class r messages from service center i to an adjacent service center j in the process system might take a multi-step path in the corresponding regular system. We add R new message classes $\bar{1}, \bar{2}, \dots, \bar{R}$ to our model for the regular system to distinguish messages being retransmitted, i.e., if a class r message at service center i makes a multi-step transition to service center j for the next service request, then we assume that there exists class \bar{r} message arrival at each intermediate service centers along the path. Hence, each service center in the regular system is presumed to execute a local, system or user process (which may involve the support of a connected I/O device) as well as simple message-passing task for inter-node communications.

We assume that the shortest path between service center i and service center j in the regular system is used for the inter-node communication. Furthermore, if there exists more than one such path, then each shortest path is used with equal probability. For convenience, we call this routing scheme as SPEP (Shortest Path with Equal Probability) routing scheme.

The service time distribution of messages is assumed to be general with rational Laplace transformation and the queuing discipline at each service center is assumed to be LCFS-PR (Last Come First Served-Preemptive Resume).

The transition probability matrix $P = \{p_{i\alpha j\beta}\}$ ($\alpha, \beta = 1, 2, \dots, R, \bar{1}, \bar{2}, \dots, \bar{R}$) describes transition of messages and can be derived for the regular system from the transition probabilities given in the process system as follows. Let $\lambda_{i\alpha j\beta}$ denote the relative rate of class β message arrival at node j from node i and class α . $\lambda_{i\alpha j\beta}$ corresponds to the average number of times a class α message at node i visits link (i, j) to become a class β message at node j . Let e_{ir} denote the relative arrival rate of class r messages at service center i in the process system, where $i = 1, 2, \dots, M$ and $r = 1, 2, \dots, R$. The e_{ir} are obtained up to multiplicative constant from the following set of linear equations:

$$e_{js} = \sum_{i=1}^M \sum_{r=1}^R e_{ir} q_{irjs},$$

where $j = 1, 2, \dots, M$ and $s = 1, 2, \dots, R$. Then, for each node pair (x, y) in the process system such that $e_{xr} > 0$ and $q_{xryr} > 0$ for some r ($r = 1, 2, \dots, R$), the total relative arrival of messages at node y from node x in the regular system is $e_{xr} q_{xryr}$ and the fraction which passes through an edge (i, j) in the set of edges of the shortest paths from node x to node y is given by

$$e_{xr} q_{xryr} P(x, i; x, y) P(i, j; x, y),$$

where $P(x, i; x, y)$ denotes the probability that a message, which has completed a local service at node x and is making a multi-step transition to node y for the next local service request, will pass through node i . and $P(i, j; x, y)$ denotes the probability that the message which is passing through node i will make a transition to an adjacent node j under SPEP routing scheme. Note that $P(x, i; x, y)$ and $P(i, j; x, y)$.

y) are given by

$$P(x, i; x, y) = \frac{\# \text{ of shortest paths from } n_x \text{ to } n_y \text{ which include } n_i}{\text{total } \# \text{ of shortest paths from } n_x \text{ to } n_y}$$

and

$$P(i, j; x, y) = \frac{\# \text{ of shortest paths from } n_j \text{ to } n_y}{\# \text{ of shortest paths from } n_i \text{ to } n_y}.$$

For the special case of fixed routing, where a specific single path is used for the inter-node communication between node x and node y , we have

$$P(x, i; x, y) = P(i, j; x, y) = \begin{cases} 1 & \text{if } n_i \text{ and } n_j \text{ are on the path.} \\ 0 & \text{otherwise.} \end{cases}$$

Considering all the node pairs (x, y) ($x, y=1, 2, \dots, M$) in the process system such that $e_{xr} > 0$ and $q_{xrr} > 0$ for $r=1, 2, \dots, R$, we get $\lambda_{i\alpha j\beta}$ ($i, j=1, 2, \dots, M$ and $\alpha, \beta=1, 2, \dots, R, \bar{1}, \bar{2}, \dots, \bar{R}$) for the regular system. Then, the transition probability matrix $P = \{p_{i\alpha j\beta}\}$ is obtained from:

$$P_{i\alpha j\beta} = \lambda_{i\alpha j\beta} / \sum_{k=1}^M (\lambda_{i\alpha kr} + \lambda_{i\alpha k\bar{r}}),$$

where $\alpha, \beta=r$ or \bar{r} , $r=1, 2, \dots, R$ and $i, j=1, 2, \dots, M$. We also get $\lambda_{i\alpha}$, the relative arrival rate of class α messages at service center i in the regular system from:

$$\lambda_{i\alpha} = \sum_{k=1}^M (\lambda_{kri\alpha} + \lambda_{k\bar{r}i\alpha}),$$

where $\alpha=r$ or \bar{r} and $r=1, 2, \dots, R$. Since each chain r has two different message classes, i.e., class r and class \bar{r} , we denote the number of class r messages in chain r as N_{rr} and class \bar{r} as $N_{r\bar{r}}$. Then, for each chain r , we have

$$N_r = N_{rr} + N_{r\bar{r}} = 1.$$

The state of the model for the regular system is defined as

$$S = (y_1, y_2, \dots, y_M),$$

where $y_i = (n_{i1}, n_{i2}, \dots, n_{iR}, n_{i\bar{1}}, n_{i\bar{2}}, \dots, n_{i\bar{R}})$. Then a feasible state of the model satisfies

$$\tilde{N} = (N_1, N_2, \dots, N_R) = \sum_{i=1}^M \hat{y}_i,$$

where

$$N_j = N_{j\bar{j}} + N_{jj} = \sum_{i=1}^M (n_{ij} + n_{i\bar{j}}) = 1 \quad (j=1, 2, \dots, R),$$

$$N = \sum_{i=1}^R N_i = \text{constant}$$

and

$$\hat{y}_i = (n_{i1} + n_{i\bar{1}}, n_{i2} + n_{i\bar{2}}, \dots, n_{iR} + n_{i\bar{R}}).$$

PERFORMANCE MEASURES

The underlying models for both process and regular system are, in essence, special cases of the general queueing network model developed by Baskett et al.,³ where the equilibrium state probabilities are shown to have the following product form:

$$P(S) = C(\tilde{N}) \prod_{i=1}^M f_i(y_i),$$

where

$$f_i(y_i) = \begin{cases} n_i! \prod_{r=1}^R \frac{1}{n_{ir}!} \left[\frac{e_{ir}}{\mu_{ir}} \right]^{n_{ir}} & \text{for process system} \\ n_i! \prod_{r=1}^R \frac{1}{n_{ir}!} \left[\frac{\lambda_{ir}}{\mu_{ir}} \right]^{n_{ir}} \prod_{\bar{r}=1}^R \frac{1}{n_{i\bar{r}}!} \left[\frac{\lambda_{i\bar{r}}}{\mu_{i\bar{r}}} \right]^{n_{i\bar{r}}} & \text{for regular system} \end{cases}$$

The normalization constant $C(\tilde{N})$ is obtained by equating the sum of these products, over all states, to unity, i.e.,

$$C(\tilde{N})^{-1} = \sum_{\substack{\text{all feasible} \\ \text{states}}} \prod_{i=1}^M f_i(y_i).$$

The direct approach to compute the normalization constant yields exponential growth in the number of algebraic operations. However, efficient computational techniques have been developed by several authors,^{2,9} which are, in essence, generalizations of Buzen's result.⁵

Let $P_i(y_i)$ denote the marginal probability that the service center i is in state y_i , where $y_i = (n_{i1}, \dots, n_{iR})$ for the process system and $(n_{i1}, \dots, n_{iR}, n_{i\bar{1}}, \dots, n_{i\bar{R}})$ for the regular system. Then, we have

$$P_i(y_i) = \sum_{\substack{\text{all states} \\ \text{s.t. node } i \\ \text{is in state } y_i}} P(y_1, y_2, \dots, y_M)$$

The mean number of class θ messages at service center i , $E(n_{i\theta})$ is given by

$$E(n_{i\theta}) = \sum_{k=1}^{N_\theta} \left\{ \sum_{\substack{\text{all states } y_i \\ \text{s.t. } n_{i\theta}=k}} P_i(y_i) \right\} k,$$

where $\theta=1, 2, \dots, R$ for the process system and $1, 2, \dots, R, \bar{1}, \bar{2}, \dots, \bar{R}$ for the regular system.

The utilization of service center i by class θ messages, $\rho_{i\theta}$ is obtained from:

$$\rho_{i\theta} = \sum_{\substack{\text{all} \\ \text{states } y_i}} P_i(y_i) \frac{n_{i\theta}}{n_i}$$

We define the mean response time of class r ($r=1, 2, \dots, R$) messages, T_r as the time for the class r message leaving an autonomous node in chain r after finishing its local service request to revisit the autonomous node in class r after finishing its service requirements in the rest of the system. Assuming that node i is the autonomous node in

chain r , the total number of the messages in the rest of the system is $N_r - E(n_{ir})$. Applying Little's result⁷ to the rest of the system, we have

$$T_r = \frac{N_r - E(n_{ir})}{v_{ir}}$$

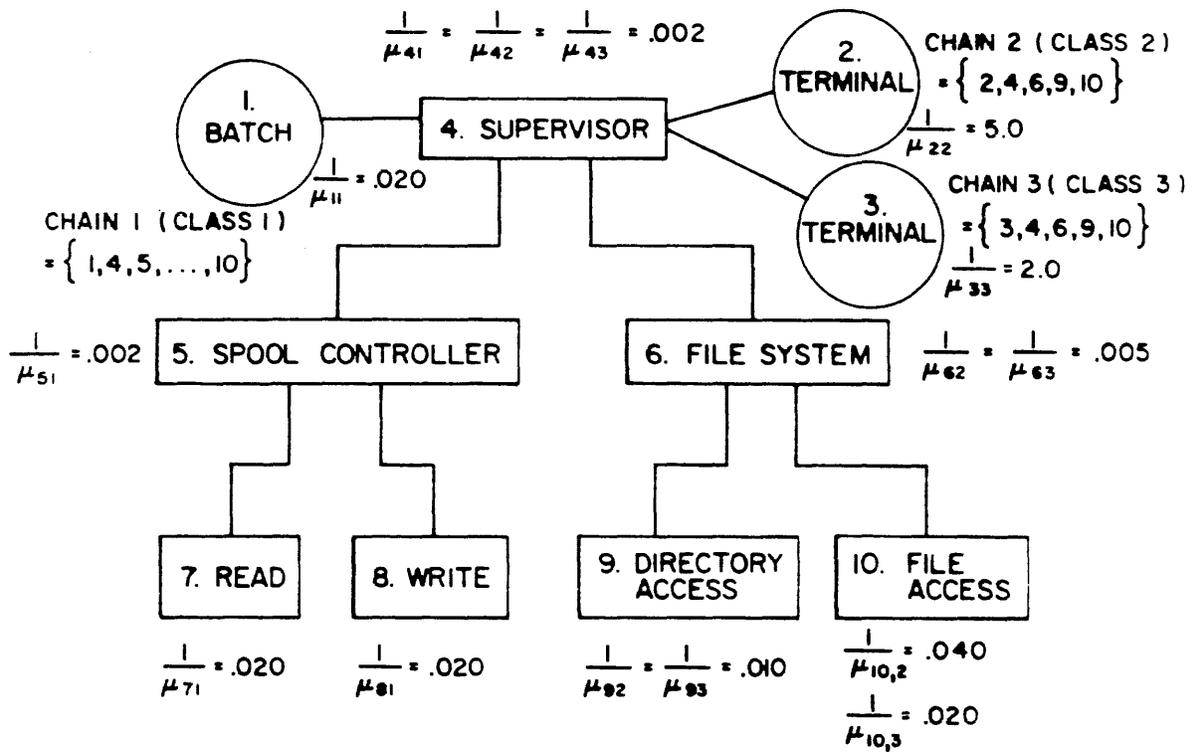
where v_{ir} is the mean departure rate of class r messages at service center i .

EXAMPLE

A 10-node example of process system with necessary parameter values is shown in Figure 1. There are three autonomous nodes, 1, 2 and 3, each of which comprises a chain.

This example, though not based on an actual system, is intended to represent a simple hierarchical operating system structure with different types of task requirements. Chain 1 represents compute-bound batch processing. Node 1 reads a program either in card image format via spool controller or from disk file via file system. Similarly, it produces output to spool controller or for storage in the disk file. Chain 2 and Chain 3 both represent interactive processing of high-level file operations, where Node 2 and Node 3 represent rather long file operation compared to Chain 3.

The regular system, onto which the process structure in Figure 1 is mapped, is shown in Figure 2. Transition prob-



TRANSITION PROBABILITIES

CHAIN 1		CHAIN 2		CHAIN 3	
$q_{4111} = .50$	$q_{6141} = .40$	$q_{4222} = .05$	$q_{4333} = .05$		
$q_{4151} = .40$	$q_{6191} = .30$	$q_{4262} = .95$	$q_{4363} = .95$		
$q_{4161} = .10$	$q_{6210} = .30$	$q_{6242} = .40$	$q_{6343} = .40$		
$q_{5141} = .50$		$q_{6292} = .30$	$q_{6393} = .30$		
$q_{5171} = .25$		$q_{62102} = .30$	$q_{63103} = .30$		
$q_{5181} = .25$					

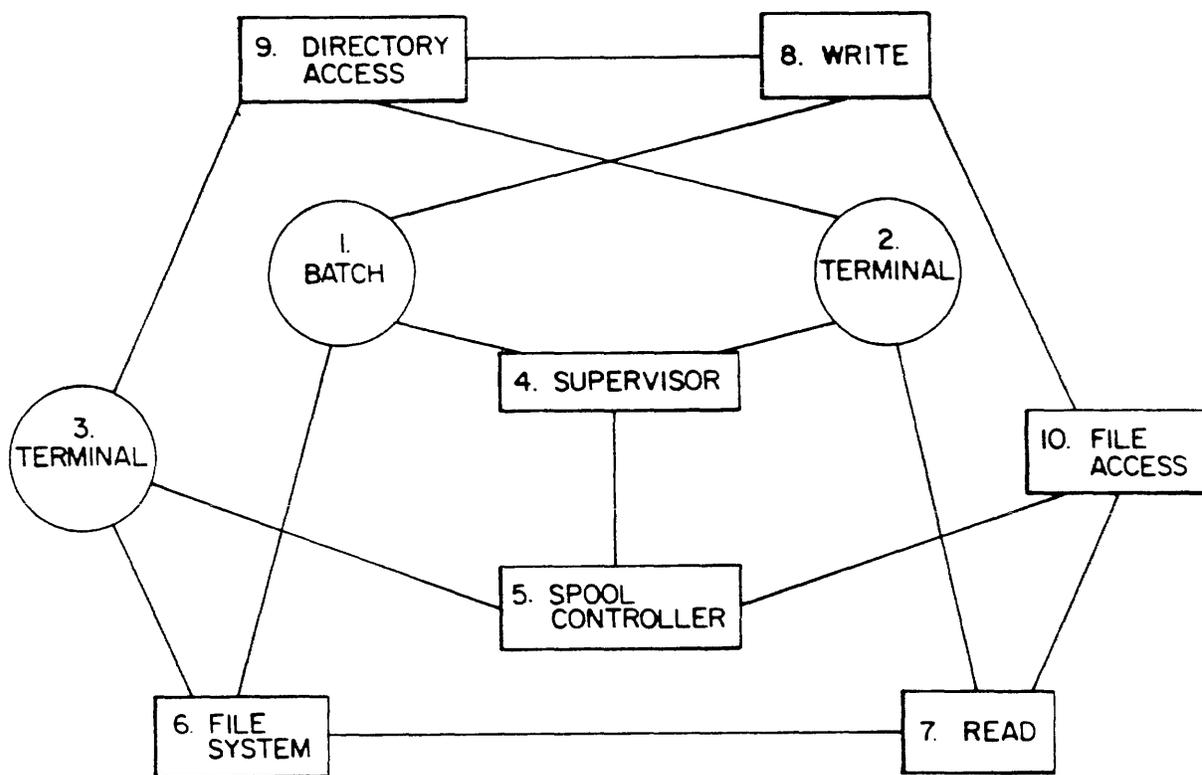
Figure 1—Process system.

abilities obtained by the method explained in the preceding section are also shown for Chain 2 as an illustration. In the network shown, there is a unique shortest path for each pair of nodes. This particular geometry is known as a (3,2) Moore graph,¹ which is also called a Peterson graph. In this case, the mapping is intentionally non-optimal so that appreciable message-passing overhead is introduced. In an optimal assignment, the process network adjacency would be preserved to the greatest extent possible. The mean service time of messages in Classes $\bar{1}$, $\bar{2}$ and $\bar{3}$ is assumed to be equal at each service center and is given by:

$$\frac{1}{\mu_{i\bar{1}}} = \frac{1}{\mu_{i\bar{3}}} = .001\gamma \text{ sec} \quad \text{and} \quad \frac{1}{\mu_{i\bar{2}}} = .002\gamma \text{ sec},$$

where $i=1, 2, \dots, 10$ and γ is a multiplication factor. The statistics obtained on performance measures for the process system is shown in Table I. Table II shows corresponding statistics for the regular system with $\gamma=1$. Comparison of the result shows the effects of message-passing overhead on service center utilizations and mean response time of the system. For example, the productive utilization of Service Center 1 is decreased by ≈ 7 percent, while overall utilization is increased by ≈ 1.5 percent, and the mean response time is increased by ≈ 12 to ≈ 25 percent for each chain due to the message-passing overhead.

Figures 3 and 4 show the effects of changing the mean service time for the messages in class $\bar{1}$, $\bar{2}$ and $\bar{3}$ on service center utilization and mean service time of the system.



$$\frac{1}{\mu_{i\bar{1}}} = \frac{1}{\mu_{i\bar{3}}} = .001\gamma \text{ sec} \quad \frac{1}{\mu_{i\bar{2}}} = .002\gamma \text{ sec} \quad i = 1, 2, \dots, 10$$

TRANSITION PROBABILITIES FOR CHAIN 2

$P_{2242} = 1.0$	$P_{621\bar{2}} = .40$	$P_{923\bar{2}} = 1.0$	$P_{3\bar{2}62} = .50$
$P_{4222} = .05$	$P_{623\bar{2}} = .30$	$P_{1027\bar{2}} = 1.0$	$P_{3\bar{2}92} = .50$
$P_{421\bar{2}} = .95$	$P_{627\bar{2}} = .30$	$P_{1\bar{2}42} = .50$	$P_{7\bar{2}62} = .50$
		$P_{1\bar{2}62} = .50$	$P_{7\bar{2}102} = .50$

Figure 2—Regular system.

TABLE I—Model Statistics of Process System

a) Overall statistics											
Node	1	2	3	4	5	6	7	8	9	10	
ρ_i	.389	.814	.718	.099	.062	.173	.156	.156	.104	.254	
$E(n_i)$.389	.814	.718	.102	.062	.193	.156	.156	.111	.300	
b) Detailed statistics											
Node	1	2	3	4	5	6	7	8	9	10	
	class										
ρ_{i0}	1.	.389	0	0	.078	.062	.049	.156	.156	.029	.058
	2.	0	.814	0	.007	0	.039	0	0	.023	.093
	3.	0	0	.718	.014	0	.085	0	0	.051	.103
$E(n_{i0})$	1.	.389	0	0	.080	.062	.055	.156	.156	.031	.071
	2.	0	.814	0	.007	0	.044	0	0	.025	.109
	3.	0	0	.718	.016	0	.094	0	0	.054	.119
c) Mean response time (sec)											
Node	1	2	3								
	0.031	1.143	0.787								

TABLE II—Model Statistics of Regular System
($\gamma=1$)

a) Overall statistics											
Node	1	2	3	4	5	6	7	8	9	10	
ρ_i	.395	.778	.704	.092	.059	.163	.169	.145	.098	.269	
$E(n_i)$.414	.778	.725	.096	.059	.181	.175	.145	.104	.323	
b) Detailed statistics											
Node	1	2	3	4	5	6	7	8	9	10	
	class										
ρ_{i0}	1.	.362	0	0	.072	.058	.045	.145	.145	.027	.054
	2.	0	.778	0	.006	0	.037	0	0	.022	.089
	3.	0	0	.680	.014	0	.081	0	0	.049	.097
	1.	.007	0	.005	0	0	0	.005	0	0	.029
	2.	.012	0	.009	0	0	0	.009	0	0	0
	3.	.013	0	.010	0	.001	0	.010	0	0	0
$E(n_{i0})$	1.	.372	0	0	.074	.058	.051	.148	.145	.029	.066
	2.	0	.778	0	.007	0	.042	0	0	.024	.107
	3.	0	0	.690	.015	0	.088	0	0	.051	.116
	1.	.007	0	.009	0	0	0	.006	0	0	.035
	2.	.017	0	.015	0	0	0	.010	0	0	0
	3.	.018	0	.010	0	.001	0	.011	0	0	0
c) Mean response time (sec)											
Node	1	2	3								
	0.035	1.433	0.911								

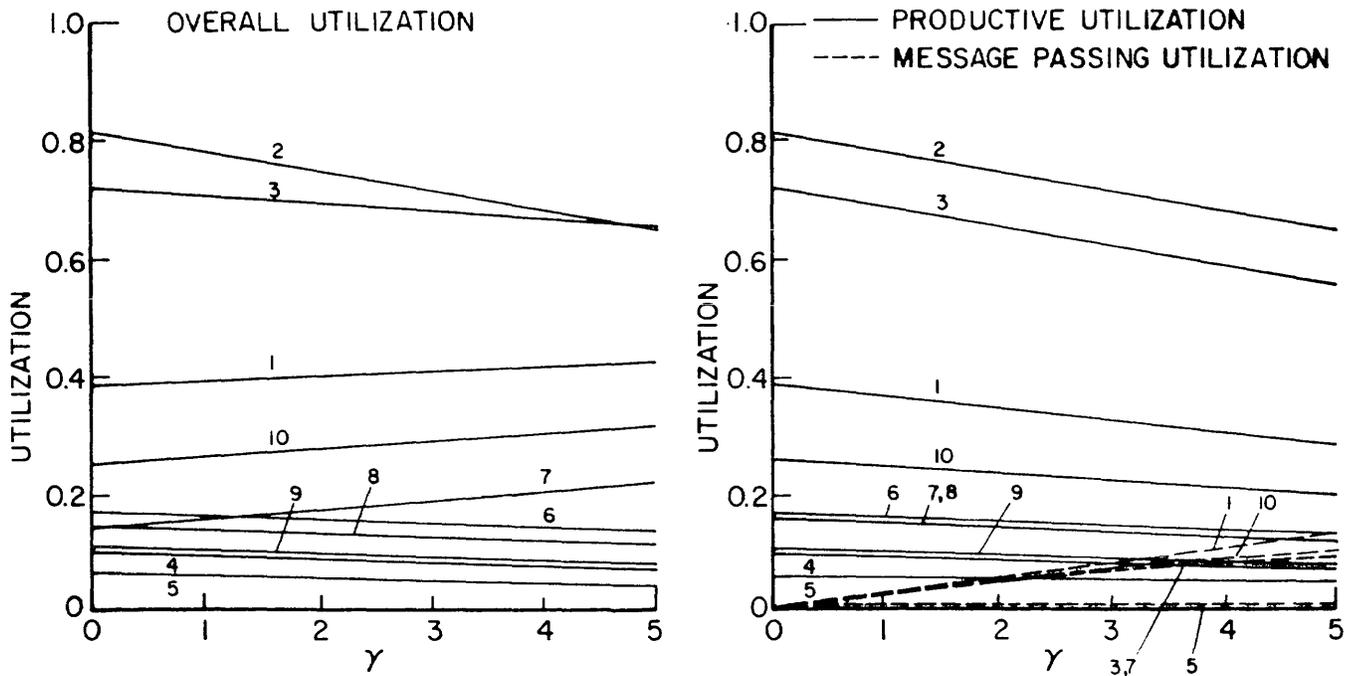


Figure 3—Service center utilizations vs. γ .

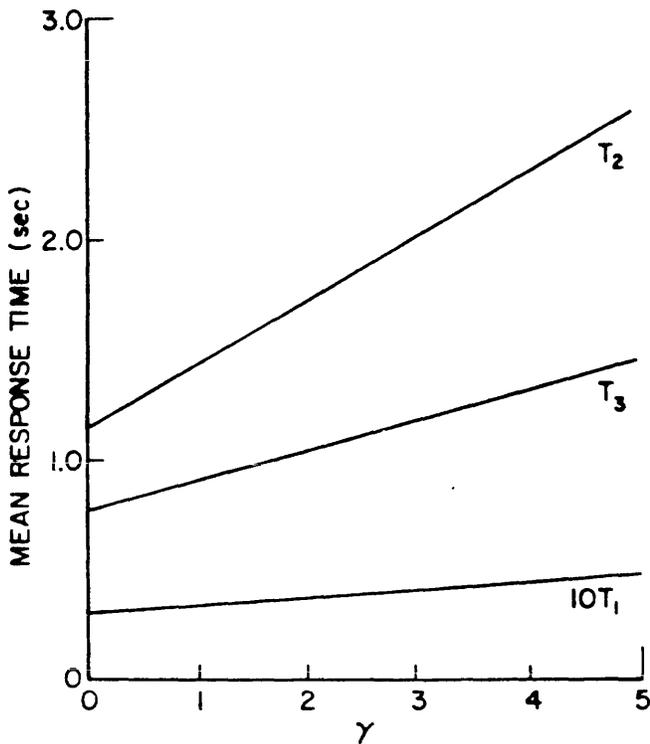


Figure 4—Mean response time vs. γ .

CONCLUSION

A network of queues model for process-structured systems is discussed and analyzed by using solution techniques recently developed. The comparison of process system and regular system shows the effects on system performance measures of the message-passing overhead introduced when the process system is mapped onto the regular system.

Although the issues concerning the mapping of a process system to a regular system and specific geometry for a regular network of low degree are not covered in this paper, they are vital considerations in considering the effects on system performance of the message-passing overhead.

The analysis technique presented could be used to design networks to meet specific performance requirements. Given a particular geometry for the regular system, for instance, the analysis technique can be used as a tool to determine a mapping which gives preference to a certain set of autonomous processes in terms of the mean response time or other performance measures.

REFERENCES

1. Arden, B., and H. Lee, "A Multi-Tree-Structured Network," *Proc. COMPCON 78 Fall*, Sept. 1978, pp. 201-210.
2. Balbo, G., S. C. Bruell and H. D. Schwetman, "Customer Classes and Closed Network Models—A Solution Technique," *Proc. 1977 IFIP Congress*, Aug. 1977, pp. 559-564.
3. Baskett, F., K. Chandy, R. Muntz and F. Palacios, "Open, Closed and

- Mixed Networks of Queues with Different Classes of Customers," *JACM*, Vol. 22, No. 2, Apr. 1975, pp. 248-260.
4. Baskett, F., and R. Muntz, "Queueing Network Models with Different Classes of Customers," *Proc. 6th. Annual IEEE Computer Society Int'l Conference*, Sept. 1972, pp. 205-209.
 5. Buzen, J. P., "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *CACM*, Vol. 16, No. 9, Sept. 1973, pp. 527-531.
 6. Cox, D. R., "A Use of Complex Probabilities in the Theory of Stochastic Processes," *Proc. Cambridge Phil. Society*, Vol. 51, 1955, pp. 313-319.
 7. Little, J., "A Proof of Queueing Formula $L=\lambda W$," *Operations Research*, Vol. 9, 1961, pp. 383-387.
 8. Muntz, R., "Networks of Queues," *Notes for Engineering 226C*, Computer Science Dept., UCLA, Jan. 1973.
 9. Wong, J. W., "Queueing Network Models for Computer Systems," Ph.D Thesis, School of Engineering and Applied Science, UCLA, Oct. 1975.

The City of New York's integrated financial management system—From mandate to working system in 18 months

by SALLY J. RUPERT

*Office of Computer Plans and Controls
New York, New York*

The development and implementation of New York City's Integrated Financial Management System (IFMS) is precedent-setting in both its scope and its purpose. It is an excellent example of the successful implementation of a large-scale computer system, a 20-million-dollar investment using the combined efforts of five consulting firms, into a complex organization, a municipal bureaucracy with 250,000 employees in 109 agencies and with a constantly shifting executive management. This paper describes how it was accomplished in terms of the development philosophy, the building of the system and, finally, the post-implementation environment.

Before beginning the presentation of how IFMS was implemented, a little history lesson seems in order. In 1975, when the fiscal crisis was a full-blown reality, the Federal and State governments issued a mandate that, unless New York City could put its financial reporting house in order and establish sufficient spending controls, no further monies would flow from their coffers to those of the City. It was from this mandate that the project named IFMS sprung. It was to be managed by two co-directors, David Woodbridge representing the then Mayor, Abraham Beame, and Steven Clifford representing the Comptroller, Harrison Goldin. The co-directors were given support and backing and authority which crossed both mayoral- and comptroller-directed functions and agencies. The City also made a substantial budget commitment to the project. The co-directors were charged with the awesome task of establishing standard municipal accounting practice throughout the City, creating entirely new central and line agency financial reporting procedures, establishing City-wide training programs, designing and implementing a fourth-generation computer system which would fully integrate the Budget, Accounting and Payroll activities of the City of New York and creating a line agency to operate, maintain and further develop the new computer system. They had exactly 18 months in which to complete the project.

And so began the creation of IFMS.

THE PROJECT DEVELOPMENT PHILOSOPHY

Large-scale projects setting a new direction and involving many designers and decision-makers tend to get bogged

down in bureaucratic red tape and development indecision. In order to alleviate this major stumbling block to successful implementation, the co-directors of IFMS developed a philosophy for getting the job done rapidly and correctly, then implemented it and stuck with it throughout the project life. That philosophy consisted of the following three premises.

The first premise said that today's business world, in both the private and the public sector, is changing with such rapidity that, if a project is not completed within a relatively short time span, it is outdated before it is implemented. That includes not only the method of doing business, but also the data processing tools to be used and the decision-makers who would use them. Therefore, no project should have a life cycle of more than 24 months. It is not worth doing if it will take longer, no matter what the size of the effort. The longer the project takes the higher the risk of obsolescence as it nears completion. And so, IFMS was planned for 18 months.

A second premise was that different methods of management and control should be used as a project progresses in its life. During the developmental portion of the project the "czar" approach was used; that is, the co-directors had final say in all decisions needed to be made by the City. This alleviated the bottleneck of soliciting decisions from various managers, sometimes with digressent goals. As the project drew close to implementation the management control expanded to a task force including members from all developmental areas, but, with the co-directors still holding final vote and veto power. Then, after the implementation, the project was placed under standard project control for maintenance. In terms of how management knew what they were controlling, here also a methodology was applied that was suited to the stage in development. Progress was monitored in two ways, deliverables and status reporting. The latter, status reporting, forced the interdependencies among all participants in the project to be addressed and honored.

The third premise said that the project could only succeed through the intelligent use of people. High-level technical people with functional and hardware/software computer specialties were dedicated to the General Design phase, at which time the project strategy was set. That strategy included not only the functional aspects of the design but also the high-order technical concept. Throughout the project,

resources were allocated to get a specific portion of the job done. No matter what may have occurred in one area of the project, resources allocated to other portions were left in place. That kept the project moving while outside resources were brought in to shore up the limping portion.

THE BUILDING OF IFMS

IFMS was constructed and implemented through the efforts of over 200 people, including not only City personnel but also five consulting firms working under the direction of the IFMS co-directors, David Woodbridge and Steven Clifford. Each consultant performed a specific function during the development and all were brought together in a single effort during the final testing and implementation. American Management Systems designed and programmed the Budget, Encumbrance Control and Accounting subsystems. Bradford National Corporation designed and programmed the Payroll subsystem. Ernst and Ernst were the developers of line agency procedures to be used in accordance with the new Charter requirements implemented through IFMS. Touche Ross set the new accounting principals and developed central agency organizations and procedures for operation under the new Budgeting, Accounting and Payroll methods. And finally, the Urban Academy developed operations manuals explaining how to complete input forms and forms flows, then trained City personnel in the use of the new system. Training for IFMS was and is still being done by the Urban Academy and consists of budget, accounting and payroll manual procedures, budget and accounting management, data analysis and on-line inquiry system use.

As work on IFMS began, project personnel worked concurrently to develop both the computerized portion of the system as well as all the attendant procedures and processes. Design efforts and all other efforts were coordinated, to ensure that all development was consistent, through the co-directors and their staffs and, as implementation drew near, a task force composed of members of each consulting firm as well as members of New York City's Office of Management and Budget, Comptroller's Office and Department of Personnel. It was chaired by the IFMS co-directors. The task force was embodied with decision-making powers which were to be exercised whenever design or procedural criteria was to be determined. Without the co-directors and the task force with authority concept, IFMS could never have been implemented in those 18 months. In an organization as complex and bureaucratic as New York's municipal government, decentralized development and decision-making would have made the decision-making process impossible. The single goal of the co-directors was to implement IFMS. The common goal of the task force was implement IFMS.

Let us now turn our attention to the building of each component of the system. The systems design for the computerized budget and accounting portion of IFMS took ten months with a team of 10-40 analysts working, at times, around the clock. The general design document totalled 32

volumes, covering only the budget and accounting subsystem in detail and giving a specification only for the Payroll subsystem. A decision was made to phase Payroll in at a later date because the risk management factor was far greater than the other two subsystems. The Payroll subsystem design consisted of 17 volumes and was produced by seven analysts. Based on this design, which was approved by the co-directors and their staffs, programming began. Throughout the programming stage of development, approximately 30 analysts and programmers contributed to the effort. Structured programming techniques were used to build the thousands of modules which comprise the system. Each program was walked through both at the unit level then again at the integration level.

Large-scale use of macros and common modules added speed to the implementation and centralized all common functions, thereby making maintenance easier. Master Tables were developed for all variable non-data base information. A complete subsystem was built to support the maintenance of these tables and common entry routines were built to access them from application programs.

Even before applications programs and technical support software were being created, another technical team was about the task of selecting the hardware configuration upon which the system would operate. The initial hardware configuration was in place in time for program unit testing. That configuration consisted of fourth-generation IBM hardware and a terminal network to support remote testing. Two CPUs were devoted to budget and accounting development while the Payroll subsystem was built on one CPU.

The software configuration consisted of IBMs—IMS used to manage IFMS's data bases, VS1 as an operating system, Data Analyzer to support report generation, and ROSCOE to support on-line testing.

IMS was chosen because it provided the speed in development needed to support the extremely tight project schedules. VS1 gave IFMS a stable proven operating system.

While system design and programming efforts were going forward, two other teams were about the task of defining central and line agency procedures and organizations. Both procedures and organizations were built to interact with the IFMS computer system. At this point in the development, communication flowed across teams from the technicians to the procedural analysts and back.

Once procedures were defined and inputs and outputs had been designed, work commenced on the development of procedures manuals and management manuals. So, another team, the manual procedures analysts, was added to the horizontal communications process.

Also, during development a team of consultants defined the agency which would be responsible for the operation of IFMS. This definition included the organization and its basic operating procedures. Approximately seven months before implementation, this agency, the Financial Information Services Agency (FISA), was formed and its managers selected. These managers, with the aid of the consultants, then commenced to hire and have trained both the technical and the operations personnel that would man the agency.

As IFMS was nearing the final stages of program testing, the task force formed an implementation group to coordinate the systems test and to make last-minute decisions.

Six weeks before July 1, 1977, the officially committed implementation day, a final integrated test began. The test consisted of two parts, a structured test with pre-determined test cases, from which agency personnel coded input, and an unstructured test in which situations were defined and central and line agency personnel interpreted the new procedures and submitted input accordingly. During the unstructured test, members of the task force were invited to submit their own test situations and input. The test was designed to exercise all newly developed procedures, manuals, organizations and of course, the computer system. FISA, the IFMS operating agency, acted in its full capacity during the test, thereby checking out its operating procedures, even including such things as courier routes and pickup and delivery points. A test coordinator was named who became the central focus for problem reports, which were completed by personnel finding errors in any of the processes, procedures, or the computer output. The test coordinator monitored problem resolution and reported to the task force at its weekly meetings. In this final step before implementation, line and central agency personnel became directly involved with the system so that on day one they were prepared to face the system, thus alleviating some of the trauma of an entirely new operating process.

As stated earlier in this paper, the Payroll subsystem implementation lagged the balance of IFMS implementation by about one year. And so, the system which went live on July 1, 1977 contained an interim computerized Payroll interface. When the Payroll subsystem began implementation in March 1978, it followed much the same process as the rest of IFMS. There was one major difference, however, in the implementation method; Budget and Accounting went live with all City agencies at once while Payroll was implemented on a phased basis, a few Agencies at a time.

POST-IMPLEMENTATION—THE OPERATIONAL ENVIRONMENT

As soon as implementation of the basic system was completed, IFMS moved to an operational status. FISA, the agency created to operate IFMS, took control gradually over the first year of operation. First, the day-to-day document processing, then job running, then job integration and production linking and library functions moved to City control. Concurrently with the operations shift, the FISA programming staff was walking through and accepting responsibility for the applications programs. Finally, the data bases and technical software were turned over to the FISA data base and systems programming area. The transition was gradual and the consultants remained in place until City personnel felt comfortable with their ability to maintain and operate this very large and complex computer system.

In order to support this system, FISA has employed over 200 data entry, operations, systems, programming and an-

alytical personnel. With the planned growth for IFMS to incorporate other Funds and other financial applications that number will increase again before another year passes.

IFMS has indeed accomplished its primary objectives, to provide the City of New York with single-source financial reporting, to provide an auditable record of every transaction processed by it, to incorporate Charter revisions into the City's financial processes, to increase the credibility of its financial information.

Agency personnel learned very quickly to complete input forms with very low error rates. Forms continued to flow and financial functions improved. The City's personnel were indeed capable of being completely retrained in a short time frame, a task thought insurmountable by many of the system's early skeptics.

The system has expanded steadily since its implementation. New development demands have been high. The communications network has grown substantially. Inquiry processing alone has multiplied to at least ten times its start volume with a weekly inquiry rate of 60,000 transactions. Two million-plus documents will flow through IFMS operations area this year. Requests for new information from the system outstrip available resources. If use of a system is a determinant of the success of the system, then IFMS was a roaring success.

To support the rapid growth of IFMS, FISA's director has had to reconfigure the hardware environment to higher-powered CPUs, substantially more disk space, and higher-speed, added capability, printing hardware. The network has grown to over 400 on-line terminals. FISA has also installed a large tape library controlled by Tape Management System to facilitate its large volume of tape-stored data and its export data subsystem which interfaces with other City computer systems. Disk storage expands again and again to support the data availability requirements of the City. Central agencies have demanded on-line availability of historical information about every document processed during a fiscal year and even beyond in many cases. FISA's hardware/software specialists continually review new products in the marketplace in order to keep FISA and consequently the City of New York at the forefront in latest generation equipment.

AND IN CONCLUSION. . .

In conclusion I would like to state again those premises which drove the method of development and implementation of IFMS and those elements of sponsor support which, when combined, made the Integrated Financial Management System of The City of New York a successful large scale computer system running in a complex organization.

Premises applied

- The world is changing so rapidly that projects not completed in a relatively short time span are obsolete before

implementation. Therefore, no project should go over two years in development.

- Different methods of management should be used at different points in the project progress, from "czar" in the beginning to task force in the end.
- Projects can only succeed through the intelligent use of people.

Action taken

- Project functional and technical concept were set early in the General design.
- Hardware configuration and operations support func-

tions were established during the General design phase so that the operating environment was ready when testing commenced.

- Both developers and sponsors took part in design evaluation.
- Both developers and users performed final testing.
- Sponsors provided almost unlimited financial support and waived decision-making to IFMS co-directors.

Obviously the Federal and State governments' mandates added additional impetus to getting the job done, but, in the end, it was the method, not the mandate, that put IFMS on the air.

Recurrent dilemmas of computer use in complex organizations

by ROB KLING and WALT SCACCHI

University of California, Irvine
Irvine, California

COMPUTER SYSTEMS AS TOOLS

Computer technology is usually spoken of as a problem-solving tool,^{33,36} a helpful device used to ease the burdens and expand the flexibility of information processing. In this narrow sense, computer technologies have in fact increased the capabilities of people and organizations to carry out complex calculations, manipulate large sets of data and access data from geographically remote locations.

These capabilities generate a corresponding and sometimes unexpected set of problems for many computer users. People who use computer systems for a variety of daily tasks must adjust to changes in computer systems, vie for adequate priority for their computing jobs, develop backup procedures when automated systems fail and periodically search for skilled programming staff. As a result, the very technology which was supposed to be an unobtrusive aid and time-saver can become very attention-demanding and a source of continual low-level conflicts. The "problem solving instrument" is capable of generating its own special problems.

Easing problems of computer use has been a traditional concern of computer scientists and many solutions have been suggested and tested. Most of these solutions, however, have assumed that computing is a fairly straightforward dialogue between a hypothetical user and a machine. Focus may rest on one party or another. Thus, hardware-based solutions which focus on expanding the flexibility and reliability of the machines emphasize components such as new peripheral devices, distributed computing, microprocessing, operating systems protection schemes or computer graphics. Likewise, software-based solutions which focus on easing the cognitive burdens of the user include new programming languages, data base manipulators, or more "natural" interfaces. Lastly, managerial solutions emphasize the organizational arrangements within which computer based-services are developed and provided. Involving users in systems design, for example, is often recommended to ensure that system specifications are appropriately developed.^{16,21,27}

Analysts can suggest sensible solutions to difficulties that computer users face in dealing with computing by segmenting the world into manageable chunks. Named topics such

as "ease of access," "software reliability" and "resource allocation" are well known labels for identified problems. This is the traditional "divide and conquer" strategy of the engineering disciplines and helps make complex production problems *manageable*. Solutions to these identified problems, however, reduce only a selected portion of the burdens faced by computer users. As computing use grows in complexity, and the number of identified "problems" and "effective solutions" increases, the likelihood that they can all be well handled by any group of service providers or instrumental users diminishes.

The routine use of computer-based services increasingly brings people in computer-using organizations into a complex set of dealings with the technology, its providers and other actors. These social relationships are both a source of service for computer users and a locus of difficulty. Factoring these relations into independent "problem areas," each with its own technical and managerial strategies for solutions, doesn't help a user comprehend the way in which computing is often *problematic*. First, no profession or service provider is usually capable of meeting all the needs and wants of its clientele. Secondly, the problematic aspects of computing arrangements often *interact*. Problems are best factored when their components interact weakly. In the case of computing, choices of which technology to use, who to staff it with, how to maintain it, and how to pay for it are often highly coupled. These are clearly social decisions as much as they are technical decisions. The social aspects of computer use are commonplace, but nevertheless they are *poorly* understood.

Our studies of computer use in a variety of settings¹⁶⁻²⁴ indicate that many users often have recurrent problems in obtaining computer services smoothly. Management analysts are quick to suggest that when users have difficulties, there must be a clearly identifiable management problem which needs a systematic solution.^{7,9} In most of the organizations we have studied, managers and staff have developed sensible strategies for dealing with many aspects of computing; but problems still recur. It is easy to blame recurrent problems on "poor managers," "stupid users" and "inadequate technology." Such sentiments are too loaded with blame and faith in simple solutions (i.e., "education," "more core") and too short on analysis to be uniformly

convincing. Simply identifying new "problems" and suggesting new, independent "solutions" may even add to the burdens of attention faced by computer users. We suggest a new approach to help understand why computer use is often problematic.

We find it helpful to expand the traditional view of computing from that of a "tool" to that of a "package." The tool metaphor, which is very appropriate for simple, individually controllable devices, such as hammers and pocket calculators, suggests that the item denoted may be used with few attendant problems. Of course, some tools may be more graceful, effective and reliable than others; but in most cases one can safely focus on the device to understand its use and operation.

In contrast, the package metaphor describes a technology which is something more than the physical device. In the case of computing, the package includes not only hardware and software facilities, but also a diverse set of skills, organizational units to supply and maintain computer-based services and data and sets of beliefs about what computing is good for and how it may be used efficaciously. Many of the difficulties that users face in exploiting computer-based systems lie in the way in which the technology is embedded in a complex set of social relationships.

Not only are most computer systems shared with other users, but programs and data are provided through several different social networks which often entail contact with different social groups.^{20,26} This complex social setting in which computing is embedded makes computing a social object, and the use of computer-based services a social act.

The primary thrust of this paper is to identify the recurrent aspects of the social world of computer users which are problematic for people who use computing to serve other ends. We have expanded our conception of computing as a potentially problematic "tool" to computing as a social object. We will now explore some more specific consequences that this expansion reveals. We would caution that while we list a set of issues which are problematic for computer users and computer specialists, and advance some hypotheses as to their relative and absolute costs and importance, we intend this discussion to be an introduction to the bundle of issues which warrant further investigation, articulation and conceptualization.

THE SOCIAL CHARACTER OF COMPUTING

Our analyses of computer use are based upon several empirical observations and theoretical claims:

1. Many people (*instrumental users*) who use computing hope it will help them be more effective in their work. The substance of that work may have little connection with computing; computer use is a means to further some other end.
2. In many important situations in which computing is used there may be many different people who are interested in utilizing the same computer-based data or reports. These people can have different understand-

ings as to the capabilities of computing and indicate different interests in the uses of computer-based analyses.

3. Much modern computing and most important automated information systems are supported in settings in which several specialized groups provide the requisite computing services and data.^{6,14,19,21,26} Automated information systems serve managers and organizational people who have little time or skill to carry out the full range of computational tasks to support their data use. Even skilled programmers rarely design, implement, test and maintain all the software they use while carrying out their work.
4. Users of computer-based services frequently report an array of difficulties in computer use.²² Complaints usually focus upon aspects of computer use which are byproducts of the social arrangements in which computer-based systems are conceptualized, developed, provided and maintained. These problems rarely focus upon computing hardware, except when users believe there is too little of it or when some party allegedly chose less suitable equipment than might be available in the market.
5. Computer-based services and information processing tasks are organized in vast array of distinctly different arrangements within and between organizations. Smooth computing use often entails the cooperation of distinct organizational groups and interests.^{19,26}
6. We view organizations as patterned arenas for conflicting and cooperating interests.^{8,35} We note there are often conflicts between the interests of participants who identify primarily with computing as their profession or career interest, and those who identify with some other social world in which computer use is primarily an instrumentality.²³ These extremes are, of course, simplified since many participants align themselves as specialists who mix computing and other substantive interests. But the grounds for conflict of interests remain similar.

These observations encourage us to view much of computer use as a complex social phenomenon in which hardware and software plays an essential, but partial role. In fact, computer use can be expected to be particularly problematic as the milieu in which it is embedded increases in *social complexity*.

ISSUES IN INSTRUMENTAL COMPUTER USE

Computing services are produced and consumed in *work settings* in which the participants take on specialized roles. The demands that instrumental users and computer specialists make upon each other and of the technology hinge, in part, on their *understandings* of the appropriate *role, capabilities* and *limitations of computing*. Typically, relations between service providers and their clients is problematic. Few service providers can meet all the wants of their clients or of the organizational participants to whom they are ac-

countable. Few clients have sufficient skill and interest to deal with technically skilled service providers on their own terms.

Application development *changes* procedures and processes for users at various intervals which may be either relatively benign or disruptive. When instrumental users rely upon automated data systems, *ensuring* that the *data* provided is of high *quality* (i.e., accurate and timely) is particularly sensitive. Part of the social interaction around computing involves establishing and maintaining *control* over the various computing resources within the organization.

Both specialists and users depend on the current state of *software development practices* to help construct reliable programs which are easy to operate and maintain. Similarly, computing creates special demands for the *time* and *attention* of users. The social aspects of computer work and computer use play a large role in shaping each computing milieu, as does the particular technology in use.

In Table I, we list the array of representative issues which we have clustered under the categories emphasized in the preceding three paragraphs. This set of issues has been selected because they appear problematic to instrumental users in studies we conducted in a variety of settings.¹⁶⁻²⁴ Some of the specific issues indicated in Table I are important to computer users in many settings; others occur infrequently. Most of these issues should be easily recognizable since they are common in settings where there is extensive computer use. These issues are briefly examined in Appendix A to indicate how each one is a byproduct of the social elements of the computing package and how it can effect the quality of computer-based services.

These issues do not exhaust those raised by the social nature of computing. But they do represent those social aspects of computing which strongly influence the patterns of computer use adopted by instrumental users. The relative importance of any of these issues is also dependent on the organizational setting where computing occurs.

THE ORGANIZATIONAL CONTEXT OF COMPUTER USE

The actual difficulties experienced in using computing depend upon the *interplay* between both technical and organizational arrangements. Consider, for example, the different impacts of data base management systems (DBMS) on the time to produce a program for a user in scientific and commercial settings.

Computer specialists may assume that a scientist utilizing a DBMS will either carry out his own programming or employ a skilled research assistant who is under his supervision. This is a result of the work organization of scientific laboratories in which each research team has dedicated research assistants to help carry out a variety of laboratory chores including data collection, reduction and analysis. If the scientist desires to change schedules or priorities in his use of the DBMS, he normally faces no bottlenecks in the process except the limitation on his own or assistant's time. Since he can regulate these alterations of priority, he is at most buffered by one queue from access to programming.

A different situation faces the instrumental computer user in a commercial firm. In commercial firms, it is rare for staff to have their own programming assistants. Programmers are usually centralized in a pool, even in user departments and scheduled through a supervisor. The commercial user may thus be further buffered from the access to computing. He may have to negotiate with a supervisor, a special committee or a review board to achieve changes in schedules or priorities in dealing with a DBMS. Each of these parties has a separate queue of requests and demands with their attendant delays. Each such queue creates additional delays for the commercial user in gaining access to programming assistance. In practice, a person may wait much longer to get on the queue of a programmer than it takes to do the work.

Even if a DBMS reduces the time required for a *programmer* to write a given program, the time it takes for *users* to

TABLE I.—Common Issues in Instrumental Computer Use

<p><i>The Work Setting of Computer Use</i></p> <ol style="list-style-type: none"> 1. The concepts users and computer specialists have of their own work and the role of computing in it. 2. The mutual perceptions of computer specialists and users. 3. Differing responsibilities among computer specialists. 4. Doing a "good job" and being rewarded for it. 5. Maintaining career mobility. <p><i>Understanding the Role and Capabilities of Computing</i></p> <ol style="list-style-type: none"> 1. Learning about computing—what computers are good for, how their particular machine might be used, etc. 2. Getting a computational task successfully completed. 3. Dealing with computing/systems jargon. 4. Getting adequate documentation for computer-based systems. <p><i>Changes in Computing Arrangements</i></p> <ol style="list-style-type: none"> 1. Loci of change. 2. Scope and rate of change. 3. Formalizing change procedures. 	<p><i>Data Quality</i></p> <ol style="list-style-type: none"> 1. Collecting input data. 2. Ensuring the correctness of processed data and analyses. <p><i>Control Over Computing</i></p> <ol style="list-style-type: none"> 1. Control over the technology. 2. Access to and control over expertise. 3. Controlling the kinds of demands made by users. 4. The "values" sought after by those individuals and organizations which promote applications development. 5. Developing and maintaining political support within the organization. <p><i>Software Development Practices</i></p> <ol style="list-style-type: none"> 1. Programming and design practices. 2. Program testing. 3. Software maintenance. 4. Software documentation.
<p><i>Attention</i></p> <ol style="list-style-type: none"> 1. The kinds of attention demanded by computing. 2. The precision and detail demanded by computing. 	

get a given computing task completed depends upon organizational arrangements. This example illustrates the way in which the social setting of computer use may influence users more than the technology in use.

STRATEGIES AND RESOURCES

The issues identified in this paper are representative of those that arise for many instrumental users and computer specialists in their daily encounters with computing. Improving the grace or ease with which computing is used hinges on coming to grips with these issues. This requires recognizing computing as a social object as much as it depends upon developing new software and new hardware. In addition, a major impact on groups using computing is the increased attention to information processing—its management and conflicts—that negotiating these issues demands. People's time, skills and organizational resources are involved in attending to these negotiations. The negotiation costs, in time, money, skills, forsaken opportunities, and sentiment borne by instrumental users may become a substantial fraction (if not the largest) of the cost of a system during its life cycle.

Computer specialists have been sensitive to some of the difficulties of computer use raised here: after all, they are commonplace. And computer scientists have been particularly adept at providing technical solutions for some of these difficulties. Generally, those technologies that diminish the "social size" of the computing package by decoupling instrumental users from some of the groups upon which they depend may alleviate some of the burdens of computing. Thus, acquiring a minicomputer may insulate a group of instrumental users from demands for machine resources made by other groups. However, it doesn't diminish the difficulties of managing data and may even increase the difficulties instrumental users face in managing skilled staff.

"Turnkey" installation of applications and hardware may reduce the instability of computing development. However, other technical improvements are more problematic from the perspective developed here. While advocates of data base management systems have stated objectives of making the development of *ad hoc* analyses easier for instrumental users,^{28,29} the social complexity of the computing milieu should increase since new specialists (such as data base administrators) are often employed. It is empirically open whether the overall environment of data base management is easier or more difficult for instrumental users to negotiate. Similarly, software engineers often propose that development aids such as test data generators would help insure the correctness of programs. From our perspective, a test data generator, however carefully crafted, is another package subject to the recurrent social histories of computing packages.

Management and social analysts who identify difficulties of computing in the social milieu often propose organizational reforms such as new pricing schemes or design disciplines that emphasize user involvement.^{16,21,27} Such strategies often resolve particular dilemmas of computer use in

a specific setting, but they do not deal directly with the large, diffuse social elements that pervade the computing package.

Our own field work in several large private firms and research laboratories indicates that effective strategies often entail large commitments of organizational resources. Chains of liaisons between instrumental users and computing service providers facilitate multiple lines of communication and smooth tensions between conflicting groups. Regular meetings and redundant forms of communication ease coordination and minimize the likelihood of major slippages between the service providers and their clients.

Technology-based strategies often require large resource commitments. We have seen, for example, one engineering firm which uses a large-scale computer for production applications, and a similarly large computer devoted solely to software development so routine operations are unlikely to be interrupted. The point is that mitigating strategies which add more machine or staffing resources can add to the existing complexity of a computing setting thereby potentially displacing one set of problems with another set of problems.

In summary, technology-based strategies often miss major portions of the computing package that include important social relations and contingencies. In addition, the best mix of technology-based strategies and socially-oriented strategies for graceful computing can consume large resources, time and money. Since most computer using groups have limited resources, one should expect "budget strategies" to be the rule rather than the exception. Given constrained resources, some interests will be better served than others and some parties should be expected to face computing dilemmas routinely. The empirical prediction would be that any problem (such as data quality, response time, appropriate consulting or adequate documentation) should be troublesome for some minority of instrumental users in even the best managed setting of computer use.²²

CONCLUSION

Much of our account has focused upon the problems attendant in routine computer use. This is not because we believe that computing is a wholly troublesome technology. On the contrary, we believe that computer use often increases the information processing effectiveness and eases the work of many instrumental users.²² However, these gains often do not come gracefully or easily. Computer use is most troublesome when the necessary social resources (such as technical expertise, demands for time and attention, staff sentiment and control over computing services) are slighted or ignored when new computing arrangements are to be provided.

From the analysis of the recurrent dilemmas of computer use presented in this paper, we draw the following conclusions:

1. The computing tool metaphor displaces attention from the social dilemmas of computing by tacitly identifying advances in computing with advances in the technical

sophistication of the equipment used. Moreover, many of the attendant difficulties in computing are *not well predicted or understood* by employing the tool metaphor.

2. Problems of computing vary with the particular computing technology in use and the organizational arrangements through which computing services are produced and used. Hardware reliability is usually more salient in on-line systems than with batch systems. Allocation of machines and staff are typically most contentious when control over each resource is centralized. Staff highly sophisticated in computing may provide the best technical assistance, but they are also the most difficult to interest in routine applications.
3. Many of the problems experienced by computer users develop from their relationships within the "computing world." The computing world is highly differentiated into specialty interests²⁴ and organized to routinize the movement of innovations from producers, through service providers to instrumental users.²³ Instrumental users face markedly more complex issues when they split their computing activities across equipment supplied from different "vendor worlds." Also, they often have little control over the pace at which small enhancements or alterations are made in supporting software supplied by groups outside their organization.
4. As technical advances in computer hardware and software simplify the technical problems of computer use faced by users, the social problems of computer use will become relatively dominant. Each of these problems has associated costs. These costs are poorly understood and have yet to appear in the figures cited for total systems costs.^{3,13,25,38}
5. The social elements of computing are typically underestimated in proposals for new computing arrangements. Social resources such as time, attention, skills, information and inclination can be costly to acquire, utilize and maintain, but discounting their role in computing results in displaced organizational costs. For example, expert consultants and good system training aids are costly to provide. But when instrumental users cannot obtain needed assistance, they recurrently find computing use to be troublesome and uncertain.
6. The package view implies that successful computer use depends on the organizational distribution of social *and* technological resources and how they are allocated or acquired. Successful computer-using organizations balance their technological investments with explicit investments in the social elements of the computing package.
7. Currently, there are no simple or uniform solutions. Alternative computing arrangements which are proposed to solve certain individual problems can exacerbate or manifest others if the social character of computing is disregarded.

Computing is a problematic technology for many instrumental users, in part, because it raises *so many social issues* which continually demand attention. We suggest that instru-

mental users and specialists alike use the list of issues presented in Table I as a diagnostic guide for assessing the impact of existing or proposed computing arrangements. As a checklist, Table I can help an analyst decide which activities a new "solution" may alter, and which it may leave untouched. Table I can also help an analyst make explicit the rich set of social features which characterize the social milieu of complex organizations.

ACKNOWLEDGMENT

Elihu Gerson, Sharon Price, and Harold Sackman provided helpful comments on an earlier draft of this paper. Philip Crabtree helped develop some of the ideas presented here. This work was supported by NSF Grant MCS77-20831.

APPENDIX A—COMMON ISSUES IN INSTRUMENTAL COMPUTER USE

The work setting of computer use

The concepts users and computer specialists have of their own work and the role of computing in it. Specialists and users have different concepts of how central computing is, should and could be to the successful performance of their jobs. To specialists, computing can be everything. Instrumental users, however, often view computers simply as a means to achieve some other ends. This difference of focus has substantial repercussions for the amount of effort people of each orientation are willing to spend learning and adapting to new computer system developments.

The mutual perceptions of computer specialists and users. Specialists can influence the involvement of users in the computing process. Shared perceptions may be important to the specialist in determining how users should be educated or to what extent users should be involved in the design, implementation and maintenance of particular systems.

Differing responsibilities among computer specialists. As an organizational unit grows and expands, the jobs within it often become more narrowly defined and specialized.² The resulting division of responsibilities and skills may increase the difficulties faced by clients of the unit when they seek a service which requires several specialists. For example, an instrumental user may find that to change an inquiry program, he or she must coordinate efforts with those of a programmer, a systems analyst, a data base manager and a teleprocessing specialist. Increasing the technical sophistication of a system often leads users to interactions with more specialists.

Doing a "good job" and being rewarded for it. People often differ on which aspects of a job are important for satisfactory performance. Some programmers emphasize satisfying user demands, while other programmers emphasize elegant code.

Despite these individual interpretations of what constitutes doing a "good job," the organizational structure may impose a reward system on specialists which emphasizes

different activities.¹⁶ The rewards may be for meeting schedules, for the number of coding lines produced or for getting to work on time. Whatever reward system exists in an organization for computing specialists, it may conflict with what specialists perceive to be important measures of job performance.²¹

Maintaining career mobility. Specialists appear no different than any other employees in being concerned about job security and career development. Specialists may feel that a strong position in the marketplace depends on one's experience with the latest technological innovations. Consequently, specialists may influence their organization to continually acquire state-of-the-art hardware and software packages.²³

Understanding the capabilities of computing

Learning about computing—What computers are good for, how their particular machine might be used, etc. Beliefs about the appropriate role and capabilities of computing vary considerably. Those who work closely with the technology often view computing as a special-purpose device which is best suited for applications something like their own. Thus accountants often view computers as "accounting engines," while urban planners may view them as statistical calculators.

Coupled with beliefs about appropriate tasks for automation are beliefs about the ease of applying computing. Computer specialists often view the technology as speedy and convenient. However, programmers (like planners, designers, managers and other professionals), can underestimate the time required to develop and implement new projects.

Getting a computational task successfully completed. When a problem can be solved with the existing computing system, users may find themselves facing a procrustean software system.³³ Rigid system designs add to the complexity which users must overcome to compute a solution to their problem. In theory, computing may be both technically and organizationally complex. In fact, it is also complicated.* Most software packages, however simple or complex, usually have idiosyncratic conventions** which arise from problems in implementation, compatibility with odd features of related systems, or simply through "poor" design. Nevertheless, an instrumental user must master and remember these conventions to utilize a software system.

In addition, the elapsed time to complete a computational task, from the point of view of an instrumental user, begins

when the task is conceived and ends when the computed job is translated into a usable form. This time frame is larger than that of the computer specialist who counts from the time that a task is well specified until a product is delivered to the user. And it is still longer than the time to complete a computational job as viewed by the computer operators. This usually equals the time to complete a job once it is being executed by a digital computer. Despite these obvious observations, the time to complete computing tasks are usually conceptualized in time frames closer to those of machine execution than to those of instrumental users.

Dealing with computing/systems jargon. Specialized languages enable work groups to communicate about their work compactly and to maintain a definition of "insiders" and "outsiders." "Jargon" is thus an inevitable element of worklife in specialized occupations. Smooth expert-client relations in computing milieus require that either one participant know the technical vocabularies and rationales of both computing and the occupational world to which it is applied, or that one of the parties be skilled at developing communicative bridges between computing and another world of discourse. If both parties possess either skill, so much the better.

Purposive use of jargon enables an actor to structure situations to her or his advantage by "snowing" the other parties in an encounter. When the legitimacy or competence of a computer specialist is brought into question, confident explanations couched in complex technical rationalities are difficult for most people to penetrate. Purposive use of jargon helps a specialist save face and protect his autonomy.

Getting adequate documentation for computer-based systems. Both computer users and specialists rely upon a variety of documents to learn the capabilities of and precise incantations for using particular system features. Different users of a given system may desire either a tutorial manual or a reference manual, although both rarely co-exist for any computer-based system. In fact, one dominant feature of computer settings is the extent to which participants depend upon clear and accurate documents to select, use and maintain computer systems, and the relative paucity of appropriate high-quality documents. Documents, like any other computing product, are produced within a social order of computer specialists, service providers, and clients. The difficulties of documentation are more those of the priorities within the computing world rather than the technical difficulties of writing.³⁰

Changing computing arrangements

Loci of change. Most computer applications evolve gradually. However, changing an application often results in altering the routine procedures for many different people who use it. New features are usually negotiated between some mix of instrumental users, computer specialists, important actors in the computer using organization, computer vendors, and outside consultants.

Computing personnel often have more requests, demands and self-initiated ideas for changes than their staffing per-

* Complexity refers to substantive logic-mathematical interrelations and difficulties: complications can arise in almost any arrangement of facts, concepts and thoughts. Complication is an undesirable characteristic of any construct; complexity may be an inherent feature.⁶

** For example, program runs may begin with an incantation such as //JOB=. Variables may be restricted to six alphanumeric characters and begin with a letter. Or once a file is processed, it may not be reprocessed until a special routine is executed. Most computer users learn to use the technology despite dozens of similarly idiosyncratic conventions. However, they add undue complication to a complex technology.

mits. Thus, they can usually select certain alterations from the larger set of requested or required alterations. While many changes in computer applications or their supporting systems are requested or "needed" by some users, certain users appear better served than others. In addition, most users must expend personal and organizational resources to ensure that changes which they desire are actually implemented. The actual dynamics of these negotiations, the resources they consume and their repercussions for both computer users and computer specialists are poorly understood.

Scope and rate of change. Changes in the computing milieu vary in frequency and scope. While it is easy to assume that low rates of change are easier for users to adapt to, that hypothesis is oversimplified. Infrequent changes of wide scope, such as changing the formats for large sets of data, may disrupt a class of users regardless of frequency. Upward compatible features which are transparent to most users may be introduced into many systems and processors with relative impunity for most users.

On the other hand, certain users often seek specific changes in both applications and support software. However, in *shared systems*, changes developed for one party are typically imposed upon all users of the same computational resources. Many technical changes that benefit one party may benefit others as well. However, there are also common conflicts between the technical needs of different users. It is an empirically open question as to how frequently technical changes are either "pareto optimal" or indicate a redistribution of computational resources. Thus, the advocacy and implementation of changes has a strong political content above and beyond the resources required to implement the change.

Formalizing change procedures. Large software systems are often used to support organizational activities. Since these systems operate in a production environment, any alteration to such a system is usually scrutinized to assure that it doesn't cause disastrous effects. Many organizations have thus instituted a series of bureaucratic procedures to be followed prior to the actual alteration of a system. These procedures can be cumbersome in certain organizational structures.

The ability of an organization group to get a particular set of changes implemented may require interaction with, and the approval of, a number of intervening individuals or committees. If there are a large number of system change requests pending, then some prioritization scheme may exist. The group seeking system changes may now have to rely on its members' negotiating skills to assure a suitable priority.

Data quality

Collecting input data. Many contingencies structure the situations in which one party collects data about the activities of a second group from a third group for use by a fourth group. Some extreme situations include those in which (a) the groups are all the same or (b) all groups are aware of each other and share information with mutual consent and are all jointly concerned that the data be accurate. The latter case might occur with bank records, for example. In cases

of high mutual commitment, data capture may be smooth and subject primarily to errors in data entry.

However, in some important situations conflicts of interest or priority can arise among the various groups. If the data is to be used to assist the fourth party to control some activities of the data subjects (as in tax reports) there is some incentive for incomplete or inaccurate reporting.¹⁷ When several organizations share information systems, providing high quality and complete information may be more important to some participants than to others.¹⁸ Thus the quality of data collected is influenced by the patterns of interest cooperation within the social order surrounding the information systems.

Ensuring the correctness of processed data. It is common to believe that once data is accurately captured by a computer-based system it will remain accurate. There are at least two conditions under which this assumption can be problematic. Sometimes data is aggregated or reorganized to be used in an analysis. As the complexity and number of the data manipulation steps increases, programmers, operators or the application system itself may introduce difficult to detect errors in the transformed data set. Since data do not reorganize themselves in useful ways without personal intervention, data analysts are an essential part of many policy analysis units, survey research centers, etc. Secondly, in some systems which are shared by many users, particularly simulations, important parameters may be changed by one party without the cognizance of other users. Digital computers are particularly useful as calculating engines when the computations are too complex, tedious, or time-consuming for hand calculation. However, it is just in such cases that verifying the validity (or stability) of the results obtained is the most difficult. While such events are rare, their dynamics are instructive.

Control over computing

Control over the technology. Maintaining effective control over computing resources is a central issue for many computer users in an organization. In addition, some higher-level administrators who are not computer users, simply view computing as an expensive line-item to be kept in check.

Since computing and information are rich organizational resources, issues over contention for control are naturally commonplace. Like other social aspects of computing, negotiations over control of specific computing resources (e.g. data, programmers, budgets, I/O devices) take time and absorb organizational resources.

Access to and control over expertise. Computing is a complex process. In spite of its complexity, its use by a variety of people is becoming widespread. Many people do not take the time or interest to learn a great deal about it. Therefore, they must rely on others to help utilize computing effectively and to handle problems and unanticipated situations. There is increasing evidence that when users have easy access to expert assistance, the computer-based systems are better accepted than in those situations where access is more difficult.²⁷

Controlling the kinds of demands made by users. Whenever a personal service is provided, the stage is set of its consumers and providers to continually negotiate the kinds of service each would most prefer. In this way, computing is little different from other personal services such as legal advice, financial counseling, or medical care.

Computer specialists develop strategies for managing the behavior of their clients to help serve their own ends and make their organizational life tractable. Since they usually work as salaried employees with little freedom to negotiate a higher wage for difficult projects or those that incur an unacceptable level of dirty work, the strategies usually entail claims about organizational contingencies. Users may be told their requests are more expensive to fulfill, will take longer time to complete or entail unexpected technical complexity (such as system redesigns) to help displace less desirable work. Since computer specialists often have a relative monopoly on the expertise essential for judging the complexity of different requests, specialists' work-moderating strategies are difficult for instrumental users to easily counter.

The "values" sought by those individuals and organizations which promote applications development. Often computing systems are developed and installed when a specific person or small group of individuals actively promotes computing within an organization.^{21,26} New computer applications are usually costly: promoters who want resources allocated to their project must often first obtain sanctions from other organizational members. Since different actors become involved in acquiring computing resources, computing will often serve many ends. For example, some actors may be seeking to enhance their administrative control, others seeking to cut costs, still others may be seeking to make their jobs easier or more interesting. Few applications can be designed to serve many different interests well. Some users of computing often face difficulties which derive from the way in which their system is "optimized" to serve the interests of some other group.

Political support within the organization. Politics deals with the allocation of goods, services, symbols and values. The distribution of computing resources is often the focus of conflicts over budgets, staff and domain. This is not incidental. Rather, it is an *intrinsic* aspect of computer use. To the extent computing resources are valued by different actors in an organization, they will seek access to them. The resulting contention with its usual conflicts, bargaining and subterfuges is similar to other kinds of organizational politics.

Some actors seek control over computing resources simply because it provides a relatively large, growing staff and consequently a growing budget. There is also some evidence that overall computing arrangements can be more strongly influenced by the political access of key actors than by the technical soundness of their preferences.³¹

Software development practices

Programming and design practices. Since the development of software has become a major expense of computing for

most organizations, considerable attention has been focused on improving software design and programming productivity. New techniques¹² and tools have been developed to assist specialists with their various tasks. Structured programming^{10,11,39} is currently emphasized to aid specialists, as well as Chief Programmer Teams¹ and automated design aids.^{4,5}

While these techniques and aids may be beneficial for the organization, they may be problematic and disruptive for specialists. They may actually make the specialists' jobs more difficult and attention-demanding. They may create changes which are frustrating for specialists accustomed to previously established procedures.

Most modern programming practices and tools are yet to be widely adopted in computing settings outside of where they were developed. Reasons for this are unclear, but we suspect that organizational contingencies (such as meeting schedule deadlines, budgetary constraints or personnel training costs) in a computing setting tend to shape the adoption and incorporation of such tools and techniques.

Program testing. In theory, one would like to be able to automatically generate a sufficient set of test data necessary to demonstrate the probable correctness of a robust class of programs. However, for programs of moderate complexity, the set of test data to exercise all paths through a program is infeasibly large.¹⁵ Nevertheless, some promising research is proceeding on various schemes to automate tests for special program conditions.^{5,32} In contrast to the research on new tools for program testing, the state of current practice does not rely upon much automation at all. Test data, for example, are usually selected manually by a programmer or a knowledgeable user.

Software maintenance. Maintenance is often considered to be everything that happens to software after user acceptance. Maintenance can range from "bug fixes" through complete redesign and redevelopment of a delivered system. Often, the people who develop the system are not the same as those who maintain it. Given that different specialists are involved in system development and maintenance, finding those people (users or specialists) with dependable understandings of a system operation can be quite salient in determining the ease, timeliness and reliable execution of maintenance tasks.

Most programming work is in maintenance, not development. However, maintenance entails ongoing interaction between users, programmers, managers, vendor representatives, etc. While current software system life-cycle costs reflect the high cost of maintenance,^{3,5} the available figures do not distinguish the costs of program alteration work from the time, skill and attention required by specialists to successfully interact with those people requesting alterations. The extent to which these interactions are negotiated and completed with ease or difficulty, may better account for the variation in minimizing or exacerbating the costs of "routine" maintenance tasks.

Software documentation. Adequate and up-to-date software documentation is continually a weak feature of most software systems. Poor documentation is not usually a result of some software development practice. We note that while

many software system manuals can be measured in inches, their adequacy and currency vary. However, reasons for the variable quality of documentation appear not to be due to the unavailability of suitable documentation support aids or deficient programmer practices. Rather, updating documentation demands time, skills in clear and concise writing and attention. Given that specialists face some number of competing demands for their services, their ability or desire to maintain documentation competes with other work demands whose completion may be more highly rewarded.

Attention

The kinds of attention demanded by computing. Computing may appear to some users and specialists as a technology that requires a person learn a great deal to effectively utilize it. Many users must (or are at least led to believe they must) use the computer efficiently because it is a scarce resource. However, the time required by a user to prepare and successfully execute (after "debugging" runs) efficient programs often displaces any net savings in terms of completing the task at hand. Concern for minimal computer resource usage versus concern for minimizing the time to complete a work task often lead to conflicting demands for the user's attention.

The precision and detail demanded by computing. As a tool, the computer is a fairly exacting device. It demands that procedures be followed explicitly. It does not allow loose or *ad hoc* procedures in handling transactions as might exist in a manual or more informal information system.

At times, users complain that their jobs are actually more difficult or less interesting with computing than they had been previous to computing. Specialists complain that it takes a special person, like a "hacker," to be truly satisfied with the detail demanded by systems and application programming.³⁷

REFERENCES

- Baker, F. T., "Chief Programmer Teams," *IBM Systems J.*, Vol. 2, No. 1, 1972, pp. 56-78.
- Blau, Peter. M., "A Formal Theory of Differentiation in Organizations," *Amer. Soc. Rev.*, Vol. 35, No. 4, April 1970, pp. 201-218.
- Boehm, Barry W., "Software and Its Impact: A Quantitative Assessment," *Datamation*, Vol. 19, No. 5, May 1973, pp. 48-59.
- Boehm, Barry W., "Software Design and Structuring," *IBM Systems J.*, Vol. 11, No. 1, 1972, pp. 56-73.
- Boehm, B., R. K. McClean and D. B. Vefrig, "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 1, March, 1975.
- Brewer, Gary, *Politicians, Bureaucrats, and Consultants*. Basic Books, New York 1974.
- Burch, John, and Felix Strater, *Information Systems: Theory and Practice*. Hamilton Publishing Co., Santa Barbara, CA., 1974.
- Collins, Randall, *Conflict Sociology*. Academic Press, New York, 1975.
- Couger, Daniel, and Robert Knapp, *System Analysis Techniques*. John Wiley and Sons, New York, 1974.
- Dahl, O. J., E. W. Dijkstra and C. A. R. Hoare, *Structured Programming*. Academic Press, London, 1972.
- Freeman, Peter, *Software Systems Principles*. S.R.A., Palo Alto, CA., 1975.
- Freeman, P. and A. Wasserman, *Software Design Techniques* (Second Edition), IEEE Press, Long Beach, Calif., 1977.
- Goldberg, J., (ed.), *Proceedings of a Symposium on the High Cost of Software*, Stanford Research Institute Project 3272. Monterey, Ca., Sept. 17-19, 1973, pp. 99-119.
- Greenberger, M., Matthew A. Crenson and Brian Crissey, *Models in the Policy Process*. Russell Sage Foundation, New York, 1976.
- Howden, W. H., "Methodology for the Generation of Program Test Data," *IEEE Trans. on Computers*, Vol. C-24, No. 5, May 1975, pp. 554-560.
- Kling, Rob, "Towards a Person-centered Computer Technology," *Proc. 1973 ACM National Conference*, pp. 387-391.
- Kling, Rob, "Computers and Social Power," *Computers and Society*, Vol. 5, No. 4, Fall 1974, pp. 6-11.
- Kling, Rob, "Value Conflicts and Social Choice in Electronics Funds Transfer Systems Developments," *Communications of the ACM*, Vol. 21, No. 8, August 1978, pp. 642-657.
- Kling, Rob, "Automated Welfare Client Tracking and Services Integration: The Political Economy of Computing," *Communications of ACM*, Vol. 21, No. 6, June 1978, pp. 484-493.
- Kling, Rob, "Information Systems in Policy Making: Computer Technology and Organizational Arrangements," *Telecommunications Policy*, Vol. 2, No. 1, March 1978, pp. 22-32.
- Kling, Rob, "The Organizational Context of User-centered Software Design," *MIS Quarterly*, Vol. 1, No. 4, December 1977, pp. 41-52.
- Kling, Rob, "The Impacts of Computing on the Work of Managers, Data Analysts, and Clerks," WP-78-64 Public Policy Research Organization University of California, Irvine, Irvine, Ca. 1978.
- Kling, Rob and Elihu Gerson, "The Social Dynamics of Technical Change in the Computing World," *Symbolic Interaction*, Vol. 1, No. 1, Fall 1977, pp. 132-146.
- Kling, Rob and Elihu Gerson, "Patterns of Segmentation and Intersection in the Computing World," *Symbolic Interaction*, Vol. 1, No. 2, Spring 1978, pp. 24-43.
- Kosy, D. W., "Air Force Command and Control Information Processing in the 1980s: Trends in Software Technology," R-1012-PR, Rand Corp., Santa Monica, Ca., 1974.
- Laudon, Kenneth, *Computers and Bureaucratic Reform*. Wiley Interscience, New York, 1974.
- Lucas, Henry C., *Why Information Systems Fail*. Columbia University Press, New York, 1975.
- Martin, James, *Computer Data-Base Organization*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.
- Nolan, Richard L., "Computer Data Bases: The Future is Now," *Harvard Business Review*, Sept.-Oct. 1973.
- Palme, Jacob, "How I Fought with Hardware and Software and Succeeded," *Software Practice and Experience*, Vol. 8, No. 1, Jan.-Feb. 1978, pp. 77-83.
- Pettigrew, A., *The Politics of Organizational Decision-Making*. Tavistock Press, London, 1973.
- Ramamoorthy, C. V., and S. F. Ho, "Testing Large Software with Automated Software Evaluation Systems," *IEEE Trans. Software Engr.*, Vol. SE-1, No. 1, March, 1975.
- Sackman, H. A. and F. W. Blackwell, "Studies in Real-World Problem-Solving With and Without Computers," R-1492-NSF. Rand Corp., Santa Monica, Ca., May 1974.
- Simon, H. A., "Designing Organizations for an Information Rich World," *Computers, Communications, and the Public Interest*, Martin Greenberger (ed.), Johns Hopkins Press, Baltimore, Md., 1971.
- Strauss, Anselm, *Negotiations*. Jossey-Bass, San Francisco, Ca., 1978.
- Streeter, M., *The Scientific Process and the Computer*. Wiley-Interscience, New York, 1974.
- Weizenbaum, J., "Science and the Compulsive Programmer," in *Computer Power and Human Reason*, W.H. Freeman and Co., San Francisco, Ca., 1975.
- Wolverton, R. W., "The Cost of Developing Large Scale Software," *TRW Software Series*, TRW-SS-7201, Redondo Beach, Ca., March 1972.
- Yourdon, Edward, *Techniques for Program Structure and Design*. Prentice-Hall, Englewood Cliffs, 1975.

Project management through the Accomplishment Value Procedure (AVP)

by DONALD J. AHARONIAN

Digital Equipment Corporation
Maynard, Massachusetts

INTRODUCTION

This paper describes a technique called the Accomplishment Value Procedure, AVP, which accurately measures the status of and provides visibility to an information systems development project. It builds upon the foundation of two of R. I. Benjamin's axioms:¹

Axiom #10—"The great leap forward is best accomplished in short, comfortable hops; if there is a 'Golden Rule' in information systems development, this is it."

Axiom #14—"If you can't plan it, you can't do it."

to which I add my own corollary:

"If you don't schedule it, it won't get done."

A persistent problem of project management has been to relate resources budgeted with work accomplished after the project begins.² AVP bridges that gap. As a tool of project management, AVP:

1. Provides the means to schedule, monitor and control a project after it has passed the planning phase.
2. Enforces a discipline for resource estimating and time scheduling that focuses on the completion of tasks.
3. Provides a method to handle changes to schedules and to resources which can be documented and displayed simply and clearly.
4. Provides a method for summarizing overall status of projects by management responsibility.

AVP does all this by focusing on the gathering of data that is plotted on a Project Visibility Chart for an individual project and a Summary Visibility Chart for a group of projects.

In doing so, AVP communicates with all levels of an organization in a consistent manner:

1. Top management sees a snapshot of the overall status of development projects.
2. Middle management sees a snapshot of individual projects.

3. Project leaders can monitor performance and compare it to schedules and estimates which point to areas of potential problems that might require management analysis.
4. Project team members can see the status of the projects they are working on.

The remainder of this paper deals with (1) a discussion of *Background: AVP in the Perspective of Project Management*, (2) a description of the *Accomplishment Value Procedure*, (3) a description of an *Example of the AVP Process*, (4) a description of *How AVP Handles Changes* in schedules and estimates, (5) a description of the *AVP Summary Process* and (6) a *Summary* section that includes conclusions and observations regarding the applicability of AVP.

BACKGROUND—AVP IN THE PERSPECTIVE OF PROJECT MANAGEMENT

Project management usually means different things to each of us. A major reason is that a project is a unique effort marshalling resources to solve a unique problem. The uniqueness has attracted special management techniques which, according to Murdick and Ross³ include "outstanding characteristics," such as:

1. *Work breakdown structuring* which is a method that decomposes the project end result, level-by-level, all the way down to something called the work package, the lowest identifiable element of work to be done.
2. *Network definition* which describes task relationships in a project and which is usually associated with PERT/CPM activities.
3. The *integration of performance/cost/time* for project planning and control.

You can locate AVP in the perspective of project management by relating the outstanding characteristics to the three basic managerial functions required as defined by Paulson⁴ as follows:

- *Planning*—" . . . the various tasks . . . must be performed to complete the project . . . involves approxi-

TABLE I.—Project Management Matrix

		Outstanding Characteristics ³		
		Work Breakdown Structuring	Network	Integration of Performance/ Cost/Time
Management Functions ⁴	Planning			
	Scheduling			AVP
	Control			AVP

mate requirements for material, equipment and manpower . . .”

- *Scheduling*—“. . . the feasible start and completion dates for each activity . . .”
- *Control*—“. . . monitoring actual performance and comparing it to that which was anticipated from the schedule. The essence of control lies in recognizing differences when they occur, determining reasons for them, and promptly evaluating effects on the schedule.”

In terms of the three “outstanding characteristics,” AVP assumes that (1) formal, or informal work breakdown structuring exists with or without a standard work breakdown

structure being built, and (2) that the relationships and dependencies among tasks are understood with or without a network being drafted. It is the third characteristic dealing with performance, cost, and time to which AVP applies—with one distinction. The distinction is that AVP takes place after planning is completed.

The matrix in Table I shows where AVP fits in the relationships between Outstanding Characteristics and Managerial Functions of Project Management. AVP addresses itself to the articulation of accomplishment (performance) and manpower (cost) related to (integrated with) time. By displaying accomplishment and manpower at points in time, AVP provides a visibility that communicates the schedule and supports control.

The quantification of the value of accomplishing each task is crucial to AVP. As each task is completed, the project is credited with the value of the task. There is no credit for a partial completion; thus the notion of “percentage complete” is avoided. The expression of percentage complete has traditionally been difficult in its execution, arbitrary in its determination and misleading in its interpretation.² How often has one heard the response that a project is “x percent complete?” In fact, projects have been known to be “90 percent” complete for months.

THE ACCOMPLISHMENT VALUE PROCEDURE

The mechanics of the procedure result in the creation and update of the Project Visibility Chart. The vehicle for this

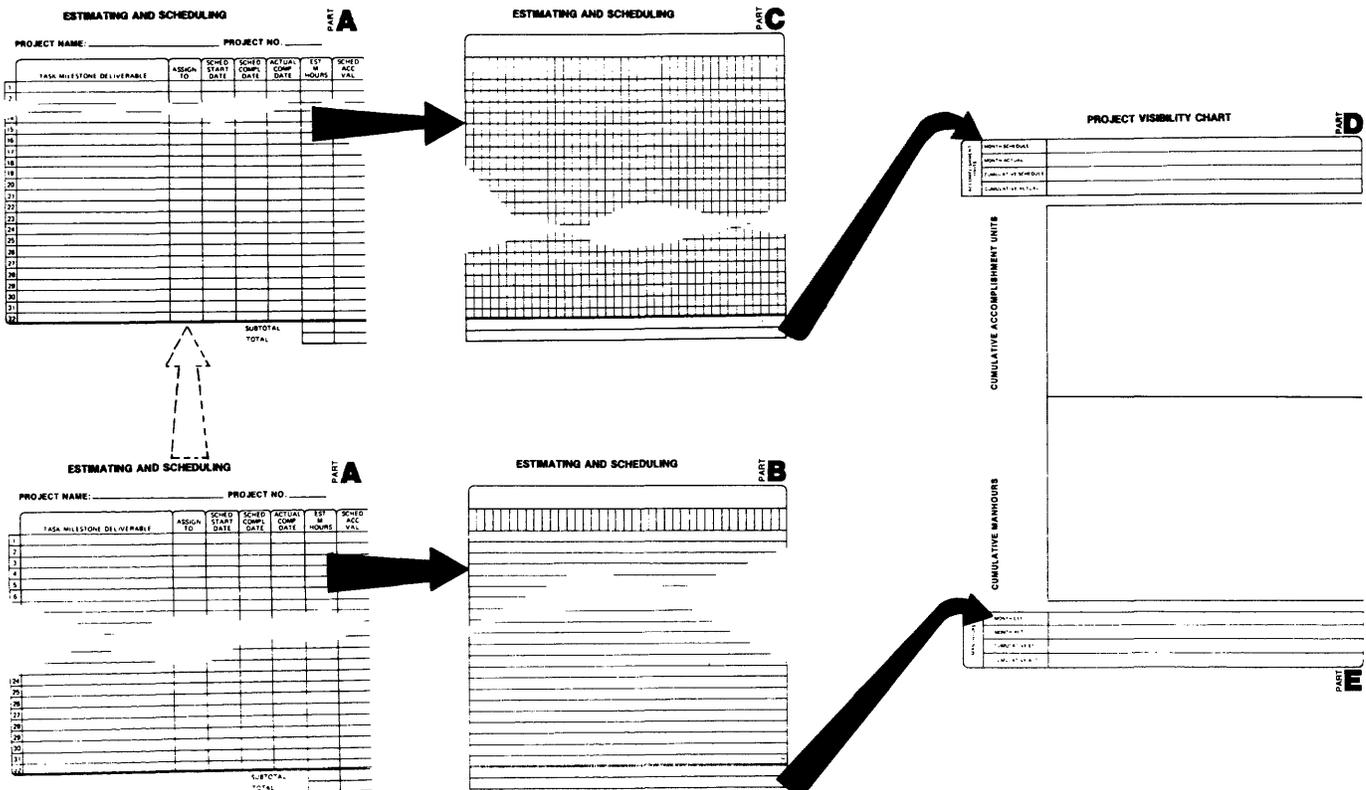


Figure 1—Blank Project Visibility Chart and Estimating and Scheduling Form (Parts A-E).

procedure is the Estimating and Scheduling Form. The form integrates the key steps of the procedure and flows through to provide the data for the Project Visibility Chart. The form is a composite of five parts, referred to as Parts A through E. Figure 1 shows all parts in the relative positions of the flow of data and information, i.e., from A to B to E and from A to C to D.

Part A is where you would insert descriptive data about each task (milestone) deliverable. The procedure assumes a documentation standard and just about any version would do. Further, the procedure amplifies the doctrine espoused by *ADP Analyzer*,⁵ which says "... documentation be completed by the end of each phase and each review period. If the documentation has not been completed, then the phase has not been completed—and the next phase cannot begin." AVP accepts any documentation standard and encourages even finer breakouts or subsets.

Part B is where you would enter the estimates of resources committed to each task/milestone/deliverable by time period (usually a month, but the procedure can handle weekly and daily). We focused on manhours because it is the key resource and it is easier to collect data regarding manpower on a timely basis.

Part C is where you enter the time span for each task/milestone/deliverable further annotated with the "value" calculated for each.

Part D is a table which is built on the accumulation of data concerning accomplishment units scheduled and actual for each time period and accumulated by the end of each time period.

Part E is a table which is built on the accumulation of data concerning manhours estimated and actual for each time period and accumulated by the end of each time period. For convenience in preparation and for easy reference and analysis, Parts D and E are physically part of the Project Visibility Chart.

In Figure 2 is an example of a Project Visibility Chart for a completed project showing in the top portion, the Cumulative Accomplishment Values, Scheduled vs. Actual and in the bottom portion, the Cumulative Manhours (resources), Estimated vs. Actual, for the history of a project. The Chart displays the status of the project expressed in Accomplishment Units as well as the expenditure of Manhours, both over the life of the project. Notice that the points plotted on each Chart are derived from the data tables, Part D and Part E, contiguous with each. The data tables are built as a result of the process previously described.

EXAMPLE OF THE AVP PROCESS

The Accomplishment Value Procedure (AVP) highlights actual completion of milestones and ignores partial comple-

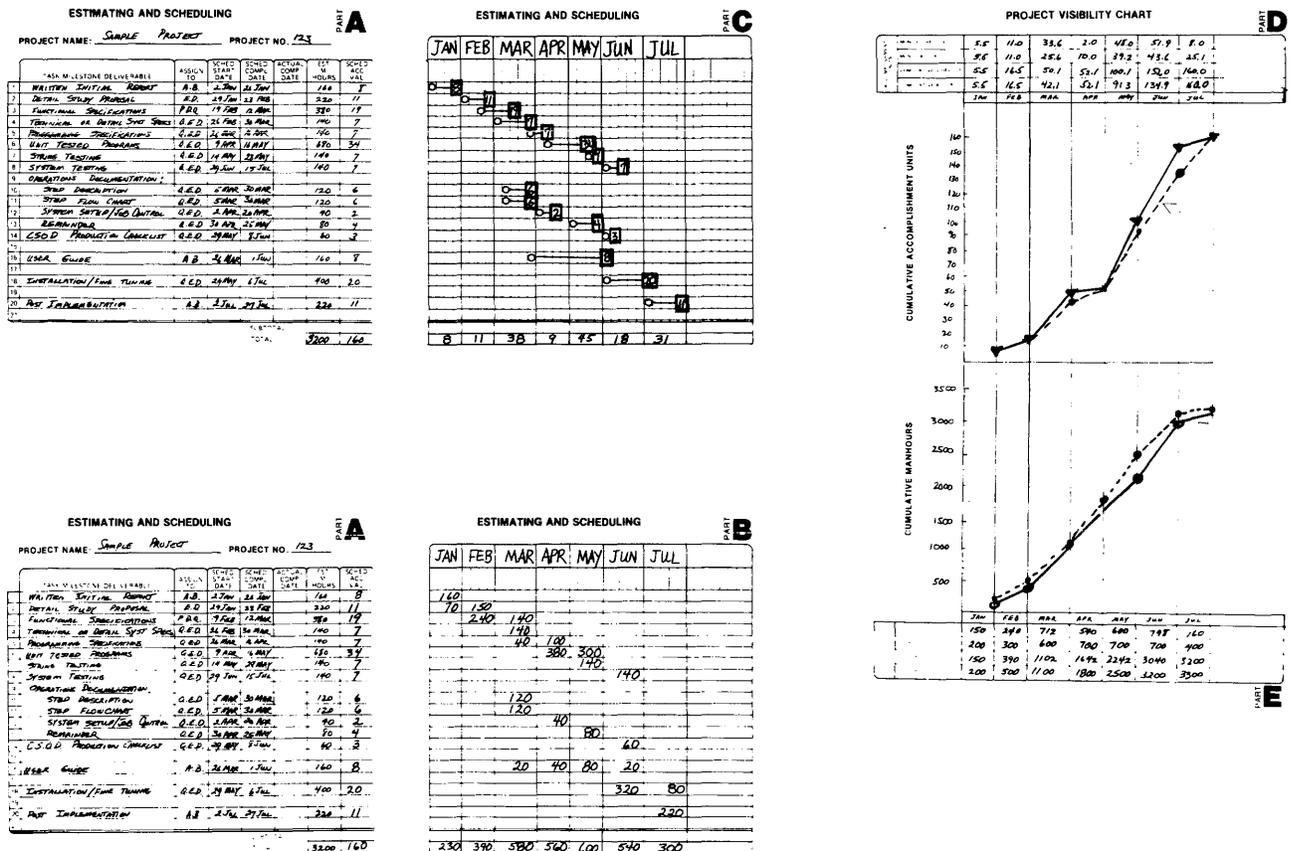


Figure 2—Completed Project Visibility Chart and Estimating and Scheduling Form (Parts A-E).

ESTIMATING AND SCHEDULING

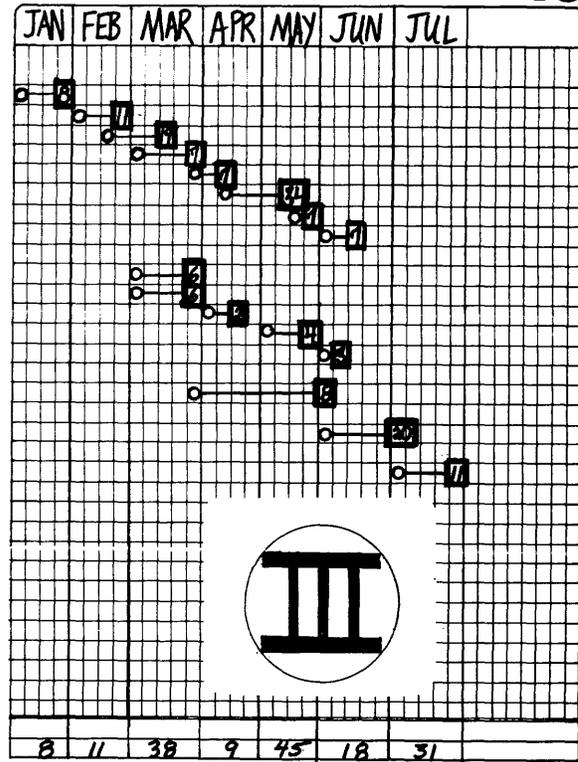
PART A

PROJECT NAME: SAMPLE PROJECT PROJECT NO. 123

TASK/MILESTONE/DELIVERABLE	ASSIGN TO	SCHED. START DATE	SCHED. COMPL. DATE	ACTUAL COMP. DATE	EST. M/ HOURS	SCHED. ACC. VAL.
1 WRITTEN INITIAL REPORT	A.B.	27 JAN	26 JAN		160	8
2 DETAIL STUDY PROPOSAL	R.D.	27 JAN	23 FEB		220	11
3 FUNCTIONAL SPECIFICATIONS P.D.	R.D.	19 FEB	12 MAR		380	19
4 TECHNICAL OR DETAIL SYS SPECS	Q.E.D.	26 FEB	30 MAR		140	7
5 PROGRAMMING SPECIFICATIONS	Q.E.D.	26 MAR	16 APR		140	7
6 UNIT TESTED PROGRAMS	Q.E.D.	9 APR	16 MAY		680	34
7 STRING TESTING	Q.E.D.	14 MAY	28 MAY		140	7
8 SYSTEM TESTING	Q.E.D.	29 JUN	15 JUL		140	7
9 OPERATIONS DOC'N:						
10 STEP DESCRIPT'N	Q.E.D.	5 MAR	30 MAR		120	6
11 STEP FLOW CHART	Q.E.D.	5 MAR	30 MAR		120	6
12 SYSTEM SETUP/TEST CTRL	Q.E.D.	2 APR	20 APR		40	2
13 REMAINDER	Q.E.D.	30 APR	25 MAY		80	4
14 C.S.D. PRODUCTION CHECKLIST	Q.E.D.	29 MAY	8 JUN		60	3
15						
16 USER GUIDE	A.B.	26 MAR	1 JUN		160	8
17						
18 INSTALLATION/FINE TUNING	Q.E.D.	29 MAY	6 JUL		400	20
19						
20 POST IMPLEMENTATION	A.B.	2 JUL	27 JUL		220	11
21						
22						
23						
24						
25						
26						
27						
28						
29						
30						
31						
32						
SUBTOTAL					3200	160
TOTAL					3200	160

ESTIMATING AND SCHEDULING

PART C



ESTIMATING AND SCHEDULING

PART A

PROJECT NAME: SAMPLE PROJECT PROJECT NO. 123

TASK/MILESTONE/DELIVERABLE	ASSIGN TO	SCHED. START DATE	SCHED. COMPL. DATE	ACTUAL COMP. DATE	EST. M/ HOURS	SCHED. ACC. VAL.
1 WRITTEN INITIAL REPORT	A.B.	27 JAN	26 JAN		160	8
2 DETAIL STUDY PROPOSAL	R.D.	27 JAN	23 FEB		220	11
3 FUNCTIONAL SPECIFICATIONS P.D.	R.D.	19 FEB	12 MAR		380	19
4 TECHNICAL OR DETAIL SYS SPECS	Q.E.D.	26 FEB	30 MAR		140	7
5 PROGRAMMING SPECIFICATIONS	Q.E.D.	26 MAR	16 APR		140	7
6 UNIT TESTED PROGRAMS	Q.E.D.	9 APR	16 MAY		680	34
7 STRING TESTING	Q.E.D.	14 MAY	28 MAY		140	7
8 SYSTEM TESTING	Q.E.D.	29 JUN	15 JUL		140	7
9 OPERATIONS DOC'N:						
10 STEP DESCRIPT'N	Q.E.D.	5 MAR	30 MAR		120	6
11 STEP FLOW CHART	Q.E.D.	5 MAR	30 MAR		120	6
12 SYSTEM SETUP/TEST CTRL	Q.E.D.	2 APR	20 APR		40	2
13 REMAINDER	Q.E.D.	30 APR	25 MAY		80	4
14 C.S.D. PRODUCTION CHECKLIST	Q.E.D.	29 MAY	8 JUN		60	3
15						
16 USER GUIDE	A.B.	26 MAR	1 JUN		160	8
17						
18 INSTALLATION/FINE TUNING	Q.E.D.	29 MAY	6 JUL		400	20
19						
20 POST IMPLEMENTATION	A.B.	2 JUL	27 JUL		220	11
21						
22						
23						
24						
25						
26						
27						
28						
29						
30						
31						
32						
SUBTOTAL					3200	160
TOTAL					3200	160

ESTIMATING AND SCHEDULING

PART B

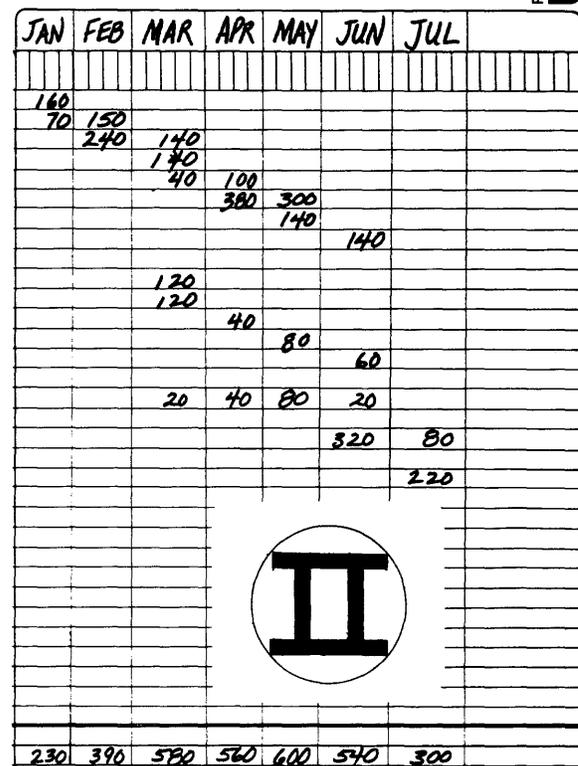


Figure 3—Example of Parts A-B and A-C filled out at the beginning of the AVP process.

tion. It provides the mechanism for identifying those milestones in the development process that have specific deliverable documents which signal the completion of a milestone.

Once the milestones are identified and resource *estimates* are associated with each, the sequence of events can be *scheduled*. Next is the calculation of the "value" of each milestone. For our purposes, value is based on the establishment of an arbitrary denomination of units which reflect the relative estimated manhour resource for each milestone/deliverable.

The Accomplishment Value is the focal point for the project as it moves to completion. The Accomplishment Value is plotted monthly on the graph along with the resources. In this example, 20 manhours was selected as equal to one Accomplishment Value. Thus, if the milestone/deliverable is estimated to need 160 manhours of resources, then its Accomplishment Value is eight.

See Figure 3, Key I, which is the Estimating and Scheduling Form—Part A filled out with the Accomplishment Values for each milestone/deliverable and represents the beginning of the process.

Having completed Part A, you can proceed to Part B where the loading of the resources over the course of the project is entered. This loading then becomes the estimated manhours (resources) for the life of the project. See Figure 3, Key II, which shows the manhour loading or MONTH EST (estimate) by month for each task/milestone/deliverable and the total MONTH EST (Estimate) for each month.

For each milestone/deliverable that you have associated with a Scheduled Start and Completion date, you can enter the 0—symbol for the start in the week in which the effort for that milestone/deliverable will begin and a □ in the week that it is scheduled to be completed and delivered.

In the square for the ending week, you can enter the Accomplishment Value. Next, add up the Accomplishment Values for each month. See Figure 3, Key III.

The values of each milestone/deliverable are combined to roll up to the total value of the project. In the course of completing the project and as each milestone/deliverable is achieved, the value of each milestone is credited toward project completion, and a measure of project status is established. Credit is given only to those milestone/deliverables that are completed; resources expended on an incomplete deliverable are given *no* credit. This criteria forces attention to establishing as many milestones as is logical and manageable—a basic rule in successful project management. As a corollary, a specific deliverable could be segmented with each segment becoming a separate milestone/deliverable that would be scheduled, estimated and accomplishment valued.

Next, you build the data tables in Parts D and E that will be the plots for curves on the Project Visibility Chart. See Figure 4.

The MONTHLY SCHEDULE Accomplishment Value in Part D is simply a posting of the totals from the bottom of Part C and from which the CUMULATIVE SCHEDULE Accomplishment Value data is calculated.

The MONTHLY EST (Estimate) manhours data in Part

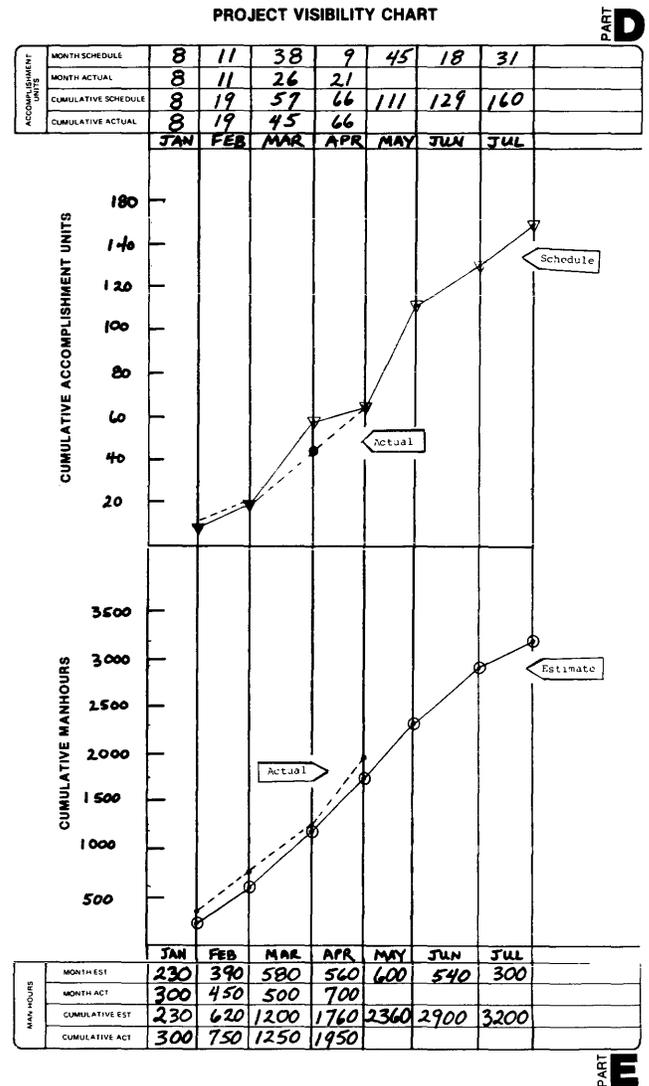


Figure 4—Project Visibility at end of April.

E is simply a posting of the totals from the bottom of Part B and from which the CUMULATIVE EST (Estimate) is calculated.

The CUMULATIVE SCHEDULE Accomplishment Value and the CUMULATIVE EST (Estimate) manhours are the plots by month on the Project Visibility Chart in Figure 4.

As the project develops, Actual Data is collected for Manhours and Accomplishment Values. In our example you can see the data recorded through April, the fourth month.

The Project Visibility Chart representing the status of the Sample Project through April would be as in Figure 4. An analysis of the charts and the data in the table above and below the charts (which are Parts D and E of the Estimating and Scheduling Form) shows that the Project fell behind the Scheduled Accomplishment during March but that we caught up during April. The recovery, however, seems to be costly since we have exceeded our estimated manhours

by nearly 200 manhours. If the trend continues, the project could get into a serious over-budget situation. This situation would require further analysis.

HOW AVP HANDLES CHANGES

The technique lends itself to a procedure for changing schedules and estimates based on changes in project scope or in availability of resources, etc. If for example, at the end of April it was determined that the project's schedule could be shortened by infusion of additional available resources, (while it may violate Brooks' Law⁶ it is optimistically espoused here for illustration purposes) then the Estimating and Scheduling Form—Parts B, C, D, and E—should be modified to reflect this change and to show the impact on the bar chart schedules. The impact of the revision with the new manpower estimate, and the Accomplishment Value Units, for May and June is readily shown in the Project Visibility Chart, Figure 5, which shows a new chart with the scheduled and estimated lines shifted upward for both Accomplishment Values and Manhours. This representation is significant since it provides visibility for changes in estimates and schedules.

THE AVP SUMMARY PROCESS

AVP further lends itself to summarizing groups of projects by management responsibility such as those of an individual cost center. A collection of Project Visibility data from several projects can be aggregated to provide visibility. By focusing on an individual fiscal month, you can display Scheduled and Estimated data for its development projects at the beginning of the month. At the end of the month, you can then aggregate the Actual data for Accomplishment and Manhours.

On the right side of Figure 6 is a Summary Visibility Chart completed for a development cost center for the month of October 1978. The Bar Graphs show the Scheduled Accomplishment Units next to the Actual Accomplishment Units in the top portion with the Estimated Manhours next to the Actual Manhours in the bottom portion.

On the left hand side of Figure 6 is the AVP Log which is the vehicle for the Summary Process. The Log provides space for recording the Schedules and Estimates for each of a group of projects at the beginning of a time period (usually Fiscal Month) and for their Actual Data at the end of the period.

While transferring Accomplishment Values from the Part Ds you have to make certain that the Unit of Measure (U/M) is consistent. If 20 manhours were used as the U/M for the Summary Process, then the procedure is to divide the U/M of each project by 20 and then to multiply the resulting fraction by the Accomplishment Value Units associated with the project. For example, if a project had scheduled 72 units with a U/M of 10, then to convert—divide 10 by 20 which results in 1/2; then multiply 1/2 by 72 which results in the

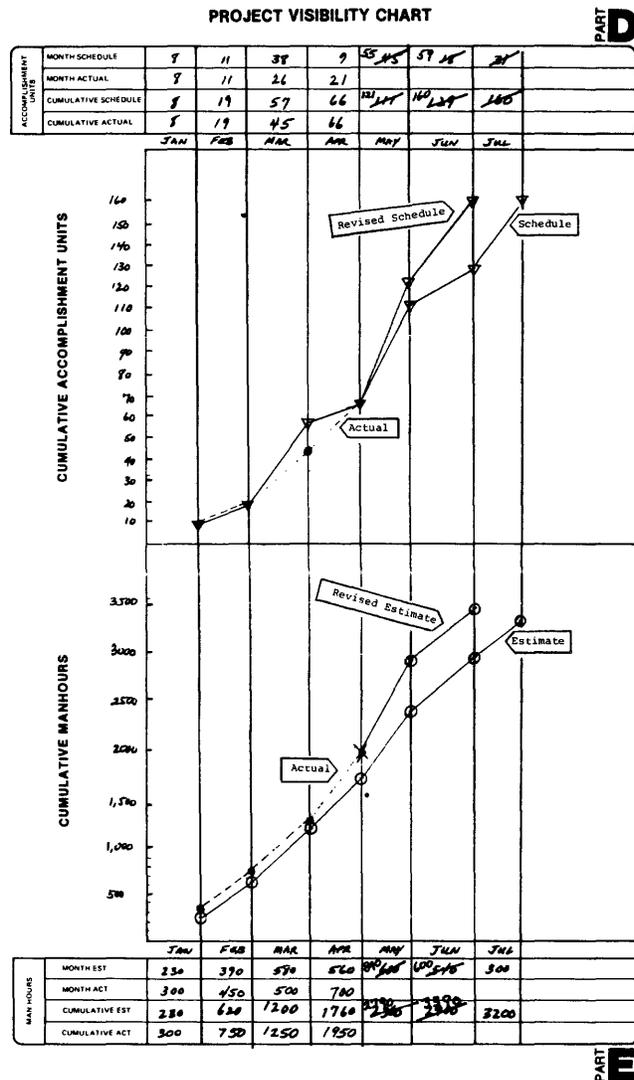


Figure 5—Project Visibility Chart showing impact of revision at end of April.

conversion of 36 Accomplishment Units. It is the latter number that you post to the Summary AVP Log.

Six projects are posted on the Summary AVP Log with a total Accomplishment Value of 120 (based on a unit of measure of 20 manhours per each Accomplishment Unit) and 2,600 manhours are estimated (and committed) to be used in the process of doing project work. At the end of the month, the actual data are posted totaling 100 Accomplishment Value Units and 3,000 manhours respectively. It is the data from the totals column that is used as a basis for constructing the bar chart.

SUMMARY

This paper has introduced the Accomplishment Value Procedure (AVP) by first discussing where it fits in the background perspective of Project Management. Then, the paper

SUMMARY VISIBILITY CHART

SUMMARY AVP LOG

Proj #	Proj. Name	Accomplishment Value				Manhours	
		Orig U/M	Sum U/M	Month Scheduled	Month Actual	Month Estimated	Month Actual
123	Accts. Rec'ble	20	20	40	40	1200	1400
124	Accts. Payable	20	20	20	-0-	360	360
125	Cash Disb.	20	20	15	15	480	480
126	Prod. Plann'g	20	20	12	12	200	300
127	Prod. Control	20	20	12	12	200	300
128	Inv. Planning	20	20	21	21	160	160
Totals				120	100	2600	3000

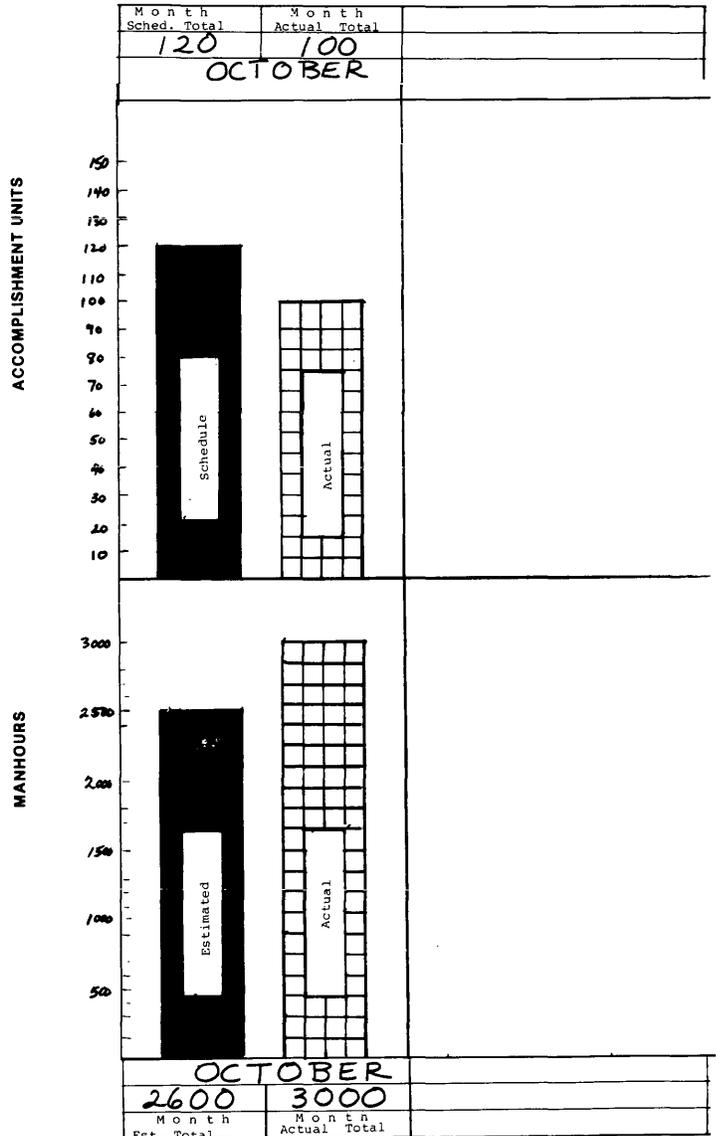


Figure 6—Summary Visibility for one month.

described the steps in the AVP process which lead up to the creation of and provides the basis for updating the Project Visibility Chart. Included in the description was an illustration of how changes in resources or in time would be shown on the Project Visibility Chart. Finally, there was a description of the AVP Summary Process which provides for tracking and displaying groups of projects of a functional organization by aggregating their Visibility Data.

AVP is currently in place in the Customer Service Operations Development (C.S.O.D.) Department which is the internal ADP/MIS systems development and computer operations organization at the Field Service Headquarters of Digital Equipment Corporation, Maynard, Mass. C.S.O.D. provides system development and computer-based support

to headquarters functions in worldwide logistics, financial control and product engineering and centralized design and development of distributed systems for decentralized implementation at 15 data centers worldwide.

The benefits of its application revolve around the simplicity of how well it can be understood by project team members and how effective it is in relating (communicating) status to users and to management.

In addition, individual project team members reach a "comfort" level with the graphical representation of the project status and spend their time with greater attention to the objectives of the project.

There are other benefits. AVP highlights the viability of resource estimates and schedules at the onset of the devel-

opment cycle and the resource consumption and accomplishment achievement during the development cycle by the smoothness or lack of smoothness of the curve on the Project Visibility Chart.

The discipline of the Accomplishment Value Procedure lends itself to contracting applications development to a vendor. By focusing on specific deliverables you could contract for progress payments synchronized with each deliverable under either a cost re-imbursable basis or a fixed-price basis.

At project completion, final payment could be held back until a reasonable warranty period has passed. Moreover, there can be variations that would provide incentives to a contractor. For example, a progress payment or a fixed-price commitment for a deliverable by a certain date would pay X dollars to the vendor but delivery one month earlier

would pay 120 percent of X dollars where the incremental 20 percent would serve as an incentive to the contractor.

REFERENCES

1. Benjamin, R. I., *Control Of The Information Systems Development Cycle*, John Wiley & Sons, N.Y., 1971.
2. Block, Ellery B., "Accomplishment/Cost: Better Project Control," *Harvard Business Review*, May-June 1971, p. 136.
3. Murdick, Robert G., and Joel E. Ross, *Information Systems For Modern Management*, Second Edition, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1975, p. 265.
4. Paulson, Boyd C., Jr., *Man-Computer Concepts For Project Management*, Technical Report No. 148, The Construction Institute, Stanford University, Stanford, CA, 1971, pp. 9-10.
5. *EDP Analyzer*, "A Structure For EDP Projects," Vol. 11, No. 5., Canning Publications, Inc., Vista, CA., May 1973, p. 13.
6. Brooks, Frederick P., "The Mythical Man-Month," *Datamation*, December 1974.

Textfax—Principle for new tools in the office of the future

by WOLFGANG HORAK and WALTER WOBORSCHIL

Siemens Central Research Laboratory
Munich, Germany

INTRODUCTION

By taking a closer look at today's office, we observe the following trend: The conventional typewriter is gradually being replaced by word-processors. These may merely be electric typewriters with a storage added or they may take on the form of highly sophisticated CRT workstations featuring screens carrying an entire standard size page and exchangeable storage media. These systems, which originally had been intended for local word-processing, are now increasingly being supplemented by communication functions, permitting direct text communication from one's own buffer to that of a business partner—i.e. to his electronic "mailbox." Whenever desired, the recipient can then call up the text from the buffer for reading or, if necessary, for editing and subsequent filing or forwarding. These functions can be summed up under the catchword "electronic mail."

First experiments on this have been performed especially in the United States, like those of the Citibank.¹ To permit not only internal, but also public text communication, national and international standards still have to be elaborated, ensuring compatibility of the various products.

Further important elements in today's office, besides text systems, are the numerous copiers and—to the extent to which international standardization progresses—also remote copiers, i.e. facsimile equipment. Copiers and remote copiers are required to duplicate or transmit documents consisting of text *and* graphics. Here, too, a fusion of individual functions can be observed, as is the case with the remote copier with local copying operation.

Let us enter an office handling, for example, quotations for technical products. In this office, the quotation texts can be generated on the text system, perhaps by using stored text segments. However, the data sheet with photos, diagrams, etc., must be prepared at the printer's in the various versions and kept in conventional files in the office. To mail the quotation copy and the data sheet to the customer, there are two possibilities. Either text and data sheet are jointly enveloped and mailed or, in case the customer happens to have a remote copier of the same type, both can be scanned and transmitted successively by the remote copier. During this process, the text is treated like a graphic and—compared with alphanumerical coding—is transmitted with unnecessary redundancy, i.e. involving too much time.

The previous example shows that, in today's office, it is still not possible to jointly

- Collect
- Process
- File
- Output as hard and soft copy
- Effectively transmit

text *and* image at the same workstation by using the same hardware and software components.

To be able to do so, new office tools are required. The underlying principle we call Textfax.² On the road toward a largely "paperless" office, we have done some research to work out this principle, trying to specify the functions of these tools and to study ways and means of implementing them.

Developments in the direction of Textfax are

- The printer plotters, where the same matrix printer is used for text and facsimile printout.
- The image processing systems for computer-aided processing of TV images.
- The system named Electronic Darkroom^{3,4} developed since 1970 at the MIT for Associated Press for editing, filing and transmitting of press photos.
- The large printing facilities for electronic photocomposition, enabling above all combined text and graphic processing for ad offices.
- The initial proposals⁵ for transmission procedures permitting combined transmission of alphanumerically coded text and coded facsimiles.
- The experimental Soft Display Word Processor from Xerox,⁶ which has a facsimile graphic generator to produce business forms or other graphics, that can be overlaid with text from the character generator.

In the following, the performance features of Textfax will first be specified more closely to subsequently go into the first results from two processing runs carried out on an experimental workstation.

PERFORMANCE FEATURES OF INTEGRATED TEXT AND FACSIMILE PROCESSING AND COMMUNICATION

Data entry

Figure 1 illustrates the various possibilities of entering text, handwriting and hand drawings and collecting mixed text/graphic material on paper or microfilm.

The *tablet* is used for inserting hand drawings to be copied into mixed text/image documents. During copying, a separate positioning step is required which is not necessary when writing directly in the softcopy with a light pen or on a touch-sensitive device (TSD) attached to the screen. This way handwritten comments, corrections and signatures can be applied to documents.

In the case of a boss/secretary workstation, e.g., the secretary has a complete system while the boss merely has a tiltable full-page screen with a TSD for handwriting and function selection. The boss can thus apply corrections to the text typed by his secretary and circulars and other sorted mail received electronically can be marked with comments before it is passed on, for example.

The *facsimile scanner* has the same resolution as defined in CCITT recommendation T.4 for Group 3 facsimile apparatus. As a compromise between facsimile quality and data quantity this resolution should suffice for most office applications and represents the standard according to which the documents are processed, filed, output and transmitted at a Textfax station. With a scanning window of approx.

215×297 mm, a page (size DIN A4=210 mm, ×297 mm, U.S. standard=215 mm×280 mm) in the form of a loose-leaf, a book or a magazine page, is scanned with a horizontal resolution of 1728 pixels per line, and a vertical resolution of 7.7 lines/min. Scanning time per page of about 10 sec. is possible. Mixed documents are entered with 4 to 8 bits per pixel, with grayscale portions after rastering being passed on for further processing with one bit per pixel just like text, graphic and handwritten portions after passing a black-and-white threshold.

On workstations having to cope with large text volumes of existing documents, *character recognition circuits* can be connected to the scanner. This way texts can be entered with low redundancy and alphanumeric coding, and not as facsimiles, i.e. in raster reproduction. In the case of mixed documents, texts in strange fonts, graphics, handwritings and grayscale images can be either masked via programmable masks or, like unrecognized characters, be entered as facsimiles. In the latter case, undesirable portions of a document to be entered can only be erased afterwards by marking the corresponding spaces in the softcopy with the cursor. In the same manner, unrecognized characters can be replaced subsequently via the keyboard by alphanumerically coded ones. The most frequently occurring fonts, such as OCR-B, letter gothic and pica, should be recognizable.

Via *video camera*, sections from leaf and book material can be entered quickly and conveniently. For example, via function keyboard and screen monitoring, the camera is positioned over the material, and the size of the section determined with the motor zoom. Thus, it is possible to enter details with a resolution larger than the standard resolution. Furthermore, construction permitting, images of three-dimensional objects and persons (e.g. photos of authors) can be entered with one and the same camera. Microfiches are either scanned at the workstation, using a suitable frontal attachment to the above camera, or with the scanner of a central microfilm file containing standard graphic segments, for example.

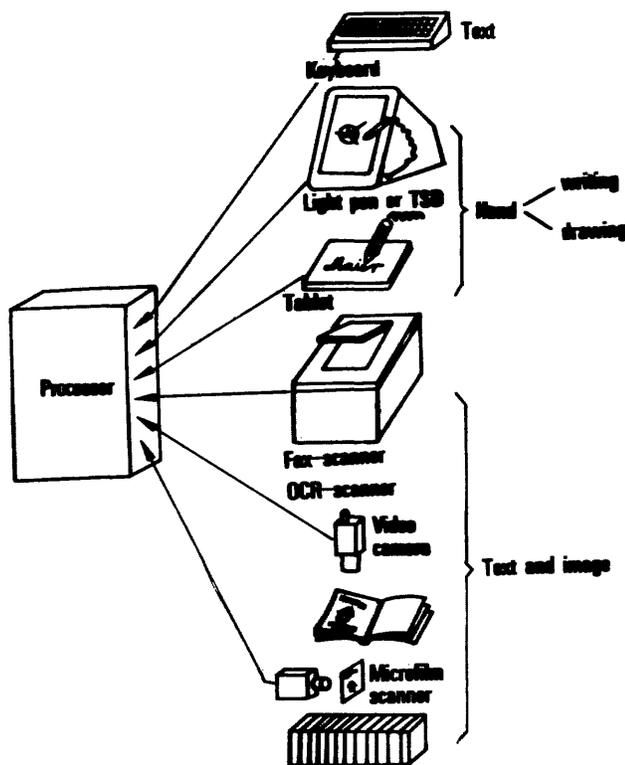


Figure 1—Entry.

Hardcopy and softcopy

The softcopy on the screen is required for monitoring input and text/facsimile editing. To feature a full page flickerless with Group 3 resolution, a video monitor with approx. 2,300 visible lines at 60 Hz frame frequency would be required. Depending on the size of image and line return, this corresponds to a beam dwell of less than four ns per pixel, and an output rate of more than 240 Mbit/s. Obtaining such data entails technological problems, above all concerning the picture tube. It is therefore recommended to instead make use of a monitor with approximately 1,200 visible lines and a 60 Hz frame. Selecting vertical format and a 44 cm screen diagonal, a full page with half the standard resolution, i.e. 3.85 lines/mm (=Group 2 resolution) can be shown flickerless in original size with black text on white background.

If half the standard resolution is insufficient, e.g. when editing smaller details, switching to full resolution is possible. In this mode only one-quarter of a vertical format page enlarged by the factor two can be represented. However,

the window, i.e. the quarter section, can be shifted over the entire page, both horizontally and vertically. The video monitor is accompanied by a refresh memory of four Mbit capacity. This corresponds to the number of black-and-white pixels of a complete page scanned with Group 3 resolution.

Hardcopies are produced with a matrix plotter. It should output the mixed text/image documents in standard resolution on regular paper DIN A4 sheets within about 10 sec. per page. As required, a processor-controlled device will cut the sheets from a paper reel of 210 or 215 mm width. Like entering, outputting and processing can be handled simultaneously.

The plotter and facsimile scanner, or videocamera or microfilm scanner can be operated concurrently in the local copying mode. Enlarging or reducing is done via software. The local copy renders text duplication by way of an impact printer's carbon copy superfluous. However, since local copying by coupling scanner and plotter via the processor loses in resolution over conventional office copiers relying on xerographic techniques, a combination of copier and facsimile scanner in one and the same device would be desirable. With local copying, the original is reproduced on the copying paper on a 1:1 scale, whereas entering calls for the original to be reproduced—e.g. on a line-by-line basis on a small CCD line.

Processing

The known functions of text editing and processing are to be supplemented by the functions of facsimile processing, to the effect that mixed text/image material can be edited just as well.

Texts are edited in alphanumeric coding, in the form entered via keyboard or character recognition circuits. Only for outputting on the screen or the plotter, and perhaps for transmitting, the text is converted into a facsimile and combined with the image information. However, as the plotter is restricted only in resolution with regard to reproducible type faces, and since the reproduction scale of the characters permits ample variations, the text editing software must allow for corresponding variables. Individual type appearance is thus ensured, right up to text graphics, without the limitations inherent in a printer's type set. Furthermore, the coordinates of facsimile fields must be taken into account in the case of automatic line wraparound and margin adjust of mixed documents.

Compared to conventional text systems, where forms can be filled out only by way of complicated screen masks, filling out forms is considerably facilitated. The actual combination of text and form is now left to the printer's hardcopy. In the case of Textfax, writing can be done on the true-to-original form featured on the screen with signets and preprinted matter. The form can be signed without bothering about the softcopy paper. The forms are stored as facsimiles in the image file from where they are called up if needed. Figure 2 presents an outline of possible functions of a modular editor for integrated text and facsimile processing.

Filing

Digitalized images involve large data volumes. While a DIN A4 page with text, alphanumerically coded, requires approx. 1.5 Kbytes, approx. 0.5 Mbytes are needed for a non-compressed DIN A4 page scanned with Group 3 resolution. On a double-density floppy disk for writing on both sides, roughly 600 text pages could be accommodated, but only two image pages. The previous figures could be increased roughly 10 times by taking recourse to single disk systems tailored to local office implementation. It is therefore necessary to develop efficient compression codes, for filing both black and white images and rastered grayscale images. Depending on image content, compression factors around 10 should be expected. Viewed from this angle, too, the floppy disk and the disk seem to be suitable for short-term filing of facsimiles only. For long-range image filing, one therefore has to avail oneself of either central image data bases with magnetic disk, magnetic tape and microfilm, or develop new image mass storage for local use at the workstation. The videotape might be one way to solve the problem. A two-hour tape should accommodate some 10,000 Group 3 facsimiles. Another way is the optical data disk for 10^{10} bit from Philips, which is capable of storing at least 2,500 Group 3 facsimiles.⁷

In the text file, e.g. on floppy disk, so-called document heads of images and mixed text/image documents are stored. They are prepared and managed like texts. The document head contains a list indicating the files in which the various component parts of the document are stored. Text and image shares can be stored in different media.

Transmission

The text/image documents prepared at a Textfax workstation can simply be transmitted as Group 3 facsimiles. To this end—as is the case with the softcopy—text components are converted into a facsimile and combined with the image components to form a full-page facsimile of approximately four million pixels. Implementation of the transmission procedure and the compression code (modified Huffman run-length code) in compliance with CCITT Standards T.30 and T.4 is the precondition for international transmission of text/image documents via telephone network to all Group 3 remote copiers, and their true-to-original output.

Due to the standardized scan/plot speed of max. 1 line/5ms a maximum transmission rate of $1728 \text{ bits}/8 \times 5 \text{ ms} \approx 44 \text{ kbit/s}$ with an assumed mean compression factor 8 is possible in traffic with Group 3 facsimile equipment without full-page buffer. However, over analog lines of the telephone network, transmission can be done at a rate of 4.8 kbits/s only, or 9.6 kbits/s at most. Depending on content, the transmission of a page thus takes approximately 1 minute. Transmission via data networks or future digital telephone networks at a signal transmission rate of 48 kbits/s would result in a transmission time of approximately 10 s, which would be in keeping with the maximum plotter speed.

If a text page is not transmitted as a facsimile, but alphanumerically coded, only approximately 3 s. are needed

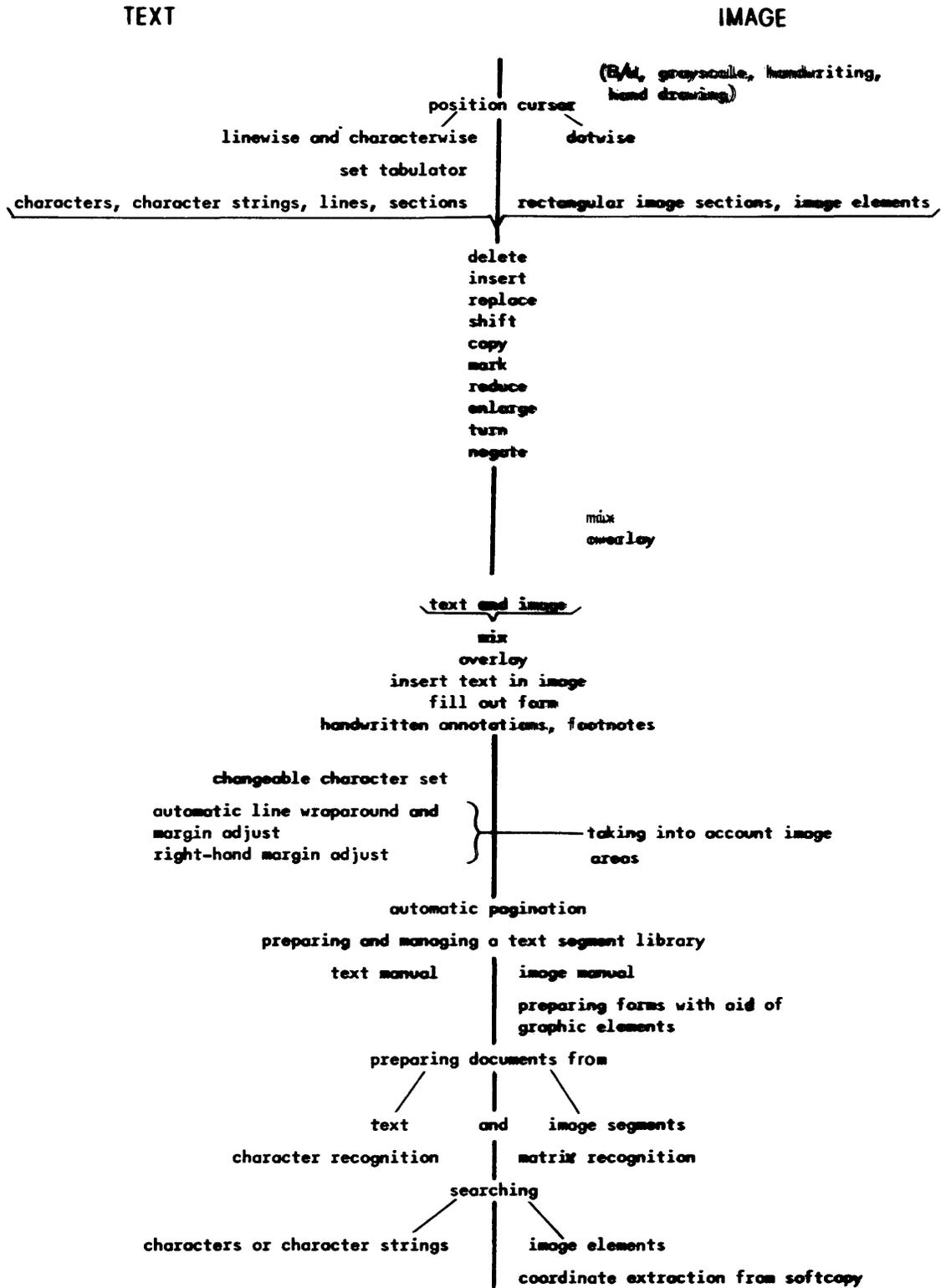


Figure 2—Processing.

TEXT

- Alphanumerically coded
 - Compatible with text stations
 - 96-character set (national)
- Character set for Latin alphabets taken from a basic table (international alphabet No. 5) and an extension table with special characters and letters
- ISO 7-bit code
 - Free text formatting on standard size longitudinal and transverse
 - Arbitrary margins, exponents, indexes, single, 1,5 and double line spacing
 - Standard 1/10 inch character spacing
 - Variable character size
 - Variable font

IMAGE

- Pixel lines compressed with modified Huffman run length code
- Compatible with CCITT T.30
- Vertical resolution 7.7 or 3.8 lines/mm
- Horizontal resolution 1728 pixels/215mm
- Any image position

- Automatic textfax switchover
- HDLC protocol
- 7 logical transmission levels
- No transmission in the opposite direction on the same circuit

Either

- Transmission on request, addressing in document header (send control protocol)
- Document header contains answerback code, phone number, distribution, date, Re:, etc
- unmanned, automatic connection setup and transmission from send to receive buffer (mailbox) with automatic dialing and automatic retry parallel to local processing
- manned transmission setup with telephone and manual switchover between voice and textfax transmission
- manned and unmanned reception with either automatic storage and indication of "incoming mail" or automatic hardcopy
- Recall of texts from the incoming mail buffer to the screen for further processing (delete, hardcopy, file edit, handwritten comments, forwarding)
- Virtual image transmission
- Automatic journaling with document headers

Figure 3—Transmission.

instead of one minute at 4.8 kbits/s, thus entailing substantial savings in transmission costs.

To be able to transmit and receive simultaneously with local processing without interrupting either process, one would ideally need a multipage send or receive buffer. Among the eight CCITT test documents the modified Huffman Code has the lowest compression factor with 5.2 with document No. 7.⁸ Correspondingly, the buffers would have to have a capacity of $n \times 0.8$ Mbits.

It is thus obvious that for reasons of economy—i.e. to operate with minimum transmission times and small buffer storages—care must be taken in transmission of mixed text/image documents that text components are transmitted in alphanumerical coding, if possible. Depending on the content of the document, switching between text and facsimile mode must therefore be possible during transmission. The most economical buffer storage capacity has still to be found by way of statistical analyses of the documents arising at the office. Assuming a *one*-page buffer for a mixed document consisting of text and image at a 50:50 ratio, we would thus arrive at a capacity of about 6 kbits+0.4 Mbits, i.e. approx. 0.4 Mbits. A page having been sent, sending can only be

continued after the receiver has confirmed the empty state of the one-page buffer.

To facilitate text transmission between Textfax and text stations, such as memory typewriters which are mapped out for communication, one should rely on the transmission standard for text stations during the text transmission phase. Corresponding international standards have not been established as yet, but they can be expected within the next few years.

Concerning transmission, seven logical levels which will be outlined by the following can be distinguished.

- The physical level with interface to the transmission equipment with signal and signalling lines for connection setup and clear-down in manned and unmanned operation. If Textfax station and telephone are connected to the same telephone line, signalling during connection setup is required to distinguish between language and non-language transmission (corresponding to the data tone). Also, in the case of conversation-interrupting transmission, the receiver must be signalled the

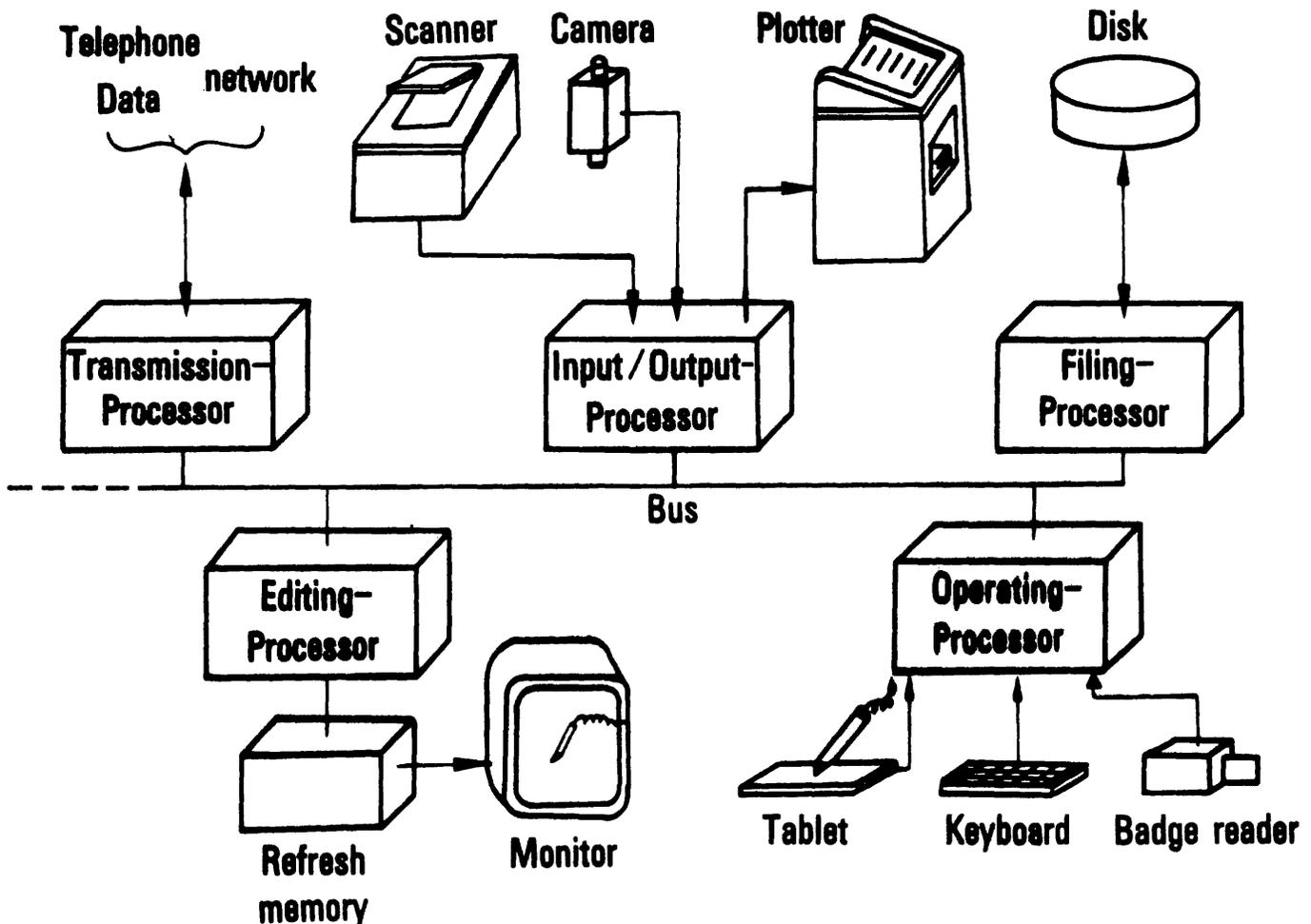


Figure 4—Multiprocessor structure of the experimental workstation.

end of Textfax transmission. Basic CCITT standards for Level 1 are

- Standards V.26 and V.27ter for modems, and V.25 for automatic calling equipment, in the case of transmission via the analog telephone network.
 - Standard X.21 in the case of transmission via data networks or digital telephone networks.
 - Standard X.21 as a subset of X.25 in the case of packet switching networks.
- The link level with the code-transparent, full-duplex HDLC protocol for protected transmission of text and facsimile blocks.
 - The packet level which is required only in case transmission is to be made via a packet switching network.
 - Level 4, which initiates the specific terminal and user-related control procedures for which no standards exist so far. For this control level, control characters must be defined to identify the further transmission procedure as text transmission, Group 3 facsimile transmission or combined text facsimile transmission.
 - Level 5 with which the end-to-end logs start out. This level provides for mutual identification of the stations according to duplex capability, character supply (alphabet, font), resolution, compression code, automatic multi-page reception, etc.
 - Level 6 as the user level. It comprises password check, format selection, form number, beginning and end of a

page, if necessary change of transmission direction, etc.

- Level 7 encompasses terminal control functions contained in the transmission code, such as control characters for text formatting (beginning of line, line spacing), change of font, change of character size, image position, text/fax switchover control characters, interrupt signals, etc.

In Figure 3 possible features of transmission services of future Textfax stations are listed. In traffic between business partners using the same forms, preprints, etc., "virtual image transmission" is important. A form, for example, is transmitted only as a name under which it is stored in the sender's and the receiver's image files, together with text and position specifications for combined copying of text and form at the place of reception.

FIRST RESULTS

Textfax experimental workstation

After this look into the future, let us now turn to the present state of our research work. In order to determine how the functions of Textfax can be implemented and how they will be accepted by the user, an experimental workstation designed in modular multiprocessor technique, which will incorporate the previously-mentioned performance

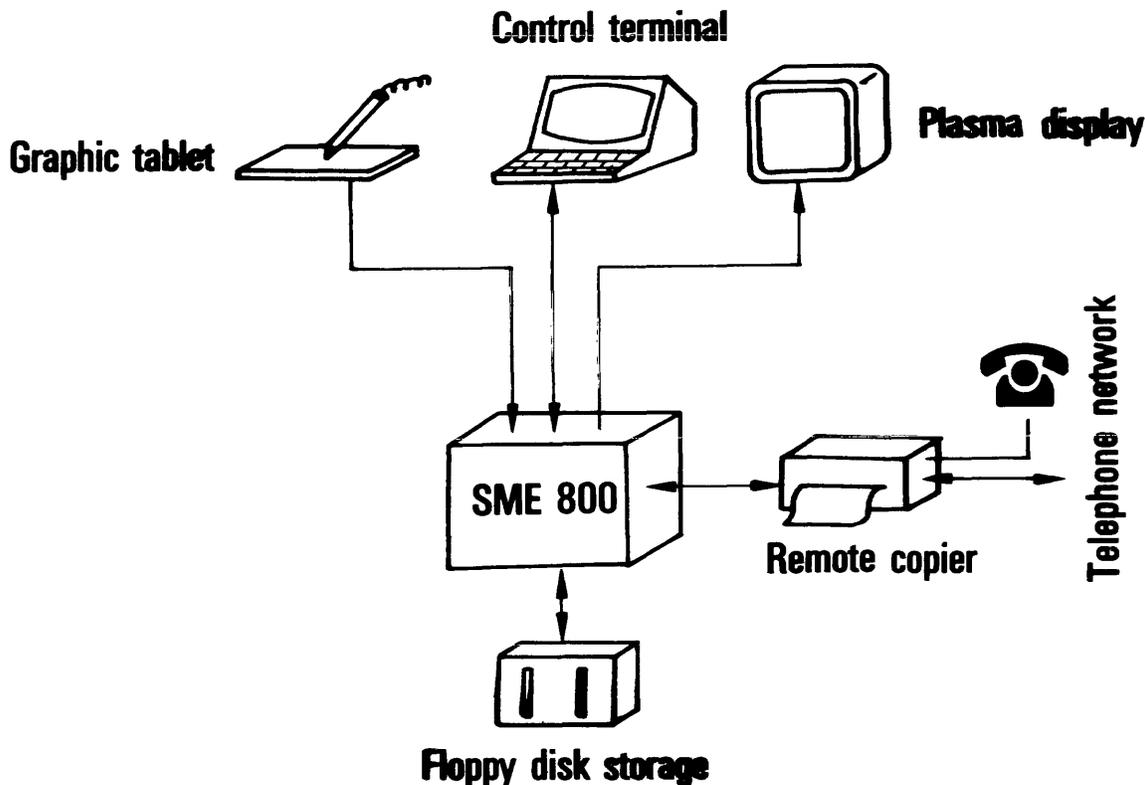


Figure 5—Initial configuration of the experimental workstation.

SIEMENS

Siemens AG · Postfach 83 27 29 · 8000 München 83

F a .
 TEXTFAX GmbH
 8000 MUENCHEN 83

Bestellung Nr. 168.320
Datum 20.10.1978

Diese Bestellung erteilen wir Ihnen zu unseren unten und unseitig stehenden Bedingungen. Die beiliegende Auftragsbestätigung erbitte wir ausgefüllt zurück.

pps. *Huber* i.V. *Maier*
 (Dr. Huber) (Maier)

Datum unserer Auftragsgabe (auf Lieferchein angeben)		Ihre Zeichen / Ihr Angebot	Pos.
		703-008 M/A	
Unser Auftragszeichen	Unsere Abteilung / Bearbeiter / % Durchwahl		
129400-9583-7012	AZE Schmiedt (0 89) 67 82-123		
Vomandem/Bestimmung - Bahnhof/Hafen		Versandt/Versandvermerk	Teiltelefon je - nein
Siemens AG Zentrale Forschung und Entwicklung Warenannahme Otto-Hahn-Ring 6 Bahnstation 8000 München 83 München-Giesing zur Weiterleitung an Schmiedt, Raum 1531		guenstigst	
		Liefertermin	1.01.1979
Position-Nr.	Bezeichnung der Lieferung/Lieferung	Menge/Einheit	Preis DM / Einheit
1	TEXTFAX-SYSTEM TFX 15 mit Aufstellung gemäss nachfolgender Skizze	1	62000.-
		Auftragsbestätigung 1-fach an Siemens AG ZFE FL KA Postfach 83 27 29 8000 München 83	
		Liefer- scheine	
Rechnung 3-fach an Siemens AG ERP Z		Postfach 185 8000 München 1	

Figure 6

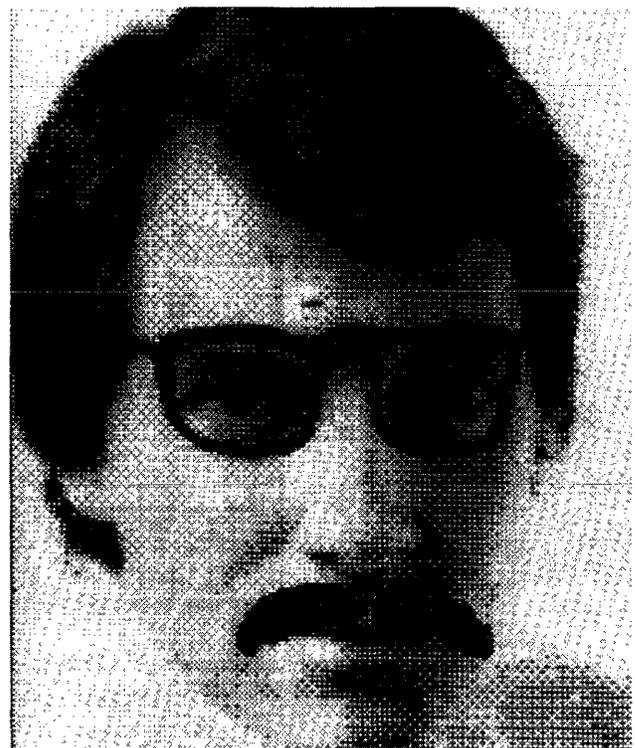
characteristics, is to be set up in the research laboratory of Siemens AG. The multiprocessor structure is shown in Figure 4. The first stage of this experimental workstation was essentially the configuration shown in Figure 5, which was used to carry out the processing activities described in the next section.

The configuration in Figure 5 is based on the Siemens microcomputer SME 800 with a 64 kB internal memory and two 250 kB floppy disk drives. Text entry and operation of the workstation are handled by a control terminal. Actual text and facsimile processing is performed on a per window basis on a 512×512 dot plasma display showing approximately one fourth of a standard size page in Group 2 resolution. The Group 2 remote copier HF 1048, attached via a digital interface, serves for data entry, hardcopy output and transmission of text/image documents. The graphics tablet can be used to insert handwritten drawings or comments in the softcopy. This configuration may be viewed as an expansion of the remote copier to include local processing functions for text and image.

An editor permitting the elementary functions listed in Figure 2 to be used on black and white images and rastered grayscale images was prepared for processing and mixing text and facsimile. Images are rastered according to the method illustrated in Reference 9, where different grayscale levels are represented by different dot densities. A special redundancy reduction method was developed offering storage economy combined with a low compute time requirement.



Figure 7



NAME: HUBER OTTO
 GEBURTSTAG: 15.12.1945
 PERS. NR.: 1589812

Huber

Figure 8

Processing examples

Example 1 in Figure 6 shows an order form filled out at the Textfax workstation. The form, which previously has been scanned via the remote copier and stored in a forms file, is displayed on the screen where it is filled with text in a window-by-window fashion. The start of the text line is marked by the cursor which may be positioned to any dot. Font and character size may be changed even within a line. When the text has been entered in the form, the drawing is recalled from the image file and copied into the order form using the cursor for positioning. The order is now ready for approval. After this, it is transmitted as a facsimile from the buffer of the employee's Textfax station to the station of the

purchasing manager, who signs the softcopy before sending the order on to the remote copier station of the supplier.

Example 2 shows processing of a rastered grayscale image. A photo was scanned by the remote copier, the resultant analog information was digitized with 4 bits per pixel and rastered using a 4×4 dot Dither matrix. Figure 7 illustrates the softcopy of the raster image on the plasma display. The black and white image can now be processed. Figure 8 shows the same head done up with glasses, moustache, data relating to the individual, and the signature. The same procedures may be used for producing wanted persons pictures (composite drawings) on a police workstation.

REFERENCES

1. White, B., "A Prototype for the Automated Office." *Datamation*, April 1977, pp. 83-90.
2. Postl, W., and W. Woborschil, "TEXTFAX—Funktionelle Integration im Büro der Zukunft," *Siemens Forsch.- u. Entwickl.-Ber.* Bd. 8, Nr. 1, 1978.
3. Troxel, D. E., "Computer Editing of News Photos," *IEEE Trans. Systems, Man, Cybern.*, Vol. SMC-5, 1975, pp. 625-629.
4. Troxel, D. E., and C. Lynn Jr., "Enhancement of News Photos," *Comp. Graphics and Image Proc.*, Vol. 7, 1978, pp. 266-281.
5. IBM Europe, "Common Control Procedures for Teletex and Facsimile Services," *CCITT COM 1*, No. 51-E, Nov. 24, 1977.
6. Shemer, J. E., and J. R. Keddy, "The Soft Display Word Processor," *COMPUTER*, Vol. 11, December 78, pp. 39-48.
7. "10¹⁰ bits optically stored on a disc," *Elektronik 1978*, No. 15, pp. 31-34.
8. Musman, H. G., and D. Preuss, "Comparison of Redundancy Reducing Codes for Facsimile Transmission of Documents," *IEEE Trans. on Comm.*, Vol. COM-25, No. 11, November 1977.
9. Judice, C. N., "Digital Video: A Buffer-Controlled Dither Processor for Animated Images," *IEEE Trans. on Comm.*, Vol. COM-25, No. 11, 1977, pp. 1433-1440.

Microcomputer programming skills

by C. WRANDLE BARTH

Goddard Space Flight Center—NASA
Greenbelt, Maryland

One of the most overwhelming aspects of the microcomputer revolution has been the speed with which hardware costs have plummeted. The software versus hardware costs ratio which prompted much of the interest in software engineering in recent years has grown dramatically as the parallel development of the microprocessors pushed down the cost of computer systems and their proliferation fueled software demand. This is increasing the need for skilled programmers for microsystems, part of which will be met by programmers currently working on big systems. In this paper we will examine some of the differences in skills and techniques that one can expect to encounter in the transition from programmer to microprogrammer. (The terms *microprogram*, *microprogrammer*, and *microprogramming* are frequently used in reference to the firmware or microcode of large machines. We will be using such terms only with reference to microprocessors.)

THE CONVERSION FROM PROGRAMMER TO MICROPROGRAMMER

Microprocessors are being used in a wide variety of applications. The programmer whose previous experience is with large batch systems may find that microprogramming for some of these applications requires a number of new skills. The hardware architecture, which high-level languages and operating systems have kept hidden from him on the big computers, may now be the primary thing with which he is working. Even the systems programmer who is accustomed to working at the level of assembly language and the machine architecture may find he must dig another level deeper, down to the logic diagrams and gates of the system. The *ands* and *ors* may have a familiar ring, but there are new things to unravel. If the application includes hardware development, signal timing diagrams will have to be read and understood to ensure the equipment being developed will work together; and testing and debugging now require oscilloscopes and probes as well as the more common traces and dumps.

These new skills are of particular importance when developing special-purpose systems that include microprocessors. When the hardware is being developed along with the software like this, one of the most important aspects of the work is the hardware/software tradeoff. Many of the func-

tions to be developed can be accomplished in either hardware or software, or a combination of the two. The greater the designer's understanding of both areas, the more valuable he becomes in being able to create an optimal system.

If one is writing programs for micros in EDP, the changes in skills and techniques are much less pronounced. It will still be necessary to be more familiar with the hardware than would be required on large systems (although not to the degree needed in systems involving dedicated microprocessors). This is mostly prompted by the limited number of sophisticated tools currently available. Even though this is certain to become less of a problem as the necessary tools are produced, the EDP microprogrammer will have to bridge the gap for some time to come.

Tumbling costs are also making real-time EDP systems more viable. Programmers of such systems may have fewer hardware concerns than those programming dedicated microprocessors; but the interfacing is certainly much harder than for the batch-style EDP systems to which their experience may have been limited.

Many of the skills and techniques required in programming microprocessors are dependent on whether the micro is part of a dedicated, general-purpose batch, or general-purpose real-time system. These distinctions, in fact, are often overlooked in the presence of the much more obvious, but somewhat more superficial transition from maxi to micro. Although much of our discussion will apply to all areas of microprogramming, parts will be appropriate only to certain applications. We will attempt to distinguish among these areas in the topics that follow.

THE CONVERSION TO PROGRAMMING DEDICATED APPLICATIONS

With each new decrease in the cost of computers, new applications for their dedicated use have become economically justifiable. The maxis were limited to the really huge projects—defense, space and production control for large factories and utilities. The minis brought computer power to the smaller factories and research laboratories. And now the micros are making computers a part of everything from cash registers to video games. Those who are new to programming-dedicated microcomputers will not only need to learn techniques they never used in batch work, but they may

also find themselves using some techniques different from those of the traditional real-time programmer as well. The old ponderous real-time applications were frequently of a nature so large or critical that the only reasonable testing approach was simulation. With the smaller systems, it is more common to be able to test directly on a working prototype, using hardware test equipment instead of simulation.

As we have already noted, the knowledge of hardware/software tradeoffs is particularly important in projects in which both are being developed together. But such tradeoffs are not easily assessed. The cost of things accomplished in hardware recurs with every copy; the software costs occur only once. Thus there is a tendency to make the hardware "weak" for economic reasons. But this pushes more and more complexity into the software while trimming the hardware capability to a bare minimum. And when the application requires every last ounce of hardware power available, other things may have to be sacrificed. Such things as the use of high-level languages, various structured programming techniques, and the avoidance of coding tricks quickly fall prey to such an environment. Building systems that are friendly to the ultimate user requires resources that may be viewed as luxuries that cannot be afforded. Even high reliability or high programmer productivity can be lost when developing in a minimal hardware system.

THOSE PAINFULLY PINCHING SHOES

Developing programs in a minimal environment has always been burdensome. Dijkstra spoke of the earliest computers that were slow and whose memories were too small as "painfully pinching shoes."¹ He said that the first programmers were pushed into coding tricks in the machine language and that they viewed programming primarily as the process of optimizing the efficiency of the computational process. Now, close to three decades later, we are faced with a brand new shoe of the very latest style, but one that is just as pinching as ever.

Many programmers view the microprocessor's relatively slow speed as immediately ruling out the use of high-level languages, regardless of the application. Optimization becomes glorified again, even optimization for its own sake. We know that optimization during coding is very error-prone. And with a dedicated computer, cycles saved on non-critical paths cannot even be made available to "other users." The use of programming tricks makes software even more unstable in these environments. (I should mention that I am not talking about programming idioms—such things as subtracting a register from itself to clear it—that become the universally acceptable way of accomplishing a task. It is the perversion of an operation code into a meaning unclear to other humans that often leads to errors.) In general, the smaller the excess hardware capacity over that minimally required, the harder our systems will be to program; the more the we will be pushed into poor programming practices; and the less reliable the end product will be.

But these are not the only aspects of machine smallness that the new microprogrammer faces. An easily overlooked

difference is that less help is available from the operating system. In many cases, there is no operating system at all; just a bare bones machine. But even where some services are available, they are far removed from the facilities that were taken for granted on the maxis. The programmer cannot count on a supervisor fixing up a division by zero, or recovering from an input error. His code must do more checking and correcting for itself.

STRUCTURED PROGRAMMING ON MICROSYSTEMS

Structured programming has always required a healthy helping of common sense. The programming manager who decrees that "all programs shall be structured" is quick to discover that really terrible programs can be written without a sign of a **goto** and with no module exceeding 50 lines. Structured programming is not a blindly mechanical process, nor is it a panacea. The more demanding the external constraints, the more that sound judgment is required in applying the principles of structured programming; but also the more that the discipline will pay off.

Some people question whether structured programs are sufficiently efficient for use on micros. Of course, many of the techniques that have been loosely grouped under the name "structured programming" really have no bearing on efficiency at all. And while I've already indicated that I feel optimization is sometimes overemphasized, there is no doubt that certain constraints must be met, particularly in programming real-time applications (either dedicated or general-purpose). It is true that some structures which frequently turn up are inefficient. Redundant tests occur when using only standard control structures. Branch instructions whose targets are other branch instructions may be generated by high-level languages or assembly macros for some of the standard control structures. And numerous short sub-routines can add substantial calling overhead to a program. Some of these problems can be overcome; others cannot.

Of major interest is the use of structured programming in high-level languages. As new languages are developed with the structured programming techniques in mind, we may find that better code is generated than would be for the current high-level languages. For example, optimizers can work better on structured code. Even our best current optimizers must abandon much of the optimization in program segments where a rat's nest of **gotos** prevents their deducing the flow of control. The optimizer for a structured programming language, however, could optimize every loop in the program since each begins with one of a small number of specific keywords. Furthermore, such optimizers could convert procedures that are called from only a single place in the program text into in-line code, thus cutting down on the calling overhead. But all of this is only what *could be*. What about what *is*? How should programs be written with the tools we have available now?

We know that most programs spend the majority of their time in a minority of their code. Even real-time programs are often time critical in relatively few places. These facts can allow us to take advantage of structured programming

techniques in our program development and then go back to post-optimize in those places where it is really critical. This also allows us to isolate our thinking about optimization from our thinking about the programming of the algorithm. Many of the techniques we use when optimizing are based on assumptions about the data of the program's variables. If we optimize as we code, it is particularly easy to make false assumptions that get violated as we also optimize other parts of the program. Post-optimization is both easier and surer since all of the parts of the puzzle are present, allowing us to completely verify any assumptions we must make. We can attack the problem piecemeal, assuring each change is valid before making another. And most importantly, we can direct our efforts to those parts of the program that are most apt to be worthwhile.

Other software engineering techniques can provide efficiency payoffs. The use of functional cohesion and decoupling² aid the complete replacement of algorithms with better ones. In a poorly modularized program, the code implementing an algorithm may be sprinkled about in such a way that it is impossible to change even when a better method is found. And replacing a poor algorithm with a better one can frequently yield a much higher optimization factor than can a ton of coding tricks.

The most important realization is that efficiency is a very relative thing. On a microprocessor dedicated to a single activity (whether used in a dedicated hardware system or in a general-purpose single-task system), wasting CPU time is of no concern if we are totally I/O bound and there is nothing else to do. Wasting memory is unimportant until it crosses a quantum boundary; and as chips become bigger and cheaper, the size of that quantum keeps increasing. Even when poor efficiency causes the system to be slower, it may be preferable. Efficiency considerations must be weighed against reliability considerations. Up to a point, a slow system that works is preferable to a fast one that fails (although a sufficiently slow system may be a failure by definition). A latent bug can be costly in any system, but particularly in a commercial device.

A NEW COMMUNITY OF USERS

One of the biggest changes that the microprogrammer faces is the new user community he serves. As a programmer of large systems, the "unsophisticated user" was a manager, a data entry clerk, a scientist, an engineer; with the micros it may well include a non-professional, an office clerk, a sales clerk, a houseperson, or a blue-collar worker. This is particularly true with programming for dedicated micros, somewhat less for real-time EDP, and least for batch EDP. However, even in this last case, the operator of such a system might well be the owner of a small business rather than a computer professional. The human interface of programs must take on a new aspect. These people will find words like "input," "record," or "string" strange or understand them differently and they are not accustomed to learning jargon. Moreover, they may not be just "down the hall" where they can ask you about some strange message

they receive when trying to run your program. They are going to put commas in their numbers and type the letter "I" when they mean the digit "1." And they expect "end" and "END" and " end" to all be the same—except when they want them to be different! But most important, they expect programs to work—and work right every time.

RELIABILITY

Consumers expect high reliability of the things they buy. No one expects to go to the store and find release 21.8 of a microwave oven! The closest things to "program fixes" that most consumers see are automobile company recalls. But if such recalls can hurt the images of the auto giants, think of what such bad public relations could do to a struggling company producing microcomputer-based systems.

And yet the overwhelming majority of programs are marketed while still sadly undertested. They are, in fact, so bad that it has become common to talk of buying program "maintenance." But programs don't break down; so what people are really buying is not maintenance at all, but a warranty. The programs are effectively guaranteed not to work as advertised; the "maintenance" assures that they will be fixed up when they inevitably fail. This viewpoint is well understood by people that have been dealing with computers every day, for they know that much software is quite bad. But the consumers are only starting to find out. This is not just limited to consumer-oriented systems, either. Customers have little sympathy for merchants whose real-time systems are down and prevent them from transacting business.

Is there a way to avoid bad software? There are some aids, for sure. Program certification (mathematical proof of correctness) offers the best hope, but is still far from practical for most programs. However, the more critical the application, the more certification may be justified, even if only on a limited basis. Certainly programs that are a part of automotive systems are going to require higher reliability as they take over more crucial functions in those systems. For the time being, the best insurance for most non-critical software is probably the peer program walk-through. Such walk-throughs will frequently turn up bugs faster and better than testing methods will, and they have the additional advantage of being educational. One of the biggest problems that is overlooked by many is how *will* field upgrades be dealt with, particularly the inevitable fixes. For in spite of all their preventive measures, if the handling of the correction of a bug that slips through is mismanaged, a company's whole reputation may go down the drain.

OLD LESSONS TO BE RELEARNED

There are a number of lessons that we have learned over the years that apply as much as ever to the micros or even more. To make sure we do not lose sight of them, we will review some here.

Use of high-level languages must be emphasized wherever

possible. Of course, there will be areas where the overhead is just too great and we must turn to assembly languages. But we must weigh this alternative carefully on a case-by-case basis. We know we can turn out better software faster using high-level languages. Even our "superprogrammers" can (and we all like to think of ourselves as superprogrammers). But if we expect to keep up with the growing software need, we must use the better tools whenever possible.

The same argument applies to using modularity. The general-purpose routine seldom takes much longer to write than a very special-purpose routine. And although it probably won't be quite as efficient, it can save programming time over and over if we can use it in future projects rather than reinventing the wheel.

The biggest cost in changing hardware is changing software. The big boys learned it when they were repeatedly cursed with "downward" (backward?) compatibility. In microcomputing hardware, six months is forever. The more a given system is tied to a given architecture, the more that system is going to become obsolete with the hardware. This may be fine for dedicated use in a consumer product where the time frame during which we plan on being in production matches the expected availability of the component hardware. But for general-purpose programs, software compatibility will be the key to longevity. This adds another reason in favor of using high-level languages. Even though different compilers for the same high-level language may require some changes in our programs, assemblers for different architectures will require even more.

One of the biggest differences between good programs and so-so programs is human engineering. This is all the more important when we consider our new user community. Programs should be written to be helpful and friendly to the user. In many cases he will be communicating in an alien environment and will not appreciate contorting himself to a system with a thousand rules he can't remember. Sometimes it requires a major increase in effort to build systems that are well engineered; but frequently simple changes can provide great conveniences to the users as well as providing additional selling points in a competitive market.

Patching is one programming technique that was falling into disrepute on the larger machines and is now making a reappearance. Patching object code rather than fixing the source is sometimes required as a temporary measure when testing micro software, particularly where the turnaround time to recompile the program is long. The key to avoiding errors when patching is maintaining the discipline of keeping

track of all such changes and assuring that they get back into the source. There is actually a greater problem than this type of patching, and it is still seen regularly in all phases of programming. That is the patching of source code contrary to the program's design. Sometimes such a patch is, in fact, the only reasonable way to fix a design bug—particularly the last bug in a large design. If the time to correct the error at the design level and then "recode" is too long, then the source patch is just as reasonable as was the object patch. And just as for the object patch, the documentation should include complete information regarding the error for future reference. But we should not be too quick to write off such errors as unfixable at their true source. If any further use is to be made of the program, it is all the more desirable to correct the design error. And in well structured programs, corrections in the design should be able to be sifted down into the code fairly straightforwardly, allowing the number of final modules that must actually be recoded to be quite small.

CONCLUSIONS

The need for programmers for microcomputers is great and will draw people from many disciplines. Professional programmers for large machines can expect to have a big head start if they move to micros and there will not be as many differences ahead as they might think. Most of the techniques that were applicable on the big machines are at least worthy of consideration for the micros, although the actual application may require some changes. The biggest differences, however, will stem more from the differences in applications than from differences in machine size. The change to real-time programming is the most important of these differences (particularly on dedicated microprocessors) and may entail dealing with hardware and machine architecture to a greater depth. The other primary difference is programming with a more computer-naive user in mind, and dealing with his limitations and expectations.

REFERENCES

1. Dijkstra, E. J., "The Humble Programmer," *CACM*, October 1972, Vol. 15, No. 10, p. 860.
2. Yourdon, E., and L. L. Constantine, *Structured Design*, Yourdon Press, 1975. pp. 116-186.

Program conversion—One successful paradigm

by CHARLES LYNN, JR., JEAN RISLEY and ROBERT WELLS

Bolt Beranek and Newman, Inc.
Cambridge, Massachusetts

The continuing rapid development of computer hardware over the years has been both a blessing to the user who sees his computing power increase and cost decrease over time, and a disappointment to those who have found that the cost of converting existing software outweighs the advantages of the new machines. For any proposed change of hardware, there are at least three options whose cost the user must carefully weigh—(1) the option of remaining with older hardware, (2) the option of completely redesigning/rewriting existent software and (3) the option of converting those portions of the existing software which are portable and rewriting the rest.

At BBN we have recently carried out the transfer of a large body of programs from one hardware environment to another. In the process of this task, we have developed an analytical approach to the problem of conversion of programs written in a partially independent higher-level language. Our technique consists of several distinct phases:

1. Identification of the programs to be converted and the environment in which they will be run.
2. Isolation and analysis of those language features which are not portable and of those features of the hardware and system environment which were used.
3. Design and implementation of a mechanical conversion process.
4. Creation and use of a procedure for giving human attention to special or difficult areas.
5. Conversion and debugging of the programs including definition and use of standards for debugging and program testing.
6. Creation of the operational environment under which the programs will be run.
7. Parallel operation of both systems with extensive comparison of their respective results.
8. Operational use of the converted system and the archiving of material from the old system.

Dividing the problem in this way makes it possible to keep control of the progress of the conversion effort. Also, performing a complete analysis before actually beginning to convert and run programs provides a better grasp of the scope of the work and tends to minimize surprises later in the project.

THE PROBLEM

Our specific problem was to move all of the current Management Information Services (MIS) functions of the company from the GE Timesharing Service* to an in-house DECSYSTEM-2020. Our primary motive was the reduction of the continually growing hardware cost in terms of machine time and storage charges. Those costs were much higher than those projected for the in-house machine, and the prohibitive cost of increasing service and development of new software effectively precluded major improvements. We were also constrained by the absolute requirement for continuity of performance—the change of environment had to be transparent to the company.

Throughout the conversion project, there were a few basic policies. The primary constraint was that the converted system behave "exactly" like the old one. This provided a strict guideline on program rewrites—nothing was to be rewritten if it could be done in the old way.

The body of programs comprising the MIS system to be converted consisted of approximately 450 source program files (70,000 source lines of code) written in three different dialects of FORTRAN (\$FORTY\$,¹ FIV² and F77⁴). There were both batch and interactive processes, and 15 hierarchical data bases with a total of 65 record types were used by the programs. The system was developed almost entirely by internal BBN MIS development personnel over several years, with some support software such as the data base management system provided by the time-sharing vendor. Program documentation was generally limited or out of date, with most documentation consisting primarily of user instruction memos.

ANALYSIS

During the analysis phase of the conversion, we divided the potential language and environmental conversion problems into specific analysis areas, with responsibility for one or more areas assumed by each member of the team. The main areas were (1) character and string handling data types

* GE is a trademark of General Electric Company; DEC, DECSYSTEM-10, and DECSYSTEM-20 are trademarks of Digital Equipment Corporation; System 1022 is a trademark of Software House.

and features present in each of the GE FORTRAN dialects but absent from the DEC FORTRAN;³ (2) data base issues (HISAM hierarchical data bases on GE,⁷ relational data-bases provided by System 1022¹⁰ on DEC); (3) input/output and file handling language features; (4) system-provided features (subroutine packages, monitor commands, SORT/MERGEs, interrupt error handling, etc.); (5) other language feature incompatibilities (ENCODE/DECODE, initialization statements, structured programming features, etc.); and (6) other facilities used, such as the data base report and batch stream command languages.

All the source code files were initially transported to the DEC system where they were kept on-line continuously. During the analysis phase we used the vendor-supplied manuals to locate potential incompatibilities in the languages, and then searched the existing on-line sources for actual occurrences of any potentially troublesome syntax.

We specifically designed a search program to accept a set of search patterns and a set of exclusion patterns, specified as regular expressions, and scan a specified group of files for line matches, which were then listed with their locations. Exclusion patterns were used to eliminate non-interesting matches, such as those found within comments. This search program and all the programs used to mechanically translate the FORTRAN sources were written in FASBOL,⁹ a compiled dialect of SNOBOL for the DECSystem-10 and DEC-SYSTEM-20 systems. SNOBOL⁵ is a powerful string processing language, and it proved ideal for the rapid development of these conversion programs.

The information collected during the analysis phase was circulated within the whole MIS group by an on-line message system (HERMES⁶). This was useful both for communication of specific points ("NAMELIST I/O is never used," "there is a lot of EQUIVALENCE with character data variables," etc.), and to provide on-line hard copy of problem areas by topic as a guide for later phases. There were group reviews and discussion sessions on the findings in each area based on the collection of messages. This information was also circulated to the development group so that they would be aware of language features that should be avoided in their development work.

MECHANICAL CONVERSION

The design of the mechanical conversion itself began with listing, by area, the specific mappings of syntax which would change a feature in one of the GE FORTRANs to acceptable DEC FORTRAN. For example, in an unformatted read, map 'READ. . .' to 'ACCEPT *.. .'; or in a type statement with initialization values, 'remove the initialization values and create a DATA statement for the variable.' Specific translations were grouped into 17 processes, each of which became a FASBOL program.

Each conversion process was conceived as a filter, taking an input file and applying a set of translations to it. The intermediate forms between these processes were not, in general, legal FORTRAN programs; for example, one of the initial filters concatenated continuation lines together to

make lines hundreds of characters long so that later filters would not have to be concerned with continuation lines. Another filter analyzed each routine to develop symbol table information about the variables used, then distributed this information as control character sequences to all occurrences of the variables. This allowed later filters to trivially determine such information as the type and dimensionality of any variable. The last FASBOL filters in the sequence removed such alterations to produce valid FORTRAN programs. Due to the largely unstructured nature of SNOBOL programs, we felt it would have been much more difficult to develop only one large SNOBOL program to perform all the translations; most of our filter programs were under three pages long and understandable as a whole.

The ordering requirements of the translation processes, shown in the graph of Figure 1, evolved based on special requirements of the processes themselves. For instance, the process to reconcile OPEN, CLOSE and unit assignments (OPCLOS) needed to occur before the one resolving problems of format modes and expressions in I/O lists (IOLIST) in order to locate FORMAT statement numbers in the READs and WRITEs. Figure 1 also shows, for example, that the relative order in which SHORT and STATE were executed does not matter, but both require LINES to have been run and both must be run before any of the other routines (DBCALL, IOTRIV, or ENVIR) may be run. The execution order which was actually used for the 17 modules is indicated by the numbers in the upper left corner of the boxes in Figure 1.

For many features, a choice had to be made between conversion to acceptable DEC forms and emulation of the GE-provided features. In most cases we favored conversion, since we hoped to minimize long-term maintenance problems caused by the holdover of alien conventions. However, the use of the HISAM data base calls and the representation of character data in strings was so difficult to convert to DEC that it was decided to emulate these functions in the converted system.

The HISAM hierarchical data base structure so dominated the logical structure of the programs that conversion of the programs to make optimal use of the System 1022 relational data base facilities would have required almost total restructuring and rewriting. We therefore decided to emulate the HISAM data base routines where possible rather than undertake extensive rewriting. The HISAM emulation consists of 14 external routines and several internal routines, comprising 1700 lines of source code. The package was written in RATFOR,⁸ a structured FORTRAN pre-processor, because of the readability and cleanliness of the structural statements it provided.

Character processing in all the FORTRAN dialects, including the target DEC language, is very dependent on the number of characters that fit into single- and double-word variables. The GE code was firmly based on four characters per word and could not be practically mapped into the five-character words that are used for literals and support routine calls on DEC. We therefore provided a minimal preprocessor (MISFOR) for long-term maintenance of the code. This

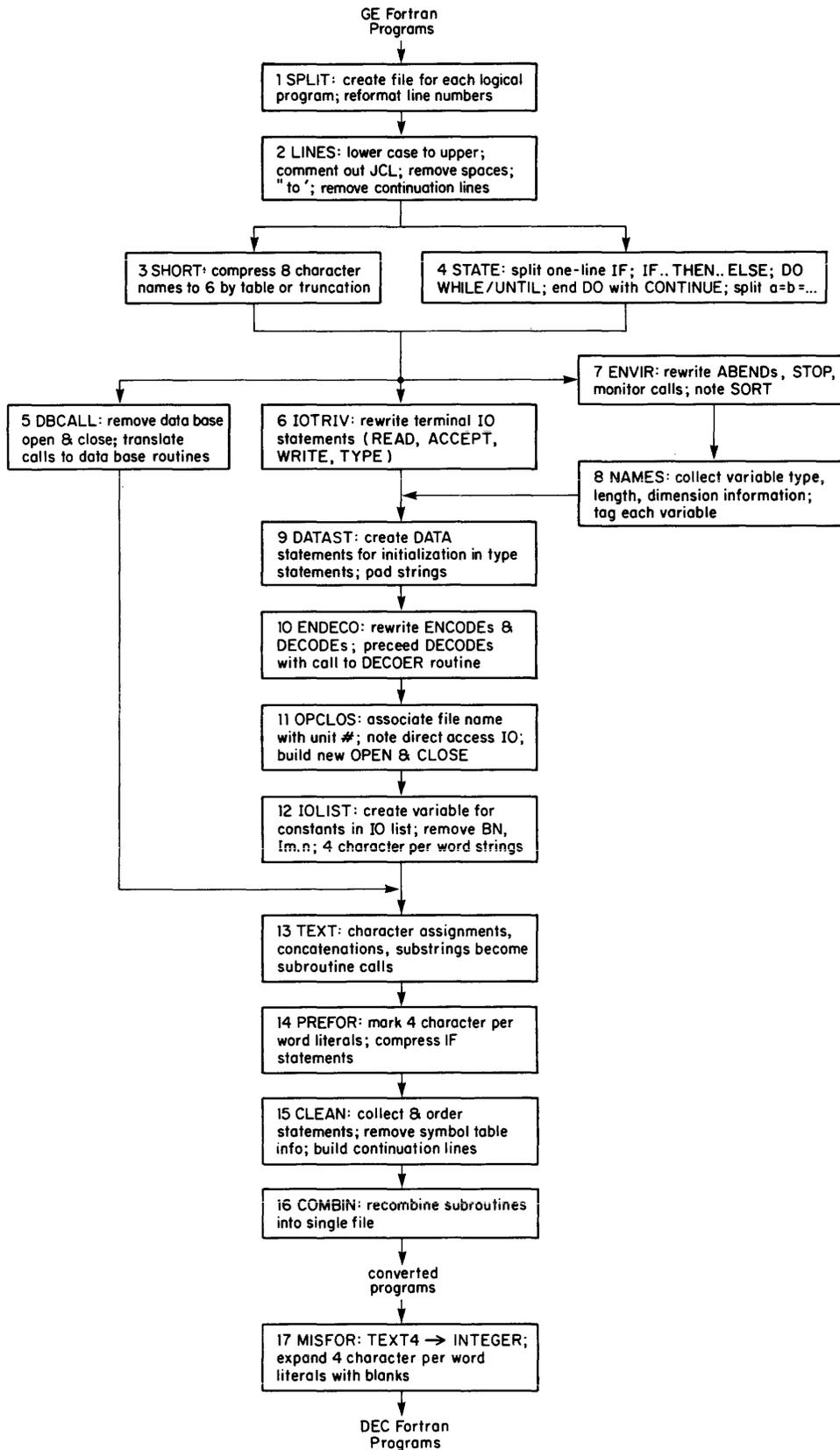


Figure 1—Execution precedence graph of translation processes.

preprocessor trivially translates a data type declaration line (TEXT4 to INTEGER) for variables containing four character/word data and translates syntactically-marked literals by inserting a blank every four characters so that DEC FORTRAN will effectively store four characters per word. Support routines have also been provided to convert four-character-per-word strings to the five-character-per-word strings needed for interfacing with DEC system routines.

Many capabilities that were language features on the GE system became calls to subroutines from the newly-created support library for the converted code. For example, the error return on DECODE, character assignment and comparison statements, and the INQUIRE for file status all became subroutine calls during the mechanical conversion.

MANUAL CONVERSION

Once the mechanical translation had been completely specified and developed, there remained a list of problems that required manual intervention. These included cases of more complicated syntax which were not used often enough to justify complete treatment in the translation programs, as well as issues such as changing file name conventions or sort command syntax which were not simply decidable. The mechanical translation both listed these troublesome items and placed a comment before them in the source program.

For the programmer performing the manual checking and editing there were a number of reference materials and memory aids. First, there was search program output with line number locations of specific potential problems, (SORT, overlays, possible bit fields, PROCEDURE calls, monitor calls, etc.). Second, there was the output of the mechanical phase with error messages marking possible problems. Third, there was a summary document and a complete specification of subroutines in the support library with calling conventions. Finally, there was a step-by-step document listing all issues expected to arise during the manual conversion. These reference materials were the primary force for completeness and consistency during the manual conversion, and they made it possible for each team member to recreate the conversion environment even when there had been distractions to other tasks.

PROGRAM CONVERSION AND DEBUGGING

For conversion, the program set was divided into subgroups of 10 to 30 programs each, usually by similarity of function or access to the same data sets. These subgroups were generally subsets of the functional groups defined by the operational system. The fact that most of the programs are organized around file input and output made it relatively straightforward to isolate each program. The actual conversion of programs tended to consist of four steps—(1) a preliminary (sometimes optional) manual edit, (2) the mechanical conversion, (3) a finalizing manual edit and (4) extensive debugging and testing. In general, each person tended to convert 10 to 20 related programs at once, moving a whole

group of programs through all of the steps. There was specialization within each group, with particular project members being most expert at understanding the complexities of sort conversion, data base usage, or data base report language usage. However, each member was responsible for conversion of groups of programs from beginning to end, with consultation if needed with the appropriate specialist.

During the manual conversion we discovered that some repeated sections of code occurred sufficiently often to merit inclusion in the support library. The requests for interactive input (TYPE, FORMAT, ACCEPT, FORMAT) were so frequent that we created functions to receive literal messages and return a value of the appropriate type. Similarly, a function was added to locate and format the date for use in many of the programs. In general, however, we tried to minimize the number of subroutines introduced so that our efforts in the conversion phase could be concentrated on conversion rather than tool development.

While most of the actual program conversion was straightforward, a few problem areas were encountered. Some of the programs required history or parameter files which had not been transferred to the new system; short files were usually re-entered manually while longer ones were transferred via tape. The time-sharing system used for the first part of the conversion (before our own DECSYSTEM-20 arrived) crashed frequently; usually only a few minutes work was lost although at one point it crashed, was down for a week, and somehow managed to lose several additional days work.

The most troublesome problems involved the parallel modification of programs. One situation involved GE programs which were modified between the time they were transferred to the new machine and the time they were actually converted; checking the dates on which the programs were last modified immediately before beginning to convert a group of programs prevented conversion of outdated modules. A second situation involved GE programs which were modified after they were converted. When this happened, the modified program was either again transported to the new system where it was mechanically compared to the earlier version, or else the development personnel making the GE modifications also modified the DEC version. The latter procedure, when not carefully controlled, led to a third situation—two people modified a converted program at the "same" time. This problem was later avoided by making each program the responsibility of one individual and making sure that no one else modified it without first checking with the program's nominal owner.

As each program was converted it was debugged. Debugging generally began with synthetic input and then proceeded with copies of actual input from the running system. The use of synthetic data made testing of programs which performed input validation and editing operations much easier since the actual input data was voluminous and would rarely, if ever, contain all possible errors. The use of large amounts of actual input from the running system made validation of the computations performed by the programs easy to check since the correct answers were already available.

It should be noted that when a discrepancy was found, it was not always because the converted program was wrong; we found several bugs in the running system. The action to be taken in such cases required careful consideration. Repairing the operational system was not always desirable—in one case it required that many people change the way values were interpreted. Therefore, the converted program was modified to run in a "simulate-the-bug" mode. In cases where numbers which were printed on reports were truncated (without any indication by the GE system) the field widths were increased. Since the data in the reports was not generally re-entered into the system, these differences did not lead to further discrepancies. In several cases, bugs in the operational system were fixed.

A disturbingly large portion of the debugging time was consumed by bugs, some amounting to a comedy of errors, which were not related to the conversion. Bugs in the DEC SORT utility provide many examples. One of its numerous errors was traced to a record in a sorted file whose record length was changed by the sort. After a couple days we were told that the bug would be fixed in the next software release. We could get it shortly but it required the next monitor release because it used new monitor calls to parse the new command string format. However, that version of the monitor was not scheduled for release for two months. Thus, all the calls to the sort utility had to be found and changed to conform to the new syntax and a routine had to be written to emulate those functions of the new monitor which the new sort used.

BATCH STREAM DEVELOPMENT

Most jobs on the GE system were run in overnight batch mode. The control file was built and submitted by a program which used a prototype and operator responses to simulated program queries. In addition to running programs, the streams created temporary backup copies of some files (it was too expensive to backup all of the files and data bases) and performed other file maintenance operations. The command language made built-in recovery mechanisms so hard to implement that none were used. Development personnel were frequently required to perform manual recovery operations when a service interrupt occurred during a job since time did not permit restoration of files from the normal GE dumps (that would require waiting until the next night). The batch streams were totally rewritten because of the large differences in the command languages and operating environments and because of the desire to improve error detection and recovery mechanisms.

The operations environment we developed on the DEC system is similar to the GE system in many ways. Batch control files are built from prototypes and operator responses; specially marked strings in the prototype, the questions, are replaced by the responses. Several advantages have resulted from running all programs from batch streams: the operations staff is free to use their terminals for other tasks, a log is automatically created for each job, and the running environment is consistently defined. Since space is

available, all major files and databases which are to be modified by a job are first copied; "deleted" files are only logically deleted. All intermediate steps are saved on the daily dump tapes before being physically deleted from the system. In the event of a crash or other large error it is relatively easy to backup to any specified point.

PARALLEL OPERATION

The validation of the system by running in parallel was a formidable task. The operational system worked under tight time constraints for data input processing and report distributions. Originally, the system had a fairly large component of interactive data entry and report request and retrieval. We made a few operational changes preparatory to running in parallel, replacing interactive data entry with batch entry, in order to get better control over input and changes to the system.

The parallel operation phase evolved into two parts. During the first part all inputs to the operational system, usually in the form of card decks, were collected and transferred via magtape to the new system. Every half-month (the company business cycle) the operational data bases and transaction files were dumped to tape and loaded onto the parallel system. About one day was required to read the tapes (11 2400-foot reels) and build the binary transaction files and System 1022 data bases. Batch control files were then generated from the stream prototypes based on the GE operations log and then run (overnight) in the same order as they had been run on the actual system. This operation, in addition to testing the individual programs, tested the batch streams, their interaction with the programs (mostly file management), and our ability to recover and backup to any specified point. While some bugs were found, most required excessive amounts of time to locate since the actual bug could have occurred anywhere within the simulated bimonthly period.

These problems led to a second mode of parallel operation during which the parallel system was run as close to the operational system as possible (typically within a day) with the reports being checked line-by-line (rounding differences of one cent were usually ignored). In order to isolate differences in data files as quickly as possible, two techniques were developed. The first was a detailed RECAP program, used on both systems, which calculated transaction file summaries. After a job stream was run, RECAP was used on the modified files: comparison of the outputs quickly isolated the differing transactions. The second technique was a complete record-by-record match of the contents of all modified data bases and some transaction files performed by a set of batch jobs and programs to sort and run file comparisons. A match was run weekly during full parallel operation.

Many of the differences encountered during the parallel operation were of a mechanical nature. Getting the same input data for both machines was a significant problem. The problem of dates had been foreseen and a modification to the DEC system date routine was made to allow the date to be pre-set. This was not sufficient, however, during the

latter stages of the parallel operation when the parallel system was run ahead of the operational one. Date discrepancies in runs made before or after midnight were common. This later resulted in several comparison mismatches which required further manual investigation to ascertain that the differences were only due to the differing dates.

Card input was also a problem. Before the card reader for the DECSYSTEM-20 arrived (about five months late) card decks were sent to a service bureau for transfer to tape. Cards were occasionally lost or out of order. Later, a program was written to allow direct inter-machine transfer of ASCII files. Its 300-baud transfer rate was satisfactory for small card decks but was too time-consuming for larger ones. When the card reader finally did arrive, the data became worse since dark spots on the backs of cards were interpreted as punched holes by GE's card reader but not by DEC's. Correcting the consequences of these differences was time-consuming and frequently led to the necessity of additional correction runs. These, in turn, led to comparison differences caused by differences in the unique batch number assigned to each run. This problem was solved by writing programs to selectively zero-specified fields in the files before the comparisons were made.

From the inception of parallel running the original operational personnel played an increasingly important role in the conversion, advising us on how to make the system easier to use, identifying problems in the parallel runs, and, during the second part, conducting the parallel runs. Their enthusiastic cooperation in the conversion was invaluable both in the technical side of the conversion, and in familiarizing them with the new system in advance of the actual switch to it.

PERSONNEL AND SCHEDULING

Our conversion team consisted primarily of four people full-time for 12 months, with expert consultations by several people including one of the original authors and one of the current operators of the system. None of the conversion team had significant previous MIS or business programming experience, but came from scientific, communications, or systems programming backgrounds.

In selection and support of personnel for a conversion project, our experience shows that a careful separation between conversion and development teams is very important. Some interaction is necessary to avoid conversion of obsolete segments and to understand more obscure sections, but, in general, keeping the conversion team separate from design issues permits complete concentration on reproducing the current functions. This tended to prevent a distraction into the history of the code, an involvement with all of the things the code could or should have been made to do, and any dependence on unverified premises about how portions of the code ought to interact.

We had two major overall design and scheduling checkpoints, one before actual conversion began and one half-way through the conversion effort. We decided initially to spend half of the time (six months) in analysis, approach

design, tool building and experimentation using one of the functional groups of the system, and at the end of that period the conversion procedure was essentially complete. The second half was originally scheduled to convert all the programs in three months, and run in parallel for three months. Actually we converted about $\frac{2}{3}$ of the programs in three months and did both conversion and running in parallel in the final three months. Converting programs, even with mechanical aids, was made more difficult by having to switch between conversion and other support tasks such as installing system software on the new machine or transporting test data and new source files to the new machine.

RESULTS

When the switch to operational use of the new system occurred, all of the regularly-scheduled programs and batch streams were operational. A few quarterly reports and reports which run irregularly on a demand basis had not been converted. A request for one of these reports caused it to be placed at the top of the queue of programs to be converted.

The conversion was essentially transparent to users of the new reports; except for the addition of report numbers and/or page numbers the actual reports are identical. The conversion was not as transparent to those who distribute the reports for mechanical reasons. Since the old system printed several reports interspersed with the batch log they were partially identified by their position in the printout. On the new system each report is printed separately so the report numbers must be used. This was initially a problem since reports which differed by either paging, sort order, class inclusion or exclusion, or run date had been given the same report number; addition of version numbers to these reports solved the problem.

The conversion was least transparent to those people, such as the contract and personnel administrators, who had performed interactive data base updating under the old system. During the parallel operation phase all updates were batched and performed by the operations staff. This method of operation has been retained, at least initially, under the new system until the scheduling and security requirements are resolved. In addition, several people had interactively requested reports under the old system; they must now request the operations staff to run these reports.

For the operations staff the day-to-day running of the system is structurally quite similar to the old system—only a few of the names have changed. The biggest difference involves having to run and distribute both those reports which were formerly requested interactively and the batched update results. The latter must be carefully scheduled since they have to be verified, and frequently corrected and re-submitted, by the people who are responsible for the data.

It is difficult to make direct comparisons of execution times or costs between the two systems due to a large number of unknown factors. The old system provided the user with two quantities—time of day, in hours and minutes, and "cost." The cost is an undisclosed function of at least

CPU, memory and I/O usage. The new system also provides the user with two quantities—elapsed time and run-time, both expressed in hundredths of seconds. The direct and secondary effects of multiple users in a time-sharing environment cause these quantities to increase with load but correction factors are not known. In general, the overnight runs on the old system required less wall clock time than the corresponding overnight run on the new system but the difference does not seem to be "significant" in the operations environment. This relationship can be reversed under appropriate loading conditions caused by additional users.

A complete cost analysis of the conversion project is difficult because many costs cannot be directly compared and several are not readily quantifiable. New development on the GE system directly increased costs; on the DEC system it only increases overall machine usage. The 4800-baud remote station was critical for the GE system and communication difficulties frequently delayed operations. After a service interrupt there was no way to quickly catch up. The DEC system does not require remote communications. After an interruption of service the operations staff can preempt the development group until things are back on schedule. Things can be scheduled more efficiently on the DEC system as system loading is more predictable.

Ignoring the factors mentioned in the last paragraph, a rough estimate of the cost of the GE services was about \$26,000 per month. This includes subscription fees, job and storage charges, communication costs, remote station lease and maintenance, shipping charges, etc. The costs associated with the DEC system are about \$13,000 per month. This figure includes two full-time operators, system hardware (appropriately depreciated) and software (including the SORT and System 1022 data base packages), maintenance, disks, tapes, cabinets, power and space. The one-time costs of the conversion from GE were about four person-years of labor and \$65,000 for computer usage (additional GE usage due to the conversion and portions of the initial mechanical conversion which occurred before the DECSYSTEM-20 was delivered). It is estimated that the conversion will pay for itself within three years.

CONCLUSIONS

We found that only a very few programs actually needed extensive rewrites. Some rewriting was required, for example, when either there were not enough I/O channels to access files in the same order as before, or because the emulation routines were prohibitively slow for the reorganization of some data bases.

The creation of standards for those aspects of the programs which can change is a process which has continued throughout the conversion. We found that the creation of interactive entry conventions applied consistently made the programs easier to read and debug without changing actual user interaction. Conventions for file and program names, once created and consistently applied, made program and data grouping more reasonable. Conventions and routines for accessing and processing dates, for example, provided

a basic consistency between programs that made functional differences easier to understand. In deciding to institute each of these standardizations, there was a careful weighing of alternatives in which the additional cost of implementation had to be shown to be negligible and the benefit to be considerably positive.

If we were to begin the project again, there are a few improvements we would make in the procedure. Attention would be given to the final operations environment before report naming and identification conventions were established. The HISAM emulation would have been written so that data base extensions could be made without having to modify the emulation package and recompile all programs using it. Time should have been spent to develop an efficient, reliable method of inter-machine data transfer. The tools to compare binary files, data bases and reports allowing error tolerances to be specified should have been developed sooner. Finally, ample time should have been allowed for the "unexpected"; about one-quarter of our time was devoted to finding bugs in purchased software, installation of new releases, delays in delivery, finding ways around problems which could not be readily fixed and system crashes.

The overall policy for a system which is undergoing conversion should be to restrain new development as much as possible until the conversion is finished. Even fixing bugs in the old system can be a problem for conversion if the fixes are not made to the converted programs as well. Any new development brings up a number of difficult issues such as whether to develop on the old or new machine, when to convert if developed on the old machine and how to run operationally on the new machine before conversion is done. We have been only partially successful in avoiding these problems, but without a deliberate management policy of only allowing the most essential changes, the conversion would have been made more difficult if not impossible.

Based on our experience, the reliable conversion of a large body of code from one hardware environment to another is possible with a detailed analytical approach. We hope that as a result of our experience others will find conversion to be less of an intimidating, uncontrolled process and more of a task for which some reasonable, effective procedures exist. Conversion may be both expensive and difficult, but it is possible.

REFERENCES

1. *FORTRAN FORTYS Mark III GCOS Background Service Reference Manual* (2200.01A), Information Services Business Division, General Electric Company, Rockville, Maryland, June 1976.
2. *FORTRAN IV (FIV/PFN) Mark III Foreground Service Reference Manual* (3102.13A), Information Services Business Division, General Electric Company, Rockville, Maryland, December 1976.
3. *FORTRAN Reference Manual* (AA-4158B-TM), Digital Equipment Corporation, Maynard, Massachusetts, April 1977.
4. *FORTRAN77(F77) Mark III Foreground Service Reference Manual* (3106.01A), Information Services Business Division, General Electric Company, Rockville, Maryland, March 1977.
5. Griswold, R. E., J. F. Poage and I. P. Polonsky, *The SNOBOL Pro-*

- gramming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.
6. Hausmann, C. L., and M. C. Grignetti, *A Tutorial Introduction to HERMES*, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, June 1976.
 7. *HISAM Mark III Foreground Service Reference Manual (5605.05A)*, Information Services Business Division, General Electric Company, Rockville, Maryland, 1977.
 8. Kernighan, B., "RATFOR—A Preprocessor for Rational FORTRAN," *Software/Practice and Experience*, Vol. 5, No. 4, 1975, pp. 395-406.
 9. Santos, P. J., "FASBOL II, A SNOBOL4 Compiler for the PDP-10," Department of Electrical Engineering and Computer Science and the Electronics Research Laboratory, University of California, Berkeley, California, June 1972.
 10. *System 1022 Database Management System*, Software House, Cambridge, Massachusetts, September, 1976.

A generalized zooming technique for pictorial database systems

by S. K. CHANG, B. S. LIN and R. WALSER

University of Illinois
Chicago, Illinois

INTRODUCTION

Pictorial information processing relies heavily on the establishment of an efficient pictorial database system. Present-day database management systems are designed primarily for efficient storage, retrieval and manipulation of alphanumeric data. Until very recently, little attention has been paid to the storage, retrieval and manipulation of non-alphanumeric information such as digitized images which require a large amount of storage even for pictures of average complexity. With the growing list of new applications in picture processing, such as geographic data processing, demographic data processing, computed tomography, whole-body scanner, earth resources survey satellite (LANDSAT) image processing, regional economic and health data processing, cartographic and mapping applications, etc., the problem of efficient, economical storage, retrieval and manipulation of vast amounts of pictorial information becomes more important and requires careful considerations.

Two problems can be distinguished in designing pictorial databases—the storage, retrieval and manipulation of a large number of pictures, and the storage, retrieval and manipulation of pictures of great complexity. Traditionally, researchers in image processing have concentrated on working with a few pictures. However, the new applications for pictorial information systems generally require that the systems be capable of handling a large number of pictures, some of which are also very complex. Consequently, new techniques must be investigated for the efficient, flexible retrieval of pictorial information from large pictorial databases.

In Reference 2, an approach to designing an integrated database system for tabular data, graphical data, and image data is described. It is based upon generalizations of the relational approach to database design.⁵ The main idea is to represent pictorial information by both logical pictures and physical pictures. A logical picture can be regarded as a model of the real image. It is defined as a hierarchically-structured collection of picture objects. The logical picture can thus be stored as relational tables in a relational database and manipulated using a relational database manipulation language. Inquiries concerning the attributes of picture objects can also be handled by this relational database man-

agement system. Once a logical picture has been identified for retrieval, the corresponding physical picture can be generated on the output device by retrieving the physical picture from an image store which is specially designed for the storage of image data.

This paper describes a generalized zooming technique which can be used for flexible information retrieval and manipulation for pictorial database system. The design of an integrated pictorial database system to support generalized zoom is then described. This system is implemented for interactive map data retrieval and manipulation in a distributed database environment. The project, called the DIMAP (Distributed Image Management and Projection) Project, is funded by the Defense Advanced Research Projects Administration.

In the following section, capabilities of the DIMAP system are described. Generalized zooming concepts, including vertical zoom, horizontal zoom and diagonal zoom, are discussed. The concept of logical pictures and physical pictures, and the correspondence of maps and *d*-maps, are discussed in the third section. The fourth section describes the image store. An example pictorial database is described in the fifth section, followed by pictorial information retrieval examples using the GRAIN language (sixth section). Dynamic zooming examples are discussed in the seventh section, and techniques for frame staging are discussed in the eighth section.

SYSTEM CAPABILITIES

System overview

The goal of the DIMAP project is to design an integrated pictorial database system which combines a relational database management system, RAIN, with an image store management system, ISMS, to enable the user to perform various zooming and panning operations, and to browse through the pictorial database. The DIMAP system provides a pictorial information retrieval language called the GRAIN language. The integrated pictorial database management system, which includes the RAIN subsystem and the ISMS subsystem, is illustrated in Figure 1.

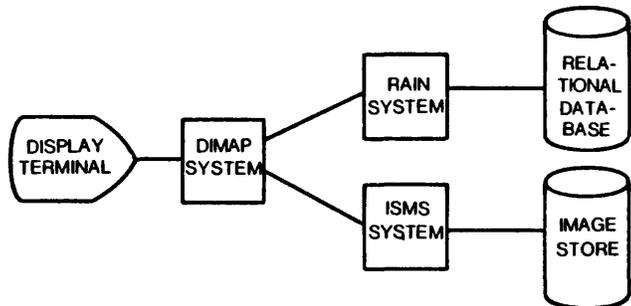


Figure 1—An integrated pictorial database management system.

Display terminal

The user interacts with the DIMAP system via a display terminal. The display terminal has the following features: (a) A color raster monitor with good resolution (512 by 512 pixels); (b) Two joysticks, one for panning, the other for zooming; (c) A cursor for pointing at picture objects in the window; (d) A functional keyboard.

A user views a map through a window in a CRT screen. The window size in the present system is 512 by 512, corresponding to the CRT screen size as well as the x - y size of one frame buffer. In later versions, it may be possible to display more than one (variable-sized) window on the screen at a time. It's possible to view a large map by panning the window in any direction over the map using a joystick. Panning proceeds in starts and stops: smooth panning is possible over an area nine times the area covered by a single window; when panning is uni-directional there is sometimes jerkiness due to loading of frame buffers.

Vertical zoom

By pushing a joystick forward, the user zooms (vertically) in for a more detailed view of a map. The current map is replaced by a more detailed map. Pulling the joystick back causes an outward zoom and loss of detail apparent in the current map.

Since the map set is organized hierarchically (see the next section and Figure 5), some picture objects in the current map may correspond to more detailed lower-level maps. The user can also zoom in on a single picture object, and request more detailed information on this picture object. If this picture object is indeed enlargeable, the current map will be replaced by another map corresponding to this picture object.

Generalized zoom

Since the DIMAP system makes use of the relational database system RAIN, the user may ask questions about non-graphical attributes of picture objects appearing in a map. A list of all attributes known to the DIMAP system about a particular picture object is obtained simply by pointing at the picture object and striking a button. A menu of

attributes appears in the window near the picture object. The user may then ask questions about those attributes. The cursor could be controlled in several ways: through the keyboard, using a joystick, or via a data tablet pen. A light pen could also serve the same function.

With the relational database system RAIN, the DIMAP system can provide generalized zoom capabilities to retrieve picture objects based upon their logical attributes. The concept of *horizontal zoom* (H-zoom) is illustrated in Figure 2. A *zoom window* is first displayed on the CRT screen, whose vertical axis corresponds to various picture objects in a picture file, and whose horizontal axis corresponds to a user-supplied *selection index* (which is obtained either by direct computation, or by table look-up). For example, one selection index could be the degree of similarity of a picture object with a given reference picture object. The *zoom line* can then be moved to set a threshold for selection of picture objects for display. If the zoom line is moved to the left, more picture objects will be selected. Thus, we are having a wide-angle view of the picture file. If the zoom line is moved to the right, fewer picture objects will be selected, meaning a close-up (telephoto) view of the picture file. This type of zoom is called horizontal zoom, because we are zooming in on subsets of picture objects belonging to a picture file. The traditional *vertical zoom* (V-zoom), on the other hand, provides close-up or wide-angle views of a single picture.

Once the zoom line is set, the *view line* can be moved to select a picture object for display. The corresponding picture then appears in the display window. In later versions, more than one viewing window may be provided.

Using functional keys, it may be possible to compute characteristics for a set of picture objects thus selected using the zoom line. We may sketch typical picture objects and atypical picture objects of the picture object set, and display their attributes. In case of raster images, an average picture object (in the sense of averaging the gray levels), can be painted, and its average attribute values displayed. We may also obtain a variable-valued logical description of the picture object set using VVL reduction techniques, extract additional features from the picture object set using pattern recognition techniques, and obtain structural information from the picture object set using syntactic parsing tech-

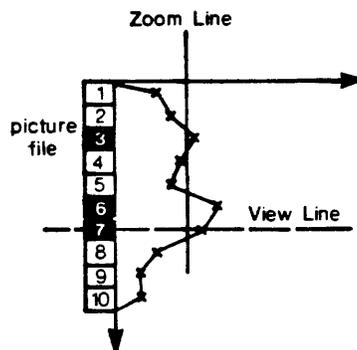


Figure 2—Horizontal zoom (H-zoom).

niques.⁶ Attribute values and texts associated with picture objects can also be displayed.

Figure 3 illustrates picture retrieval by successive horizontal zooms. By moving the view line from one position to another and striking a function key, all picture objects between these limits having selection index above the zoom threshold will be selected. A reduced picture file can then be constructed. The zoom line can again be used to further reduce the picture object set, perhaps using a different (user-supplied) selection index. Finally, by striking another function key, the view line is set in the automatic mode, and pictures appear one by one in the viewing window in rapid succession. If these pictures are successive frames ordered chronologically, a movie is produced.

The concept of horizontal zoom can be further generalized to provide correlation capabilities among picture files. This is called *diagonal zoom* (D-zoom), as illustrated in Figure 4. Suppose picture files A and B are to be correlated, based upon a (user-defined) relation among picture objects. For example, picture file A may consist of prototypes of various types of tanks, and picture file B consists of picture objects to be classified. The user may first select a subset from file A by setting zoom line A. All picture objects in file B related to picture objects in this subset of file A can be selected (using the correlation matrix as shown in Figure 4). The user

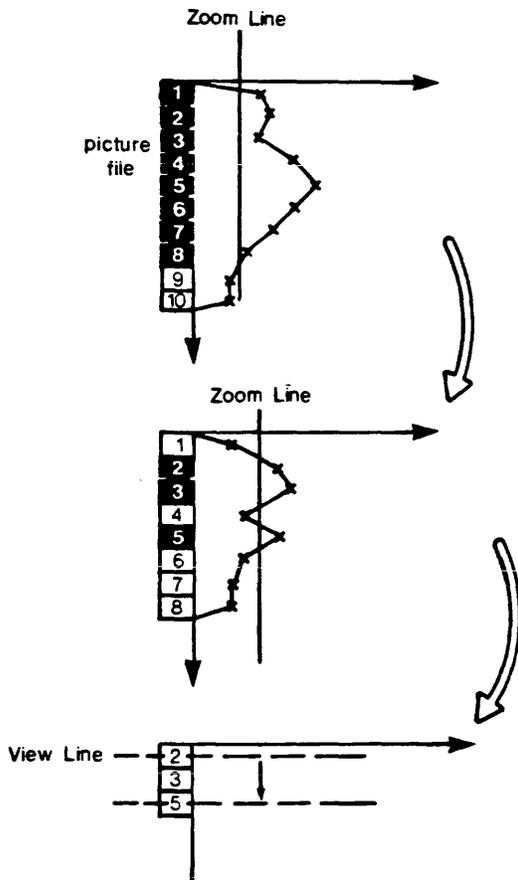


Figure 3—Successive horizontal zooms.

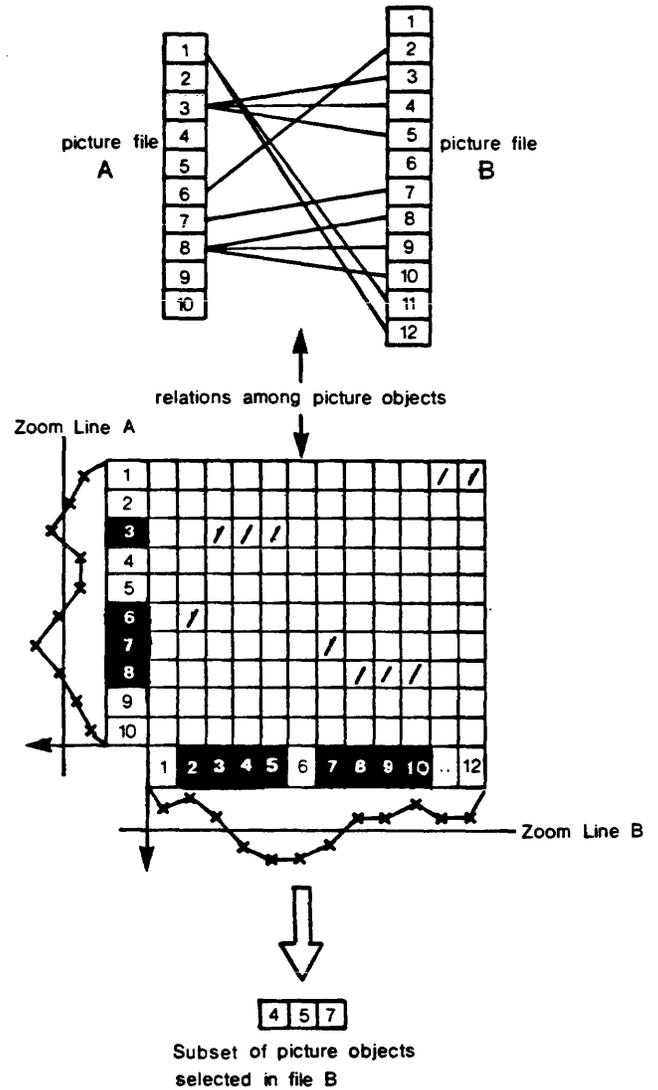


Figure 4—Diagonal zoom (D-zoom).

may further prune the resulting set using zoom line B. The final subset of selected picture objects in file B can then be displayed.

To summarize, V-zoom and H-zoom can be used to select a subset of picture objects from a single picture file. D-zoom can be regarded as generalized H-zoom and can be used to select a related subset of picture objects from multiple picture files.

Map overlay and panning

A map is composed of a collection of *overlays* (see Figure 5), which the user may select individually. In order to concentrate on selected features, the user tunes out overlays simply by pushing buttons. A terrain map, for example, may show elevation contours, roads, vegetation and cities. These four features could be plotted separately, perhaps on clear

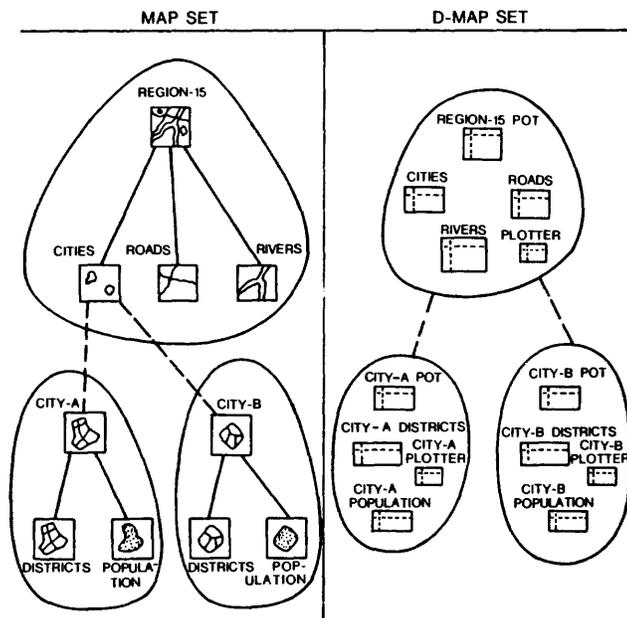


Figure 5—Correspondence of map set and d-map set.

plastic sheets. When the sheets are overlaid, a complete map is obtained.

In conventional maps, on paper, "What you see is what you get." The map reader can't tune out certain features in order to concentrate on others. There's no such constraint in the DIMAP system. At any time the user may show a single or many features in a particular region. The only restriction is that features which interfere visually with one another may not be displayed at the same time. Consider, for example, the two features "vegetation" and "political system." Assume both are area features, meaning their frames consist of colored (or shaded) regions in the viewing window. Displaying both simultaneously would produce a masking effect. The color actually seen by the user would not be the intended color of either, unless the intended color of the two areas happens to be the same; and in the latter case information would surely be distorted because there's little chance that vegetation and political system would correspond in an exact way.

The dynamic overlay capability of the DIMAP system is implemented through the use of the image planes (see the fourth section). At load time, features (picture object set) can be arbitrarily associated with a particular image plane. The user can then move the display window horizontally over an image plane from one frame to another, which is called *panning*.

CONCEPT OF LOGICAL VS. PHYSICAL PICTURES

A map isn't stored in the relational data base in the way it appears in the window to the user. Rather, maps are generated by various processes that transform relational data into a visual form. The overall process of transforming

relational data for display is called *materialization*. Clearly, there is a close correspondence between information in the relational database and information on the display screen. In fact, it's the same information, represented in two different ways. In Reference 2, it has been proposed that only logical pictures are stored in the relational database, and physical pictures are stored in a separate image store.

To make the distinction among logical pictures and physical pictures conceptually clear, the following terminology is adopted:

- | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> ● Relational Database d-map set d-map d-frame/logical picture relation tuple | <ul style="list-style-type: none"> ● Image Store map set map frame/physical picture/image picture object set/features/overlay picture object/feature |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

A *map set* is a hierarchical collection of maps, whose logical representation is called a *d-map set*. A *d-map set* is the entire collection of d-map relations in the database. The correspondence of map set and d-map set is illustrated in Figure 5.

In Figure 5, the top map is REGION-15, consisting of three overlays CITIES, ROADS, and RIVERS. In the CITIES overlay, there are two enlargeable picture objects CITY-A and CITY-B, each corresponding to another map. The map CITY-A in turn consists of two overlays, DISTRICT and POPULATION.

A *map* is composed from one or many overlays, whose logical representation is called a *d-map*. A *d-map* is a set of relations in the data base, which defines a complete map. The correspondence of map and d-map is illustrated in Figure 6. In addition to the relations CITIES, ROADS and RIVERS, there are two special relations POT and PLOTTER, whose functions will be explained below.

The smallest unit for visual display is called a *frame*, whose logical representation is called a *d-frame*. A *d-frame* is a set of relations from which a single frame buffer may be loaded. Each relation in a d-frame corresponds to a group of *picture objects* (i.e. *features*) of the same class. The correspondence of frame and d-frame is illustrated in Figure 7. It should be noted that these d-frame relations are restricted relations obtained from the d-map relations.

The *physical picture* in a frame is also called an *image*. Thus, physical picture, frame and image are regarded as synonymous, whose logical representation is called a *logical picture*.

For each d-map, there is a special relation called POT (Picture Object Table), which contains detailed definitions of all the d-map relations. An example will be given in a later section.

Since all picture objects represented in a d-frame relation are of the same class, their visual interpretations are similar. The visual interpretations of tuples in the relation PEOPLE, for example, are alike insofar as they all depict a head, two arms, and two legs. We associate with each d-frame relation

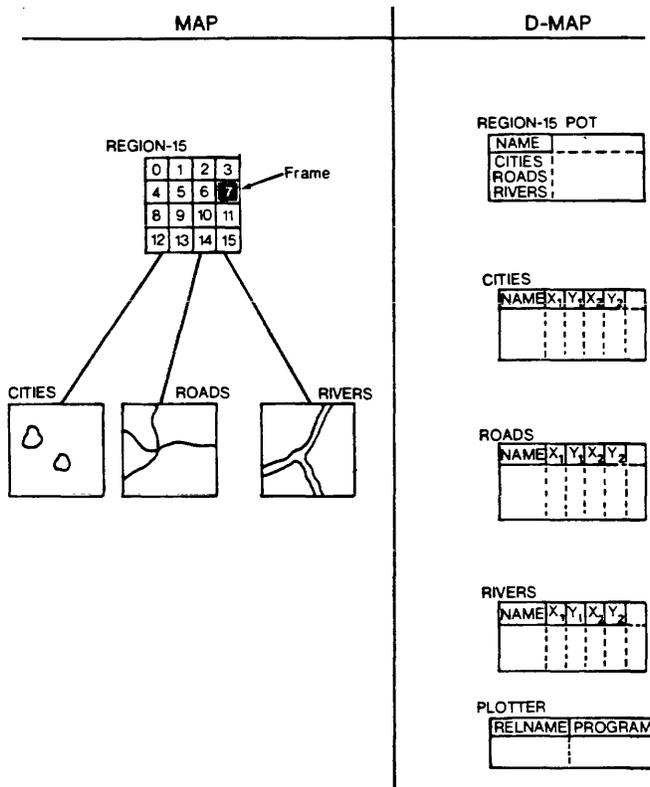


Figure 6—Correspondence of map and d-map.

a graphics program that can draw a stereotypical picture object that is characteristic of the class of picture objects corresponding to the relation. The graphics program associated with PEOPLE knows, of course, how to draw people. It doesn't know, however, how to draw specific people, like "Mary Scott" or "Ray Roth." It takes the information needed to draw a specific person (assuming it's capable of drawing details about people) from the tuple corresponding to the person in question.

A typical d-frame is illustrated in Figure 7. There is a graphics program associated with every relation in the d-frame. The association is made via the special relation named PLOTTER, illustrated in detail in Figure 8. In the simplest implementation scheme, the "program" attribute in plotter takes program names as values. These are the names of graphical programs stored in regular executable UNIX files. The information the program needs to create a particular frame is found in the associated relation. This implies that the graphics program must somehow communicate with the RAIN database system.

Support for materialization, i.e. the process through which relations are given a visual interpretation, is one of DIMAP's central tasks. The problem is how to associate a d-frame with a frame buffer so that materialization may proceed as quickly as possible, leaving a pleasing visual impression with the user. This problem will be discussed in a later section.

THE IMAGE STORE

A detailed system diagram is illustrated in Figure 9. From the user's viewpoint, the DIMAP system can be used to retrieve a logical picture, called a d-frame, which is stored in relational tabular form. A logical picture, or a d-frame, consists of a number of relational tables which are retrieved from the pictorial database using GRAIN commands. The

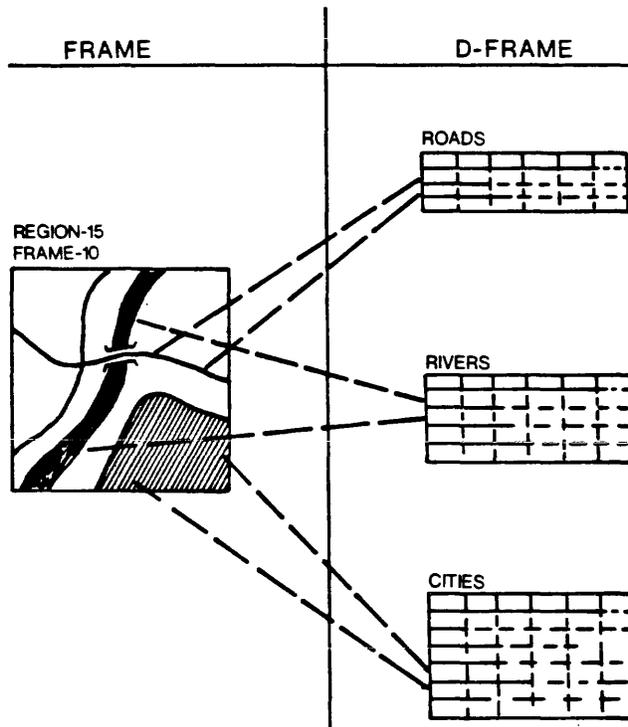


Figure 7—Correspondence of frame and d-frame. (Note: d-frame relations are restricted relations obtained from d-map relations.)

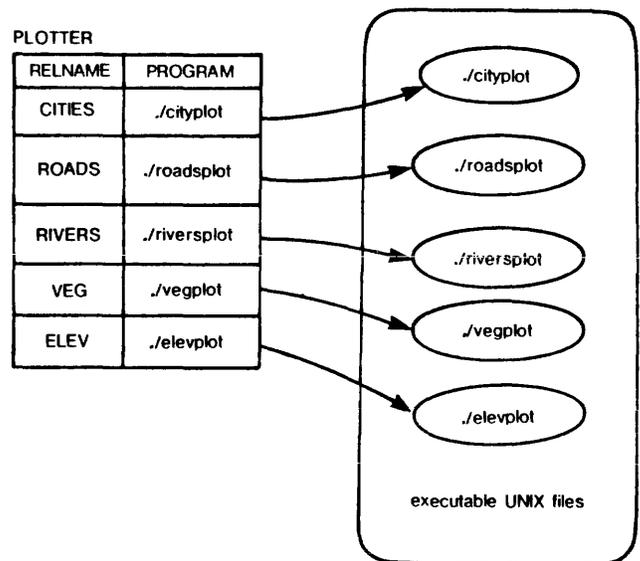


Figure 8—PLOTTER relation.

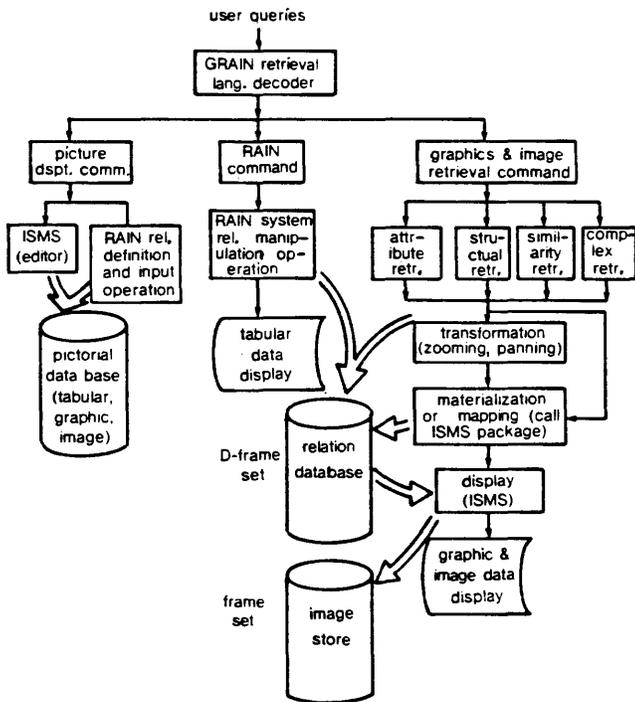


Figure 9—Detailed system diagram of DIMAP.

GRAIN commands can be used to retrieve pictorial information the user needed via attribute information, structural relationship, similarity measure, and complex image processing operations such as color level and gray level manipulation. At the bottom level, the ISMS system can be used to materialize logical pictures into physical pictures, called picture frames or simply frames. Using the display and print commands provided by the GRAIN language, the user can have flexible access to graphic, image, and tabular information. The design of the integrated pictorial database system, the relational database system RAIN, and the image store management system ISMS, can be found in References 2, 11 and 9, respectively.

The viewing window is refreshed out of the *image store*, consisting of four *image planes*, as depicted in Figure 10. The basic unit in the image store is a four-(or larger) bit register called an *image cell*. The four bits can be used to code color or gray level information. On display, the value in an image cell is interpreted visually as a point in a picture. Thus, an image store cell corresponds to a pixel. Each image plane has 2048 by 2048 memory cells, and is partitioned into nine areas, called *frame buffers*. Each frame buffer has 512 by 512 memory cells; thus, a frame buffer has the same number of memory cells as the viewing window has pixels.

Ideally, the image store would be implemented in hardware. This would be expensive in practice, however. To reduce cost, the frame buffers in the present DIMAP system are disk-resident. The result is a less expensive but slower system. The storage requirement (in UNIX blocks) of the disk-resident frame buffers is computed as follows: Since one frame buffer has $512 \times 512 \times 4 \text{ bits} = 131,072 \text{ bytes} = 256$

UNIX blocks, and one image plane has 9 frame buffers, or $256 \times 9 = 2304$ UNIX blocks, the total storage requirement for four image planes is 9216 UNIX blocks, which can be accommodated by a reasonably large disk system.

It's assumed that a typical digitized map is so large that it won't fit within a single viewing window, nor even within a single image plane. This means that pixels in the map greatly outnumber points on the display screen. Thus it's necessary to partition the information associated with the map. Such a partition is called a frame.

In the simplest case, a map fits entirely within one frame buffer. Thus the *x* dimension of the map is the same or less than a frame buffer's maximum width; and likewise for the map's *y* dimension and the frame buffer's maximum height. The d-map for such a map consists of just one d-frame. A frame buffer may be loaded from this single d-frame, and the map may be viewed in its entirety within the buffer, without panning.

In the worst case, the map is very large and the d-map consists of a large number of d-frames, DF_1, \dots, DF_n . Since the user can view only one frame at a time it will often be necessary to move the window from one frame to another. Using the image store, it should be possible to achieve a smooth (if slow) pan.

The panning problem, from the database point of view, is to load the frame buffers from the proper d-frames as the user moves the window over the map. It should appear to the user that he is peering down through a window that can be moved laterally in any direction.

AN EXAMPLE PICTORIAL DATABASE

An example pictorial database will be described here, which will be used in future experiments. This database includes aerial features, linear features, and point features of a 10km by 10km area in West Germany around the city of Fulda. In order to define the pictorial database, approximately 20 elementary relations have been defined.

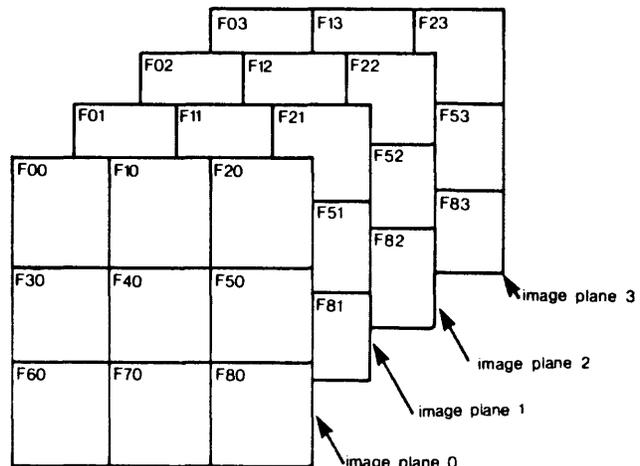


Figure 10—The image store.



Figure 11—Typical map overlays for aerial features.

There are three types of features in the example database. Bridge relation and tunnel relation are point features. Highway relation, railroad relation and waterway relation are linear features. Land use overlay consists of five land distribution relations—urban, forest, cropland, meadow and others. These are aerial features. Vegetation relation and soil type relation are also aerial features. Typical map overlays for aerial features are illustrated in Figure 11.

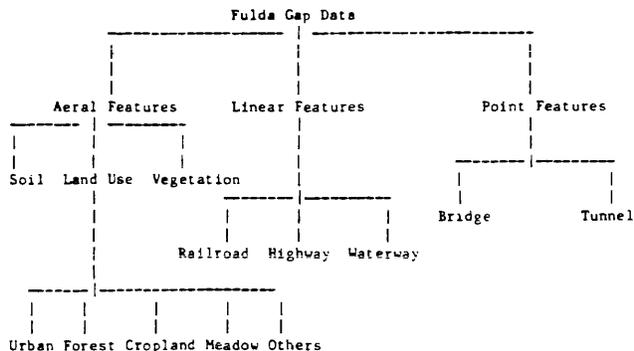


Figure 12—Fulda Gap example database structure.

The structure of the Fulda Gap example database to represent the relationship among these elementary relations is illustrated in Figure 12.

From Figure 12, we can create a relation table, called POT (picture object table), to represent this Fulda Gap example database structure. In Figure 13, the data type field value *R* means raster data format, *L* means line data format, and *P* means point data format. If two relations both have

name	oname	data type	operator	relation type	descriptor definition	no. of tuples
urban	landuse	R	+	E-	d1	400
forest	landuse	R	+	E-	d2	2000
cropland	landuse	R	+	E-	d3	1500
meadow	landuse	R	+	E-	d4	1500
others	landuse	R	+	E-	d5	1000
veget.	aeral	R	-	E-	d6	6400
soil	aeral	R	-	E-	d7	6400
landuse	aeral	R	-	C-	--	--
highway	linear	L	+	E-	d9	30
railroad	linear	L	+	E-	da	20
waterway	linear	L	+	E-	db	15
bridge	point	P	+	E-	dc	1
tunnel	point	P	+	E-	dd	1
aeral	Fulda	R	+	C-	--	--
linear	Fulda	L	+	C-	--	--
point	Fulda	P	+	C-	--	--
Fulda	---	*	*	C-	--	--

Figure 13—POT for Fulda Gap example database.

+ operators or one with + operator and the other with - operator, and with data type *L* and/or *R*, then these two relations can be combined to materialize in the same frame buffer. If two relations both have - operators, then these two relations cannot be combined to materialize in the same frame buffer.

Elementary relations are indicated by type *E* in Figure 13. A composite relation³ is indicated by type *C*. The symbol "--" indicates this relation contains no picture object that can be enlarged. A "+" symbol would indicate this relation contains enlargeable picture objects. The user can then V-zoom in on such picture objects. The remaining fields of POT relation are pointers to descriptor definitions and number of tuples in a relation.

GRAIN RETRIEVAL EXAMPLES

The syntax of GRAIN retrieval language is given in Appendix 1 of Reference 1. GRAIN also contains the RAIN language as a proper subset for manipulation of relational tables. In this section, we present some pictorial retrieval examples for DIMAP by using Fulda Gap example database. First we present several important commands of the GRAIN command language.²

1. *Display* *(frame name)*—Display the physical picture stored in frame buffer *(frame name)* on the CRT screen.
2. *Sketch* *(picture name)*—The logical picture *(picture name)* is plotted on the CRT screen as a line drawing. *(picture name)* can also be a composite picture object which is defined in terms of other picture objects. If the clause "into *(frame name)*" is added to this command, then the logical picture selected will be converted into physical picture and stored in the frame buffer called *(frame name)*.
3. *Paint* *(picture name)*—The physical picture corresponding to the logical picture called *(picture name)* is displayed on the CRT screen as a raster image or a line drawing depending on the picture data type. If the clause "into *(frame name)*" is added, the physical picture is stored in the frame buffer called *(frame name)*.
4. *Draw* *(picture name)*—The line drawing for a new picture object *(picture name)* is to be created. The draw command invokes a graphics editor, which utilizes the graphic system for interactive generation of line drawing.

These are the more important GRAIN commands. In what follows, pictorial information retrieval using the above commands will be illustrated by examples. Pictorial information retrieval can be classified into several categories: attribute retrieval, structural retrieval, similarity retrieval, and complex retrieval.

Attribute retrieval

In attribute retrieval, pictorial objects are retrieved by their logical attributes. For example, in order to sketch the railroads, the sketch command can be used:

sketch picture; name equal 'railroad.'

To select certain railroads, the following command can be used:

*sketch railroad; rgage greater than '120,' or
sketch picture; name equal 'railroad'; rgage greater than
'120,'*

where 'rgage' means the rail gage.

To generate urban land distribution of land use for Fulda Gap, we can say:

paint picture; name equal 'urban.'

Structural retrieval

In structural retrieval, picture objects are retrieved by structural properties, such as component, container, left, right, up, down. For example, the following command sketches the picture objects having a component picture object with name 'urban':

sketch picture; component (name equal 'urban').

In the previous statement, a line drawing for land use will be sketched.

Similarity retrieval

This command can be used to retrieve picture objects which are similar to a given picture using a certain similarity measure. For example, to retrieve all highways which are similar to a given highway called 'h3,' the command is:

*sketch highway; similar (highway-name equal 'h3') using
'M1,'*

where 'M1' is the similarity measure routine which the user supplies for testing similarity among picture objects. For raster images, the user-supplied routine may range from simple template matching to sophisticated hierarchical structural matching. If raster image data is converted into contour data, then several efficient similarity measures can be applied.

Complex retrieval

Complex pictorial information retrieval involves the combination and processing of tabular, graphical, as well as

image data. Complex pictorial information retrieval can also be handled using GRAIN commands. For example, to paint a portion of a picture object with certain color, and paint the other portion with different color, the commands are:

```
paint picture; name equal 'urban'; with 'red'; into frame-x.
paint picture; name equal 'forest'; with 'green'; into frame-x.
display frame-x.
```

If we have a special routine to convert line segment format data to coordinate format data, then we can use the following command to check which highway passes through certain city or town or village:

```
pass=temp(*(x,y)) urban
sketch pass.
```

where the relation '*temp*' is a temporary relation to store the converted coordinate format data for highway relation. The first statement is a RAIN equi-join command.

DYNAMIC ZOOMING EXAMPLES

As mentioned in the second section, the DIMAP system supports panning and zooming operations for pictorial information retrieval. We again use Fulda Gap database in the following examples.

Panning transformation

To pan around certain portion of land occupied by forest, the statement is:

```
paint picture; name equal 'forest'; (forest-x less than or equal '40') and (forest-x greater than or equal '70') and (forest-y less than or equal '20') and (forest-y greater than or equal '30').
```

Zooming transformations

Three kinds of zoom—horizontal zoom, vertical zoom and diagonal zoom are supported.

Horizontal zoom (H-zoom)

To select the land occupied by coniferous forest and mixed forest, the commands are:

```
paint picture; name equal 'forest'; forest-class greater than '1.'
```

Vertical zoom (V-zoom)

For V-zoom within a map, it is almost the same as panning transformation, except that the origin and coordinate spacing should be specified.

```
paint picture; name equal 'forest'; (forest-x greater than or equal '20') and (forest-x less than or equal '40') and (forest-y greater than or equal '10') and (forest-y less than or equal '30'); ((forest-x minus '2') mod '2') equal '0'; ((forest-y minus '4') mod '2') equal 0.
```

For V-zoom on enlargeable picture objects, we first select a picture object, and then load a new d-map corresponding to that picture object.

```
load CITY-A.
sketch picture.
```

Diagonal zoom (D-zoom)

This is the generalized H-zoom operation. This transformation finds all picture objects which are similar to a group of picture objects. For example, in order to find all highways which are similar to two highways '*h1*' or '*h2*', the command is:

```
get picture; similar (highway-name equal 'h1') or similar (highway-name equal 'h2') using 'M2'; into TEMP.
```

TECHNIQUES FOR FRAME STAGING

In the previous cases, zoom operations are performed by dynamically constructing a d-frame using GRAIN retrieval commands. The advantage of dynamic zoom is its flexibility. The disadvantage of dynamic zoom is that it may be too time-consuming to construct d-frames dynamically. For efficiency reasons, we need also consider the problem of *staging* of d-frames.

The problem can be conceived as in Figure 14a where a window is shown over an image plane. The image plane, in turn, is (conceptually) over a d-map. Since the map is larger than the image plane, it isn't possible to have the entire map in the image plane at one time. Relations from the d-map must therefore be materialized on a selected basis into the image plane. It may appear to the user that the window may move anywhere over the map, even though, plainly, this isn't physically straightforward. We need an algorithm which will indicate which frame to load from which relation, and when.

A potential solution is depicted in Figure 14b. The image plane, consisting of nine frame buffers (shown from the "top"), is emphasized by heavy black lines. The viewing window is shown dashed, and the d-map (corresponding to a collection of d-frames in the relational database) is shown

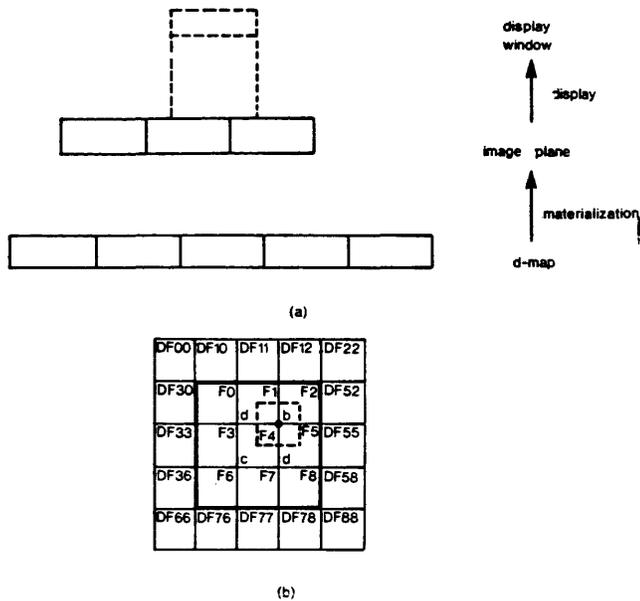


Figure 14—Frame buffer staging concepts (a) and panning of window on image plane (b).

underneath the image plane. It's helpful to notice that Figure 14b is a top view of the arrangement shown in Figure 14a.

To see the solution to the frame buffer staging problem (and, as a corollary, the panning problem) first imagine that the window (shown dashed) is positioned precisely over the central frame buffer. With the window in this position the problem is simple: just let hardware project the image in the central frame buffer up to the CRT screen. But now notice that if the user moves the viewing window at all it will partially cover not just one but four frame buffers. Precisely which four buffers are affected can be determined by noting which of the four central vertices is covered. It's easy to see that only one vertex may be covered at a time. Suppose vertex *b* is covered. This indicates that the viewing window is moving toward the upper right hand corner of the d-map. There's a strong likelihood that frames lying in that vicinity will have to be displayed. This is taken as a cue by the system to mean that the available frame buffers—F0, F3, F6, F7, and F8—are to be loaded. Assuming there are procedures for properly mapping frame buffers into the viewing window, the question is: from which d-frames should the frame buffers be loaded? A staging rule that would work in the case illustrated in Figure 14b is as follows:

If the window covers vertex *b*, then DF11 is materialized into F0, DF12 into F3, DF22 into F6, DF52 into F7 and DF55 into F8.

Other staging rules can be similarly formulated.

IMPLEMENTATION STATUS AND DISCUSSIONS

Prototype RAIN and ISMS subsystems have already been implemented. At the Knowledge Systems Laboratory, we are currently implementing RAIN II (a better version to replace RAIN), and the DIMAP system.

The following research/development problems are currently being considered: (a) The design of a pictorial relational algebra for logical picture manipulation, which can be regarded as enhancement of the traditional relational algebra; (b) Evaluation of paging techniques for efficient storage of frames in image store,¹⁰ and staging techniques for efficient retrieval of frames from d-map; (c) Evaluation of system capabilities by combining the DIMAP system with a knowledge base system to test policy analysis applications; (d) Consideration of database decomposition in a distributed database environment.

REFERENCES

1. Chang, S. K., B. S. Lin and R. Walsler, "A Generalized Zooming Technique for Pictorial Database System," *Technical Report KSL-20*, Knowledge Systems Laboratory, University of Illinois at Chicago Circle, October 1978.
2. Chang, S. K., J. Reuss and B. H. McCormick, "An Integrated Relational Database System for Pictures," *Proc. of 1977 IEEE Workshop on Picture Data Description and Management*, Chicago, Illinois, April 21-22, 1977, pp. 142-149.
3. Chang, S. K., M. O'Brien, J. Read, R. Borovec, W. H. Cheng and J. S. Ke, "Design Considerations of a Database System in a Clinical Network Environment," *Proc. of National Computer Conference*, New York, June 1976, pp. 277-286.
4. Chang, T. L., "Similarity Measures," *Proc. of Third International Joint Conference on Pattern Recognition*, San Diego, California, November 1976.
5. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, Vol. 13, No. 6, June 1970.
6. Fu, K. S., *Syntactic Methods in Pattern Recognition*, Academic Press, 1974.
7. Kunii, T., S. Weyl and J. M. Tenenbaum, "A Relational Database Schema for Describing Complex Pictures with Color and Texture," *Proc. of the Second International Joint Conference on Pattern Recognition*, Lyngby-Copenhagen, Denmark, August 1974.
8. McKeown, D. M., Jr., and D. R. Reddy, "A Hierarchical Symbolic Representation for an Image Database," *Proc. of IEEE Workshop on Picture Data Description and Management*, Chicago, April 21-22, 1977, pp. 40-44.
9. Reuss, J. L., "Introduction to the Image Store Management System," Technical Report, Knowledge Systems Laboratory, University of Illinois at Chicago Circle, March 1, 1977.
10. Reuss, J. L., S. K. Chang and B. H. McCormick, "Paging Techniques for a Pictorial Database," Technical Report, Knowledge Systems Laboratory, University of Illinois at Chicago Circle, June 15, 1977.
11. Walsler, R., et al., "RAIN Version 2 Specification," *Technical Report KSL-20*, Knowledge Systems Laboratory, September 1978.

An approach to real-time scan conversion*

by FRANKLIN C. CROW

University of Texas
Austin, Texas

INTRODUCTION

Scan conversion—that is, the transformation of line segment endpoint coordinates into a collection of scanline segments suitable for raster display—is important because raster displays have many advantages over random-scan, or calligraphic, displays. The calligraphic displays require extensive special-purpose hardware to generate line segments, or “vectors,” and to drive the beam deflection circuits of the CRT. Furthermore, by its very nature, the calligraphic display is subject to damage caused by software defects; a program which directs the beam to the same portion of the CRT face for too long can damage the phosphors, creating a permanent dark spot.

On the other hand, the raster display can be driven by simple digital signals and is immune to software-induced damage. The raster display uses a technology shared by millions of television receivers around the world. This means lower costs through mass production and more flexibility through associated devices designed to store, transmit, project and make hard copies from video signals. For these same reasons, research into new displays is almost entirely concentrated on TV-compatible proposals. Thus, inexpensive displays for computer graphics are most likely to use raster displays in the future.¹⁷

There are a number of current products offering raster-graphic displays using digital image memories with a bit for every picture element of the display.¹⁻³ Such memories provide a very straightforward way to perform scan conversion and are quite appropriate for primarily static images. However, dynamic images pose difficult problems since moving portions of the image must be cleared from the memory and then re-drawn for each successive frame. Furthermore, any static portions of the image which coincide with cleared portions of the display will themselves be partially cleared, leaving unsightly gaps (Figure 1). It would be preferable to generate all dynamic lines together, in scan order, 30 to 60 times a second.

Ideally, static portions of an image should be stored in an image memory while the moving portions are dynamically scan-converted. However, there are arguments for dynamically scan-converting the entire image, assuming that scan-

conversion can be made to run fast enough. If vectors are to be colored or grayscale tricks used to smooth the lines (as discussed later in this paper) then several bits must be used to define the characteristics of each picture element. This requires a rather large amount of memory for an entire image of any worthwhile resolution (307,200 times N bits for a 640 by 480 element image).

There is a reasonably clear trade of memory size against processor power in the decision between an image memory and real-time scan conversion. The image memory needs a processor for generating vectors, etc. But, it doesn't need the power necessary for the techniques discussed here. Buying considerably more processor power could eliminate a few megabits of memory. It is not clear, given the rapid pace of development in both processor and memory systems, which alternative will be more economical ten years from now.

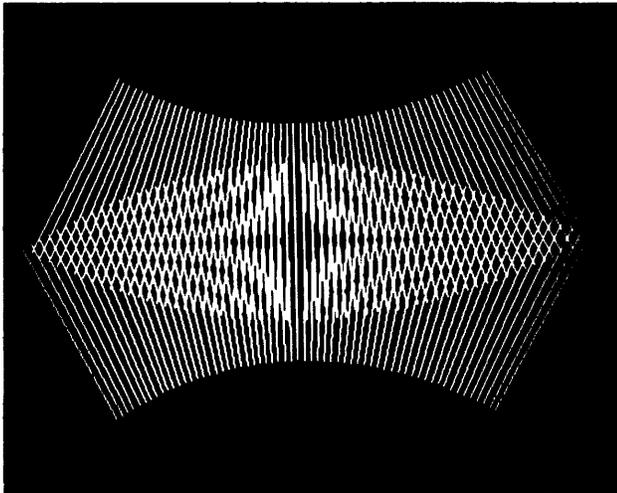
There are, of course, compromises. The image memory can be divided into character-sized cells and memory allocated only to those cells through which a line passes.^{2,10,11} This would allow important savings when using color or grayscale. It is also possible to use separate image memories for static and dynamic portions of the image, allowing the dynamic portion to be cleared after every frame display. This sort of functionally-divided form of display has been used successfully with direct-view storage tubes.¹⁹

There have been at least two previous efforts to develop systems using real-time scan conversion. Cheek⁶ reported a system in which vectors were chopped into short lengths and then grouped into horizontal strips of the display. Lindner and Tozzi¹⁴ have worked on a system which takes an approach similar to that of this paper but appears much more complicated. The approach taken here is heavily influenced by experience with scan-ordered hidden-surface algorithms; similarity between adjacent scanlines is depended upon to minimize computations. The basic scan-conversion algorithm, described next, was initially used, by the author, in a software implementation at the University of Utah in 1973.

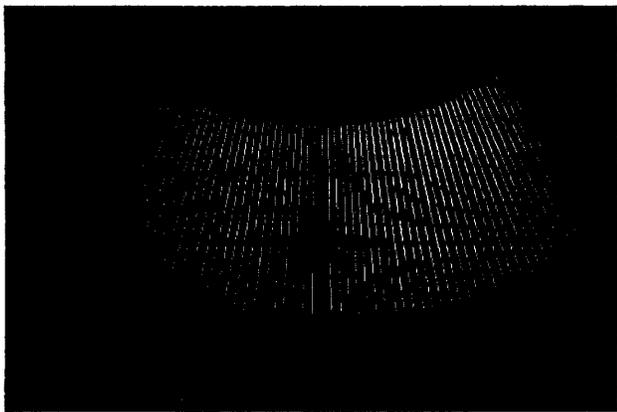
THE BASIC SCAN CONVERSION ALGORITHM

A line segment has a great deal of “coherence.” That is, given one part of a line segment, the rest is easily extrapo-

* This work was supported in part by The National Science Foundation under grant # MCS76-83889.



(a)



(b)

Figure 1—The effect of selective erasure using a digital image memory.

lated. Thus very simple changes suffice to update the scan segment description for a vector from one scanline to the next. Digital vector generators use this property to reduce vector drawing to a series of incremental operations.^{4,9,13,15,16,18} The algorithms developed here differ from previously published methods in that all vectors are generated in an interleaved order dictated by the raster scan pattern. Earlier methods generate vectors individually, using the most convenient order for the algorithm involved.

The scan conversion algorithm is composed of three reasonably distinct tasks. First, lines in the display list must be sorted by the order in which they first appear in the scan. Second, for each scanline the position and length of the scan segment representing each vector crossing that scanline must be computed. Finally, the scan segments for each scanline in turn must be sent in proper order to the display.

The conversion process is spread over a pipeline consisting of a general purpose image update processor, a *Y*-sorted buffer, a microprogrammed scanline processor, an *X*-sorted scanline buffer and, finally, a hardwired picture element processor (Figure 2). The two buffers are implicitly sorted by writing into predetermined slots, one for each scanline

in the *Y*-sorted buffer and one for each picture element in the *X*-sorted buffer.

The design process was heavily influenced by the desire to use general purpose processors wherever possible in order to be in the best position to take advantage of future advances in microprocessor components. Thus, while the scanline processor could no doubt be more effectively implemented in random logic, a bit-sliced microprocessor was chosen to maximize flexibility.

Very modest design goals have been set for the first implementation, a machine capable of maintaining a few hundred vectors with no more than about 50 intersecting any one scanline. For this effort a display resolution of 320 by 240 picture elements at 60 fields per second will be used, roughly the resolution available from an inexpensive home television set. The eventual design goal is at least 1000 by 750 picture elements, or about an order of magnitude improvement. For the moment, the more modest goal allows concentrating on the algorithms and minimizes the sort of difficulties which arise from pushing digital circuitry to state-of-the-art limits.

THE IMAGE UPDATE PROCESSOR

At the head of the scan conversion pipeline, the image update processor is dedicated to keeping track of the vectors

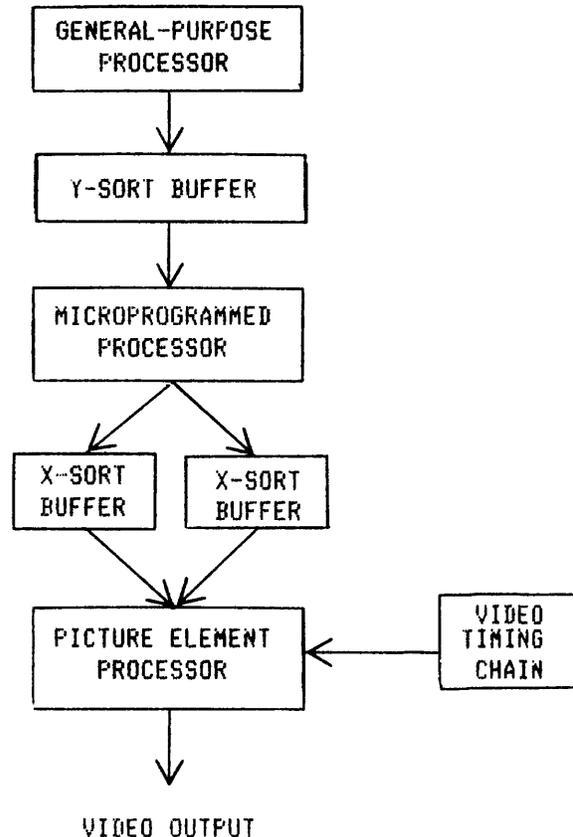


Figure 2—The basic elements of the scan conversion pipeline.

to be displayed (the "display list") and loading the *Y*-sorted buffer. The display list may be formatted to suit the application at hand, the architecture of the host computer, the architecture of the software system or whatever other constraints exist. In short, the display list organization is of no concern here. Suggestions for structuring display lists can be found in Reference 16.

The important task for scan conversion is loading the *Y*-sorted buffer. For the convenience of later, more tightly time-bound elements of the pipeline, the *Y*-sorted buffer stores vector descriptions in a different form. Only the *X*-coordinate of the higher end of the vector is stored, the *Y*-coordinate being implied by the position of the entry. An increment for the *X*-coordinate serves to define the direction of the vector since the *Y*-increment is assumed to be one. All that is left to completely define the line is a length measure. This is supplied as the number of scanlines spanned by the vector.

The calculations involved in computing the buffer entries from vector endpoint coordinates are dominated (in small processors at least) by one division step. This division is necessary to computing the increment. The upper endpoint *X*-coordinate is available directly, after a compare of the *Y*-coordinates. Nearly as simply, the number of scanlines spanned is given by the difference of the *Y*-coordinates plus one. However, the increment is the difference in *X*-coordinates divided by the number of scanlines spanned.

Updating an image consisting of 200 lines at a rate of 30 times a second allows 166 microseconds per vector. Current 16-bit microprocessors with built-in multiply and divide can execute the necessary instructions in about that same time. Since the image update rate can vary from 60 times a second down to around 20 times a second without destroying the smoothness of the motion, there is some leeway available. Use of a minicomputer or one of the more powerful 16-bit microprocessors now appearing should supply adequate power for the image update function.

THE *Y*-SORTED BUFFER

The vector entries, as produced by the image update processor are stored in the *Y*-sorted buffer in a manner which makes it easy for the scanline processor to access the information it needs. Specifically, the scanline processor must be able to readily retrieve all the vectors whose upper endpoints lie on a given scanline. Therefore, the *Y*-sorted buffer is organized as a fixed length array of list heads, each of which is either null or points into a memory containing linked lists of vector entries (Figure 3).

All unused vector entries in the buffer are similarly linked in a separate list to make allocation and deallocation of the fixed-sized vector entries a simple operation. Algorithms for this sort of memory management can be found in Knuth.¹²

The *Y*-sorted buffer can be used in one of two modes. If the drawing displayed is very dynamic, it is simplest to recreate the entire set of vector entries for each image update. However, for partially-static drawings and those with only translational motion, processor cycles may be saved by modifying the existing structure. Using a doubly-linked list for each scanline, a vector entry can be removed from one scanline list and appended to another. If only the position and not the direction of the vector has been changed, it suffices to change the upper end *X*-coordinate in the vector entry. All other numbers remain the same.

The latter mode, of course, involves contention for access to the buffer. Both the image update processor and the scanline processor must access the buffer. Since the scanline processor is under greater time constraints, it is given priority.

THE SCANLINE UPDATE PROCESSOR

The contents of the *Y*-sorted buffer are used to generate a set of scan segments for each scanline. The scan segments

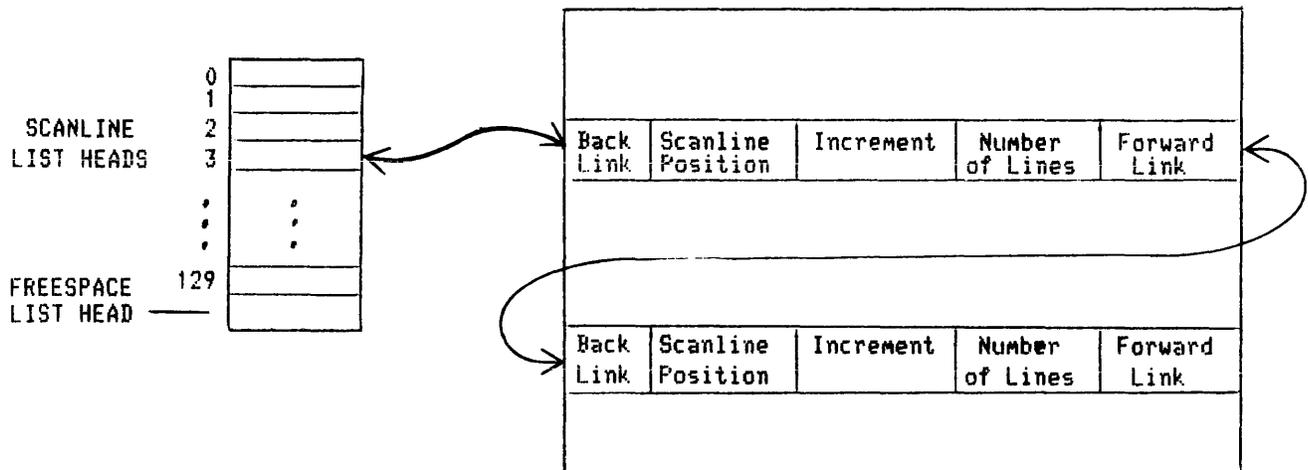


Figure 3—The *Y*-sort buffer and its data structure.

consist of a start position and a run length (the number of picture elements to intensify) for each vector which intersects a given scanline. The scanline processor generates the scan segments for each scanline in turn, moving from the top of the drawing to the bottom.

A scan segment is easily produced from the current *X*-coordinate of a vector and the increment giving the position of the vector at the next scanline. The start position is just the vector position; the run length is just the integer part of the sum of the increment and the fractional part of the vector position.

As the scanline processor works its way down the picture, a "scanline array" containing those vectors which intersect the current scanline must be maintained. As each new scanline is processed, three operations are necessary to maintain the scanline array. First, old vectors in the list which lie entirely above the current scanline must be discarded. Then, vectors which do intersect the current scanline must be updated to find the current point of intersection. Finally, new vectors whose topmost end coincides with the current scanline must be added to the array.

Recall, the *Y*-sorted buffer contains three data on each vector: (1) The horizontal position of the topmost end, (2) the increment which will give the position at the next scanline and (3) the number of scanlines spanned by the vector. Three similar quantities must be maintained by the scanline processor for all vectors intersecting the current scanline (Figure 4).

At each scanline, the scanline array is processed. For each array position with a valid entry, the increment is added to the fractional part of the vector position. The integer portion of the result is then stored in the *X*-sorted scan buffer as the run length. The increment is then added to the vector position and the result stored for use at the next scanline. The number of scanlines spanned is then decremented. If the result is zero, the entry is tagged invalid.

Position

	Current Position	Increment	Number of Lines Left
0	52.25	4.36	43
1	69.45	0.01	2
2	314.0	-1.23	15
3	5.0	-0.67	0
⋮	⋮	⋮	⋮
49			

Figure 4—Information stored in the scanline processor's data memory.

New vector descriptions are inserted where invalid entries are found or created while generating a scan segment. At each such occurrence, the *Y*-sorted buffer is checked for a new vector which, if found, is then loaded into the available array position and processed to generate a scan segment.

Given the standard scan rate of 15,750 lines per second, the scanline processor has 63.5 microseconds to produce a scanline or 1270 nanoseconds per vector, assuming the initial design goal of 50 vectors per scanline. Pipelining the microinstruction fetch, a vector can be processed in six microcycles: (1) Fetch the entry, (2) sum for the run length, (3) store the run length, (4) sum for the position on the next scanline, (5) decrement the scanlines spanned and (6) store the updated entry. Adding a new entry requires two or three more cycles to transfer the entry and update a list pointer. Thus a somewhat relaxed 200-nanosecond cycle time can be used without overly specializing the processor. This is well within the capabilities of current four-bit processor slices.

THE X-SORTED SCAN BUFFER

The scan buffer is actually two buffers. One receives scan segments from the scanline processor while the other is read by the picture element processor. After each scanline is processed the two buffers are functionally switched (Figure 2).

The scanline processor sends a run length to be stored at an address given by the accompanying vector position (Figure 5). A read-modify-write cycle on the given address is used to fetch the previously-stored run length, compare it with the incoming run length and store the larger of the two. This process automatically resolves the problem of overlapping scan segments starting at the same picture element.

At the end of each scanline, one buffer should contain all the scan segments for the next scanline stored in *X*-sorted order while the other should be zeroed, ready to accept another set of scan segments. This implies that the picture element processor must clear the memory as it reads it. Given 320 picture elements per scanline, the buffer must cycle at around 150 nanoseconds. If the picture element processor is to clear the memory in a read-modify-write cycle, then a 70-nanosecond memory is needed. Alterna-

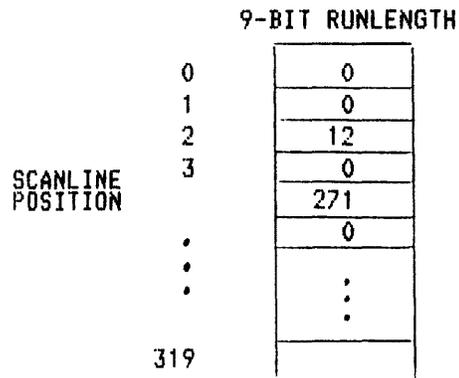


Figure 5—Structure of the *X*-sorted buffer.

tively, the memory can be interleaved on the least significant bit or a bulk clear executed during the beam flyback time (about ten microseconds).

THE PICTURE ELEMENT PROCESSOR

The final element in the picture production pipeline produces a sequence of pulses which, when mixed with synchronization signals for a video monitor, result in intensified scan segments properly placed on the display. This involves reading the *X*-sorted scan buffer, setting a flip-flop at the beginning of a scan segment and resetting that flip-flop at the end of a scan segment.

A counter is used to divide the visible portion of a scanline into 320 parts and to address the *X*-sorted scan buffer. A down counter running at the same rate is used to define scan segment lengths. When the down counter is loaded, the output flip-flop is set. When the down counter reaches zero, the flip-flop is reset.

The down counter is loaded directly from the *X*-sorted scan buffer. Before loading, however, the downcounter contents are compared with the run length from the scan buffer. Only if the magnitude of the incoming run exceeds the content of the down counter is the down counter reloaded. This ensures that overlapping scan segments are properly handled. A series of overlapping segments will produce a single long intensified strip on the display.

The 150 nanoseconds allowed by the 320-element scanline is ample time to perform the compare-and-load operation. Accesses to the scanline buffer are overlapped with the compare operation using an intermediate register.

SPEEDING UP THE IMPLEMENTATION—MORE LINES AND HIGHER RESOLUTION

It should be clear that the picture element processor is designed to handle the case where there is a new scan segment at every picture element. Therefore arbitrarily complicated drawings can be handled at the tail end of the pipeline. However, higher resolution may require producing a picture element as often as every 15 nanoseconds, requiring high-power circuitry and greater concurrency for the scanline buffer and comparator.

The word-processing industry is currently moving to high-resolution monitors in an effort to make the display look as much as possible like a standard 8½-by-11 typewritten page. Because of this, high-resolution raster monitors are now available at costs as low as a few hundred dollars. The semiconductor industry can be expected to eventually produce high-resolution versions of the display controller chips now being produced for standard-resolution monitors greatly simplifying the problems in driving the faster displays.

Current restrictions on the number of vectors which can be displayed lie in the scanline update processor. Bit-sliced microprocessors are not currently fast enough to allow more than 100 vectors or so to intersect a given scanline. The algorithm executed by the scanline processor is simple

enough to be readily translated to random logic. Estimates indicate a potential for increasing the processing rate by as much as an order of magnitude by such means. This would allow up to 1000 vectors on a scanline on a 240-line display or roughly 300 on a 750-line display. Wild guesses suggest that the chip count and cost of the scanline processor would increase by a factor of three to five. However, this violates the philosophy of minimizing special-purpose circuitry.

The other approach to speeding up the scanline processor involves running several microprocessors concurrently. Unfortunately, adequately speedy processors are not yet cheap enough that more than one or two of them can be considered economical in a supposedly low-cost terminal. If cost considerations are ignored under the supposition that the semiconductor industry will solve that sort of problem in due course, then a collection of processors could be arranged to deliver updated vector entries at a rate of one per microcycle. The processor cost and complexity could be expected to increase by a factor of five to ten.

An order of magnitude increase in performance allows a 1000 element by 750 line display with up to 300 vectors crossing any one scanline. This would allow display of roughly 60 lines of 100 or so legible characters, or (equivalently) 18,000 short vectors, or 3600 vector-inches on a 16" by 12" screen (19" diagonal). These figures are for 60 frames per second. Stroke-writing displays with equivalent or better specifications currently start at around \$20,000. The sort of display system discussed here should cost considerably less than one-half that amount. Whether it could be marketed at such a low price, however, is open to question.

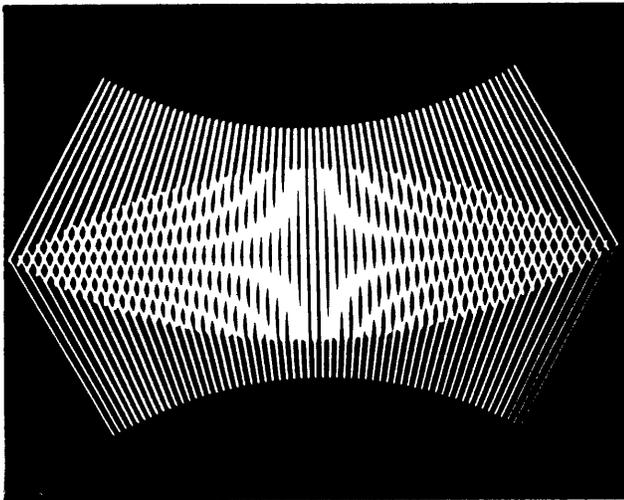
HIGHER QUALITY LINES

One good look at Figure 1 will reveal the major aesthetic problem with scan-converted vectors. They have ugly kinks which are all too evident at any but impractically high resolutions (compare with Figure 6 made on a calligraphic display). Getting rid of these kinks is a difficult, but not impossible proposition. The observer of the display can be tricked into perceiving smooth lines on the display by the judicious use of grayscale techniques.^{7,8}

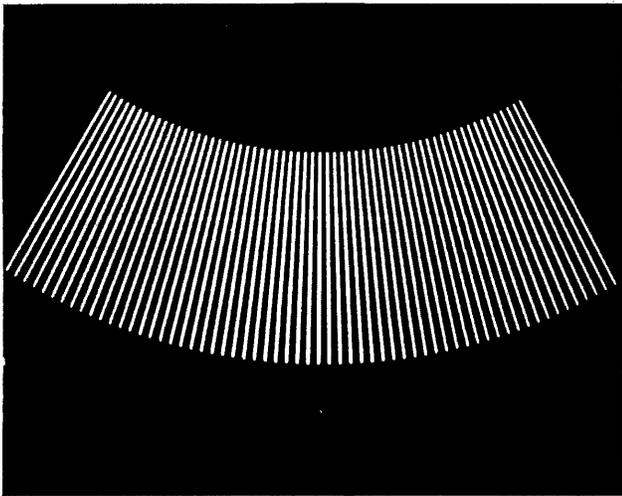
When using these techniques, the scan segments become gray-level functions instead of just run lengths. This necessitates a more complicated picture element processor. However, the scanline processor and *Y*-sorted buffer need not be changed.

Gray levels for producing any scan segment may be stored in a single table which stores the universal intensity profile for all scan segments. Any given scan segment may be produced by supplying an index into the table for the first picture element, the number of elements and the distance between table entries to be retrieved.

Therefore it becomes the job of the picture element processor to generate these numbers from the scanline position and position increment maintained by the scanline processor. The index of the first table entry is computed from the fractional part of the vector scanline position. The number of elements and the distance between table entries, in turn,



(a)



(b)

Figure 6—The pattern of Figure 1 on a calligraphic display.

must be computed from the vector position increment. The microcode for the scanline processor can easily be modified to deliver these two numbers instead of the truncated scanline position and run length.

The distance between table entries is obtained from the reciprocal of the vector position increment using a table of scaled reciprocals. The index for the first entry is given by the product of the fractional part of the vector position and the distance between table entries. The number of entries to be used is just twice the run length used previously.

Note that each scan segment must now be treated individually. The problem of overlapping scan segments becomes much more acute. Furthermore, each scan segment must now be twice as long as before. The likelihood that a number of nearly horizontal vectors will cause sufficient overlap to swamp the processor is quite high.

The gray levels of overlapping vectors must be arithmetically combined to determine the gray level for affected

picture elements. To be absolutely correct about combining the intensities of overlapping vectors, some measure of the area each vector occupies in a picture element and the area of overlap between such vectors would be necessary. Although such calculations have been used for shaded raster images,^{5,7} the additional quality obtained isn't worth the expense in this application. Experiments indicate that a simple sum, truncated to the maximum allowable intensity where necessary, gives acceptable results.⁸

It appears unlikely that all this arithmetic can be performed on the fly as the line is scanned out. Therefore, a buffer is needed in which the grey levels to be displayed are stored. Two such buffers may be used. While one is providing grey levels to the display, the other may be used for building the next scanline (Figures 7,8).

The scan segment information provided by the scanline processor is acted upon by a picture element processor which loads its output into the scanline buffer via a read-sum-write cycle, accumulating intensity at a pixel until saturation. This arrangement eliminates the strict timing constraints involved in scanning directly from an X -sorted buffer of run lengths.

For simple images, the picture element processor should be less than three times as complex as in the initial design.

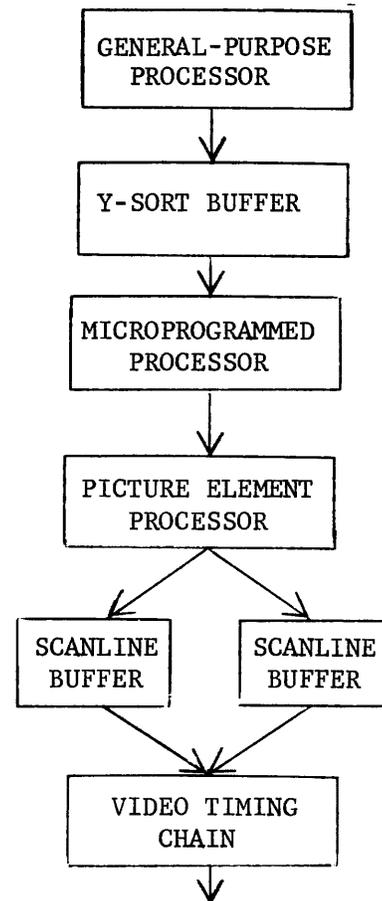


Figure 7—The basic elements of a scan conversion pipeline for smooth vectors.

8-BIT INTENSITY	
0	0
1	50
2	255
3	123
4	0
⋮	⋮
⋮	⋮
319	

Figure 8—Scanline buffer for smooth vectors.

However, a high-performance implementation would require using several picture element processors in parallel.

CONCLUSIONS

The algorithms for the approach to scan conversion presented here have been demonstrated in software. The translation to a hardware implementation for a limited number of lines and modest resolution should pose no problems. The expansion of the concept to higher resolutions and smooth vectors is expected to provide some challenge, but no insurmountable problems.

The failure modes exhibited when the scan converter is overloaded are totally different from the flicker seen on an overloaded calligraphic display system. There are two choices for a failure mode: either repeat the last scanline, or leave out some scan segments. The former method will cause noticeable stripes on the screen, the latter will cause some vectors to disappear on certain scanlines and perhaps leave other vectors out altogether.

A safer failure mode could be engineered by going to lower resolution whenever overload is detected. At a 60-hz refresh rate one slightly bad frame produced while discovering overflow would probably be acceptable. The degraded mode of operation for low-resolution (320 by 240) would provide only 120 lines vertically. Surely, most users would find this intolerable. On the other hand, a high resolution implementation might run in the degraded mode quite successfully.

A practical approach to real-time scan conversion has been described which can be implemented straightforwardly using currently widely available parts. Projected trends in LSI development indicate that high-resolution implementations competitive with low-end calligraphic displays could be produced at quite reasonable cost within a few years.

REFERENCES

- 2648A Graphics Terminal, Hewlett-Packard, Palo Alto, California.
- 4025 Terminal, Tektronix Inc., Beaverton, Oregon.
- IGT 100 Interactive Graphics Terminal, Computer Products Inc., Anaheim, California.
- Bresenham, J. E., "Algorithm for Computer Control of a Digital Plotter," *IBM Systems Journal*, Vol. 4, No. 1, 1965.
- Catmull, E. A., "Hidden-Surface Algorithm with Anti-Aliasing," *Fifth Annual Conference on Computer Graphics and Interactive Technique (SIGGRAPH '78)*, August 1978.
- Cheek, T. B., "A Graphic Display System Using Raster-Scan Monitors and Real-Time Scan Conversion," *SID 1973, Symposium Digest of Technical Papers*, May 1973.
- Crow, F. C., "The Aliasing Problem in Computer-Synthesized Shaded Images," *CACM*, November 1977.
- Crow, F. C., "The Use of Grayscale for Improved Raster Display of Vectors and Characters," *5th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '78)*, August 1978.
- Jordan, B. W., and R. C. Barrett, "A Scan Conversion Algorithm with Reduced Storage Requirements," *CACM*, November 1973.
- Jordan, B. W., and R. C. Barrett, "A Cell-Organized Raster Display for Line Drawings," *CACM*, February 1974.
- Jordan, B. W., and R. C. Barrett, "Scan Conversion Algorithms for a Cell-Organized Raster Display," *CACM*, March 1974.
- Knuth, D. E., *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Chapter 2, 2nd Edition, Addison-Wesley, 1973.
- Kriz, S., "Hardware for High-Speed Digital Vector Drawing," *SID 1973 Symposium Digest of Technical Papers*.
- Lindner, R. and C. Tozzi, "Detailed Concept for a Realization of an Advanced TV Raster Display Terminal," Technical Report GDV-77-1, Technische Hochschule Darmstadt, 1977.
- Matherat, P., "A Chip for Low-Cost Raster-Scan Graphic Display," *Fifth Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '78)*, August 1978.
- Newman, W. M. and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, Inc., 1973.
- Newman, W. M., "Trends in Graphic Display Design," *IEEE-TC*, December 1976.
- Stockton, F. G., "Algorithm 162," *CACM*, April 1963.
- Thanhouser, N., "Intermixing Refresh and Direct View Storage Graphics," *Third Annual Conference on Computer Graphics, Interactive Techniques and Image Processing (SIGGRAPH '76)*, July 1976.

The evolution and architecture of a high-speed workstation for interactive graphics

by WILLIAM L. PAISNER

California Computer Products, Inc.
Anaheim, California

INTRODUCTION

In 1975, California Computer Products, Inc. (CalComp) embarked on the design of a multi-station Interactive Graphics System. As a major part of this design effort, a fresh look was taken at the requirements for the workstation, which, after all, is the operator's sole point of contact with the system. The needs of the operator were perceived as follows:

- To examine the working drawing—any part at any magnification.
- To interact with the drawing—pointing where useful, typing where useful.
—To perform the above rapidly so that he functions in a result-oriented rather than mechanics-oriented environment.^{1,3,4}
- To receive prompting as necessary for complex operations.

In addition, the design goal of multiple simultaneously operating workstations added the following requirements:

- A workstation must have a substantial share of the distributed processing load so that rapid response time can be maintained at all times.
- A workstation must share major peripherals with other workstations.

The remainder of the paper illustrates the workstation design which evolved from these needs and requirements, concentrating on the architecture of an innovative high-speed graphics processor at the workstation core.

STRUCTURE OF CALCOMP IGS 500 SYSTEM

The system is based around a CalComp 16/40 16-bit minicomputer. It incorporates from one to four large high-speed disks which contain program and drawings. The system is structured as in Figure 1, which is drawn to emphasize the workstation elements.

The requirement of speed and distributed intelligence was

satisfied by the inclusion of a "Picture Processor." The description of the architecture and capabilities of this high-speed graphics processor occupies the remaining sections of this paper.

The Alphanumeric CRT/Keyboard provides the operator with high-speed prompting (9600 baud) and allows keyboard input wherever relevant and useful. The tablet or digitizer allows the operator to point at the drawing being digitized/edited and in general allows a close interaction with the drawing through local functions supported by the Picture Processor. The tablet and keyboard can both also be used for menu selection.

A three-axis joystick allows the operator, locally supported by the Picture Processor, to pan and zoom over the *entire* drawing in real time. As will be seen in the next section, the drawing is resident in the Picture Processor for the duration of the session. The speed of the Picture Processor allows a single scan (re-display) of the drawing through the joystick-defined window in 16-200 milliseconds, depending on drawing complexity.

Vector graphics are presented on a raster scan display, operating at 60 Hz, non-interlaced. This provides a bright, flicker free display at minimum cost. An optional two-bit gray scale is available to support gridding and cursor operations.

It should be emphasized that the drawing contained in the Picture Processor memory is a working copy of the archived drawing on the disk. It is in an application-structured hierarchical form—it is in "database" coordinates, not "screen" coordinates. During the session, this is the only copy of the drawing to be modified. At the end of the session it is returned through the central minicomputer and packed onto the disk. This concept of a single resident application-structured drawing, acted upon by a local high-speed graphic processor, is the heart of the workstation and has allowed significant advances to take place in operator interaction.

GENERAL ARCHITECTURE OF THE PICTURE PROCESSOR

The Picture Processor connects to the Host Computer, a 16-bit minicomputer, through a high-speed parallel interface.

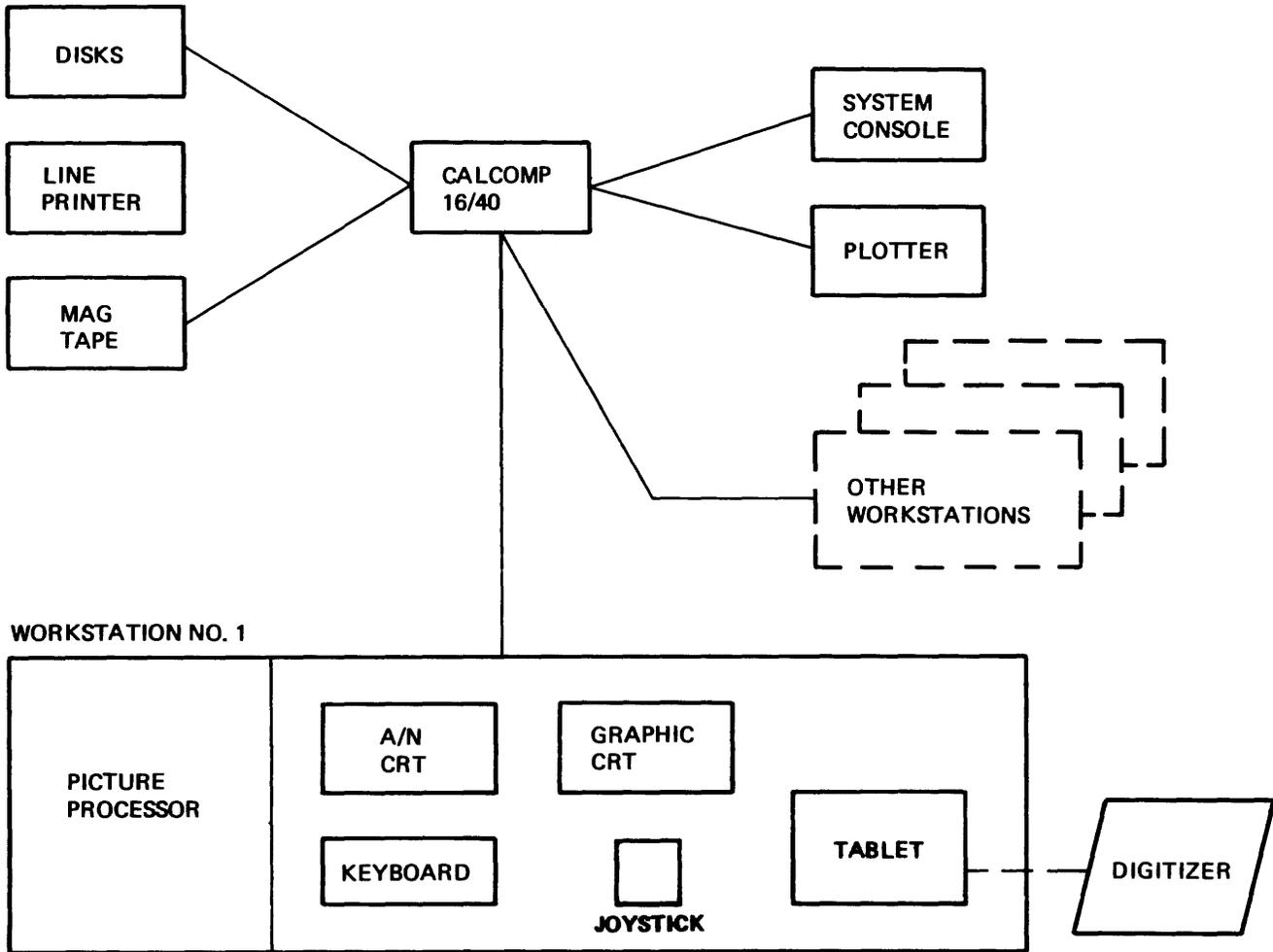


Figure 1

The interface has full handshake and is capable of speeds up to two MBYTES/second. As discussed earlier, the Picture Processor contains a large (64-256 KBYTE) Data Base Memory which holds the drawing during the interactive session. The memory is structured as 64K 32-bit words. A small area of the memory (less than 200 words) is used to pass command and outputs to and from the Host Computer, Local Function Manager, and Display Manager. (See Figure 2.)

The Data Base Memory is, in effect, shared by these three processors during the session. The Host Computer is a 16-bit general purpose minicomputer distributed and partially manufactured by CalComp under license from SEMS Corporation. Through the Host Input/Output module, it manipulates the drawing in the Data Base Memory and issues commands to the remaining two processors. The Local Function Manager is a 6800-based microcomputer which manages the joystick and tablet/digitizer and which serves as a "local" Host Computer to perform functions which reduce the load on the central minicomputer. The position of the local Function Manager within the Picture Processor architecture allows it to perform a significant number of

useful tasks. These will be described in more detail later in this paper. The Display Manager is a high-speed (200 nsec. cycle time) custom-designed microcomputer whose primary task is to scan the drawing in Data Base Memory and output vectors to a pipelined sequence of graphics processing modules which then drive the graphics CRT.

The actual pipeline is a byte-structured 10 MBYTE/second path with a FIFO at the entrance to each module. The pipeline has its own language, using control bytes and data bytes, which allows a standard interface for each processing module and supports the introduction of new ones in future designs. So that the Display Manager may make additional use of these modules, the pipeline is fed back to the Display Manager, allowing it to accept data after processing. This capability is fully exploited in the Picture Processor and is described in the next section.

The graphics CRT is a 15-inch video monitor driven through frame buffer technology. The basic workstation contains one monochrome graphic CRT, switch-selected reverse video and a resolution of 300x416 pixels. An additional frame buffer plane can be added for an optional two-bit gray scale. The Picture Processor can support up to three

additional graphics CRTs, each with optional two-bit gray scale. The 300×416 resolution was selected as the basic display because the operator's ability to rapidly pan and zoom over the drawing made it less necessary to observe larger areas of the drawing at one time. This, in turn, allowed a less expensive display system, with savings in the size (and cost) of the frame buffers and in the selection of the video monitor. At the time of submission of this paper, larger and higher resolution monochrome and color CRT display systems are under development and will be available as options.

The ability of the Picture Processor to allow rapid motion on the screen required a corresponding sophistication in the display system. The graphics CRT is actually driven by two independent frame buffers which are used in various ways during a session. While pan/zoom is taking place, the frame buffers are used as a double-buffered output system. While

one buffer is refreshing the CRT, the other is being loaded with new information from the graphics pipeline. When this loading is complete, taking one to 13 frame times, the buffers are swapped, the other buffer is cleared and loading of new data begins again. The result is a smooth, no-flicker motion of the drawing on the screen.

Another mode of frame buffer operation supports local "dragging," in which an element of the drawing is attached to the tablet stylus and moved around on the screen. In this case, the initial screen image is displayed through one frame buffer. The object to be moved is undrawn from the buffer and redrawn into the other frame buffer, thus avoiding the typical holes which are left when only one buffer is available. During this operation, the contents of both frame buffers appear on the screen but a priority logic is used to ensure that the intensity level of the object being dragged overrides that of the objects over which it passes.

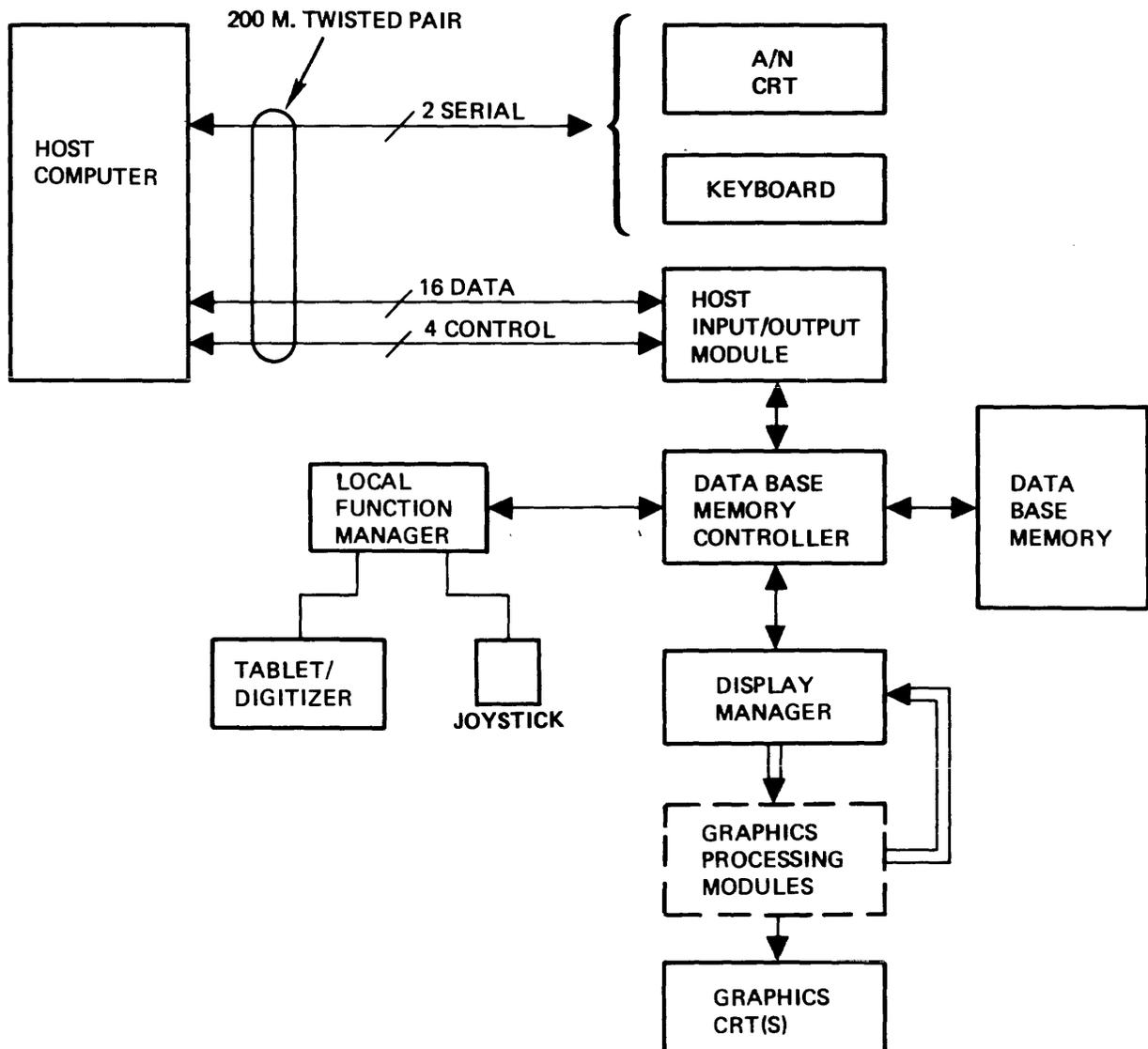


Figure 2

INFORMATION FLOW WITHIN THE PICTURE PROCESSOR

The functions performed by the Picture Processor fall into two major categories. The first group are those involving scan and interpretation of the drawing in Data Base Memory and are performed through the Display Manager/Graphics Processing pipeline path. The second group are those involving the joystick and tablet and the functions which involve them. These are performed by the Local Function Manager which in many cases makes use of the Display Manager/pipeline functions as part of its operation. Figure 3 illustrates the major functions in the first group. The Local Function Manager is described in the next section.

The drawing data base is fetched from disk and loaded into Data Base Memory when the operator types the drawing

name. The disk file is not accessed again for graphical information until the drawing is saved during or at the end of the session. A drawing can be loaded in a few seconds. The drawing structure in Data Base Memory consists of two "files." The Control File has fixed length entries and contains information common to all graphical objects—origin, geometry type, boxing parameters, and certain parametric information such as drawing level, subtype, and a pointer to disk-resident application-dependent properties of the object. Each Control File entry also contains a pointer to a related Geometry File entry. The Geometry File has variable length entries—the structure of each entry depends on its geometry type.

Geometry types supported by the Picture Processor are:

Line—A series of connected or unconnected line segments.

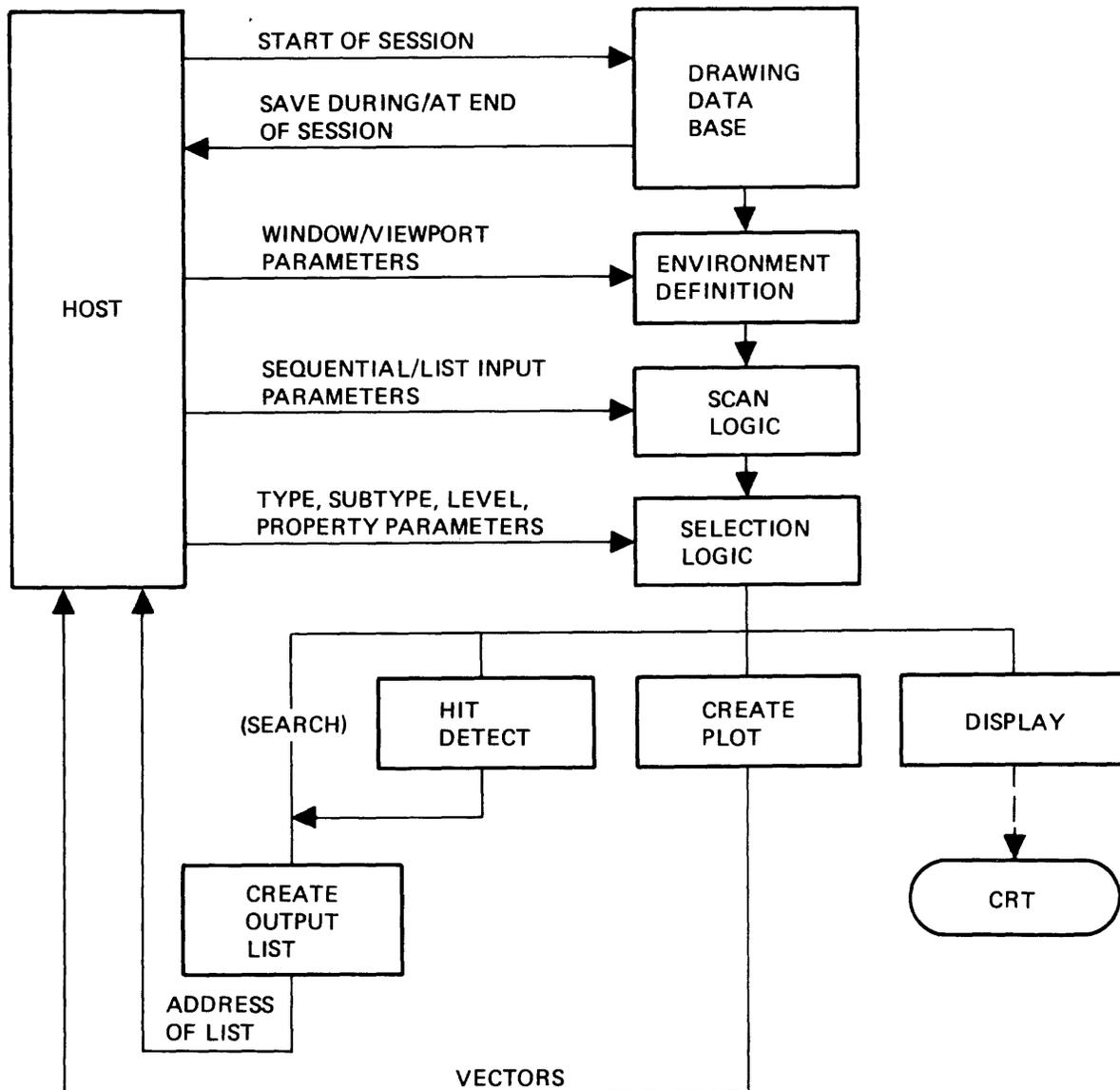


Figure 3

Short line—A series of short connected or unconnected line segments.

Arc—A circular arc, up to and including a full circle.

Text—A character string with optional size, rotation, and inter-character spacing. All characters come from a selection of system or user-defined fonts and are completely arbitrary. A step-and-repeat capability is provided that allows an individual character, of arbitrary structure, to be repeated in a string up to 256 times.

Group—A collection of any of the above entities or other groups. The collection is treated as an entity and may be scaled, rotated, and placed in multiple locations in the drawing. Independent *X, Y* scaling is provided, allowing for example, an ellipse to be generated from an Arc group member.

From this description, it can be seen that it is a hierarchical structure, and in fact most drawings contain a variety of multi-level groups. This sharing of geometries allows the data base to become extremely compact, allowing a smaller, lower cost memory to be used. The data base structure allows for 24-bit precision in all coordinates except for Short Line (12-bit) and text font construction. The Picture Processor Graphics Processing Modules, however, allow 24-bit data in window specifications and the origins of objects at the top of the hierarchy—all other coordinate information is limited to 16 bits. This decision was made to permit reasonable construction costs for the critical Picture Processor boards.

The Host computer directs the Display Manager operation by placing a command block in a reserved area of Data Base Memory. Once the command block has been placed, the Display Manager generally follows the sequence in Figure 3 and the Host is free, during this sequence, to service other workstations, plot, or do other background tasks.

Initially, the Display Manager defines the window, in data base coordinates, and the viewport, in screen coordinates, to be used for this command sequence. In some modes, namely Search and Hit Detect, only the window is defined; for simple Search, neither are defined. Next the scan parameters are examined. The Display Manager can sequentially scan the Control File within the parameter limits, or it can process only those Control File entries whose addresses appear in a list in Data Base Memory. In the latter case, only the address of the list is given as a parameter. List input is a powerful tool for the Display Manager, as it is also able to generate lists in the same format.

As each Control File entry is accessed, the Display Manager compares its level, type, subtype and property pointer against a selection block given to it as a parameter. Selection parameters can be lists or ranges and a Control File entry can be either included or excluded if it "passes" selection. The speed of the Display Manager allows an entry to be tested in microseconds. Selection serves a variety of functions for the application software: among them is the ability to avoid or reduce the "hit ambiguity" problem by allowing only certain objects to be "hit" by the user. In practice, the combination of memory-resident drawing and a very high speed processor (Display Manager) has had a synergistic

effect in the selection mechanism and new uses are still being found for this capability.

Once a Control File entry is selected, it is processed according to one of four basic modes of operation:

Search—No further processing is done. The address of the selected Control entry is added to a list being built in Data Base Memory. At the end of processing of the Control File, the address of this list is returned to the Host computer through Data Base Memory. Note especially that this list is in the form accepted by the Display Manager as list input and can therefore, if desired, be resubmitted to the Display Manager in a subsequent command for display or even further search operations.

Display—The selected Control File entries are processed to extract their coordinates. The associated Geometry File entries are located and interpreted to produce vectors. These vectors are then output to the pipeline where the Graphics Processing Modules perform the necessary transformations into screen coordinates. The end result is to load a frame buffer with the proper bits for the raster display.

Plot—Processing is identical to Display except that instead of the vectors being "drawn" in the frame buffer, they are returned through the pipeline (see Figure 2) back to the Display Manager which then places them in a buffer in Data Base Memory for the Host computer. In this case, the "viewport" definition is chosen so that the precision of the output vectors match the precision of the plotter to be used. The Host then queues the plot data for actual plotting. This capability allows a complete plot file to be generated in seconds with a minimum of Host computer involvement.

Hit Detect—In this mode, the window definition is typically (although not necessarily) a very small region in data base coordinates surrounding the data base coordinate location of the tablet/digitizer stylus. The processing of Control File and Geometry File entries proceeds as before, but for this mode, the Graphics Processing Modules are used differently. Vectors are output to the pipeline but processing terminates at the Window/Clip module (described later in this paper). The module generates for each draw vector, a 2-byte hit response word and outputs that word to the remainder of the pipeline. The Display Manager, on receiving this response word, generates an output list entry in Data Base Memory, just as in Search Mode. At the end of processing, the address of this list is returned to the Host computer, where, as before, it can be re-input to the Display Manager in a subsequent command, for example, Display (for flashing the hit object).

THE LOCAL FUNCTION MANAGER (LFM)

As previously described, the high-speed processing capability of the Picture Processor is located in the Display Manager/Graphics Processing modules pipeline. The Display Manager receives command information from and outputs results, if any, to the Data Base Memory. Because both

the Host computer and the LFM are capable of reading and writing Data Base Memory, the full power of the pipeline is available to the LFM as well. In this mode of operation, the Host computer places command information for the LFM in Data Base Memory. Included in that information are the proper command blocks for the LFM to use when invoking the Display Manager. The flow of information is therefore as shown in Figure 4.

The variety of LFM tasks within the Picture Processor is best illustrated by describing six of the major ones:

- Panning and zooming
- Picking
- Dragging
- Flashing
- Performing application-specific functions
- Managing Picture Processor diagnostics

Panning and zooming

The LFM samples the X, Y and Z axes of the joystick 60 times a second. If a change has occurred, the values are used to modify the center and size of the window(s) given to the LFM by the Host computer when the Pan/Zoom command was issued. The LFM then, using command blocks given it by the Host, invokes the Display Manager to erase the current frame buffer, redisplay the drawing through the modified window, and swap frame buffers. This process repeats as often as possible (≤ 60 times a second) while the operator is manipulating the joystick. The process of calculating the window modifications from the joystick position was subject to considerable optimization during development of the Picture Processor to improve the "feel" of the joystick. The joystick is the operator's movie camera—its operation is required to be smooth, self-evident, immediately rewarding in terms of screen motion, and, most important, to require no mental effort whatsoever from an experienced operator. The LFM modifies both the window center/size and the update rate as a table-lookup function of joystick position. Note that once the initial command was issued, all pan/zoom activity has been completely local.

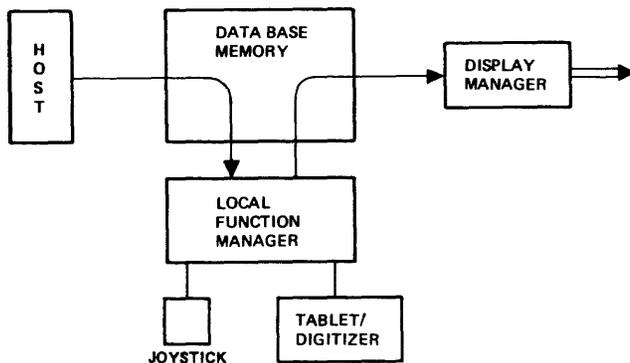


Figure 4

Picking

This is the process used by an operator to identify a particular screen object. The LFM reads the tablet stylus position in screen (tablet) coordinates. Using the viewport in which the access is made and its associated window, the LFM uses one of the Graphics Processing modules (invoked through the Display Manager) to "inverse map" the stylus coordinates into data base coordinates. A small window centered at these coordinates is created and the Display Manager used in Hit Detect mode to determine if an object is being pointed to. The LFM interrogates the list output by the Display Manager and, if null, continues to monitor the stylus until it is lifted. At this point, or if an object is "hit," the LFM quits and raises status to interrupt the Host computer. From the time that the Pick command is issued to the first hit, if any, all operation is local to the workstation.

Dragging

This capability allows the operator to "attach" an on-screen object to the tablet stylus and to literally drag it around on the screen. As part of the command, the Host has identified the object to be dragged, usually from a preceding Pick. The LFM uses the inverse mapped stylus coordinates, obtained as in the Pick operation, to update the origin of the object. The LFM then uses the Display Manager to erase the priority frame buffer and to redraw the object at its new location. This process continues as the operator moves the stylus and terminates when he lifts the stylus. An important aspect of this operation is that the updating takes place on the real drawing object in actual data base coordinates. Thus, when the operation terminates, the drawing is correctly updated and no Host involvement has taken place since the initial command was issued. Due to the high speed of the Picture Processor, objects of any complexity can be dragged in real time.

Flashing

Flashing is a simple LFM function during which an object or list of objects with appropriate Display Manager command blocks is given to the LFM by the Host. The LFM from then on, unless told to stop by the Host, periodically draws and blanks the object(s), using the Display Manager as usual.

Performing application-specific functions

In addition to PROM-resident firmware, the LFM also contains a read-write memory, most of which is available for additional firmware. The LFM contains a loader and the Host may, as part of the initialization of a given application software package, download specific functions to be executed in the LFM. Such functions can be chosen to further offload the Host or to make use of the joystick or tablet in

ways not provided by the standard LFM firmware. This is a powerful capability in its ability to support future application areas.

Managing Picture Processor diagnostics

When power is first applied to the workstation, the LFM performs an extensive, carefully sequenced fault detection and isolation process. It tests itself, its peripherals, its access to Data Base Memory, the memory itself, the Display Manager and each of the Graphics Processing Modules. As the tests are performed, each using only those capabilities proven by the previous test, indicator lights are decremented on the edge of the LFM board. When all tests are successful, a status line is raised to the Host computer to indicate an available workstation. If a failure occurred, the board or boards at fault can be deduced from an examination of the indicator lights. The presence of this test, together with loop-through capabilities in the Host Input/Output Module (Figure 1) provides an exceptional degree of confidence in the workstation at the start of a session.

GRAPHICS PROCESSING MODULES

The modules in Figure 5 perform all of the high-speed work of the Picture Processor. The architecture of the Display Manager and of the pipeline itself were described earlier in this paper. Of the remaining modules in Figure 5, all but the Video Function module are custom designed microcomputers with a cycle time of 200 nsec non-overlapped. All modules shown operate on standard 10 MHz and 5 MHz clocks generated by and distributed from a timing module, not shown here, which also generates the video sync and other timing signals.

Matrix transform module

The Display Manager outputs vectors for an object (Control File-Geometry File pair) which are in the local coordinate system of the object. These vectors may require scaling, rotation, and translation to transform them into true data base coordinates. The Matrix Transform module stores a working matrix, M_w , internally as

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \\ M_{31} & M_{32} \end{bmatrix}$$

As vectors are input from the pipeline, they are transformed by this matrix as follows

$$(X', Y') = (X, Y, 1) M_w$$

and output to the pipeline following the module. The matrix can be cleared to the identity by a command to the module. If nested transformations are required, as in multi-level groups or italic text, a new matrix can be sent via the pipeline to the Matrix Transform module where it will be concatenated onto the working matrix as follows:

$$M_w' = M_{input} M_w$$

It should be noted that for compatibility, a (0 0 1) column is appended to both matrices before concatenation—the resulting (0 0 1) column is then deleted before M_w' is stored. The matrix can also be output to the pipeline, back to the Display Manager, where it can be saved either in local storage or Data Base Memory. This saving (and subsequent restoring) of the matrix occurs as part of context changes during the traversal of nested groups—also in specialized areas of Text and Arc generation.

The Matrix Transform module represents all scaling and rotational terms in twos complement with 14 fractional bits. This allows the values ± 1 to be represented exactly which

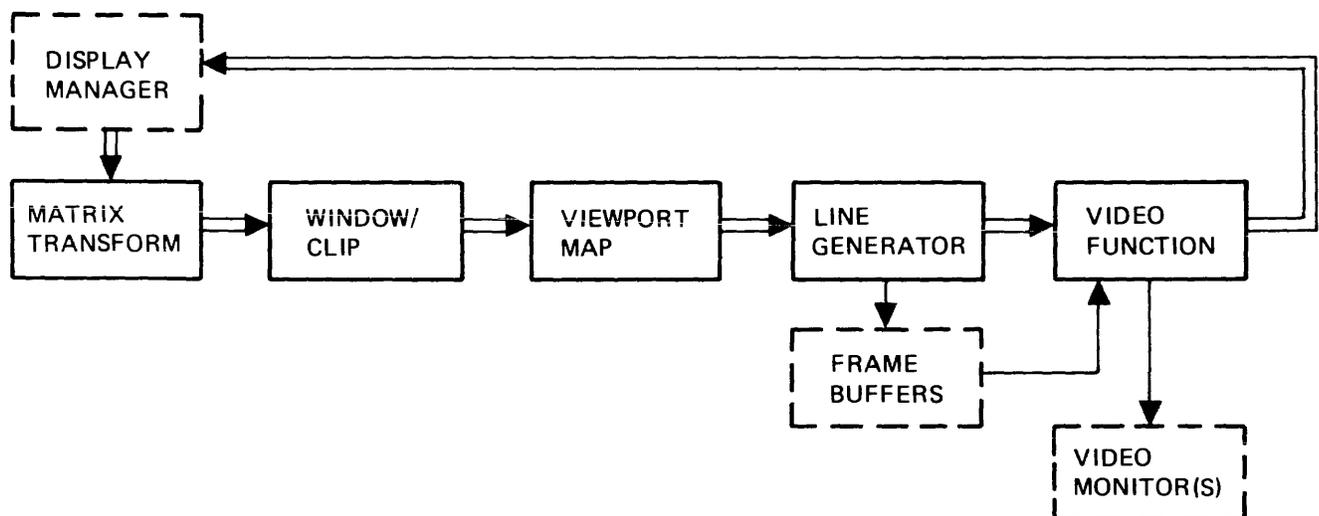


Figure 5

reduces the accumulated error present in nested transformations. In practice, visual feedback has tended to compensate for any such error and nesting levels of five or six (of a maximum of 14) are used routinely.

The Matrix Transform module performs all calculations in 16-bit two's complement fixed-point arithmetic. In vector multiplication the microprogram sequences a set of serial multipliers and adders which calculate X' and Y' simultaneously. Typical processing times are 5.8 microseconds for vector transformation and 17.4 microseconds for a full matrix concatenation.

Window/Clip module

During the Environment Definition phase of operation (see Figure 3), the Display Manager has output the current window size to this module. As each top-level object in the hierarchy is output, the difference between its 24-bit origin and the 24-bit window center, called the "offset," is calculated by the Display Manager and output to the Window/Clip module. Using this information, the module then performs a clipping algorithm on incoming vectors, and in Display and Plot modes, outputs the clipped vectors, if any, to the pipeline. In Hit Detect mode, only a result code is output. Vectors entering the Window/Clip module are in an "object-centered" data base coordinate system. When they leave, they have been clipped to the window edges and translated so that they are in a window-centered data base coordinate system.

The algorithm used is a modified version of the Clipping Divider.² The hardware operates in 18-bit precision and, using essentially two microprocessors operating from the same microprogram, performs X and Y clipping simultaneously. Typical clipping times range from five to 20 microseconds per vector.

Viewport map module

During the Environment Definition phase, the Display Manager has output the current window size, viewport size, and viewport center to this module. Since the incoming vectors to this module are in window-centered data base coordinates, a simple linear transformation turns them into screen coordinates as follows:

$$X_s = X_w(VXSIZE/WXSIZE) + VXCTR$$

$$Y_s = Y_w(VYSIZE/WYSIZE) + VYCTR$$

where

- (X_w, Y_w) —Window-centered vector in data base coordinates.
- $(WXSIZE, WYSIZE)$ —A diagonal vector from the center of the window to the upper right corner, in data base coordinates.
- $(VXSIZE, VYSIZE)$ —A diagonal vector from the center of the viewport to the upper

right corner, in screen coordinates.

$(VXCTR, VYCTR)$ —The center of the viewport, in screen coordinates.

(X_s, Y_s) —The screen vector in screen coordinates.

The Viewport Map module implements this transformation and also the inverse, in which a screen coordinate vector is input, resulting in a window-centered data base coordinate vector. In both cases, vectors are input from the pipeline and the resulting vector is output through the pipeline to the next module.

Calculations are performed in 16-bit precision. Multiplications are done in fixed point for multipliers of less than one—in floating point for multipliers greater than one. Since, in general, this module operates on fewer vectors than the preceding modules (since clipping has taken place), X and Y are produced sequentially and the total time to map a vector is 13 microseconds.

Line generator module

This module receives screen coordinate vectors of four types—absolute move, absolute draw, relative move and relative draw. In Display mode, the module is enabled. It takes all "draw" vectors and performs a line algorithm in firmware to turn on bits in the current frame buffer along the path of the line. For gray scale, two bits are written at each pixel position along the line. For other modes, this module is disabled and passes all received vectors through itself to the pipeline.

The line algorithm is essentially the same eight-vector incremental algorithm used in CalComp plotter software and hardware, biased so that a line is drawn identically when drawn from either end. The line generator operates in 12-bit precision so not to limit future display resolution.

Video function module

This module has several tasks, all related to display. It manages the frame buffer outputs by receiving commands through the pipeline which direct it to connect the two frame buffer outputs together to the screen through a priority scheme, or to connect only one at a time, as used by the pan/zoom LFM operation. It can optionally synchronize the connection of frame buffer output to the next video frame to avoid flicker during rapid motion.

The Video Function module also manages the actual video outputs, monochrome and gray scale, as well as providing a reverse video capability (black on white). It should be noted that this module is in the pipeline only so that it can receive buffer connection commands from the Display Manager—all other pipeline information is passed through unchanged.

PIPELINE THROUGHPUT

The pipeline architecture allows the processing of the Display Manager, Matrix Transform, Window/Clip, Viewport Map and Line Generator modules to operate essentially at the speed of the slowest module. For a given number of displayed lines, only the latter two modules have a constant processing time—the others depend on the drawing structure and the current window in use. The design goal called for 5000 visible one-half-inch vectors, with an arbitrary mix of primitive and group structures, to be displayed in 100 milliseconds. In general this has been met for all except pure text, in which the inclusion of an italic transformation at the character level has raised the display time for 1200 visible four-segment characters to approximately 220 milliseconds.

ACKNOWLEDGMENTS

The architectural definition, design and implementation of the Picture Processor was performed by a small group consisting of the author, Robert Trousdale, Walter Reed and Robert Tingley. Thanks are due to the CalComp management for their unswerving support during the gestation period, to Lefty Miyahara and Vee Fanning for their faultless work in building the first prototype boards and to Jan Wen-

del for her patient typing of 13 revisions of the Picture Processor Interface Manual.

REFERENCES

1. Newman, W. M., and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, Inc., 1973.
2. Sproull, R. F., and I. E. Sutherland, "A clipping divider," *Proc. AFIPS FJCC 1968*, Vol. 33, pp. 765-775.
3. Foley, J. D., and V. L. Wallace, "The act of natural graphic man-machine conversation," *Proc. IEEE*, Vol. 62, No. 4, April 1974.
4. Pooch, U. W., "A user-oriented computer graphics system," *Proc. SIGGRAPH*, Vol. 10, No. 2, 1976, pp. 25-31.
5. Morris, L. R., "Fast transformation of three dimensional graphics structures via mixed-point arithmetic," *Comput. and Graphics*, Vol. 2, Pergamon Press, 1976, pp. 7-10.
6. Jordan, B. W., Jr., W. J. Lennon and B. D. Holm, "An improved algorithm for the generation of non-parametric curves," *IEEE Trans. on Computers*, Vol. C-22, No. 12, December 1973.
7. Bresenham, J., "A linear algorithm for incremental digital display of circular arcs," *Comm. ACM.*, Vol. 20, No. 2, February 1977.
8. Woodsford, P. A., "The HRD-1 LASER display system," *SIGGRAPH Proc.*, Vol. 10, No. 2, 1976, pp. 68-73.
9. Baskett, F., and L. Shustek, "The design of a low cost video graphics terminal," *SIGGRAPH Proc.*, Vol. 10, No. 2, 1976, pp. 235-240.
10. Sutherland, I. E., and G. W. Hodgman, "Reentrant polygon clipping," *Comm. ACM*, Vol. 17, No. 1, 1974, pp. 32-42.
11. "Status report of the Graphics Standards Planning Committee of ACM/SIGGRAPH," *Proc. SIGGRAPH*, Vol. 11, No. 3, Fall 1977.

A mathematical model for distributed free space

by Y. H. CHIN

The Cleveland State University
Cleveland, Ohio

and

S. H. YU

Telecommunication Laboratories
Chungli, Taiwan

INTRODUCTION

To have fast response time is often a requirement for a data base system especially in the on-line environment such as inventory control, stock quotation or hotel/airline reservation. This requirement for fast response time can be easily obtained by carefully organizing the file at loading time. Due to subsequent insertions, the file structure designed with fast response time would be damaged because insertions were stored in overflow area. As more insertions are added, the response time will be lengthened since accessing records in overflow area takes more time than in home area. When the response time exceeds the tolerance limit that a user can stand, a reorganization is required. In general, reorganization of a file is a costly and time-consuming job and should be avoided as much as possible. In order to maintain a fast response time and to avoid frequent reorganizations, a technique called "distributed free space"¹⁰ within home area was introduced.

When data description and instances are loaded into a physical storage device by a data base management system (DBMS), the access methods (or file manager), which are a portion of DBMS, allocate one or more storage rooms, each of them includes spaces for both initial records and future insertions. Such a storage room allocated at loading time to accommodate both initial records and insertions is called a "data storage area (DSA)." Figure 1 shows an example of DSA. There will be no problem of future deletion of data because many DBMSs have utility routines to reclaim the vacated space and merge them to the realm of distributed free space for future insertions. So the effect of deletion is not discussed here. Many commercial DBMSs have such facilities. Examples are VSAM in IBM's IMS, CYBER RECORD MANAGER in CDC's DMS-170, CINCOM's TOTAL, MRI's System 2000 and Cullinane's IDMS, etc.

In commercial access methods, there are parameters provided for users to claim an amount of free space at creation. In general, a user may overestimate or underestimate the amount of distributed free space he needs. In order to determine how much distributed free space a user should

claim, Chin² presents a mathematical model to estimate the size of free space so that insertions do not cause the fast response to exceed the pre-set limit. The model in Reference 2 is derived based on the worst case, namely all insertions are added into a single DSA. As a result, that model reserves too much free space. In this paper, we present a new model, which reserves less amount of distributed free storage space than Chin's model, without increasing the fast response time.

In the next section the models are discussed. We illustrate the simulation tests in the third section. Finally, characteristics of the models and consequences of the experimental tests are discussed in the fourth section.

MATHEMATICAL MODEL FOR DISTRIBUTED FREE SPACE

For clearness, let us restate Chin's problem, assumption, and classification on access methods.

- *Problem and Assumptions*—If the number of initial records and the subsequent insertion rate of a file are both known a priori, how much free space should be claimed such that the probability of overflow is less than any pre-specified value?
- *Classification on Access Method*—The various access methods can be classified into two groups by way of data organization; namely, ordered access method (OAM) and non-ordered access method (NAM). The former refers to those access methods which require data in an ordered sequence with respect to some field values such as index sequential access method or binary search. The later refers to those access methods in which data are not required to be in an ordered sequence for storage or retrieval such as hashing function or non-ordered sequential searching. Hereafter, the size of "distributed free space" has been analyzed on the base of these two classes of access methods.

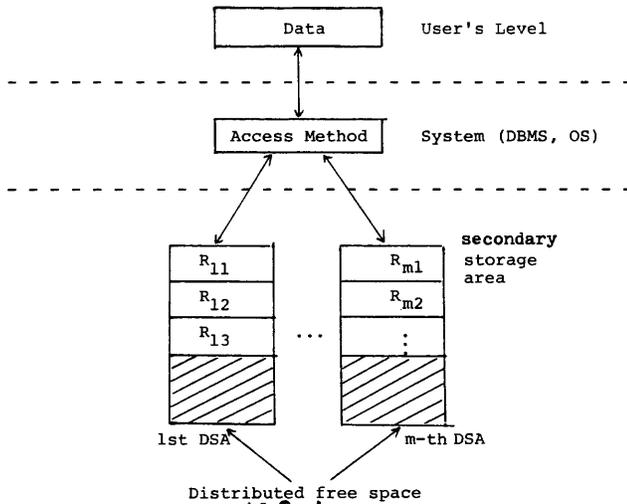


Figure 1—Data storage area (DSA).

Notations

- R : The total number of initial records within a file at load time.
- I : The number of records inserted into the file in time t units subsequent to the load time.
- S_i : The number of initial records loaded into the i th DSA at load time.
- r_i : The number of insertions added to the i th DSA.
- $E(r_i | S_i)$: The number of expected insertions added to the i th DSA given S_i initial records within t units after load time.
- I_i : The amount of free space should be pre-allocated in the i th DSA.
- $VAR(r_i | S_i)$: The variance of random variable r_i with a given S_i .
- m : The number of DSAs created at loading time.
- B : The insertion ratio of the file at time t units after the load time. It is a given value which is used to estimate the quantity of I . $B=I/R$.

Ordered access method

When a file is created by using an OAM such as IBM's VSAM, records in a DSA are stored, maintained and retrieved with respect to a pre-ordered sequence which depends upon the value of a selected attribute. It bases on the mathematical model, called B-tree, which is developed by Bayer and McCreight.¹

Traditionally, each DSA is initially loaded with the same number of records at file creation time. If n fixed-length records are loaded in m DSAs at load time, then $S_1 = S_2 = \dots = S_m = n$, and $\sum_{i=1}^m S_i = R$. Ideally, one would like to have the probability of inserting records into each DSA

to be the same, which is the case if the number of initial records in each DSA is the same and the ranges of key-values are evenly distributed among all DSAs. However, if DSAs have the same number of initial records, it is impossible to make the range of key-value to be evenly distributed among all DSAs. Different DSA will have different ranges of key values. Hence the probability of inserting records into different DSA is different. Keehn and Lacy⁶ show that the probability of adding exactly x insertions to a DSA with n initial records is as follows:

$$P_{S_i=n}(r_i=x) = P_n(x) = \frac{\binom{I}{x} \binom{R}{n}}{\binom{R+I}{n+x}} \frac{n}{n+x} \quad (1)$$

$$0 \leq x \leq I, 1 \leq n \leq R, \text{ and } \sum_{x=0}^I P_n(x) = 1$$

Curves of $P_5(x)$, $P_{10}(x)$ and $P_{20}(x)$ are shown in Figure 2.

The probability $P_n(x)$ in (1) is true regardless of the key-value distribution F_k so long as that both the R initial records and I subsequent insertions are originated from the same F_k .

Using (1), we find that the expected number of records to be added into a DSA with n initial records is

$$\mu_i = E(r_i | S_i = n) = \sum_{x=1}^I x P_n(x) = \frac{nI}{R+1}^* \text{ for } 1 \leq i \leq m \quad (2)$$

Its variance is

$$\sigma_i^2 = VAR(r_i | S_i = n) = \frac{nI}{R+1} \left[\frac{(n+1)(I-1)}{R+2} + 1 - \frac{nI}{R+1} \right]^* \text{ for } 1 \leq i \leq m \quad (3)$$

Since μ_i of (2) is only an "expected estimation," the

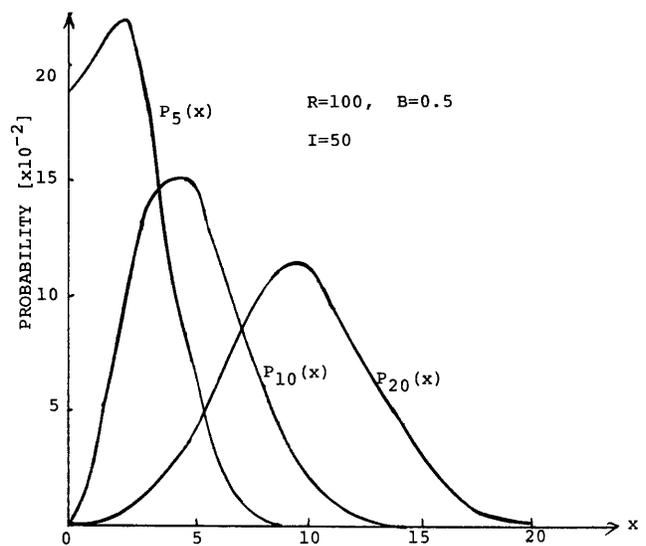


Figure 2

* Derivations are given in Appendix.

probability of overflow could not be known even if μ_i records are reserved as free space within the i th DSA. Intuitively, an "appropriate free space" within a DSA will not only be able to accommodate μ_i insertions, but also give an arbitrarily small amount of deviation for μ_i insertions. For these reasons, we use the Central Limit Theorem to determine an "appropriate free space," I_i , for the i th DSA as follows:

Step 1—Use (2) and (3) to find the expected value and standard deviation of random variable r_i .

Step 2—By Central Limit Theorem, the probability density function (pdf) for random variable $z_i=(r_i-\mu_i)/\sigma_i$ approaches a normal distribution for a large sample. We are interested in the probability that the number of subsequent insertions added to the i th DSA is less than I_i , i.e., $P(r_i \leq I_i)$. Hence,

$$\begin{aligned} P(r_i \leq I_i) &= P\left(\frac{r_i - \mu_i}{\sigma_i} \leq \frac{I_i - \mu_i}{\sigma_i}\right) \\ &= P\left(z_i \leq \frac{I_i - \mu_i}{\sigma_i}\right) \\ &= \Phi\left(\frac{I_i - \mu_i}{\sigma_i}\right) = \Phi(z) \end{aligned} \quad (4)$$

Step 3—Choose the probability $\Phi(z)$ as large as needed so that we can lower the percent of overflow.* For a value z from the normal table,⁴ we can obtain $I_i = \mu_i + z\sigma_i$.

Example 1— $R=1000$, $I=500$, $n=100$, $m=10$
From (2), (3) $\mu_i = 100 \cdot 500 / 1001 = 50$
 $\sigma_i = 8.06$

From these values of μ_i and σ_i , we know that the degree of deviation away from a "mean" value of 50 is 8.06. This value of σ_i signals us that we have a non-zero overflow probability of $\mu_i=50$ records pre-allocated as free space. In order to avoid the overflow problem, I_i should be adjusted according to Steps 2 and 3.

From the normal table,⁴ we select a value for z such that this selection makes $\Phi(z)$ approach one. In our example, selection of $\Phi(z)=0.9995$ makes $z=3.29$.

Using $z=3.29$ as an adjustable coefficient and by (4), I_i should be equal to $50 + \lceil 3.29 \cdot 8.06 \rceil = 77$. Therefore, the size of DSA should be 177 records. The additional 77 records are allocated for subsequent insertions. Under such arrangement, the probability of overflow will be less than 0.0005.

Non-ordered access method

When a file is created by using NAM such as hashing or sequential access method, records within a DSA are stored,

* When $\Phi(z)$ approaches one, it really means that the probability of more than I_i insertions being added to the i th DSA is very small. That is, $\Phi(r_i > I_i) \rightarrow \epsilon$. When $\Phi(r_i > I_i)$ is small, it means the probability of adding insertions outside of the i th DSA is small; therefore, the probability of overflow is small whenever $\Phi(z)$ is large.

retrieved and maintained in a random, non-ordered sequence. For instance, if a file of 1000 records is created by using a hashing function with divisor 11, then the divisor may be represented as 11 DSAs whose sizes are undefined at the moment. Due to the key distribution of original records, each DSA may be initially loaded with different amounts of records. Assume S_i initial records are loaded into the i th DSA at load time, and r_i insertions are added to the i th DSA within period t . Then, we cannot use the model derived in the last section since both the key distribution of initial records and the function of an access method play an important role in determining the number of insertions r_i .

The earlier model of Chin² uses the concept that all I insertions may be added into a single DSA and hence focuses his analysis on one DSA only. It performs well for a file organization with a few DSAs. But when the number of DSAs increases, it will over-allocate free space and thus yield low storage space utilization.

A different approach to this problem can be illustrated by the following example. Suppose the key-values of a given file originate from a "global key space," denoted as N , and suppose the key-values of R initial records and I insertions are members of N . Now let N be the nine-digit social security number system, which contains 10^9 distinct members. These 10^9 members can be partitioned into seven disjoint clusters, say $N_1, N_2, N_3, \dots, N_7$, when a division method with divisor 7 is used.

According to a certain NAM, N can be partitioned into m disjoint clusters N_1, N_2, \dots, N_m , which correspond to m DSAs. Let the i th cluster contain both S_i initial records and r_i subsequent insertions. All those records will eventually be mapped into the corresponding i th DSA through loading and insertion operations. From the point view of clustering, the probability of having x records, whether they are stored at loading or subsequent insertion time, added to a DSA is a hypergeometric distribution. Without ambiguity, we will use N and N_j , to represent the size of global key space and the size of the j th cluster, respectively.

Consider a population of N individuals, of which N_1 are in the cluster 1, N_2 are in the cluster 2, \dots , and N_m are in the cluster m , with $\sum_{j=1}^m N_j = N$. Suppose a sample of size R is chosen from N individuals without replacement. Then the joint distribution of the random variables, S_1, S_2, \dots, S_m , which represent the numbers of individuals in clusters N_1, N_2, \dots, N_m in the sample space R , is defined as

$$P(S_1, S_2, \dots, S_m) = \prod_{j=1}^m \binom{N_j}{S_j} / \binom{N}{R} \quad (5)$$

with

$$\sum_{j=1}^m S_j = R, \quad 0 \leq S_j \leq \min(N_j, R)$$

for $j=1, 2, \dots, m$.

This is called by Johnson⁵ the multi-variate hypergeometric distribution with parameters N_1, N_2, \dots, N_m, R . Therefore, the probability distribution function of S_i is hypergeo-

metric with parameters R, N, N_i :

$$P(S_i|N, N_i, R) = \binom{N_i}{S_i} \binom{N-N_i}{R-S_i} / \binom{N}{R} \quad (6)$$

$$0 \leq S_i \leq \min(N_i, R) \quad \text{for } 1 \leq i \leq m$$

where R, N , and N_i represent the file size, the number of elements in global key space and the number of elements in the i th cluster, respectively. Since (6) is a hypergeometric distribution, its mean and variance can be written as follows:⁵

$$E(S_i) = R * N_i / N \quad (7)$$

$$\text{VAR}(S_i) = R * (N_i / N) * (1 - (N_i / N) * (N - R)) / (N - 1) \quad (8)$$

Unfortunately, N_i is unknown and cannot be determined analytically since different hashing functions generate different clusters. However, we can estimate N_i from the known value S_i . For any particular set S_i, R, N , the value of N_i for which $P(S_i|N, N_i, R)$ is the largest is denoted by \hat{N}_i . It is called the maximum likelihood estimator of N_i .⁴ For convenience, let (6) be abbreviated as $q_{S_i}(N_i)$. Then

$$q_{S_i}(X) = \binom{X}{S_i} \binom{N-X}{R-S_i} / \binom{N}{R}$$

and

$$q_{S_i}(X+1) = \binom{X+1}{S_i} \binom{N-X-1}{R-S_i} / \binom{N}{R}$$

The ratio $q_{S_i}(X+1)/q_{S_i}(X)$ equals to $(X+1)(N-X-R-S_i)/(X+1-S_i)(N-X)$, simple calculation shows that when this ratio is equal to one, $q_{S_i}(N_i)$ achieves its maximum. When

$$\frac{q_{S_i}(X+1)}{q_{S_i}(X)} = \frac{(X+1)(N-X-R+S_i)}{(X+1-S_i)(N-X)} = 1,$$

we have $X = S_i * (N+1) / R$.

Hence, the maximum likelihood estimators \hat{N}_i is the greatest integer less than or equal to $S_i(N+1)/R$. That is

$$\hat{N}_i = \lfloor S_i(N+1)/R \rfloor. \quad (9)$$

In a similar way, the probability of x out of I insertions that are added to the i th DSA with S_i initial records is also a hypergeometric distribution:

$$P(r_i=x|N, N_i, R, I, S_i) = \binom{N_i-S_i}{x} \binom{N-R-(N_i-S_i)}{I-x} / \binom{N-R}{I} \quad (10)$$

$$\text{for } 0 \leq x \leq \min(I, N_i - S_i)$$

We use $N_i - S_i$ instead of N_i for the reason that S_i initial records were chosen at load time and there are only $N_i - S_i$ records left in the i th cluster. The insertions are originated from these $N_i - S_i$ records out of a sample of $N - R$ records.

Now the values of all parameters are known. The expected number of insertions r_i to be added into the i th DSA with S_i initial records is

$$E(r_i|S_i) = I * (N_i - S_i) / (N - R) \quad (11)$$

Its variance is

$$\text{VAR}(r_i|S_i) = I \left[\frac{N_i - S_i}{N - R} \right] \left[1 - \frac{N_i - S_i}{N - R} \right] \left[\frac{N - R - I}{N - R - 1} \right] \quad (12)$$

Using the same strategy as OAM, the free space to be left in the i th DSA is $E(r_i|S_i) + z * \text{VAR}(r_i|S_i)^{1/2}$, where z is an adjustable coefficient as noted in OAM.

Example 2—A file created by using a hashing function through a selected three-digit field. Suppose there are five DSAs available for the file and each file is initially loaded with 16, 11, 26, 52 and 295 records, respectively. If 200 insertions ($I=200$) are estimated to enter into the file within the t time units, then the corresponding size of each DSA is tabulated below:

$$R=400, \quad I=200, \quad N=\{000 \sim 999\}, \quad z=3.29$$

ith DSA	S_i	$E(r_i S_i)$	$\text{VAR}(r_i S_i)$	Distributed	
				Free Space	Size of DSA
1	16	8	4.9	15	31
2	11	5	3.2	11	22
3	26	13	7.9	22	48
4	52	26	14.9	38	90
5	295	147	25.9	164	459
				400	650

Using $S_4=52$ as an illustration, we know $N=10^3$, $R=400$, and $I=200$.

From Equation 9,

$$\hat{N}_4 = N_4 = \frac{52}{400} (10^3 + 1) = 129.$$

From Equation 11,

$$E(r_4|S_4) = 200(129 - 52) / (1000 - 400) = 26$$

From Equation 12,

$$\text{VAR}(r_4|S_4) = 14.9$$

$$I_4 = E(r_4|S_4) + 3.29 * \text{VAR}^{1/2}(r_4|S_4) = 38.$$

Therefore, the size of the 4th DSA is equal to 90 records.

SIMULATION AND DISCUSSION

Methods of simulation

Since data can be stored, retrieved and maintained differently by different access methods, experiments are done separately for each method. In OAM, the total number of

TABLE I—OAM with Various m
 $R=5000, B=0.1, I=500, z=3.29$

No. of initial records S_i	No. of DSA m	Distributed Free Space F	Average No. of Overflows	Average Additional Accesses	Total Space Utilization TSU	Free Space Utilization FSU
20	250	1750	1	0.00034	.8146	.2850
25	200	1400	2	0.00062	.8590	.3555
50	100	1300	0	0.00001	.8730	.3846
100	50	1050	0	0.00003	.9091	.4761
200	25	875	0	0.00000	.9362	.5714
500	10	730	0	0.00000	.9599	.6849
1000	5	655	0	0.00000	.9726	.7634

records in a DSA are dominated by the following two factors—(i) the number of initially loaded records in each DSA and (ii) the key distribution of these initial records and the subsequent insertions. As both data must be chosen from a specific global key space with a distribution F_k ,⁶ we generated 10 sets of key-value each with 1000-5000 distinct elements from a selected distribution. Five of them are sorted and used as initial records, the second five testing data sets are used as subsequent insertions. Therefore a test consists of a set of 25 repetitions. Experiments are divided into two parts, one with changing number of DSAs and the other with different insertion ratio. The results are given in Table I and Table II.

In NAM, the dominative factors include those in OAM as well as which hashing functions are used. Here we did many different experiments. The procedure of generating test files was the same as that used in OAM. At first, we tested whether the DSA size has an effect on the model by changing the number of DSAs available for the file. Next we changed the insertion ratio. The results are given in Tables III and IV. Since the hashing function is a dominant factor, we also did our experiments on various hashing functions such as division transformation, radix transformation, random transformation, etc. The results are given in Table V. In addition, comparisons between our model and Chin's model are made in terms of total distributed free space, average overflows,

TABLE II—OAM with Various Insertion Ratios
 $R=5000, S_i=250, m=20, z=3.29$

Insertion Ratio B	Insertion I	Distributed Free Space F	Average # of Overflows	Average Additional Accesses	Total Space Utilization TSU	Free Space Utilization FSU
0.1	500	840	0	0.0	.9418	.5952
0.2	1000	1500	0	0.0	.9231	.6667
0.3	1500	2120	0	0.0	.9129	.7075
0.4	2000	2760	0	0.0	.9021	.7246
0.5	2500	3380	0	0.0	.8949	.7396
0.6	3000	3980	0	0.00019	.8908	.7536
0.7	3500	4600	0	0.00009	.8854	.7608
0.8	4000	5200	0	0.0	.8824	.7692
0.9	4500	5800	0	0.0	.8796	.7759
1.0	5000	6420	0	0.0	.8757	.7789

TABLE III—NAM with Various m
 $R=5000, I=500, B=0.1, \text{METHOD}=\text{DIVISION}, z=3.29$

No. of DSA m	Distributed Free Space F	Avg. of Overflows	Average Additional Accesses	Total Space Utilization TSU	Free Space Utilization FSU
113	1273-1281	0	0.00012	.8762	.3913
67	1095-1100	0	0.0	.9019	.4554
34	920- 925	0	0.00001	.9288	.5424
23	843- 847	0	0.00000	.9410	.5917
13	754- 756	0	0.00002	.9558	.6627
7	679- 681	0	0.0	.9683	.7353

average additional accesses, total storage utilization and free space utilization. Results are given in Table VI.

As for the file organization, we use a conventional indexed sequential method for OAM, and a division transformation for NAM. We use both of them as main access methods because of their popularity. If overflow occurs, we use chaining as an overflow handling technique.

The values in the fourth column of the Tables is calculated as follows: Each record located in its home area is required one access. If an overflowed chain has length L , then a given overflow record took $L+1$ accesses. If I_i was the amount of free space reserved in the i th DSA, and x insertions were added to that DSA, then overflow occurred for $x > I_i$. The number of accesses to fetch all these records which are sequentially chained together is $0+1+2+\dots+(x-I_i)=n_{ai}$. Then the average additional access per record is equal to $\sum_{i=1}^m n_{ai} / (R+I)$.

Discussion of results

Based on the results obtained from both OAM and NAM, we see that there are almost no overflows whenever F records are reserved, where F equals $\sum_{i=1}^m I_i$. But if more than F insertions were added, the number of overflows would increase rapidly. It means that the distributed free space reserved by the model is sufficient to maintain the short response time and yet not much storage space is wasted.

When the number of DSAs increases, more free space will be allocated. The collection of free space of all DSAs

TABLE IV—NAM with Various m and B
 $R=5000, \text{METHOD}=\text{DIVISION}, z=3.29$

Insertion Ratio B	No. of DSA m	Distributed Free Space F	Avg. of Overflows	Average Additional Accesses	Total Space Utilization TSU	Free Space Utilization FSU
0.1	67	1095-1100	0	0.0	.9019	.4554
0.2		1842-1846	0	0.00013	.8766	.5421
0.4		3185-3192	0	0.00025	.8549	.6272
0.1	23	843- 847	0	0.0	.9410	.5917
0.2		1486-1489	0	0.0	.9248	.6721
0.4		2687-2689	0	0.00006	.9105	.7440
0.1	13	754- 756	0	0.00002	.9558	.6627
0.2		1358-1360	0	0.0	.9435	.7356
0.4		2509-2510	0	0.0	.9322	.7971

TABLE V—Results of Various Methods of NAM
R=5000, I=500, m=67, z=3.39

Method	Distributed Free Space F	Average Number of Overflows	Average Additional Accesses	Total Space Utilization TSU	Free Space Utilization FSU
Division	1095-1100	0	0.0	.9019	.4554
Random transformation	1092-1097	0	0.00002	.9022	.4562
Radix	1094-1099	0	0.00008	.9020	.4554

reduces the storage utilization as indicated in Figures 3a and 4a. As shown in Figures 3b and 4b, when more DSAs are claimed at loading time, more distributed free space is allocated.

In Figure 5, as insertion ratio increases, the free space utilization will increase also. The reason is as follows: A high insertion ratio means a larger amount of insertions will be added into a file. Therefore, the amount of pre-allocated free space is more effectively utilized than that of a small insertion ratio. On the contrary, total space utilization decreases as insertion ratio increases. The reason is that a high insertion ratio pre-allocates a larger portion of free space than that of small insertion ratio.

From the Column 4 of Table V, the free space reserved for each hashing function is nearly the same but the addi-

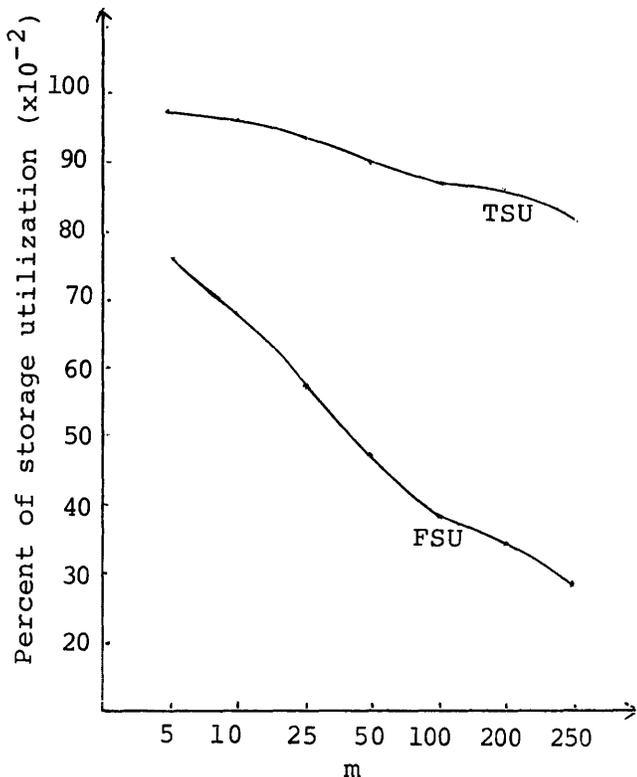


Figure 3a—Effect of number of DSAs on storage utilization of OAM.
R=5000, B=0.1, I=500, z=3.29.

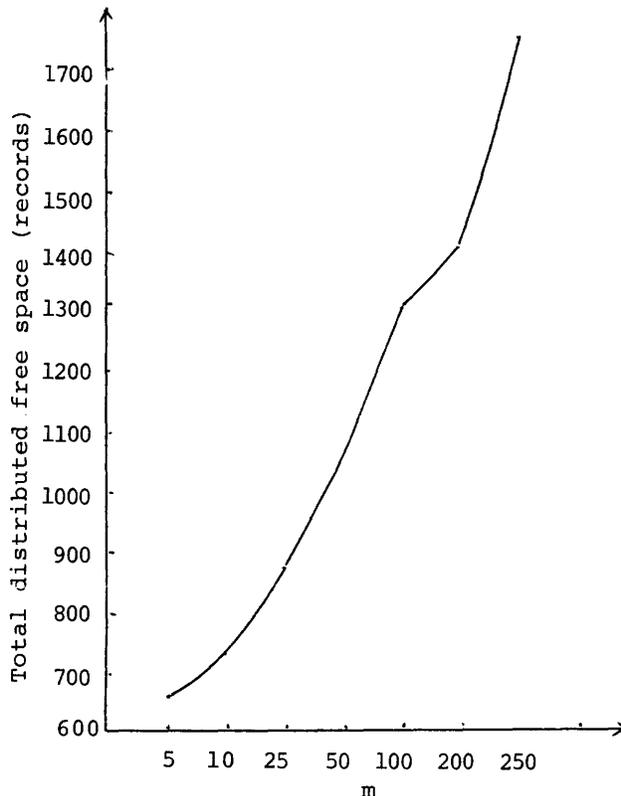


Figure 3b—Total distributed free space for OAM on various m. R=5000, B=0.1, I=500, z=3.29.

tional accesses have a slight difference. The average additional accesses of division method, random method and radix method are 0, 0.00002 and 0.00008, respectively. The result is the same as in Reference 9; namely, division method often outperforms other methods.

Figure 6 shows the comparison of storage space between our method and Chin's. As shown by this figure, the new model (NEW in Figure 6) is superior than the old model (OLD in Figure 6), especially when a large number of DSAs are allocated to the file.

TABLE VIa—Comparison Between Chin's and New Method
R=1000, B=0.2, I=200, METHOD=DIVISION, z=3.29

m	Total Distribution Free Space F		Average Overflows		Average Additional Accesses		Total Storage Utilization TSU		Free Space Utilization FSU	
	NEW	OLD	NEW	OLD	NEW	OLD	NEW	OLD	NEW	OLD
101	404	2181	6	0	0.00653	0.0	.8501	.3806	.4791	.0957
67	402	1850	2	0	0.00295	0.0	.8456	.4208	.4718	.1080
37	407	1407	0	0	0.00017	0.0	.8527	.4980	.4909	.1419
23	368	1092	0	0	0.0	0.0	.9772	.5736	.5435	.1831
13	325	770	0	0	0.0	0.0	.9057	.6771	.6154	.2590
7	287	500	0	0	0.0	0.0	.9324	.7999	.6969	.3999
5	270	387	0	0	0.0	0.0	.9449	.8657	.7407	.5179
3	246	253	0	0	0.00010	0.0	.9630	.9575	.8125	.7899
2	228	178	0	22	0.0	0.15193	.9772	.9999	.8772	.9999
1	200	94	0	106	0.0	4.7258	1.	1.	1.	1.

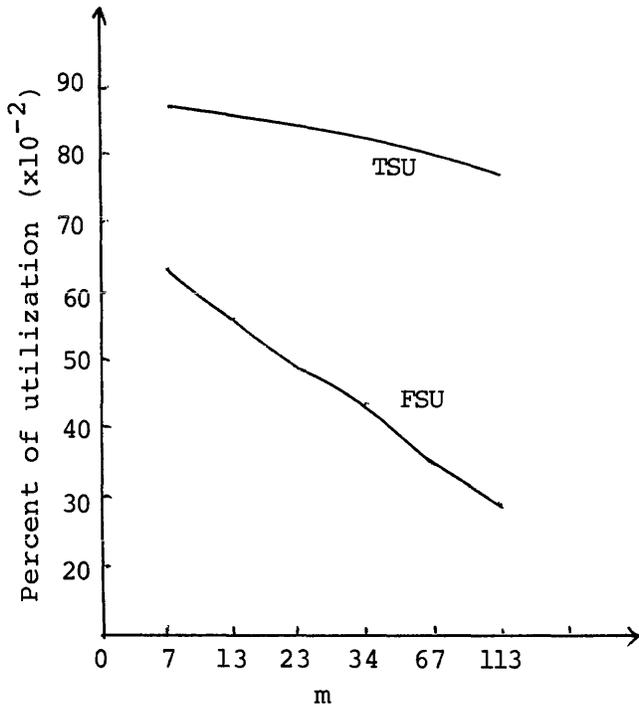


Figure 4a—Effect of m on storage utilization of NAM. $R=5000$, $I=500$, $B=0.1$, $METHOD=DIVISION$, $z=3.29$.

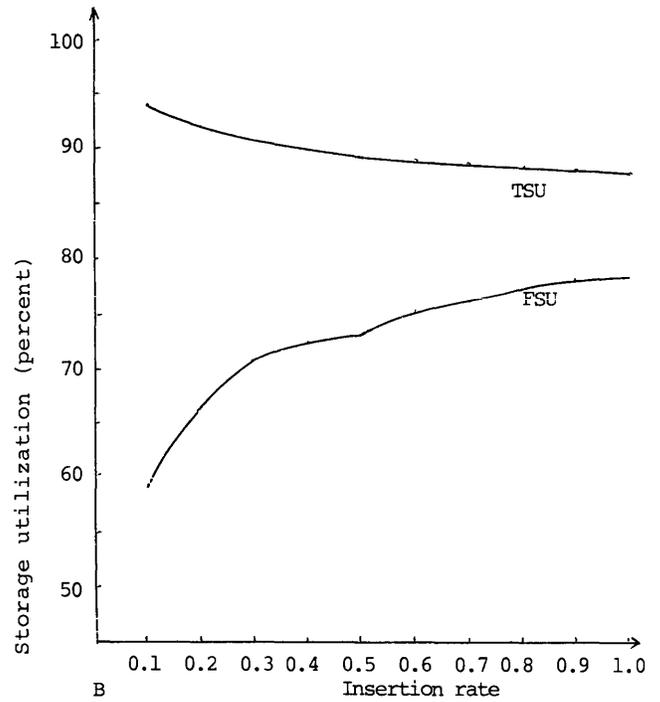


Figure 5a—Effect of B (insertion ratio) on storage utilization of OAM. $R=5000$, $n=250$, $M=20$, $z=3.29$.

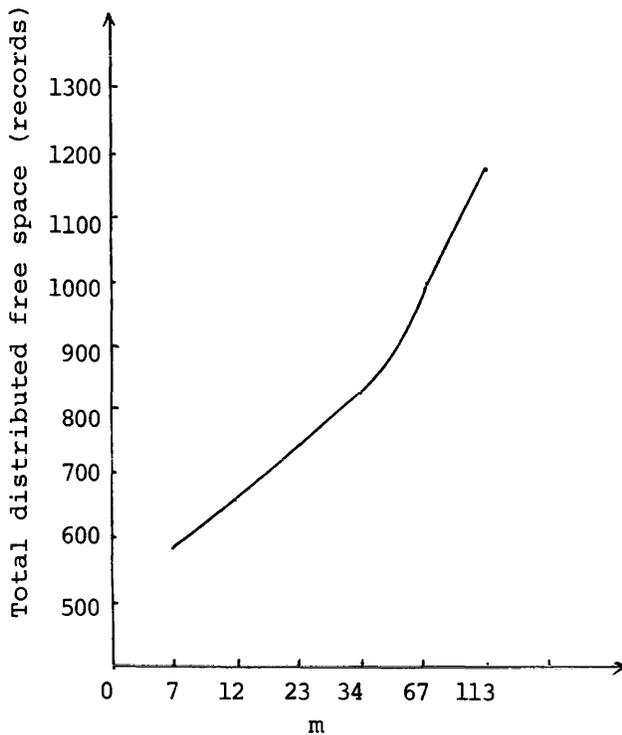


Figure 4b—Total distributed free space for NAM on various m . $R=5000$, $I=500$, $B=0.1$, $METHOD=DIVISION$, $z=3.29$.

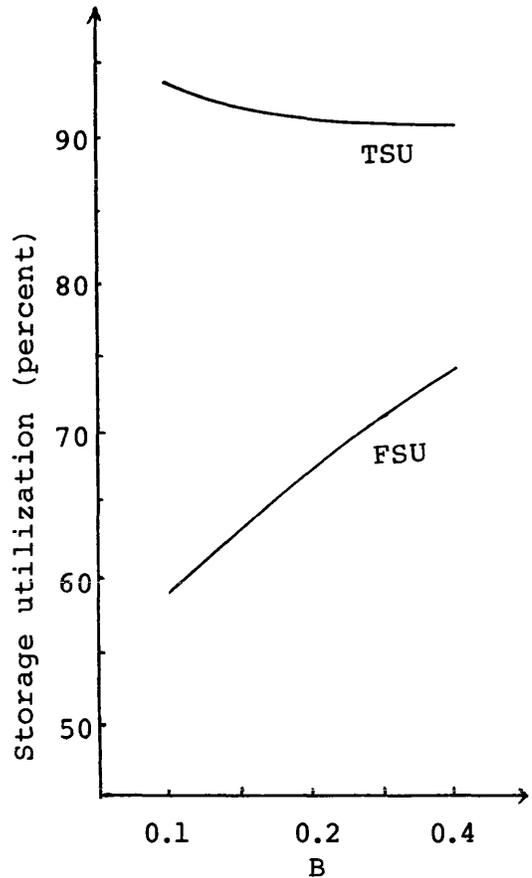


Figure 5b—Effect of B (insertion ratio) on storage utilization of NAM. $R=5000$, $METHOD=DIVISION$, $m=23$, $z=3.29$.

TABLE VIb—Comparison Between Chin's and New Method
 $R=1000, B=0.5, I=500, \text{METHOD}=\text{DIVISION}, z=3.29$

m	Total Distribution Free Space		Average No. of Overflows		Average Additional Accesses		Total Storage Utilization TSU		Free Space Utilization FSU	
	NEW	OLD	NEW	OLD	NEW	OLD	NEW	OLD	NEW	OLD
101	1010	2970	1	0	0.00117	0.0	.7457	.3782	.4938	.1686
67	1005	2618	0	0	0.00037	0.0	.7479	.4146	.4970	.1910
37	851	2124	0	0	0.00013	0.0	.8103	.4806	.5873	.2358
23	782	1736	0	0	0.0	0.0	.8475	.5486	.6394	.2883
13	715	1320	0	0	0.0	0.0	.8748	.6460	.6993	.3783
7	651	932	0	0	0.0	0.0	.9085	.7762	.7680	.5362
5	615	755	0	0	0.0	0.0	.9288	.8529	.8130	.6633
3	576	528	1	7	0.00208	0.0409	.9511	.9772	.8663	.9339
2	—	386	—	113	—	2.33829	—	1.0	—	1.0

Figure 7 shows the comparison between response time of the new model and the old model. As indicated in Figures 6 and 7, the fast response times are nearly the same for both models, but the new model reserves less amount of free storage space than the old model does.

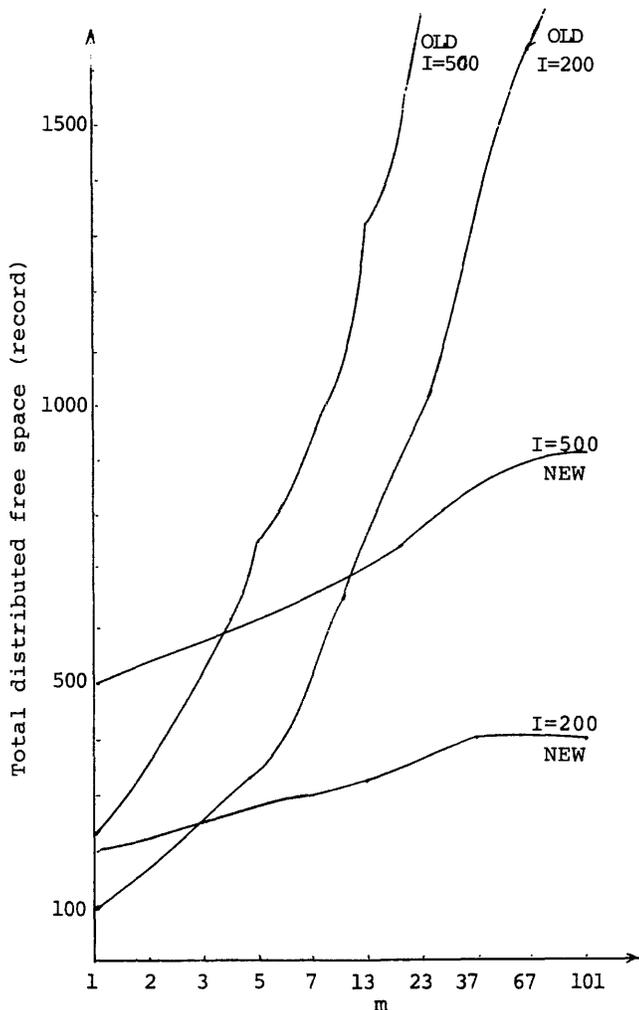


Figure 6a—NAM. $R=1000, z=3.29, \text{METHOD}=\text{DIVISION}$.

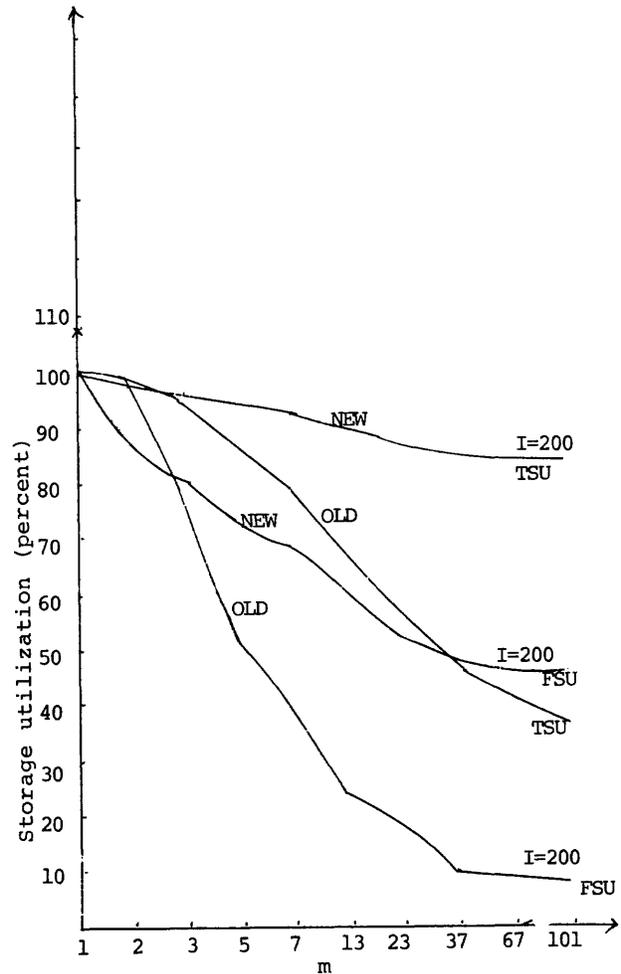


Figure 6b—NAM. $R=1000, z=3.29, \text{METHOD}=\text{DIVISION}$.

A set of tests was done to verify the correctness of the mathematical models. For free space ranged from u_i to $u_i + z\sigma_i$ the probability of overflow calculated from testing data is nearly equal to the corresponding probability calculated from the models.

CONCLUSION

In this paper we have developed a new mathematical model to determine a sufficient "distributed free space" within a DSA in terms of the number of records. In either OAM or NAM, we assume initial records and subsequent insertions are selected from the same distribution. But in NAM, we further assume that the size of global key space N can be calculated from the length of key field. In practice this may not be true. However, in general, N is much larger than the file size so that this assumption causes no serious problem.

Although the experiments have shown that there is almost no overflow when F records are reserved as free space for subsequent insertions, it is not easy to know mathematically that how long the fast response time can be kept. The reason

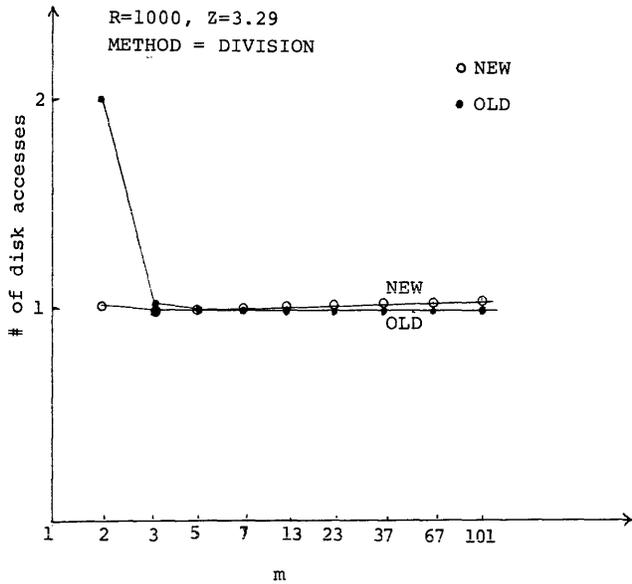


Figure 7—Response time comparison for NAM.

is that various applications have different rates and period of insertions.

From the testing results, we know the storage space utilization is economical for a larger DSA. How to partition a large DSA into a number of small blocks has been discussed in Reference 2.

If the storage cost does not exceed reorganization cost, it is economical to distribute free space within home area even if distributed free space has poor storage utilization. If storage cost exceeds reorganization cost, it is very costly to leave large amount of free space. In this case, optimal reorganization point should be considered and the associated topics have been studied in References 11-13.

REFERENCES

1. Bayer, R., and E. McCreight, "Organization and Maintenance of Large Ordered Indexes," *Acta Informatica*, Vol. 1, 1972, pp. 173-189.
2. Chin, Y. H., "An Analysis of Distributed Free Space in Operation and Data Management Environment," *IEEE Trans. on SE.*, Vol. SE-4, No. 5, 1978, pp. 436-440.
3. *DOS/VS Data Management Guide*, Form No. GC53-5372-3, IBM Corporation.
4. Feller, W., *An Introduction to Probability Theory and Its Applications*, Vol. 1, John Wiley & Sons Inc., New York, N. Y., 1973, pp. 176-177.
5. Johnson, N. L., and S. Kotz, *Discrete Distributions, Univariate Distributions-1*, Houghton Mifflin, Boston, 1969.
6. Keehn, D. G., and J. O. Lacy, "VSAM Data Set Design Parameters," *IBM System Journal*, Vol. 13, No. 3, 1974, pp. 186-212.
7. Knuth, D. E., *The Art of Computer Programming*, Vol. 1, Addison-Wesley, Reading, Mass., 1973, p. 58.
8. Knuth, D. E., *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, Mass. 1973, pp. 473-479.
9. Lum, V. Y., P. S. T. Yuen and M. Dodd, "Key-to-address Transform Technique: A Fundamental Performance Study on Large Existing Formatted Files," *Comm. ACM*, Vol. 14, No. 4, April 1971, pp. 228-239.
10. Martin, James, *Computer Data-Base Organization*, Prentice-Hall, Englewood Cliffs, N. J., 1977.
11. Maruyama, K., and S. E. Smith, "Optimal Reorganization of Distributed

12. Shneiderman, B., "Optimum Data Base Reorganization Points," *Comm. ACM*, Vol. 16, No. 6, June 1973, pp. 362-365.
13. Yao, S. B., K. S. Das and T. J. Teorey, "A Dynamic Database Reorganization Algorithm," *ACM TODS*, Vol. 1, No. 2, June 1976, pp. 159-174.

APPENDIX

$$\begin{aligned}
 E(r_i | S_i = n) &= \sum_{i=1}^I iP(i) = \sum \frac{i \binom{I}{i} \binom{R}{n}}{\binom{R+I}{n+i}} * \frac{n}{n+i} \\
 &= n \binom{R}{n} \sum \frac{I!}{(I-i)!(i-1)!} \\
 &\quad * \frac{(n+i-1)!(R+I-n-i)!}{(R+I)!} \\
 &= \frac{n \binom{R}{n} I!}{(R+I)!} \sum \frac{(n+i-1)!(R+I-n-i)!}{(i-1)!(I-i)!}
 \end{aligned}$$

Let $J=R-n$

$$\begin{aligned}
 r &= R-n+I \\
 &= \frac{n \binom{R}{n} I!}{(R+I)!} * n! * J! \sum \binom{n-1+i}{n} \binom{r-i}{J} * \\
 &= \frac{n \binom{R}{n} I! n! (R-n)!}{(R+I)!} \binom{n+r}{J+n+1} \\
 &= \frac{n \binom{R}{n} I! n! (R-n)!}{(R+I)!} \binom{R+I}{R+1} \\
 &= \frac{nR! I! n! (R-n)! (R+I)!}{n! (R-n)! (R+I)! (I-1)! (R+1)!} \\
 &= \frac{nI}{R+1}
 \end{aligned}$$

$$\begin{aligned}
 E(r_i^2 | S_i = n) &= \sum_{i=1}^I i^2 P(i) = \sum i(i-1)P(i) + \sum iP(i) \\
 &= \sum \frac{i(i-1) \binom{I}{i} \binom{R}{n}}{\binom{R+I}{n+i}} * \frac{n}{n+i} \\
 &\quad + E(r_i | S_i = n) \\
 &= \frac{nR! I!}{(R+I)! (R-n)! n!}
 \end{aligned}$$

* 1.2.6-(25).⁷

$$\begin{aligned}
& \sum \frac{(n+i-1)!(R+I-n-i)!}{(i-2)!(I-i)!} \\
& + E(r_i | S_i = n) \\
& = \frac{nR!I!(n+1)!(R-n)!}{(R+I)!(R-n)!n!} \\
& \sum \binom{n-1+i}{n+1} \binom{R+I-n-i}{R-n} \\
& + E(r_i | S_i = n) \\
& = \frac{nR!I!(n+1)!(R-n)!}{(R+I)!(R-n)!n!} \\
& \times \binom{n-1+R+I-n+1}{n+1+R-n+1} + E(r_i | S_i = n) \\
& = \frac{nR!I!(n+1)!(R-n)!}{(R+I)!(R-n)!n!} \binom{R+I}{R+2} \\
& + E(r_i | S_i = n) \\
& = \frac{n(n+1)I(I-1)}{(R+2)(R+1)} + \frac{nI}{R+1} \\
\text{VAR}(r_i | S_i = n) & = \frac{nI}{R+1} \left[\frac{(n+1)(I-1)}{R+2} + 1 - \frac{nI}{R+1} \right]
\end{aligned}$$

Forecasting computer resource utilization using key volume indicators

by DAVID E. Y. SARNA

Price Waterhouse & Co.
New York, New York

INTRODUCTION

The purpose of capacity planning is to determine how much computer power we have overall, how much we are using and, most importantly, how much is left and how much is required. There is some justification for the current interest in capacity planning. We have recently improved our ability to predict the quality of service, i.e. the turnaround time that can be expected from a given configuration when a known workload is imposed. This is a basic activity in capacity planning.

If we use the right tools, and obtain the right measurements, we can determine the service requirements of existing workloads. Capacity planning tools, usually based on queueing theory, enable us to predict with remarkable accuracy the effect on utilization, thruput, turnaround and response times to be expected for a given change in computer workload or configuration. The challenge then comes not in predicting how much raw capacity remains unused on the existing computer, but rather in predicting the expected workload. The specific equipment required can be determined straightforwardly once the expected workload has been established.

CURRENT FORECASTING TECHNIQUES

Forecasting growth in computer utilization is the key to successful capacity planning. There are three basic techniques that have been used up to now, with varying degrees of success, to forecast computer utilization:

1. Divide up the total current usage by department. Ask each department to estimate next year's requirements in terms of CPU-seconds and EXCPs.
2. Take this year's job accounting figures and adjust them for expected change over the next year.
3. Apply a trend analysis to this year's job accounting tapes to obtain an (overall) forecast for next year.

What are the disadvantages of these techniques?

1. *Ask the user*—Try forecasting your monthly requirements for dishwashing liquid! To most users, CPU-

seconds or EXCPs are quantities even more obscure and difficult to estimate.

2. *Estimate based on last year's results*—This is the classic seat-of-the-pants approach. Some are better at it than others.
3. *Watch the trend*—This is a step in the right direction. Use of this technique presupposes that next year's usage pattern will mirror this year's. If sufficient data are used and the trending techniques are sufficiently sophisticated, good results can often be obtained. However, there is no assurance that the current trend will continue. For example, an installation may experience flat growth this year, but may then experience a large workload increase when the applications currently under development go on-line. Simple trend analysis is not going to predict that type of growth. In a decentralized environment, such as is found in many RJE-oriented shops, this problem is particularly acute.

KEY VOLUME INDICATORS

We have been using a concept called key volume indicators (KVI) to overcome deficiencies in forecasting using existing methods. The tools required are a job accounting analysis program and a regression program found in statistical software packages and available on many pocket calculators. The underlying principle behind this technique is that the end user, given sufficient information, understands his business best. He should be responsible for predicting his own needs. He cannot be expected to predict computer usage, but he probably can predict business-related growth with a fair degree of success. The key volume indicator is a way of relating an application's inputs and outputs to computer utilization. Appropriate key volume indicators (KVI) must be chosen in order to prepare reliable forecasts. The key volume indicators must relate to computer usage and at the same time be business- and application-related in order to be forecastable by the users. Procedures have been developed to assist in selecting statistically valid indicators based on the accumulated volume and computer usage data relating to an application. A useful by-product of the process will be the availability of unit costs for many applications,

permitting comparisons among user constituencies and better cost estimates for planned applications.

Once the indicators have been identified, the user will prepare a forecast in terms of the key volume indicator units, and the computer will translate these units into a forecast of computer resources. EXCPs, CPU-seconds, or any other measure of utilization can be forecast using this technique.

TYPES OF WORKLOADS

Growth in corporate computer use will come primarily from three sources: (a) Increased workloads for existing applications, (b) shifts from batch to on-line processing and (c) new applications. Other factors influencing use are related to changes in processing time caused by program modifications, new techniques such as a conversion from sequential processing to a data base management system, or changes in run frequency.

DETERMINING KEY VOLUME INDICATORS FOR AN APPLICATION

To determine key volume indicators, potential indicators are selected for their forecastability, relationship to the application and availability of historical data. Then, historical volume and computer utilization data is examined statistically, in order to select the potential volume indicators with the greatest correlation to computer utilization over time.

MONITORING AND IMPROVING FORECASTING ABILITY

Forecasts will improve because (1) users will be able to do a better job of forecasting key volume indicators than computer-related measures such as CPU seconds or EXCPs, (2) the indicators will be chosen carefully and bear a known relationship to computer utilization and (3) users will prepare forecasts periodically and as they gain experience in actual usage as compared to their forecasts, their forecasts should improve.

STANDARD COSTS

An additional benefit from the use of key volume indicators is the collection of data relating to the computer resources required per KVI unit of work. This data can be used to develop standard costs for applications, and to compare the standard costs with actual costs to highlight variances for further study and possible management action. Comparisons of the costs of similar applications among the divisions should assist in identifying inefficient programs and in reducing costs.

DETERMINATION OF THE ADEQUACY OF INSTALLED EQUIPMENT TO MEET PROJECTED REQUIREMENTS

Simulation models based on queueing network theory can be used to predict the effect on response times and CPU utilization of workload changes. For a given computer configuration and workload, the model will predict the average CPU utilization, the average batch job turnaround times and the average terminal response times to be expected due to workload changes. The model also can be used to predict the effect of volumes different from the forecasts.

NEW APPLICATIONS

Usage data is not available for applications not yet installed. If a similar application is installed at another division, the key volume indicators and coefficients may be borrowed as a first approximation. If such comparable data is not available, the analyst will have to base his estimate on the time required to process the approximate number of CPU instructions and I/O operations per transaction. If a new application will replace an existing application, care must be taken to deduct from the total forecast all resource utilization to be displaced by the new application. After the application is placed in production, and the usage data becomes available, the procedure for existing applications should be followed.

ON-LINE APPLICATIONS

On-line application usage can be projected using basically the same procedures as for batch systems, using key volume indicators appropriate to the on-line system. However, interactive programming systems pose a special problem. We have found that the number of programmers is an appropriate KVI since in most shops the amount of computer utilization for interactive systems is directly related to the number of programmers doing interactive programming, and this relationship is fairly constant from month to month.

STEPS IN DEVELOPING THE FORECAST

Forecasts for existing and new applications are prepared in terms of key volume indicator work units. These figures are extended by coefficients produced by the statistical routines to give forecasts of computer resource utilization. A forecast is also prepared for all other computer workloads. The sum of all the individual forecasts gives the total computer utilization forecast, which may be adjusted to take into account additional factors, such as overall improvements to operations as a result of equipment changes, or performance improvement programs. The procedure for forecasting computer utilization will be summarized. More detailed procedures are provided in the appendices.

1. List the potential key volume indicators. For example, potential indicators for an order/billing/accounts receivable system commonly found in computer shops might include: number of invoices, number of orders, number of updates, number of line items, number of parts in the file, and dollar invoice value.
2. Sort and summarize the job accounting records (SMF tapes) by major application system. At least six months' worth of records should be used. Please note that summarization should be by major application. In many shops one or two letters of the job name or other job accounting information is used to identify the major application.
3. Summarize the total monthly resource consumption by CPU-seconds, EXCPs, monthly computer charges, or any other quantity to be forecast. The resource usage data are referred to as dependent variables.
4. If a significant portion of the application is run on a daily basis, divide the figures obtained in Step 3 by the number of workdays per month.
5. Obtain from the appropriate user department actual monthly volume figures for the potential key volume indicators identified in Step 1, for the accounting periods used in Step 2. If the computer usage was divided by the number of workdays per month in Step 4, divide the potential key volume indicators by the same figure. The data for the key volume indicators are referred to as independent variables.
6. Perform stepwise multiple linear regression using the potential indicators as the independent variables and CPU-seconds, or other quantities to be forecast as the dependent variables. The important outputs from the regression program are (a) regression coefficients and (b) a standard error of estimation. The regression coef-

ficients for each computer resource, when extended by a forecast expressed in terms of key volume indicators, predict the expected utilization of the resources by the application being forecast.

Potential key volume indicators are ranked by the regression program in order of correlation. The standard error of estimation tells us how well the coefficients explain or predict the input data. An indication of the "goodness" of fit is given by the coefficient of determination. A high value indicates a good fit and a low value indicates a poor fit.

Generally speaking, we have found that one or two potential key volume indicators will account for most of the utilization and these become the key volume indicators. At one client, we found that the number of invoices could be used to explain almost all utilization within an order-entry/billing/accounts receivable application. (See Figures 1 and 2).

FORECASTING PROCEDURE

How are key volume indicators used in practice? Assume that the number of monthly invoices has been found to be an appropriate key volume indicator. The user would then be given a report showing the number of invoices produced each month over the past year. He is asked to prepare a quarterly forecast of invoices—the key volume indicator—to be generated over the next year. His forecast is extended by the regression coefficients to obtain the resource forecast. This process is continued for all major applications. In most shops, Pareto's law applies. Most of the utilization can be accounted for by a relatively few major applications, and only these applications need to be forecast individually. All

	OCT	NOV	DEC	JAN	FEB	MAR	APR
1. Key Volume Indicators							
Numbers of invoices	24,017	21,570	23,411	21,644	23,476	26,311	24,774
Number of updates	3,145	2,960	2,012	2,500	2,709	2,108	2,013
Number of open-term records	49,752	51,552	49,896	45,144	53,136	48,384	55,512
2. CPU Utilization							
Measured (minutes)	273	268	262	280	250	221	230
Regression: invoices and updates. $R^2=.77$, $a=430.74$, $b=.01$, $c=.01$	260.5	279.6	250.7	272.8	259.4	226.3	238.0
Regression: invoices and open items. $R^2=.80$, $a=585.24$, $b=-.01$, $c=.00202$	252.3	272.9	258.4	285.1	251.3	233.5	233.0
Regression: invoices only. $R^2=.70$, $a=505.64$, $b=-.01$	251.0	277.0	257.4	276.2	256.8	226.7	243.0
3. EXCP Utilization							
Measured (000)	4205	4268	3970	4384	3908	3595	3651
Regression: invoices and updates. $R^2=.86$, $a=6624.65$, $b=-.13$, $c=.22$	4087.5	4365.9	3915.5	4254.3	4061.5	3558.4	3737.0
Regression: invoices and updates. $R^2=.86$, $a=8866.79$, $b=-.15$, $c=-.03$	3956.6	4265.2	4041.4	4433.2	3941.5	3658.4	3684.0
Regression: invoice only. $R^2=.78$, $a=7768.21$, $b=-.16$	3930.8	4321.7	4027.6	4309.9	4017.2	3564.2	3809.0

Figure 1—Key volume indicators in an accounts receivable application.

The above figure shows the measured resource consumption and the expected utilization predicted by the regression equation using the indicated KVIs. It can be seen that the best correlation was achieved using both invoices and open items as KVIs. However, forecasting using only invoices as the KVI still gives an acceptable correlation. Other potential KVIs, such as volume of vendor master records, did not correlate well.

other utilization is lumped together, and the number of jobs used as the key volume indicator. Regression is performed and the regression coefficients obtained. These coefficients will be extended by the overall growth forecast for the coming year. Note that, where a CPU utilization forecast is being prepared, the sum of the individual forecasts must be multiplied by the ratio of the total CPU time recorded by a hardware or software monitor to the total recorded by job accounting (the "capture ratio") to obtain a forecast for the total CPU consumption. This correction is necessary because most job accounting systems do not allocate all of the CPU time to individual application programs. For example if it is known that only two-thirds of the total CPU time is allocated by a job accounting system, the total result forecast would be multiplied by three over two to obtain a more accurate estimate of the total CPU consumption being forecast.

FORECASTING ACCURACY

What unusual conditions may arise when preparing forecasts using key volume indicators? The most obvious possibility is that the standard error reported by the regression program is unacceptably large. This indicates either that the key volume indicators selected are not, in fact, good predictors of computer utilization, or that the data are incomplete or reflect a temporary exceptional condition. This question can be resolved by inspecting the output from the regression program.

Most regression program listings contain the following:

<i>Regression Program Output</i>	<i>Explanation</i>
Independent variables:	Key volume indicators
Dependent variables:	Resources utilization being forecast:
<ul style="list-style-type: none"> • Expected values • Actual values • Variance 	<ul style="list-style-type: none"> • Predicted values • Input values • Difference between expected and actual values

If the variance is large for most months, the potential key volume indicators were not found to correlate well with utilization, and different indicators must be selected. However, if the data generally correlate well, but correlate poorly for a few months, this would also impact the standard error of estimation. If possible, the data should be researched to determine whether there were any unusual conditions relating to processing the application system for those months where the variance is large. The non-representative data should then be disregarded and the regression program run once again, using the remaining data. This will often produce an acceptable result.

Another item of concern in some shops relates to distribution of the workload by shift. The procedure described will calculate a total load for the CPU. Where an installation's workload is distributed unevenly over the day, it may be necessary to calculate the expected load on a per-shift

basis. To do this, the accounting data should be sorted using the shift as the major sort key and the application name as the minor sort key. Regression would then be performed for each shift individually. The assumption here is that the distribution of utilization throughout the day will be the same next year as in the past year. If this assumption is definitely known to be incorrect, the forecast should be appropriately adjusted.

Other adjustments to the forecast may be necessary. For example, a computer performance improvement program may be under way, and an overall reduction of perhaps 20 percent in CPU utilization may be anticipated. The forecast would then be reduced by the amount of expected performance improvement.

ADVANTAGES OF KVI FORECASTS

What are some of the major advantages to be expected from using key volume indicators to predict computer utilization? The main advantage is that the forecasts themselves will be prepared by the end-user who will now be forecasting in terms he can understand. Forecast usage is then accurately related to resource consumption. Once use of key volume indicators is instituted, the reliability of the forecast can be monitored. This feedback can be expected, over time, to improve the forecast's accuracy. Moreover, if the user exceeds his forecast utilization, he cannot demand that the computer center handle the unexpected workload with the same ease that it handles scheduled work. On the other hand, users who consistently overforecast their requirements should be penalized by the pricing algorithm employed in the computer center. For example, the rate charged the user should be based on forecast usage. Unforecast usage should be charged at a higher rate. Usage forecast but not used should be charged for, although perhaps at a lower rate than charges for actual usage.

As previously noted, other benefits expected through use of key volume forecasts are the accumulation of standard cost information and the suitability of the data generated for input to a queueing model. This makes it easy to predict not only the total utilization "demand" but also the ability of existing and/or planned configurations to handle the expected workload.

UTILIZATION FORECASTS AND CAPACITY PLANNING

One last observation relates to the accuracy of forecast: In our experience with this technique, accuracy was surprisingly good. However, a key point to remember, though, is that computer capacity does not generally come in very small increments. The purpose of preparing a capacity forecast is to determine the need for additions or changes to the computer configuration. The forecasting accuracy is sufficient if it can correctly predict the need for equipment changes. One useful way to test the sensitivity of the capacity prediction to small changes in user forecasts is to

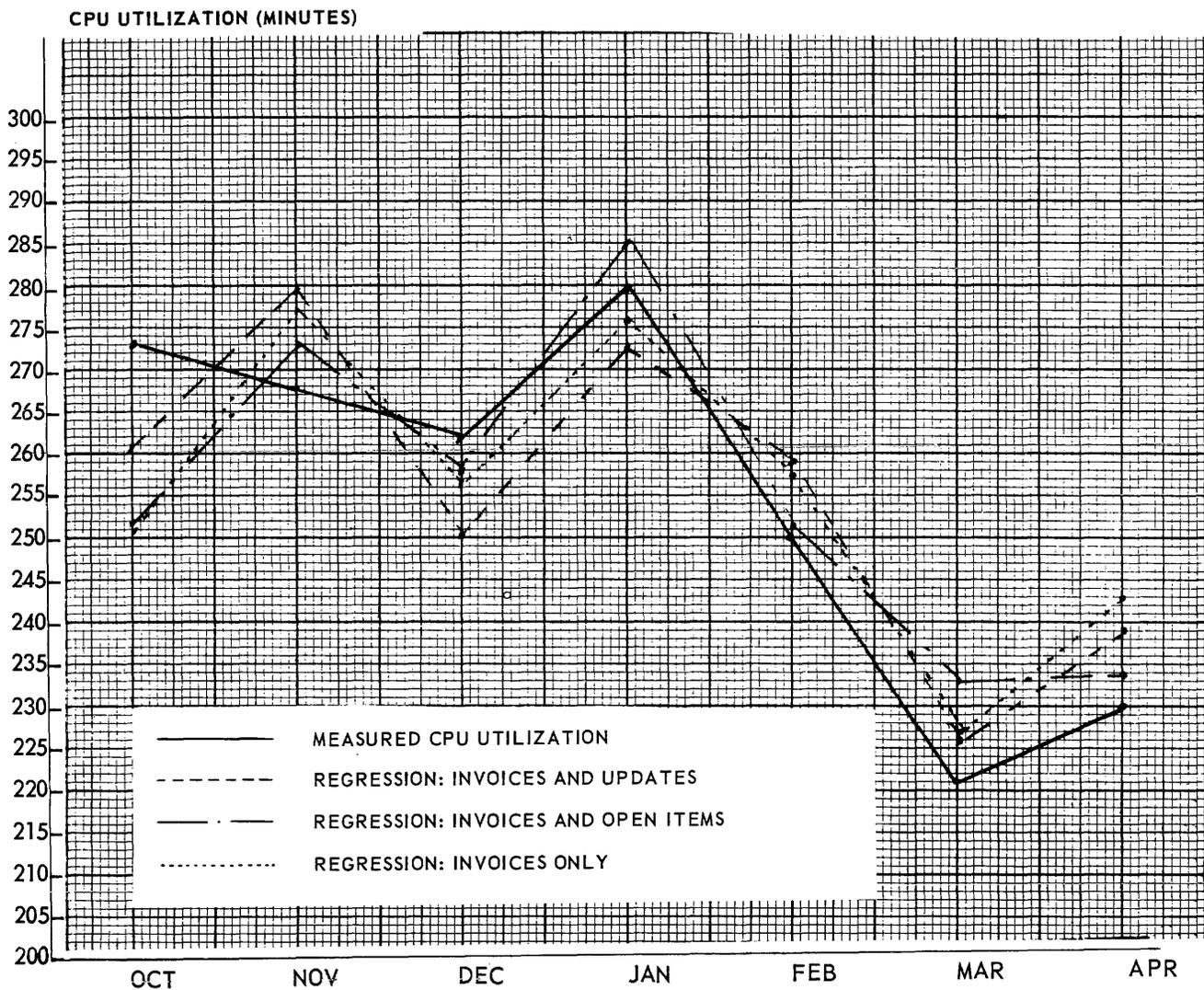


Figure 2—Actual and predicted CPU utilization in an accounts receivable application.

bracket the projected forecasts when preparing the capacity plan. This is especially easy to accomplish where a model is employed. As is well known, up to a point, computers are very tolerant of additional workloads imposed upon them; they respond with only small changes in turnaround time and thruput until a critical point or knee is reached. Loads in excess of the critical values will cause serious deterioration in service. By appropriately bracketing the forecast assumptions, it would be possible to estimate the equipment's ability to handle the probable range of the expected computer workloads.

We have found the use of key volume indicators to be a useful method of improving the accuracy of computer utilization forecasts.

REFERENCES

1. Bonner, L., "An Introduction to Capacity Planning," GG22-9001-00. IBM Washington Systems Center Technical Bulletin, January 1977.

2. Beizer, B., *Micro-Analysis of Computer System Performance*, Van Nostrand Reinhold Co., 1978.
3. Buzen, J. P., "Principles of Computer Performance Modelling and Prediction," *Performance Modelling and Prediction*, Vol. 2, 1977.
4. Buzen, J. P. et al., "BEST/1™—Design of a tool for computer system capacity planning," *AFIPS Conference Proceedings*, Vol. 47, pp. 447-455.
5. Denning, P. J., and J. P. Buzen, "The Operational Analysis of Queueing Network Models," *Computing Surveys*, Vol. 10, No. 3, pp. 225-261.
6. Wesolowsky, G. O., *Multiple Regression and Analysis of Variance*, John Wiley & Sons, Inc., 1976.

APPENDIX 1—PROCEDURE FOR DEVELOPING EQUIPMENT UTILIZATION FORECASTS USING KEY VOLUME INDICATORS

1. Select the key volume indicator for each application for which a utilization forecast is to be prepared, by carrying out the appropriate procedure (batch, on-line or new) for determining key volume indicators in Ap-

pendix 2, 3, or 4. As a result of this step, regression coefficients will also be obtained for each application.

2. Obtain the users' forecasts for each application in terms of the key volume indicators identified in Step 1.
3. To forecast CPU seconds per month and I/O count per month, extend the volume forecasts by the corresponding regression coefficients obtained in Step 1.
4. Adjust each forecast to take into account other factors, such as expected
 - Changes in run frequency or additional runs.
 - Equipment and software changes.
 - Changes in resource utilization due to optimization or tuning.
 - Any other factors expected to influence computer use.
5. Forecast the net additional utilization from all other applications and for system software. To do this, for the most recent month, subtract from the total monthly utilization (CPU seconds and I/O count) that portion of utilization represented by applications forecast individually in Step 3, above. This result represents the net additional utilization from all other applications ("other work"). Determine regression coefficients for the net additional utilization for other work by carrying out the procedure for determining key volume indicators. Use the number of jobs as the potential key volume indicator. As in Step 3, extend the regression coefficients obtained by the net number of jobs forecast for other work to obtain a forecast of CPU seconds per month or I/O count per month.
6. In order to obtain the forecast of total CPU seconds and I/O counts, add the individual forecasts resulting from Step 4 to the net additional utilization developed in Step 5, giving a forecast of total utilization.
7. To correct for inaccuracies in recording CPU utilization by job accounting routines, multiply the total CPU utilization obtained in Step 6 by the ratio of total CPU utilization, as measured by a software or hardware monitor to the total CPU utilization recorded by the job accounting programs. Other resource measures, such as I/O counts, are usually accurately recorded by job accounting programs and do not require adjustments.
8. The ratio of the forecast to current utilization gives the growth percent. This growth percent is used by capacity planning tools such as queueing models to predict the effect of the changed workload on terminal response times and job turnaround times.

APPENDIX 2—PROCEDURE FOR SELECTING KEY VOLUME INDICATORS FOR INSTALLED BATCH APPLICATIONS

1. For each application, list all transaction types and the organization (key or sequence) of each major file and report. These are potential key volume indicators. Also

identify the job names used to identify the application to the operating system.

2. From this list, identify logical intersections within the application. For example, paychecks/employee and updates/employee are logical intersections within a payroll system, and line-items/order is a logical intersection for an order entry system. These also are potential key volume indicators.
3. List any other potential key volume indicators.
4. Reduce the list to those potential indicators that would be forecastable by the divisions, and for which historical data could be collected. These are called "predictors."
5. Obtain the monthly volume data by application for these predictors, together with the corresponding CPU seconds and total I/O count, which are figures that should be available in the job accounting reports. These are the major measures of equipment utilization most subject to fluctuation and directly related to individual applications and, therefore, these are the statistics that will be forecast.
6. Use a statistical package, such as BMD, developed by the University of California, or SPSS, developed by the National Opinion Research Center of the University of Chicago, to perform stepwise linear regression using the predictors as the independent variables and the CPU seconds and I/O counts as the dependent variables. The regression program should be used for each application to relate CPU seconds to each predictor and also I/O count to each predictor. The advantage of running the regression program separately for CPU seconds and for I/O count is that it is possible that certain predictors will correlate well with CPU seconds and others will correlate well with I/O count. Input to the regression program consists of the monthly volumes for each predictor, and the CPU seconds or I/O counts for that application. In addition, certain regression programs must be provided parameters which limit the number of program iterations to the most likely range of coefficient combinations and weightings. Appropriate values to be specified are 1.0 for the "F-level for inclusion" parameter, 0.5 for the "F-level for deletion" parameter, and a "tolerance level" of .001. These values will minimize the computation required.

The regression program calculates the correlation between predictor volume and computer utilization (CPU seconds and I/O count). This correlation is expressed in values called regression coefficients for each predictor. The regression program automatically bypasses predictors that do not correlate well with computer utilization. Output of the program is a rank listing of predictors and regression coefficients in order of best correlation. For each predictor following the first, the program gives the additional correlation that is obtained by using that predictor in addition to the prior, better predictors.

Also given by the program is the multiple correlation coefficient called R^2 . This is the precision of the regres-

sion coefficient for the group of best predictors selected by the program. R^2 is calculated by the statistical program based on the variance between the actual utilization data and the utilization predicted by the regression program. The value of R^2 can range from 0.0, for an imperfect fit, to 1.0 for a perfect fit. An R^2 value of 0.7 or more is usually adequate for planning since the objective of a utilization forecast is equipment capacity planning, which usually involves large increments of capacity. A small value for R^2 may indicate either that the predictors are unsatisfactory key volume indicators or that certain historical data associated with the predictors may have been skewed by some additional factor. Therefore remove the one or two months' data (called "cases" in the regression computer output listing) with highest "residual," that is, the largest difference between the predicted and empirical results. Rerun the regression program. If the value of R^2 is satisfactory, proceed to Step 7. If the precision ratio is still unsatisfactory, additional predictors must be chosen.

7. When a satisfactory precision ratio has been achieved, the predictors selected in the final regression step should be used as key volume indicators.

APPENDIX 3—PROCEDURE FOR SELECTING KEY VOLUME INDICATORS FOR NEW APPLICATIONS

For new applications, no usage data would be available and a different approach must be used.

1. If forecasts have been prepared for a similar application (perhaps one in use at another division) the regression coefficients obtained for that application may be borrowed and used in developing equipment utilization forecasts.
2. If there are no similar applications, the following "rules of thumb" may be used in place of the resource utilization data normally collected:
 - a. Based on the feasibility studies and system design documentation, estimate the number of I/O operations by multiplying the average number of I/O operations for each transaction type by the expected monthly transaction volume. Use this estimate in place of actual I/O counts.
 - b. Average CPU utilization can be estimated by using the average CPU utilization per key volume unit. In one test on an IBM 370/158 computer, the average CPU utilization for batch portion of the accounts receivable application was 0.6 seconds per invoice and for on-line CICS applications the average CPU time per transaction was found to be 0.3 seconds. These rates appear to be representative. Therefore, to estimate for any other CPU of similar architecture, multiply these figures by the ratio of the relative CPU speeds of the 370/158 and the other CPU.

c. If a more detailed estimate is desired, the following procedures can be used:

- Estimate the number of CPU instructions required for processing each transaction, based on the system design specifications. Multiply the number of I/O operations obtained in Step 2a by the average number of machine instructions per I/O operation to obtain the number of CPU instructions for I/O processing. Add the number of CPU instructions for transaction processing to the number of CPU instructions for input/output processing to obtain an estimate of total CPU instructions per transaction. Multiply by the average machine instruction time to obtain estimated CPU time per transaction. Multiply the CPU time per transaction by the expected monthly transaction volume to obtain expected total monthly CPU utilization for the application.
 - For the IBM 370/158 running under the OS/SVS operating system and using the COBOL/VS compiler, the estimated number of COBOL statements executed can be multiplied by 16 to give the estimated number of machine instructions. About 2,000 instructions are required for each EXCP. The average machine instruction time is 1.24 microseconds.
3. Subtract the computer resources used by any existing programs expected to be displaced by the new application from the estimates obtained in Step 1 or Step 2 to obtain the net increase expected from the planned new application.
 4. Include the estimates developed above when carrying out Step 6 of the procedure for developing the equipment utilization forecasts (Appendix 1).
 5. Once the application is placed into production and meaningful utilization figures become available (probably after a three-month shakedown period), the appropriate procedure for determining key volume indicators should be carried out using actual utilization statistics.

APPENDIX 4—PROCEDURE FOR SELECTING KEY VOLUME INDICATORS FOR ON-LINE APPLICATIONS

On-line systems can be projected using basically the same procedures as for batch systems.

1. List the potential key volume indicators. The basic unit of work for an on-line application is the transaction. Most on-line teleprocessing monitors collect statistics about the number of executions of each transaction type, and also the number of transactions per terminal. The major transaction types and the number of terminals are thus potential key volume indicators. For on-

- line programming systems, the number of programmers is frequently an appropriate key volume indicator.
2. Reduce the list to those potential indicators that would be forecastable by the divisions, and for which historical data could be collected. These are called "predictors."
 3. Obtain from the teleprocessing monitor monthly data for those predictors together with the corresponding utilization, i.e., CPU seconds and total I/O count.
 4. Perform Steps 6 and 7 of Appendix 2 (*Procedure for Determining Key Volume Indicators for Batch Applications*).

ACKNOWLEDGMENT

The author acknowledges with thanks the assistance of Peter V. Cohen.

Workflow—A technique for analyzing JES systems

by H. PAT ARTIS

Bell Laboratories
Piscataway, New Jersey

INTRODUCTION

During the last decade, the mode of operation of many centralized computer installations has been significantly changed by the widespread use of remote job entry (RJE) facilities. The increasing usage of remote work stations presents the analyst a significant problem in sizing remotes to efficiently handle the flow of jobs. Currently, few, if any, tools exist to aid the analyst in this area.

A tool called Workflow Analysis has been developed to assist the analyst in identifying bottlenecks and sizing JES systems. This tool processes the System Management Facility¹ (SMF) log file to produce graphical reports on the queues managed by a JES subsystem.

BACKGROUND

RJE was initially provided to IBM users by the HASP and ASP² spooling packages in the late 1960s. Since then, the number of installations using RJE facilities and the percentage of their system's workloads handled by the remotes have steadily increased. When the MVS operating system was introduced, HASP and ASP evolved from optional spooling packages to subsystems called JES2 and JES3. Today, a typical JES2 or JES3 system serves 20 to 30 remotes with a high percentage of the system's workload being submitted by the remotes.

Coincident with the evolution of the software to support RJE has been the development by IBM and other vendors of a wide range of potential remote devices. They range in size from a 360 or 370 CPU work station supporting multiple readers and printers to small work stations or minicomputers that emulate a remote's protocol. With the variety of available hardware, a user can size a remote from a single 300 to 500 line-per-minute (LPM) printer to a CPU that can handle multiple 2000 LPM printers. The development by many vendors of moderately-sized* and -priced devices has hastened the proliferation of remotes by allowing more users to request and justify their own workstations.

* These remote stations typically consist of a 300 card-per-minute (CPM) reader and a 1000 line-per-minute (LPM) printer. These values are rated speeds that are usually degraded by transmission line capacity and other influences.

Experience has shown that only one or two poorly-sized remotes can significantly skew turnaround statistics for an entire JES system. Also, it is generally the users who control the flow of work to and from the remotes by their selection of an input station and specification of an output destination. Hence, a tool was needed not only to assist in the sizing of remotes but to monitor and report on their daily utilization.

JES MEASURABLES

To identify and comment on the measurable quantities in a JES system it is useful to consider the diagram in Figure 1. As shown in the figure, at the center of the system is one or more processors that process the jobs submitted by the users.** There are three methods for entering a job into the system. They are

- Reading in the job at a local input device.
- Reading in the job at a remote.
- Submitting the job to the system's internal reader. The internal reader is available to TSO users and any job currently executing in the system.

When a job is entered into the system it is queued for execution. The input queues are maintained by job class and JES provides priority queuing within each job class.

Once a job has been processed, its output is queued for spooling. The output queues are maintained by destination,*** output class and output forms type. Priority queuing is also available for the output queues.

It should also be noted that jobs are free to enter and exit the system at any point specified by the user. The user may also specify that various portions of a job's output be routed to different or multiple destinations.

One of the most convenient methods for studying a JES system is to measure the content of the queues on an interval basis. The following measurements of system activities are

** JES2 systems typically contain a single processor while JES3 systems consist of two or more processors. However, some users employ the shared spool facility of JES2 to control multiple processors.

*** The term "destination" refers to the location to which a file is to be spooled. A file may be spooled locally or to a remote work station.

JES SYSTEM

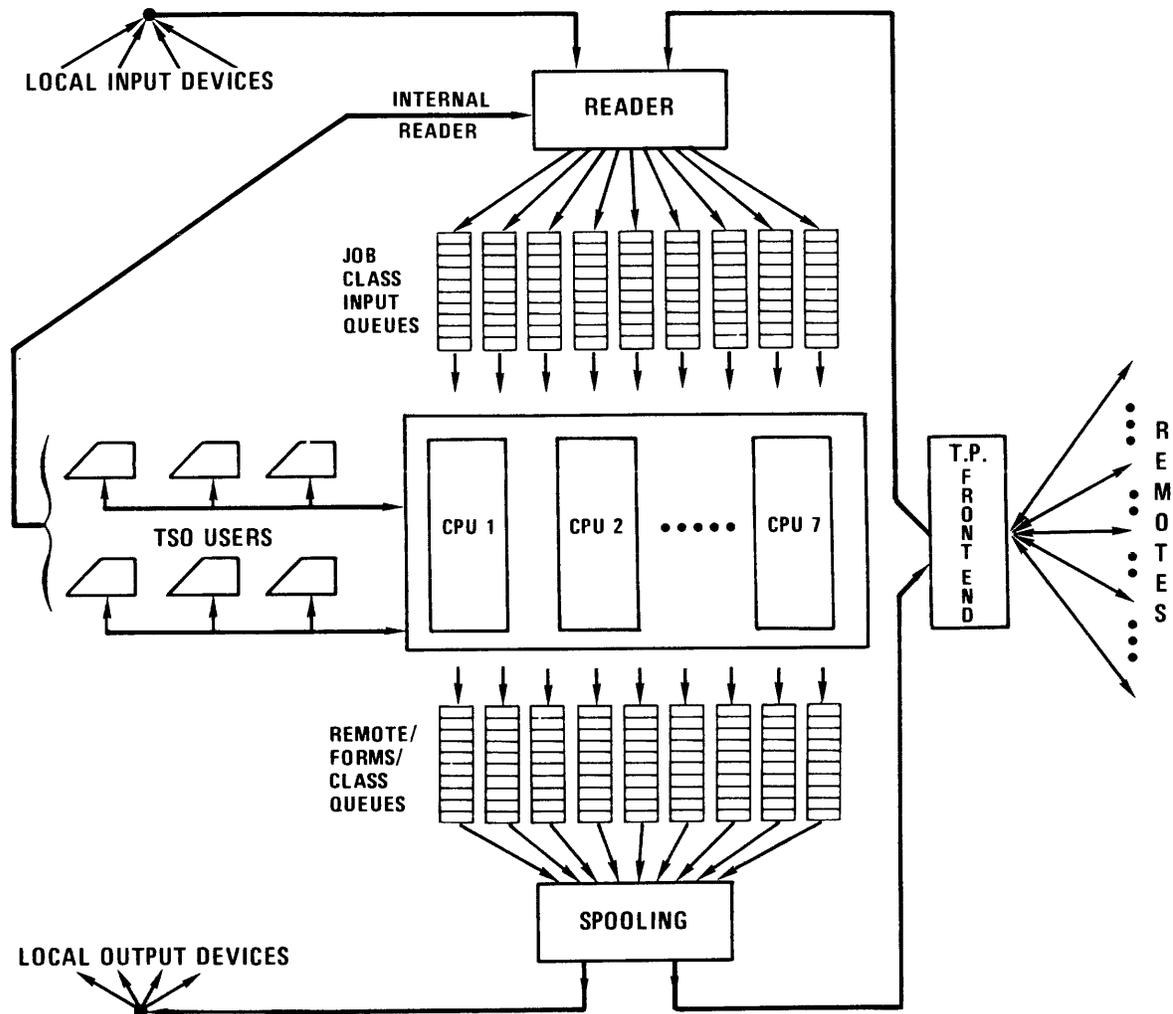


Figure 1

considered to be important for each interval:

Jobs Read—The number of jobs read into the system during the interval. This measure allows the analyst to study the load arrival pattern for the entire system. The cumulative value of this measure gives the total number of jobs that have entered the system since the start of the study.

Jobs Purged—The number of jobs purged from the system during the interval. The difference of the cumulative values for Total Jobs Read and Total Jobs Purged is number of jobs in the system.

Input Queue—The average number of jobs of all classes waiting in the input queues during the interval. This value provides a measure of the total amount of work queued for the processor(s).

Output Queue—The average number of jobs waiting in the output queues for all locations during the interval. A comparison between this measurement and the Input Queue measurement provides an indication of how well the system's output facilities are matched to the load arrival pattern.

Executing—The average number of jobs currently executing on all processors during the interval. This measure is the sum of the average multiprogramming levels of all of the processors.

Job Queues—The average number of jobs of each class queued for execution during the interval. This measurement allows the analyst to study the arrival patterns and the system's capability for serving each job class.

Furthermore, the following measurements of location (i.e.

local or a remote workstation) activities are considered to be important for each interval:

Jobs Read from Location—The number of jobs read into the system from a location during the interval. This measurement allows the analyst to study the work patterns of users at different locations.†

Jobs Backlogged from Location—The average for the interval of the number of jobs in the system that originated from a location. This measurement allows the analyst to compute the percentage of the system's current backlog that originated at any location.

Lines Backlogged to Location—The average for the interval of the number of logical records (print and punch) queued for spooling at a location. This measure allows the analyst to determine how well the hardware at a given location is matched to the load which it is assigned.

Files Backlogged to Location—The average for the interval of the number of files (print and punch) queued for spooling at a location. This measure allows the analyst to estimate how many users are waiting for output at the location.

The jobs processed by the system may also be measured. These measures are collected for each job, by job class and for all jobs. The measures are:

Input Queue Time—The input queuing delay from the time a job was read into the system until it was selected for execution.

Output Queue Time—The output queuing delay from the time the job completed execution until it was purged from the system.

Total Queuing Time—The total of the input and output queuing delays. This is the total non-productive time the jobs spent in the system.

Execution Time—The duration the job spent in execution.

Turnaround Time—The total time the job spent in the system. This measure is the total of the job's execution, input queuing and output queuing times.

IMPLEMENTATION

The measurements described in the previous section can be implemented using the data available in the JES Job Purge (Type 26) and the JES Output Writer (Type 6) SMF records. The JES Job Purge record contains the following items used by the algorithm:

- JES Job Number
- JES Input Device (location)
- Job Class

† The measure "Jobs Purged by Location" is not suggested since jobs need not be returned to the location at which they entered the system. One of the best examples of this is jobs submitted to the internal reader from TSO users. The output from these jobs is routed to some location convenient to the user for printing.

JOB STATISTICS

JOB CLASS	NUMBER OF JOBS	INPUT QUEUE (MIN)	OUTPUT QUEUE (MIN)	EXECUTION (MIN)	TOTAL QUEUE (MIN)	TURN AROUND TIME (MIN)
A	12,457	2	35	2	37	39
B	4,108	6	35	9	41	50
C	5,555	25	64	5	89	94
E	1,431	4	14	1	18	19
J	997	1	38	2	39	41
M	571	20	21	3	41	44
N	493	2	23	13	25	38
ALL JOBS	25,612	8	40	4	48	52

Figure 2

- Reader Start Time and Date, T_1
- Execution Start Time and Date, T_2
- Execution End Time and Date, T_3
- Job Purge Time and Date, T_6

Since a job may spool multiple output files there may be more than one JES Output Writer record for the job. The record for the i th output file contains

- JES Job Number
- JES Output Device (location)
- Number of Logical Records Printed
- Writer Start Time and Date, $T_{(4,i)}$
- Writer End Time and Date, $T_{(5,i)}$

The Type 26 and all of the Type 6 SMF records for a job may be assembled using the JES job number as a key.

Each of the variables to be measured must be represented by a table whose size is determined by the granularity of the interval size chosen. Currently, the existing implementation uses a granularity of five minutes and requires 288 full words for each table to represent one day.

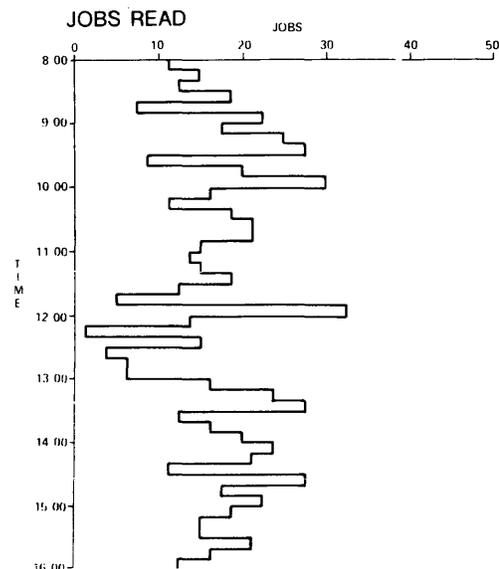


Figure 3

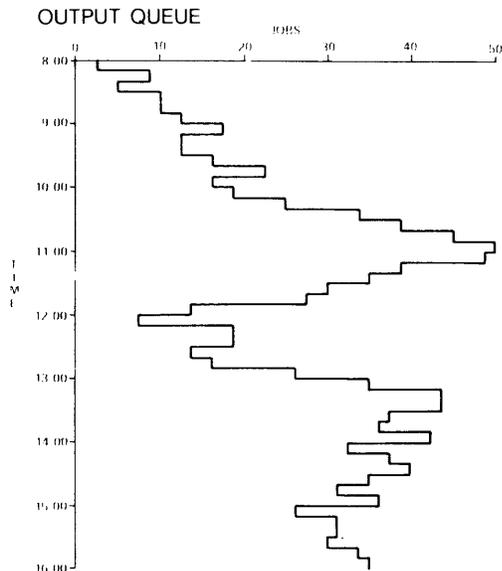


Figure 4

Using the tables and the data from the SMF Type 6 and Type 26 records, the previously-listed measures can be implemented as follows:

- Jobs Read*—Increment the table at time T_1 .
- Jobs Purged*—Increment the table at time T_6 .
- Input Queue*—Increment the table from time T_1 to time T_2 .
- Output Queue*—Increment the table from time T_3 to time T_6 .
- Executing*—Increment the table from time T_2 to time T_3 .
- Job Queues*—Increment the table for the job class specified in the Type 26 record from time T_1 to time T_2 .
- Jobs Read from Location*—Increment the table for the input location specified in the Type 26 record at time T_1 .
- Jobs Backlogged from Location*—Increment the table for the input location specified in the Type 26 record from time T_1 to time T_6 .
- Lines Backlogged to Location*—For the i th output file for a job increment the table for the output location specified in the Type 6 record from time T_3 to time $T_{(4,i)}$ or $T_{(5,i)}$ by the number of logical records to be spooled.*
- Files Backlogged to Location*—For the i th output file for a job increment the table for the output location specified in the Type 6 record from time T_3 to time $T_{(5,i)}$.

§ Unfortunately, the Type 6 SMF record does not indicate when an output file was queued. Hence, all files are usually assumed to be queued for output at the termination of the job's execution. Although this is generally true, JES does provide the user the ability to dynamically free a file to the output file during execution. In such cases, the output is assumed to have been queued since T_2 .

* The analyst may either choose to credit the output as being backlogged until the writer starts or stops. One could also choose to credit the complete output as being backlogged to $T_{(4,i)}$ and then use a linearly declining value until $T_{(5,i)}$ when zero lines would remain. The current implementation uses $T_{(5,i)}$.

- Input Queue Time*—The difference between T_2 and T_1 .
- Output Queue Time*—The difference between T_6 and T_3 .
- Total Queuing Time*—The difference between T_6 and T_1 , plus the difference between T_6 and T_3 .
- Execution Time*—The difference between T_3 and T_2 .
- Turnaround Time*—The difference between T_6 and T_1 .

CASE STUDY

A study was conducted of a moderately-loaded 370/168 JES2 system that supported TSO users and seven remote

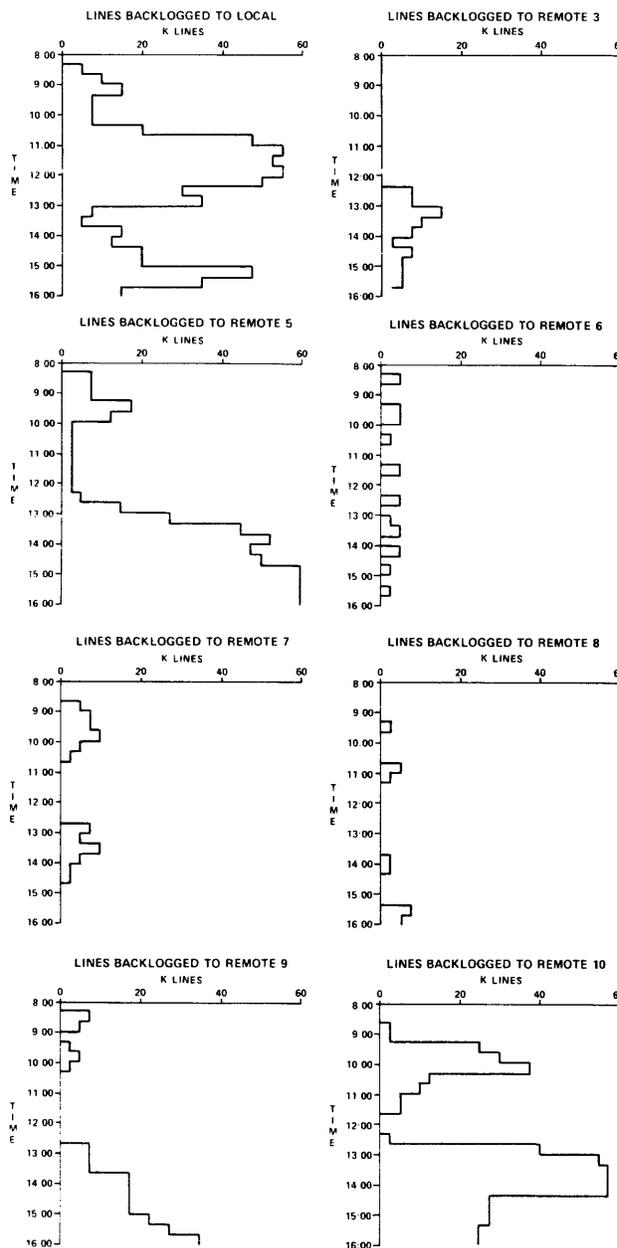


Figure 5

workstations in a testing environment. Although this system only processed about one thousand small jobs per day, the system's service objective of a 30-minute average turnaround time for batch jobs was not being achieved. One month of SMF data was obtained for the system.

The first step of the study was the analysis of the turnaround time statistics for the month. The turnaround time statistics were compiled by job class and summarized for all jobs. As shown in Figure 2, the average turnaround time for all jobs processed by the system was 52 minutes. However, the average input queuing and execution times only represented 12 minutes, 23 percent, of the average turnaround time. The low values of input queuing and execution times corresponded with the observations of moderate load and relatively small jobs. The high average value of output queuing time, 40 minutes, indicated the presence of a bottleneck in the system's output spooling.

The prime shift period, eight AM to four PM, was analyzed for a number of days in the month using the Workflow Analysis tool. In each of the days studied, the same output spooling bottlenecks were evident. A typical day is discussed in the following paragraph.

The arrival pattern of the jobs, Figure 3, is very typical for testing installations. The arrival rate of jobs peaks just before and after lunch and prior to each coffee break. The system's output queue length, Figure 4, follows the arrival pattern of the jobs. However, the output backlog does not decline as rapidly in the afternoon as it does in the morning. This led to an investigation of the Lines Backlogged to

Location reports for the local and remote stations. As shown in Figure 5, Remote Stations 3, 6, 7 and 8 are lightly loaded throughout the entire day. However, remote stations 5 and 10 present significant output bottlenecks in the afternoons with backlogs of 50 to 60 thousand lines. Since the remotes are configured with printers that average only 450 LPM, these backlogs represent more than a two-hour output delay for jobs routed to either of the remotes. It is the output delay of these two remote stations, used by about 20 percent of the jobs, that results in the system's service objective not being met. Although similar output backlogs occur at the local station, it is not a concern since the local station has several high-speed printers with a net effective rate of more than five thousand LPM.

REMARKS

The tool presented in this paper is relatively uncomplicated and is very straightforward to implement. However, it provides a great deal of information to the analyst about the flow of jobs thru a JES system that is not currently available from other sources.

REFERENCES

1. *System Management Facility*, IBM, July 1977, GC28-0706.
2. Lorin, Harold, *Parallelism in Hardware and Software: Real and Apparent Concurrency* Prentice-Hall, 1972.

MMPS—A reconfigurable multi-microprocessor simulator system

by DANIEL KLEIN

Carnegie-Mellon Institute of Research
Pittsburgh, Pennsylvania

INTRODUCTION

The computing industry is currently undergoing a quiet revolution. Along with the recent advances in LSI design and production, a new trend in computation is forming. Many corporations are realizing that an important part of the future of computers lies in distributed data processing. The emphasis here is not on distributed data, for this field has been extensively (though certainly not completely) researched by computer scientists, but on distributed *processing*.

While there exist a great many design and debugging tools for uni-processor systems, such as simulator systems, as well as off-the-shelf in-circuit emulators for almost any mini- or microprocessor on the market, there exist few real design or debugging tools for multi-processing systems. While it is relatively easy to simulate any uni-process machine on almost any other, there is no easy method of either simulating a multiprocessor or of simulating any algorithm which is designed to run on such a system. And while it is possible to hand code a multi-processor simulator, this is certainly not a desirable method. Any design modifications must be incorporated into the current simulator by reprogramming (possibly large) sections of the code. A network simulator cannot limit itself to the simulation of a specific network, with fixed characteristics and configurations. Rather, what is needed is a software system development tool which is a general purpose network simulation package to provide a simulation environment for the design, checkout and maintenance of multiprocessor computer networks.

What we have set out to do is to construct tools for the network designer. A multi-microprocessor design language and dynamically reconfigurable multi-microprocessor simulator (MMPS), (which can be used in conjunction with a reconfigurable hardware network emulator, the BUCS system,¹ provides an aid to an efficient design methodology, and is described here.

DDP system description and simulation is an important step in the development of computer network systems. It not only tests the correctness of the overall design scheme, but also that of the individual elements, with relation to speed, instruction set, unit MTBF, etc. It should also test for possible run-time hazardous conditions which might arise, such as integrity problems, before the actual construc-

tion of the network, thus enabling the designer to bypass large amounts of testbed development.

The simulator should have the capability to provide both hardware and software level breakpoint tracing, as well as a static method of tracing the overall system performance, for later evaluation and possible redesign. Such a system, coupled with a full complement of production aids such as cross-assemblers, cross-compilers and cross-interpreters, is a highly desirable precursor system to the actual construction of the network system. If this system also has the ability to interface with some high-level hardware descriptive language, so that previously undefined pieces of hardware may be easily included into the library of simulator components already available, then this system becomes a very powerful multi-processing development tool.

Unfortunately, most of the existing computer hardware descriptive languages (CHDLs) such as LOGICSPEC, AHPL, CDL, DDL and CONLAN are designed to specify, or to reduce to, the gate level of operation within the described piece of hardware. While this may suffice for describing small logic elements, such as interface devices, or even small processors, it most certainly is not ideal for describing a large computer, and even less suited for describing a network of computers.

From the system standpoint, a designer of a multi-processing network only needs to know about the interactions between the individual processors, not how each of them performs on the minute scale. Realistically, all that is needed for each element of the network is a knowledge of how fast an element can accept an input, how fast it provides an output, and that the time between input and output is a black-box function of the data provided. The black-box programming, is of course effected by the designer of the individual elements, and "hooks" into the box should be provided, should observations of the gut-level functions of the elements be desired. However, for the most part, all the system designer need know is that the black-box exists, and that it functions as specified. Gate level CHDLs are simply not suited for this purpose. On the one hand, they are exceptionally slow when simulating large systems, and on the other hand, they are not designed to be reconfigurable, as the specifications state.

There exists at least one CHDL that approximates the

desired functions. In its current state, ISPS² is semi-interpretive, but it has the capabilities to be a compiled language. In ISPS, it is sufficient to declare a memory subunit of a specific size, and not need to specify (or have automatically specified) the specific gate level functionings of a memory module. The register transfer level that ISPS reduces to is on a much higher level than the gate level reduction of most other CHDLs. Being an instruction set processor, ISPS can provide the black-box simulation desired with a small description of the hardware. A Digital Equipment Corporation PDP-8 can be described in ISPS on a page or two of code, while LOGICSPEC, for example, would require many more pages of description to perform the same operations in a greater amount of time. (An ISPS description of the Manchester Mark-1 computer, the simplest ISPS description available, appears in the Appendix). Although ISPS is better suited for a multiprocessor simulator environment than any of its companion hardware languages, it would still require some modifications to optimally interface it with a reconfigurable network simulator.³ These modifications are, however, beyond of the scope of this paper, and will be dealt with at some later date.

USER LEVEL DESCRIPTION OF MMPS

The MMPS system is currently being designed on the UNIX time-sharing system,⁴ produced at Bell Laboratories. This system was chosen for its availability on small main-frame systems, and for the multiprocessing capabilities it provides.

When the user initializes MMPS, he has available to him a number of "build" commands, for declaring and connecting elements of his network together. Once the network is defined, there are a set of "trace" commands for defining breakpoints and for passively monitoring actions within the network. There is also a set of "peek/poke" commands for loading and dumping memory, and setting or reading the values of certain busses. Additionally, on-line help is always available for any command or set of commands.

The main goal in designing the command set of MMPS was to parallel the predicted actions of a hardware designer. Thus through the use of a **DECLARE** command, the user can define any number of duplicate elements of particular (predefined) system elements. Thus, a sample command:

```
DECLARE 8085[2], MEMORY[1](4,8), MEMORY[2](8)
```

would define two 8085s, one memory with four bits of addressing space and eight bits of data, and two memories with eight bits of address and data. Using the standard pinouts of the 8085, the user could then **CONNECT** the various elements together via a generalized backplane by issuing a command of the form:

```
CONNECT 8085[*](12-15), MEMORY[1](0-3) TO  
WIRE(0-3).
```

This command would connect four of the 8085's address lines, and the four address lines of memory module 1 to-

gether on backplane wires zero through three. This effectively makes a common memory unit, shared by both processors. If the user wants to disconnect a wire, all he need do is use the **DISCONNECT** command to do so. Notice, that all of these commands are dynamic, and may be done at any time during the simulation. Thus, if the user wished to test a fault tolerant system, various elements could be **DISCONNECTED** during the run, and faulty elements could be **CONNECTED** in, thus effectively simulating a failing unit.

Special consideration has also been made for non-standard features in a network. The user has the facility with which to clock each processor at a different speed (via the **CLOCK** command), and to specify wired **OR** connections (via the **WIREDOR** command). In this way, potential difficulties may be overcome. There is also a way to specify the default floating value of any particular wire. Through the use of the **FLOAT** command, lines may be declared to float high or low, depending on the application. Special consideration has also been made for the accommodation of bidirectional busses, and tri-state bus features. Depending on the type of simulation, I/O may be simulated in any of a number of modes, including ASCII or binary/octal/hex dump modes.

The "peek/poke" commands are designed to emulate the loading and dumping of memory, but on a much higher level than is currently available to the technician. While it is of course possible to load memory from the simulator, mimicking the insertion of a different PROM, it is also possible to dump memory at any time, examine the signals on any of the wires and, most importantly, assert different signals than are currently on the bus. Since the user has the capability to "freeze" the state of the system at any time, he may dynamically alter the state of the system, and continue its operation. Additionally, through the use of the **SAVE** command, the user may save the current state of a network simulation, to be **RESTORED** at a later time.

However, of all the commands, the "trace" commands give the MMPS simulator its most power. To make a simulator better than the hardware it is emulating, the simulator must provide features that the hardware cannot. The MMPS system does this by providing both breakpoint and passive tracing facilities. MMPS provides passive tracing in the form of an active hardware bus monitor, which records in a file all bus transfers across the backplane. Once the system has been halted, this log file may be examined by a group of people at their leisure. MMPS also has three other levels of selectable tracing. Operations which occur local to a processor (such as a register to register move, or addition of two loaded operands) can be **TRACED** in a log file that is local to the processor. Since all of the log files have their entries keyed by time, multiple log files may be merged and compared to detect possible race or lock conditions. Additionally, operations may be switched to **SIGNAL** the user of their occurrence without freezing the system, or by causing a **BREAK** in operations, and holding the system in a frozen state following an action. The access of certain memory locations (such as the entry point to a subroutine) may be monitored, as well as the values of internal processor registers. Certain classes of information, such as PC overflow and halt instruction execution, are not switchable, but are

considered serious enough to warn the user of their occurrence, regardless. The system *can* be continued after this type of breakpoint error, though.

OPERATIONAL LEVEL DESCRIPTION OF MMPS

Since MMPS is not only simulating the parallel processing of a network, but also spawning a series of parallel UNIX processes to do this simulation, special consideration had to be given to the problem of simultaneous actions in the network. The problem of two processes reading or writing the same bus *at the same time* must be dealt with, as these actions can and will happen in the real network. To cope with this problem, MMPS uses an asynchronous queue to schedule the actions of the individual elements. When two elements have simultaneous actions, the scheduler allows both to occur, and then updates the state of the busses in any of a number of user selectable ways. Simultaneous reads present no problem, but simultaneous read/write operations do. The user may select a random order of evaluation, or may elect to have all reads processed before writes, or all writes processed before reads. If the wired OR capability has been selected, simultaneous writes will be dealt with according to the specified logic, and if not, an error is generated (since two processes writing at the same time to a non-wired OR bus will cause that bus to blow).

The best way to describe the mechanism of synchronization of MMPS is to call it a packet-switched network. Whenever a process needs to interact with the "outside world," it sends a packet to the "overlord" indicating which units it wishes to communicate with. Of course, a unit (or process) may interact only with those elements to which it has been **CONNECTED**. Upon receipt of this message, the "overlord" process sends as many copies of this message as are necessary to the other subunits of the network. Thus, a sample interchange between a processor and memory would be as follows:

1. Processor puts address on bus, and sets memory read request line on control bus. (Message is sent from

processor simulator to "overlord," and then to memory simulator.)

2. Memory gets signal on control bus, reads address bus, and gets the appropriate memory location. (Message is received by memory simulator, and the appropriate calculations and simulated delays occur.)
3. Memory places contents of selected address on data bus. (Message is sent from the memory simulator to the "overlord," and then to the processor simulator.)
4. Processor, after appropriate delay, reads data bus, and completes the memory fetch. (Message is received by processor simulator, and cycle is complete.)

Since simulated times may vary considerably from real times, each message also contains the time of its occurrence, so that the actions may be properly synchronized. Notice, also, that the distinction is made between internal and external actions. That is, if a process need not communicate with the "outside" world for a while, and need only update its internal registers, when it again writes a message, the time field will reflect this time spent "inside."

Since it would increase the general simulation time to have a process poll a particular bus (as does the hardware) to implement interrupt lines, a special feature has been built into the "overlord" whereby a process can advise which of its lines an interrupt can be generated on. The process can then go into an idle state (i.e. not polling, but perhaps doing other operations), and be automatically interrupted when the indicated line is asserted by another process. This method takes full advantage of the asynchronous method of coordinating the various subprocesses.

EXAMPLE

The following is a sample interchange between the system designer and the MMPS system. In this example, the user will connect two imaginary processors, called MINIs, to a common memory. The MINI is used for simplicity's sake. As in the MMPS simulator, commentary (which is ignored by the command parser) is preceded with a "!", and ends with a carriage return.

```

>! First, the user must declare the processors, the memory, and the
>! associated backplane wires.
>
>DECLARE MINI[2],
+>     MEMORY[1] <4:4>,
+>     WIRE[10];
>
>! Notice that the memory is declared to have four bits of addressing,
>! and four bits of data space. The control lines are assumed. The
>! nine backplane wires are for connecting the processors to the
>! memory units, etc.
>
>! Now the clock speeds are set on the processors
>
>CLOCK MINI[1] at .5MHz,
+>     MINI[2] at .75MHz;
```

```

>
>! Now the elements are connected. The address and data lines of
>! both the processors and the memory are connected together, along
>! with the control lines. One I/O line of each processor is
>! connected to the other for handshaking purposes. (This is declared
>! to be wired OR). The remaining I/O line per processor is connected
>! to a wire for monitoring purposes.
>
>CONNECT MINI[*] <0:3>, MEMORY[1] <0:3>
+>          TO WIRE <0:3>;
>
>CONNECT MINI[*] <4:7>, MEMORY[1]<4:7>
+>          TO WIRE <4:7>;
>
>CONNECT MINI[*] <8> TO WIRE <8>;
>
>WIREDOR WIRE <8>;
>
>CONNECT MINI[1] <9> TO WIRE <9>;
>CONNECT MINI[2] <9> TO WIRE <10>;
>
>! Next establish the tracing and monitoring points. First,
>! determine what are the valid points in the memory.
>
>NAMES MEMORY[1];
MREAD MWRITE DATA ADDRESS CONTROL
>
>! Set the traces names.
>
>TRACE MEMORY[1] : MREAD, MWRITE, ADDRESS;
>
>! Print out what is being traced, and also what is being signaled
>! on in the memory.
>
>PRTRACE MEMORY[1];
MREAD MWRITE ADDRESS
>PBREAK MEMORY[1];
      <No break names>
>
>! Load the memory, starting at location 5.
>
>LOAD MEMORY[1] FROM 5 WITH PROG
Locations 5—15 loaded from PROG; Total 11 words.
      <End of memory reached before end of file!>
>
>! Set the default tracing on the MINI's, and go
>
>TRACE MINI[*] : DEFAULT;
>NOBREAK MINI[*];
MINI[1] : Are you sure? Y
MINI[2] : Are you sure? Y
>GO
.
.
.
BREAK on MINI[2] : Halt executed at time 5135ms.
>
>! Now dump memory.

```

```

>
>DUMP MEMORY[1] TO DMPDIL
Locations 0-15 dumped to DMPFIL; Total 16 words.
>QUIT
%
```

The logfiles (called MINI.1, MINI.2, and MEMORY.1) can now be examined. Additionally, BUSREQ is a file that contains all information transferred along the busses, including that information that came out of the I/O lines to wires nine and ten. In case the user wanted to recreate the system above, he could utilize the USE command with the file COMLOG, which contains a running summary of the commands given to MMPS.

CONCLUSION

MMPS provides a highly flexible tool for the network designer. As such, it can be used by many applications users, including the communications and data base management fields. Since MMPS is a simulator, it of course runs considerably more slowly than the hardware it is simulating. There are plans to interface MMPS with a downloadable hardware emulator, giving the added advantage of being able to look at detail with the simulator, and an overall picture with the emulator. With these tools, network design and implementation speed should be greatly increased.

REFERENCES

1. McConnell, Tron, Ronald Krutz and Don Gregg, *BUCS—A Bus Utilization Control System* (a reconfigurable hardware emulator), CMIR, 1978.
2. Barbacci, Mario, Gary Barnes, Roderick Cattel and Daniel Sieworek, *The ISPS Computer Description Language*, Department of Computer Science, Carnegie-Mellon University, 1978.
3. Rose, Charles, and Donald Hewitt, *The Human Engineering of a System Design Environment for Microprocessor-Based Systems*, Case Western Reserve University, 1978.
4. Ritchie, Dennis, and Ken Thompson, "The UNIX Time-Sharing System," *CACM*, Vol. 17, No. 7, July 1974, pp. 365-375.
5. Drongowski, Paul, "Capability Requirements in a Multimicroprocessor, Hardware/Software Simulation Environment," *Symposium on Design Automation and Microprocessors*, IEEE-ACM, 1977.

APPENDIX

The following ISP description is given courtesy of Mario Barbacci of the Department of Computer Science, Carnegie-

Mellon University, Pittsburgh. It describes a very simple computer, and is provided for those who would desire an example of an actual ISPS computer description.

```

MARK1 :=
BEGIN
```

! The Manchester Mark-I architecture is described.

! The Mark-I was an early (circa 1948) computer.

```

**MP . STATE**
  m[0:8191](0:31),

**PC . STATE**
  pi\present.instruction (0:15),
  f\function (0:2) := pi (0:2),
  s (0:12) := pi (3:15),
  cr\control.register (0:12),
  acc\accumulator (0:31),
**INSTRUCTION . EXECUTION**{TC}
```

```

icycle\instruction.cycle :=
BEGIN
REPEAT
  BEGIN
    pi ← m[CR] (0:15) NEXT
  DECODE f =)
    BEGIN
      #0 := cr ← m[s],
      #1 := cr ← cr + m[s],
      #2 := acc ← - m[s],
      #3 := m[s] ← acc,
      #4:#5 := acc ← acc - m[s],
      #6 := IF acc LSS 0 => cr ← cr + 1,
      #7 := STOP ( )
    END NEXT
  cr ← cr + 1
END
END
END
```


A (31,15) Reed-Solomon code for large memory systems

by RAYMOND S. LIM

NASA-Ames Research Center
Moffett Field, California

SUMMARY

This paper describes the encoding and the decoding of a (31,15) Reed-Solomon Code for multiple-burst error correction for large memory systems. The decoding procedure consists of four steps—(1) syndrome calculation, (2) error-location polynomial calculation, (3) error-location numbers calculation and (4) error values calculation. The principal features of the design are the use of a hardware shift register for both high-speed encoding and syndrome calculation, and the use of a commercially available (31,15) decoder for decoding Steps 2, 3 and 4.

INTRODUCTION

With present technology, very large memory systems ($\geq 10^{12}$ bits) designed for the archival storage of digital data are critically dependent on electronic error correction systems (EECS) for ensuring system viability and integrity.¹⁻⁴ In the IBM 3850 Mass Storage System, the EECS used is an Extended Group Coded Recording capable of correcting up to 32 eight-bit bytes of data in a 208-byte data block. In the CDC 38500 Mass Memory System, the EECS used is a modified Group Coded Recording similar to that used in the IBM 2400 Series magnetic tape systems. The use of magnetic tape systems for archival storage of digital data depends even more critically on EECS to make them viable. The EECS, devised by Brown and Seller, and used in the IBM 2400 series magnetic tape system, is not adequate for long-term archival storage of data.⁵⁻⁷

At the Institute for Advanced Computation (IAC), archival storage systems such as the UNICON 690, magnetic tape systems, and other mass memory systems are no exceptions. The viability of these systems depends critically on EECS. Instead of designing a different EECS for each particular archival system, it is advantageous to design a single EECS powerful enough to serve all systems within the Institute.

This paper describes a (31,15) eight-error-correcting Reed-Solomon (RS) code and a decoding procedure suitable for implementation using present technology. This code is not a binary code, but a q-ary code with code symbols from $GF(2^5)$; i.e., each code symbol consists of five bits. For the I4-TENEX system—PDP-10, PDP-11, and ILLIAC IV com-

puters—the RS code is planned to have two modes of operation—the 36-bit mode and the 16/8-bit mode. Decoding will be implemented by hardware which consists of four steps—(1) syndrome calculation, (2) error-location polynomial calculation, (3) error-location numbers calculation and (4) error values calculation. The encoding process is performed by a hardware shift register in 31 clock cycles (50 ns per cycle). The syndrome calculation in Step 1 is also performed by the same encoding shift register in 31 clock cycles. Decoding Steps 2, 3 and 4 are performed by a commercially available (31,15) $GF(2^5)$ decoder⁸ in one msec maximum, depending on the number of errors which actually occurred.

REED-SOLOMON CODES

Reed-Solomon (RS) codes⁹⁻¹² are the most powerful of the known classes of block codes capable of correcting random errors and multiple-burst errors. From coding theory, there are codes with code symbols from a q -symbol alphabet if p is a prime number and q is any power of p . These codes are called q-ary Bose-Chaudhuri-Hocquenghem (BCH) codes. RS codes, with code symbols from a Galois field of q elements $GF(q)$, are a special subclass of BCH codes.

For present engineering applications only binary codes derived from RS codes are of interest. For this reason, $GF(q)$ will be restricted to $GF(2^m)$ where m is a positive integer. The field $GF(2^m)$ is formed by a primitive polynomial of degree m with α as the primitive element of the field. In the algebra of a Galois field, α is also called the n th root of unity in $GF(2^m)$ since $\alpha^n = 1$ for $n = 2^m - 1$. With $q = 2^m$, the code symbols of an RS code are α^i , $i = \infty, 0, 1, 2, \dots, 2^m - 2$, which are the 2^m distinct elements of $GF(2^m)$. The notation $\alpha^\infty = 0$ is used here.

The RS codes have $\alpha, \alpha^2, \alpha^3, \dots, \alpha^{2t}$ as roots. Since the minimum polynomial with root α^j is simply $(x + \alpha^j)$, the generator polynomial $g(X)$ of a t -error-correcting RS code of length $2^m - 1$ is

$$g(X) = (X + \alpha)(X + \alpha^2), \dots, (X + \alpha^{2t}) \quad (1)$$

The code word polynomials generated by $g(X)$ consist of the multiples of $g(X)$ modulo $X^{2^m - 1} + 1$, and have $\alpha, \alpha^2, \alpha^3, \dots, \alpha^{2t}$ as roots. Since $g(X)$ has degree $2t$, and α is a primitive n th root of unity in $GF(2^m)$, the RS code generated

by $g(X)$ is a t -error-correcting cyclic code with the following parameters:

$$\left. \begin{aligned} \text{Code length (symbols)} \quad n &= 2^m - 1 \\ \text{Number of parity check symbols} \quad n - k &= 2t \\ \text{Minimum distance} \quad d &= 2t + 1 \\ \text{Number of information symbols} \quad k &= 2^m - 1 - 2t \end{aligned} \right\} (2)$$

Since a symbol in $GF(2^m)$ can be expressed as an m -tuple over $GF(2)$, the parameters of an RS code over $GF(2)$ are simply m times larger.

ERROR-CORRECTING CAPABILITY

RS codes over $GF(2^m)$ are very effective in correcting random and burst errors. Since each code symbol is an m -tuple over $GF(2)$, a t -error-correcting RS code is capable of correcting any error pattern that affects t or fewer m -bit symbols. For example, since a burst of length $3m + 1$ cannot affect more than four m -bit symbols, a four-symbol correcting code can correct any single burst of length $3m + 1$ or less. It can also simultaneously correct any combination of two bursts of length $m + 1$ or less because each burst can affect no more than two symbols. At the same time, it can correct any combination of four or less random errors. In general, the RS code with error correcting capability t can be used to correct any of the following errors:

1. All single bursts of length b_1 , no matter where they start, if $b_1 \leq m(t-1) + 1$.
2. Two bursts of length no longer than b_2 each, no matter where each burst starts, if $b_2 \leq m[(t/2) - 1] + 1$, or any p bursts of length no longer than b_p each, no matter where each burst starts, if $b_p \leq m[(t/p) - 1] + 1$.

From the previous discussion, it follows that the RS code can be used to correct random errors, single-burst errors, or multiple-burst errors.

CODE SELECTION

RS codes offer the designer a wide range of code parameters as indicated in Equation 2. In coding theory, a block code with parameters n and k is denoted as (n, k) . For the IAC I4-TENEX system that consists of computers with work lengths of 16 bits (PDP-11s), 36 bits (PDP-10s), and 32/64 bits (ILLIAC IV), the best choice for fitting these word lengths is the (31,15) code. The formats for the 36-bit and the 16/8-bit are shown in Figure 1.

ENCODING

There are two methods for encoding linear cyclic codes—the serial shift register method and the parallel matrix method. Because of hardware complexity in the parallel method, only the serial method is described in this paper.

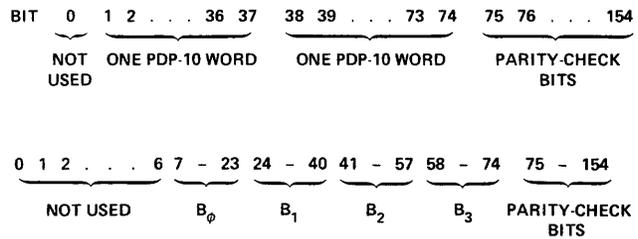


Figure 1—Data formats of the (31,15) RS code for 36-bit and 16/8-bit modes. Unused leading bits are always zero.

Let $M(X)$ be a message polynomial with k symbols encoded into a code polynomial (code word) $V(X)$ with n symbols. In the serial shift register method, encoding in systematic form is accomplished by dividing $x^{n-k}M(X)$ by $g(X)$ and appending the remainder $r(X)$ to $X^{n-k}M(X)$; i.e.,

$$V(X) = r(X) + X^{n-k}M(X) = q(X)g(X) \quad (3)$$

where $q(x)$ is the quotient. This indicates that $[r(X) + X^{n-k}M(X)]$ is a multiple of $g(X)$ and, therefore, is a code polynomial generated by $g(X)$. The code word generated is

$$(r_0 r_1 r_2, \dots, r_{n-k-1} m_0 m_1, \dots, m_{k-1})$$

$\left| \begin{array}{c} \leftarrow \text{parity check} \rightarrow \\ \text{bits} \end{array} \right| \left| \begin{array}{c} \leftarrow \text{message} \rightarrow \\ \text{bits} \end{array} \right|$

and the most significant symbol of the message, m_{k-1} , is sent first.

There are two shift register methods for encoding linear cyclic codes. One method uses an $(n-k)$ -stage shift register, and the other uses a k -stage shift register.⁹ In practice, the $(n-k)$ -stage shift register is more commonly used unless $n-k$ is much greater than k . For encoding the (31,15) RS code, an $(n-k)=16$ stages shift register (Figure 2) can be used to implement Equation 3. The feedback multipliers g_0, g_1, \dots, g_{15} are coefficients of the generator polynomial $g(X)$, where $g(X)$ from Equation 1 is

$$g(X) = (X + \alpha)(X + \alpha^2), \dots, (X + \alpha^{15})(X + \alpha^{16}) \quad (4)$$

Multiplying out Equation 4 and selecting the primitive polynomial $p(X)$ in $GF(2^5)$ to be

$$p(X) = X^5 + X^2 + 1 \quad (5)$$

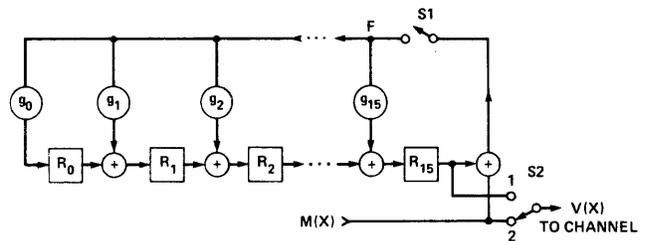


Figure 2—Encoder for (31,15) code. g_i is a field element from $GF(2^5)$ and R_i is a five-tuple shift register stage.

the coefficients are

$$\begin{aligned} g_0 &= \alpha^{12} & g_4 &= \alpha^{14} & g_8 &= \alpha^{25} & g_{12} &= \alpha^8 & g_{16} &= 1 \\ g_1 &= \alpha^{18} & g_5 &= \alpha^{23} & g_9 &= \alpha^{21} & g_{13} &= 1 \\ g_2 &= \alpha^{22} & g_6 &= \alpha^4 & g_{10} &= \alpha & g_{14} &= \alpha^{13} \\ g_3 &= \alpha^{23} & g_7 &= \alpha^7 & g_{11} &= \alpha^3 & g_{15} &= \alpha^{23} \end{aligned}$$

Each R_i register stage is a five-tuple shift register. The operation of the encoder is as follows. With $S1$ set for feedback and $S2$ set to position 2, k information symbols are shifted into the encoder and simultaneously sent to the channel. Then $S1$ is turned to disable the feedback and $S2$ is turned to position 1; the 16 parity-check symbols stored in the encoder now are shifted out to the channel, clearing the shift registers.

DECODING

Let $V(X)$ be the transmitted code word, $E(X)$ be the channel noise-error pattern, and $R(X)$ be the received code word and represented as follows:

$$\left. \begin{aligned} V(X) &= v_0 + v_1x + v_2x^2 + \dots + v_{n-1}x^{n-1} \\ E(X) &= e_0 + e_1x + e_2x^2 + \dots + e_{n-1}x^{n-1} \\ R(X) &= r_0 + r_1x + r_2x^2 + \dots + r_{n-1}x^{n-1} \end{aligned} \right\} \quad (6)$$

where v_i , e_i and r_i are elements of $GF(2^m)$, $i=0, 1, 2, \dots, n-1$. At the decoder

$$R(X) = V(X) + E(X) \quad (7)$$

The error pattern $E(X)$ can be described by a list of values and locations of its non-zero components. For the decoding procedure to be described, the error location will be given in terms of an error-location number which is simply α^j for the $(n-j)$ th symbol. Let x_j be the error location number and e_j be the error value. Then for each non-zero component of $E(X)$, a pair of field elements (x_j, e_j) is required to describe that error. If $E(X)$ has p errors, then p pairs (x_j, e_j) are required to describe the errors. Any decoding procedure is a procedure for locating these p pairs of (x_j, e_j) if $p \leq t$.

Assume that $E(X)$ is an error pattern of p errors at locations j_1, j_2, \dots, j_p . Then

$$E(X) = e_{j_1}x^{j_1} + e_{j_2}x^{j_2} + \dots + e_{j_p}x^{j_p} \quad (8)$$

where $p \leq t$ and $0 \leq j_1 < j_2 < \dots < j_p \leq n-1$. The first step in decoding is to check whether $V(X)$ is a code word by calculating the syndrome. If the syndrome is zero, then either $V(X)$ actually has no errors or $V(X)$ has an undetectable error. In either case, $V(X)$ is accepted as errorless. A non-zero syndrome indicates that an error has been detected; the error may or may not be correctable. For the RS codes, the syndrome is defined as a vector S with $2t$ components as follows:⁷⁻¹⁰

$$S_i = R(\alpha^i) = r_0 + r_1\alpha^i + r_2(\alpha^i)^2 + \dots + r_{n-1}(\alpha^i)^{n-1} \quad (9)$$

for $i=1, 2, 3, \dots, 2t$. Combining Equations 7 and 9, the

result is

$$S_i = V(\alpha^i) + E(\alpha^i) \quad (10)$$

Since $V(\alpha^i) = 0$.

$$S_i = E(\alpha^i) = \sum_{l=1}^p e_{j_l} X_{j_l}^i \quad (11)$$

An effective decoding procedure is described below and a design implementation for the (31,15) RS code is given. This procedure consists of four major steps as follows:⁹⁻¹²

1. Calculate the syndrome $S = (S_1, S_2, \dots, S_{2t})$ from $R(X)$.
2. Calculate the error location polynomial $\sigma(X)$ from S .
3. Determine the error locations X_j by finding the roots of $\sigma(X)$.
4. Calculate the error value e_j from X_j and S .

Decoding Step 1 is straightforward. Steps 2-4 are difficult and are very time-consuming.⁹⁻¹²

Step 1—Syndrome calculation

For large t ($t \geq 4$), a good way to calculate the syndrome is to use the $g(X)$ encoding shift register as shown in Figure 2, except that $R(X)$ is excluded-ored with the output from g_0 to form the input to R_0 . This will result in some saving in logic because this shift register is already used for encoding. If $V(X)$ has no errors, the S calculated is always zero. However, if $V(X)$ has an error, the S calculated by $g(X)$ is not the same as the S calculated by $R(\alpha^i)$ of Equation 9, but they are related. This relationship is described below.

From Equation 9, let

$$S = (S_1, S_2, \dots, S_{2t}) \quad (12)$$

be the syndrome calculated by $R(\alpha^i)$ with

$$S_i = R(\alpha^i) \quad (13)$$

for $i=1, 2, \dots, 2t$. Let

$$S^* = (S_1^*, S_2^*, \dots, S_{2t}^*) \quad (14)$$

be the remainder calculated by dividing $R(X)$ by $g(X)$. The remainder S^* is another form of the syndrome but $S^* \neq S$. Using the Euclidean division algorithm, the result of $R(X)/g(X)$ is

$$R(X) = Q(X)g(X) + S^*(X) \quad (15)$$

where $Q(X)$ is the quotient and

$$S^*(X) = S_1^* + S_2^*X + S_3^*X^2 + \dots + S_{2t}^*X^{2t-1} \quad (16)$$

is the remainder. Substituting X by α^i in Equation 15 gives the result

$$R(\alpha^i) = Q(\alpha^i)g(\alpha^i) + S^*(\alpha^i) = S^*(\alpha^i) \quad (17)$$

since $Q(\alpha^i)g(\alpha^i) = 0$. From Equations 13, 16 and 17, the

relationship between S and S_1^* is

$$S_i = S^*(\alpha^i) = S_1^* + S_2^*(\alpha^i) + S_3^*(\alpha^i)^2 + \dots + S_{2t}^*(\alpha^i)^{2t-1} \quad (18)$$

for $i=1, 2, 3, \dots, 2t$. In our design, the actual value of S is not needed because decoding Steps 2-4 will be performed by the GF1TM decoder. The GF1TM decoder has a built-in capability for calculating S , albeit at a lower speed.

Steps 2-4—Error correction

The implementation of decoding Steps 2-4 in our design will use the GF1TM decoder, which is a commercially available device.⁸ Viewing the GF1TM decoder as an LSI chip, its functional block diagram is shown in Figure 3. The received code word $R(X)$ has 31 five-bit digits. Assuming the the GF1TM has been initialized by pulsing the INIT line and that the DATE-IN ENABLE line is activated, $R(X)$ is input, one digit at a time, to the DI lines. After the last digit (31st) of $R(X)$ is input, the GF1TM decoder immediately starts the decoding procedure to correct the errors in $R(X)$. The decoding time is a variable depending on the actual random errors as well as erasures in $R(X)$. The GF1TM can correct at a maximum of $2t+s \leq 16$ errors, where t is the number of random errors and s is the number of erasure errors. Input code symbols that have erasure errors are indicated by the ERASURE INDICATOR line. In the worst case, the maximum decoding time is one millisecond.

The corrected output of $R(X)$ is $V(X)$ and $V(X)$ is available on the DATA OUT lines as soon as the OUTPUT READY line is activated by the GF1TM. The GF1TM output has a total of 19 digits. The first four digits are labels and are used to indicate correctable and uncorrectable status

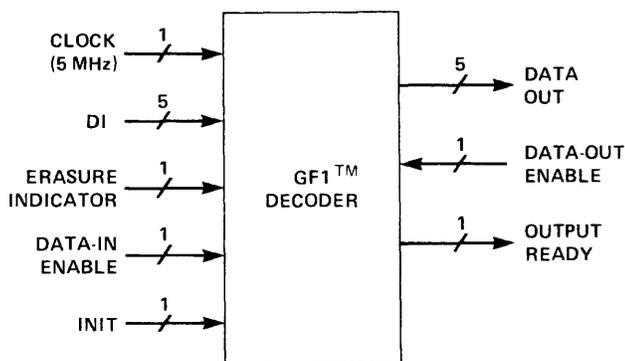


Figure 3—Block diagram of GF1TM decoder.

and the actual number of errors which have occurred. The last 15 digits are the information digits of $V(X)$.

CONCLUSION

A decoding procedure has been described for a (31,15) eight-error-correcting Reed-Solomon (RS) code. The code parameters are chosen for cyclic codes so that encoding and syndrome calculation can be implemented by shift registers, each requiring 31 clock cycles. Using current STTL technology, a clock cycle can be as short as 50 ns so that encoding or syndrome calculation can be accomplished in 1.55 μ s (about 48 megabits per second). For error correction, decoding Steps 2-4 are implemented by the GF1TM decoder,⁸ which is a commercial (31,15) RS decoder. This decoder can correct $2t+s \leq 16$ errors where t is the number of random errors and s is the number of erasure errors. The maximum decoding time of the GF1TM is one millisecond.

ACKNOWLEDGMENT

The author wishes to thank Dr. Shu Lin, Dr. E. R. Berlekamp and Dr. D. K. Stevenson for reading and commenting on the work reported herein.

REFERENCES

1. *Introduction to the IBM 3850 Mass Storage System (MSS)*, IBM Document GA-32-0028-2, July 1975, p. 57.
2. *IBM 3850 Mass Storage System Installation Guide*, IBM Document GA-32-0030-1, April 1976, p. 17.
3. *38500 Mass Storage System General Information Manual*, Control Data Corporation Document 22294400, 1976.
4. *38500 Mass Storage Reference Manual*, Control Data Corporation Document 22083000, 1976.
5. Brown, D. T., and F. F. Sellers, Jr., "Error Correction for IBM 800-bit-per-inch Magnetic Tape," *IBM J. Res. Develop.*, July 1970, pp. 384-389.
6. Berlekamp, E. R., "Algebraic Codes for Improving the Reliability of Tape Storage," *AFIPS Conference Proceedings*, Vol. 44, May 19-22, 1975, pp. 497-499.
7. Poland, W. B., G. E. Prine and T. L. Jones, "Archival Performance of NASA GFSC Digital Magnetic Tape," *AFIPS Conference Proceedings*, Vol. 42, Part II, 1973.
8. *Technical Data. Reed-Solomon Decoder, Model GF1TM*, Cyclotomics, Inc., 2140 Shattuck Ave., Berkeley, Calif., 94704, June 1978.
9. Lin, S., *An Introduction to Error-Correcting Codes*, Prentice-Hall, Englewood Cliffs, New Jersey, 1970.
10. Peterson, W. W., and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd Edition, MIT Press, Cambridge, Mass., 1972.
11. Berlekamp, E. R., *Algebraic Coding Theory*, McGraw-Hill Book Co., Inc., New York, 1968.
12. Lim, R. S., "A Decoding Procedure for the Reed-Solomon Codes," NASA TP-1286, August 1978.

English dictionary searching with little extra space

by DOUGLAS COMER

Purdue University
W. Lafayette, Indiana

INTRODUCTION

When text is typeset using a computer-based system, it can also be checked for spelling errors automatically and efficiently. Several methods of spelling error detection have been proposed. Morris *et al.* [MORR75] study statistical properties of English words, and describe an algorithm to catch possible typos by examining the relative frequency of trigrams (3-letter combinations). Kernighan *et al.* [KERN78] report a more conventional approach which looks up each word in a machine readable spelling dictionary.¹ Knuth [KNUT73] comments on dictionary storage, and suggests an organization intended to reduce space requirements. Another, somewhat unsophisticated technique sorts all words in a document and prints those which occur infrequently. Of course, habitually misspelled words escape unnoticed in a frequency based system.

Both [KERN78] and [KNUT73] mention removing suffixes from the spelling words and storing only the stems. When a word is to be looked up, its suffixes are removed by the same method, and only the stem is checked. This method has the advantage of requiring less space, but the disadvantage of not always catching illegal stem and suffix combinations. For example, the typo "computions" would pass the spelling test because removal of the suffix "ions" leaves the valid stem "comput."

Because there are several hundred thousand words in the English language, a complete spelling dictionary is too large to keep in main memory. Fortunately, a complete spelling dictionary has almost no value for catching typos: misspellings commonly turn out to be obscure or archaic terms which appear in the dictionary. Even a dictionary of 40,000 words may be too large to be useful. For example, the 40,000 word dictionary used by Purdue University's computing center includes terms such as "de," "hod," "ila," "lo," "moo," and "pul," which would probably be typos in technical prose.

While one cannot give an optimum size for an on-line dictionary, it is clear that "the bigger the better" does not apply. Since writers use most words infrequently and a few words very frequently, the best scheme seems to be:

1. Start with a small core of, say, 10,000 commonly used English words.
2. Add new words to the dictionary only as users request them.

This way, an appropriate dictionary evolves without unnecessary or obscure words. A small dictionary, grown by user request, is less likely to accept misspellings, and it can be managed more efficiently as well.

Carrying the evolution strategy further, researchers at Bell Labs derived a spelling dictionary solely from user's documents [KERN78]. The resulting dictionary is one-third the size of their original spelling dictionary. It has proved to be more useful, however, since it contains all the heavily used technical jargon as well as common words. Of course, if the user population spans a wide variety of interests, each group should probably grow its own augmentation dictionary, keeping the core for common English words. Any term appearing in all augmentation dictionaries should be deleted and moved to the common core.

Small spelling dictionaries often fit into main memory and can be searched without accessing secondary storage. The words themselves usually occupy a large portion of the available space, however, leaving only a little extra space for the program and data structures. This paper explores searching techniques in the context of a small spelling dictionary, and presents a technique which exploits a small amount of extra space to lower access costs. The second section reviews conventional search methods; the third section presents the trie-binary search (which uses a modest amount of memory). The fourth section examines an application to a particular spelling dictionary, and the fifth section presents performance statistics for typical input data. The final section suggests extensions and improvements.

DICTIONARY SEARCHING TECHNIQUES

We view a *spelling dictionary* D as a static set of keys k_i , $1 \leq i \leq n$, each key being composed of m letters. For each word w in the input text, the search procedure must answer the membership question, "is w in D ?" We assume that:

¹ More precisely, a word list; this paper uses the terms interchangeably.

1. The dictionary fits into main memory and occupies a fraction of the available space, α , $0 < \alpha \leq 1$.
2. The dictionary is not to be compressed (see [MORR75] for compression techniques).
3. The dictionary may be preprocessed (eg. sorted) to decrease search times. In particular, an index may be built to speed retrieval.

The general problem of data storage and retrieval for static sets of keys has received wide attention in the literature. Knuth [KNUT73] provides a summary. When the keys themselves occupy all but a small fraction of the available memory space, only a few candidate strategies emerge:

- sequential search —each probe eliminates 1 key.
- binary search —each probe eliminates half the keys.
- interpolation search —each probe eliminates more than half the keys in a uniform (or nearly uniform) distribution.
- hash table search —each probe eliminates most of the remaining keys (given sufficient memory).
- partial index —a small, auxiliary data structure is searched to eliminate all but a subset of the keys. Then, one of the above methods is used to search the subset.

The first four methods have been analyzed, and expressions for the expected number of comparisons per look-up have been derived under the assumption of a uniform distribution of keys. Sequential searching examines one-half of the dictionary for each look-up, and is obviously inferior to binary searching, which makes an average of

$$C_{Binary} = \log_2 n - 1 \tag{1}$$

comparisons in a dictionary of n words [KNUT73].

Interpolation search requires only $C_1 \cdot \log_2(\log_2 n) + C_2$ comparisons on the average for nearly uniform distributions [YAO76]. In practice, however, the constants make interpolation search too expensive, especially with a highly skewed distribution like that of a spelling dictionary [KNUT73].

The performance of hashing depends on the fraction of the space that is occupied, α , and can be estimated as:

$$C_{Hash} = (1 + 1/(1 - \alpha))/2 \tag{2}$$

comparisons for a successful search, and

$$C'_{Hash} = (1 + 1/(1 - \alpha)^2)/2 \tag{3}$$

comparisons for an unsuccessful search [KNUT73].

Ignoring for the moment partial index methods, one must choose between binary searching and hashing. In the case of an English spelling dictionary which contains less than 2^{16} words, binary searching requires less than 15 probes on the average, and 17 probes in the worst case. From (2) we have that hashing is inferior for $\alpha > .966$. For example, when only

2 percent of the space is unused, hashing requires over 25 probes for an average successful search, and 1250 probes for an average unsuccessful search.

TAKING ADVANTAGE OF A LITTLE EXTRA SPACE

As shown in Figure 1 when only a small fraction of memory is free, binary searching requires fewer comparisons on the average than hashing, even though a binary search uses none of the extra space that is available. This section presents a partial index method that uses only a little extra space; the next section demonstrates that it makes fewer comparisons on the average than a binary search.

Figure 2 illustrates the use of a partial index mechanism. During each look-up, the spelling program searches the index to identify the appropriate subset of the dictionary to search. Presumably, the index identifies the proper subset rapidly. The program then uses a binary search to explore the specified subset.

One particular index method, called a *trie-binary search* keeps the dictionary in sorted form, using a trie index [FRED60] to identify the correct subset to search. As shown in Figure 3, the simplest trie index consists of an array of 26 pointers, one for each letter of the alphabet, which give the starting location of words starting with that letter. Using the first letter of a word to index into the array, one can quickly locate the subset of all words starting with that letter. The array is referred to as a *node* of the trie; many standard English dictionaries provide a 1-node trie in the form of a thumb index to help the user find the appropriate section faster.

A trie index can be extended to more than one level easily by allowing a pointer in the first level to point to another trie node instead of directly into the dictionary. For example, if the level one pointer for "c" pointed to a second node, then the second node would contain the starting locations of the subsets of words beginning with "ca," "cb,"

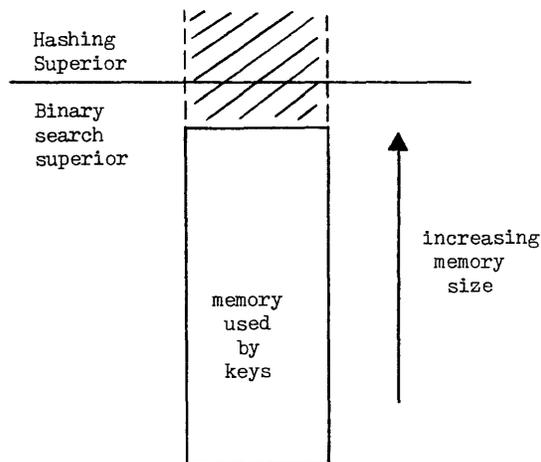


Figure 1—The relationship between memory size and optimal search strategies. Binary search is superior to hashing when only a little space remains free, even though it does not use the free space.

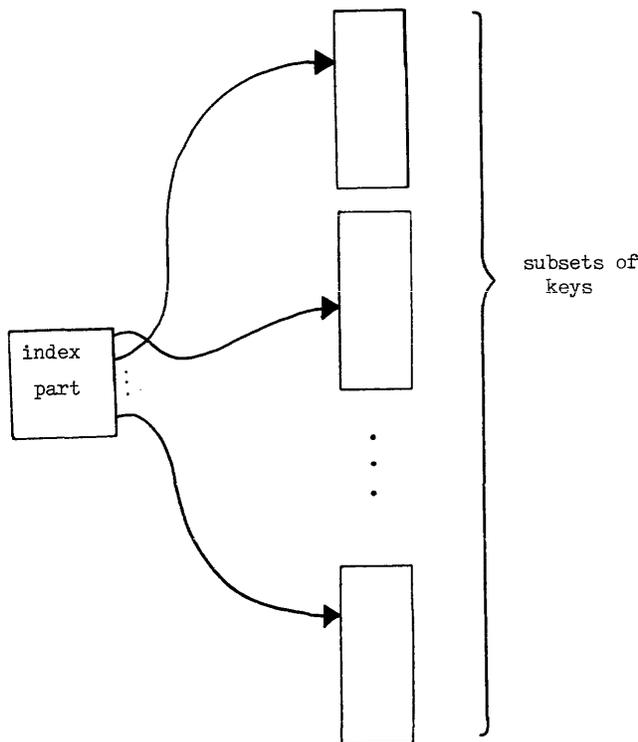


Figure 2—The idea of an index. A short search in the index data structure identifies some subset of the keys which is then searched for the exact key.

... , "cz." If a third level node were inserted following the pointer in a second level node for "co," then the third level node would give the starting locations of words beginning with "coa," "cob," ... , "coz."

The following terminology will be useful in the sequel. Pointers in the trie which point into the dictionary are called *leaf pointers*; all others are called *trie pointers*. The level one node of a trie is called its *root*, and a node which can be reached by following a pointer from a level *i* node is said to lie at *level i+1*.

Searching in the trie begins by indexing into the root node based on the first character. Decisions at subsequent nodes depend on subsequent letters. In general, an *i*th level node in the trie further restricts the subset of words to be searched by testing the *i*th letter.

Since storage space is limited, care must be taken to build the trie so that each node allocated reduces the average number of comparisons by the maximum amount possible. For example, if space exists for only two nodes, dividing the set of words starting with letter "s" results in fewer comparisons than dividing the set of words starting with "a." The idea of allocating nodes to produce lowest cost searches leads to the following algorithm for trie construction:

Algorithm A: (construct minimum cost trie using available storage)
construct a 1-node trie for the dictionary;
while storage is not exhausted *do*

```

begin
  find dictionary pointer p which yields best improvement in search time when its set is divided;
  insert a new node after pointer p
end
end A;
    
```

Algorithm A exhibits a high running time unless one employs an efficient method for finding the optimum dictionary pointer to be replaced. For uniformly distributed sets of keys, dividing a largest set is optimal; for non-uniform distributions the size of the set is not always related to the effect that division of that set would have on the cost. However, assuming that division of the largest remaining subset produces a good, if not optimum, trie leads to an efficient, practical algorithm for index construction:

Algorithm B: (construct a low cost trie efficiently)
construct a 1-node trie for the dictionary;
select a maximum set size, *t*;
while a dictionary pointer, *p*, points to a subset of more than *t* keys *do*
 insert a new node after pointer *p*
end B;

Note that the tries constructed by either algorithm do not have fixed depth. Of course, each pointer in a variable depth trie must have an indication of whether it points into the dictionary or to another level of the trie, so an additional bit is required to store the indicator. Furthermore, pointers into the dictionary should include both a starting and ending

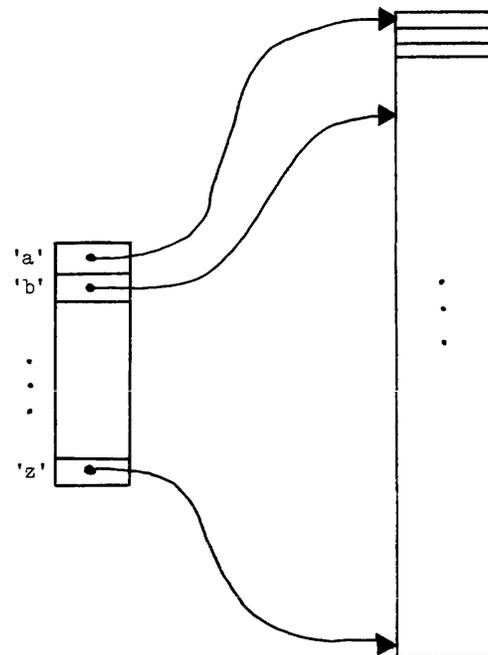


Figure 3—The simplest trie—binary search. The index consists of one array of 26 pointers giving the starting locations of words beginning with 'a,' 'b,' ... , 'z.'

location, since finding adjacent pointers in the trie would be time consuming. Even with these additions, the variable depth trie uses far less storage to achieve the same search time as a fixed depth trie.

The choice of the maximum set size, t , is crucial to fitting the trie into a small amount of memory. If t is too large, not enough nodes will be generated, and search efficiency will be lost. If t is too small, the algorithm will run out of space before the trie can be completed, and search performance may be poor. The next section discusses the relationship between storage used and the maximum set size, giving data from a sample dictionary.

PERFORMANCE OF TRIE-BINARY SEARCH

This section describes the performance of the trie-binary search strategy on a typical spelling dictionary. Usually, such empirical data has limited value because key distributions vary from file to file. In English, the relative distribution of keys remains fairly constant over a large range of dictionary sizes. Thus, the performance study presented here applies directly to other English dictionaries.

The sample dictionary consists of 16,949 words, and was originally formed from computer tapes of newspapers. Gradually, some technical terms have been added, and obscure and nonsense words have been eliminated (although words like "MR", "MRS", "AVE", "ST", and "BLDG" remain).

With space for 26 pointers (1 node), trie-binary, binary, and hash search were implemented, and the actual number of probes² necessary to find each word in the dictionary was recorded. Using the collected data, the average and worst case number of probes for a successful search were calculated. Using equations (1), (2) and (3), and assuming that the trie divided the set of words into 26 equal size subsets, predicted average and worst case probes were computed. Table I summarizes the results.

The trie-binary search (using 1 node) requires only 75 percent of the probes needed by a straight binary search, and clearly performs better with only 26 extra storage locations. Two questions arise immediately: how does trie-

TABLE I

		Binary Search	Trie-Binary	Hashing
Observed	avg.	13.05	9.80	—
	worst	15	12	—
Predicted	avg.	13.05	9.35	652
	worst	15	11	16949

The average and worst case search times for 16949 word dictionary, and the predicted values assuming equal size subsets are produced by the trie. The observed values for hashing are omitted because of the excessive computation costs.

² Throughout this paper we count each access in the index as one probe. Thus, if two characters are tested in the trie and three comparisons are made in the dictionary, the lookup requires five probes.

binary search perform with more space, and what is the relationship between the maximum set size, t , and the number of nodes allocated?

The graph shown in Figure 4 answers the first question in the case of the sample dictionary. It is clear from this plot that the best tradeoff space for time occurs with a small amount of space. For example, Table II summarizes the number of nodes necessary to lower the average number of probes in steps of one. Each successive reduction requires more space.

For the dictionary in question, a choice of 100-120 nodes represents a good compromise and brings the average number of probes for a successful search down to about 6.25 (or 48 percent of that used by a binary search). Doubling the space to 200-240 nodes further reduces the average number of probes by only .5, and hardly seems worthwhile.

The relationship between the maximum set size, t , and the number of nodes allocated, p , is important because the fast trie building algorithm depends on t , while the user can most easily estimate p . Knuth [KNUT73] estimates that

$$p = n / (t \cdot \ln b)$$

for a file of uniformly distributed keys, where b is the branch factor of each node. Using a branching of 26 for English yields

$$p = n / (3.3 t)$$

For the sample spelling dictionary, the estimate turns out to be close to 50 percent low for some t . This discrepancy can be explained easily by the non-uniform distribution of keys. It turns out that an average branching factor of around 8 provides a closer estimate of p :

$$p = n / (2.08 t) \quad (4)$$

Figure 5 shows both the estimated and actual number of nodes needed as a function of the maximum set size, t . Unfortunately, the number of nodes rises rapidly for small t , the steep slope making selection of optimum t difficult. However, the cost of trie construction is small, and the task will be performed once in a preprocessing phase, so version A of the trie building algorithm can be used to find optimum t , if necessary. As noted earlier, adding a few extra nodes does not lower the average number of probes drastically. Thus, a conservative estimate of t will not degrade performance severely when version B of the algorithm is used.

TABLE II

total nodes allocated	average probes
1	9.8
9	8.8
24	7.8
77	6.8
236	5.8

The number of nodes necessary to lower average probes in steps of 1. Each successive reduction takes increasingly more space.

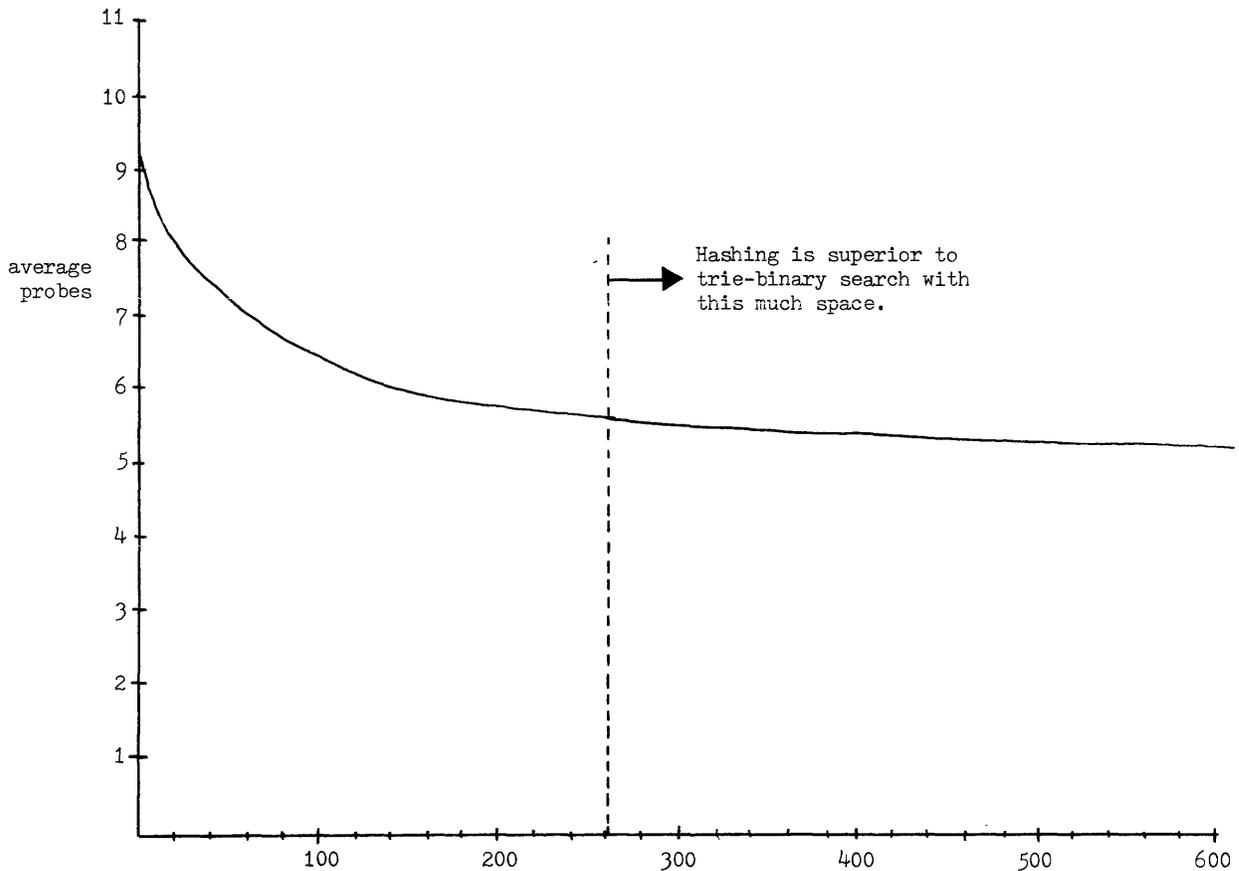


Figure 4—The average number of probes needed to find a word as a function of the number of nodes allocated in a trie. This data comes from an actual dictionary of 16,949 words.

IMPLEMENTATION PERFORMANCE

To see how trie-binary search performs in practice, algorithm B was implemented and compared to binary search. The implementation was done in Pascal, and the programs shared all code not involving the searching. Each program was run and timed ten times on a CDC 6500, using as input a text file of 7841 words arranged on 1359 lines. The maximum set size for the trie-binary search was $t=125$, which resulted in a trie with 77 nodes (using approximately 3 percent extra storage). The performance from runs with median search times is given in Table III.

TABLE III

	read dictionary	build trie	search	read text
binary search	991 ms	—	4908 ms	4203 ms
trie-binary search	998 ms	1642 ms	2242 ms	4193 ms

CPU times from a CDC 6500 for processing a text file of 7841 words using binary search and trie-binary search. The column "read text" includes CPU time for scanning and extracting the words as well.

Note that trie-binary search requires only about 46 percent as much CPU time during searching as a binary search on the sample input. Furthermore, binary search requires 20 percent more CPU time than trie-binary search, even if the trie is built "on the fly." When the trie is built and stored, the time required to read it is insignificant compared to the time required to read the dictionary.

OTHER INDEX STRUCTURES

This section considers several other index structures, and compares them to trie-binary search. The first variation, called a *length-binary* search, works as follows: words in the dictionary as sorted by length, and within each length lexicographically. An index is created which consists of a vector of pointers giving the starting location of each length group. Searching proceeds by indexing the length vector to find the locations of words of the appropriate length. Having located the appropriate subset, the spelling program searches it using a binary search. Length-binary search has the advantage that equal length words are stored contiguously so that memory can be compacted.

For many computer architectures, a combination of length- and trie-binary search leads to the best use of stor-

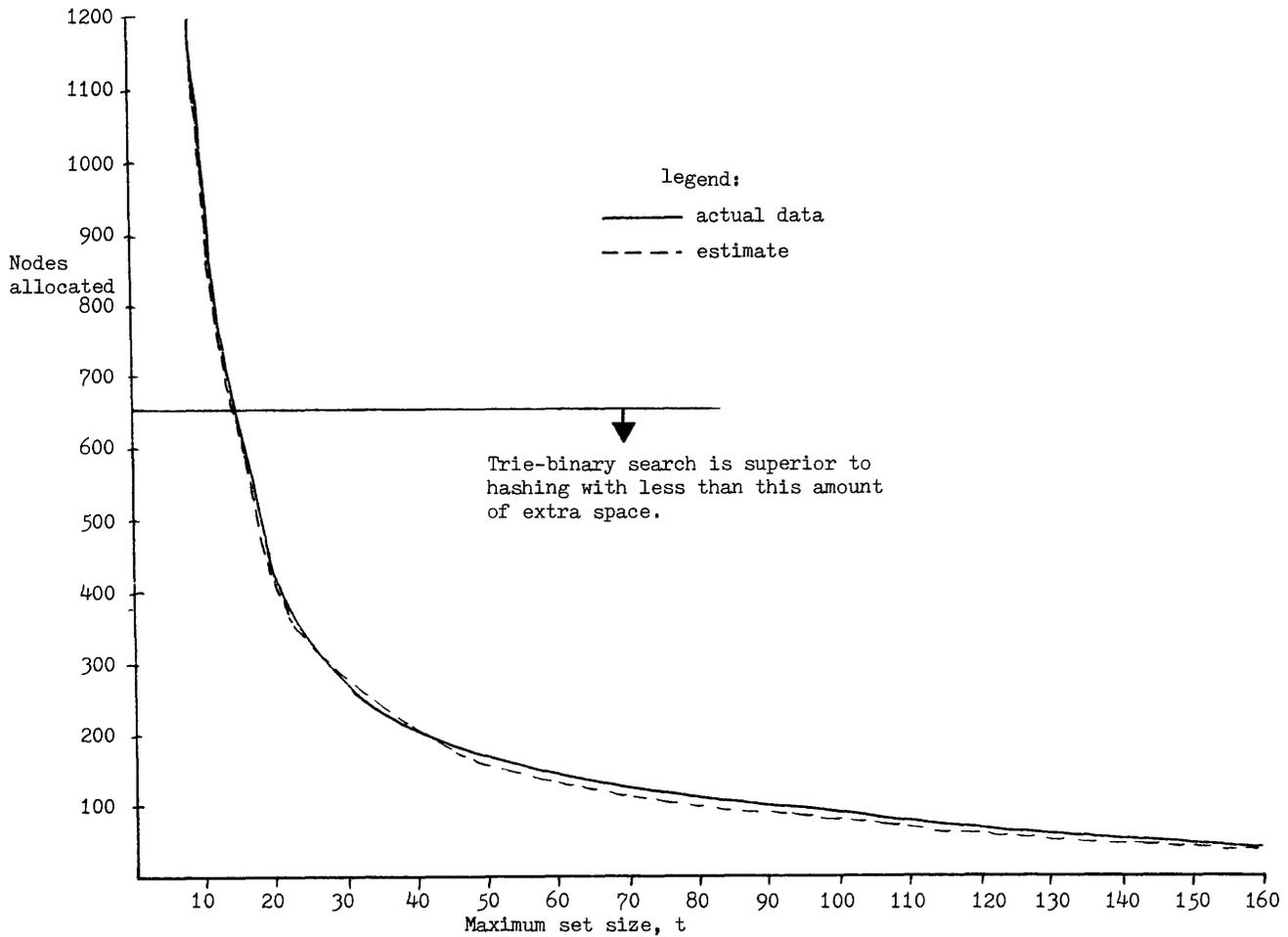


Figure 5—The relationship between nodes allocated and maximum set size, t, for a spelling dictionary of 16,949 words. The estimate assumes uniform distribution of keys.

age. At the root node of the trie, the length is used as an index; successive nodes rely on character indexing. Because words are stored compactly, a significant amount of memory is available for the trie.

Another variation is related to the work of Coffman and Eve [COFF70] who consider storage of keys in a tree. To improve the tree balance, they suggest hashing the key first, and using the bits of the hashed value to make decisions in the tree. Figure 6 shows how hashing can be applied to trie-binary search. First, one divides the dictionary into subsets by hashing the words into a fixed number of "buckets." Each subset is stored contiguously, in lexicographic order, and its starting position is stored contiguously, in a 1-level trie node. During a search operation, the program hashes the input word to obtain an index into the trie node, from which it obtains a pointer to the appropriate subset. In fact, one can view trie-binary searching as a hashed trie search in which the hash code is given by the character ordinal.

One final variation frequently suggested involves a tree-structured index which guarantees the dictionary will be divided into equal size subsets. Figure 7 shows how the method, which is based on the Indexed Sequential Access

Method [GHOS69] organizes the tree. By selecting K_1 keys at equally-spaced intervals in the dictionary and placing them in the root, the tree divides the file into equal size subsets. Of course, a program to search the multi-level tree makes approximately $\log_2 K_1$ comparisons at the root before finding the correct pointer to follow (assuming a binary search). If the search continues through m levels, the method makes an average of:

$$C_t \approx \log_2 n - 1 - m \tag{5}$$

comparisons and follows m pointers, where n is the number of keys in the dictionary. We assume here that the number of keys promoted into the index is insignificant compared to the total number of keys.

For trie-binary search we assumed that the cost of following one pointer was the same as the cost of performing one comparison. Applying the same assumption here implies that an Indexed Sequential tree costs the same to search as a straight binary search when the entire set of keys resides in main memory. Therefore, trie-binary search is far superior.

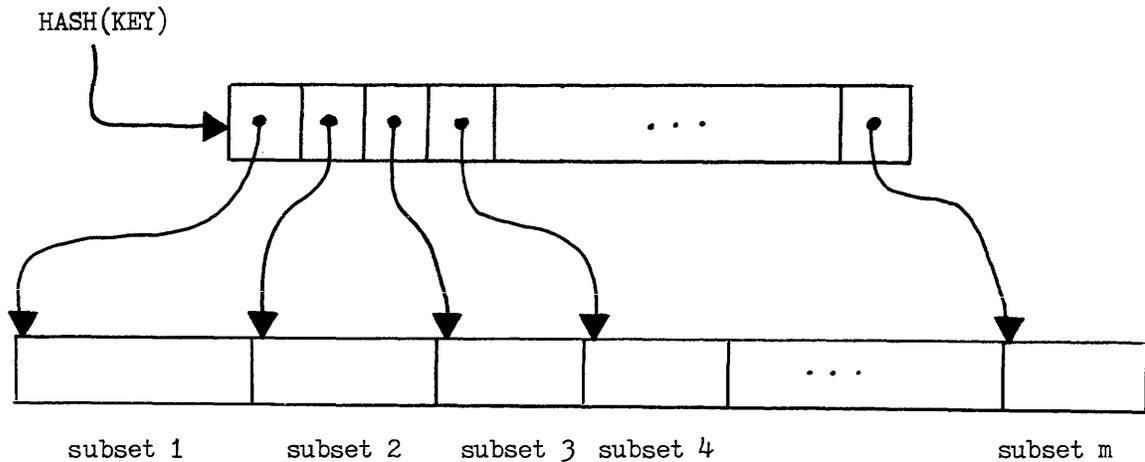


Figure 6—A 1-node trie using hashing. To locate a key, its hashed value is used to index a vector which points to the correct subset. The subset must be searched to locate the exact match. Note that the subsets may differ in size.

CONCLUSIONS

This paper has presented a data structure and algorithm, the trie-binary search for dictionary searching. The method performs well compared to a binary search while using very little extra space beyond that required to store the keys themselves. Trie-binary search has been applied to a typical spelling dictionary, and statistics have been gathered on its performance. For the distribution of words in a typical spell-

ing dictionary, trie-binary search makes 52 percent fewer probes than binary search, while using less than 3 percent extra space. For an input document of roughly 30 pages, trie-binary search required 74 percent as much total CPU time (46 percent as much CPU time during searching) as a binary search.

Several questions remain unanswered. It would be interesting to find out how trie-binary search performs on sets of keys other than spelling dictionaries. One might conjecture

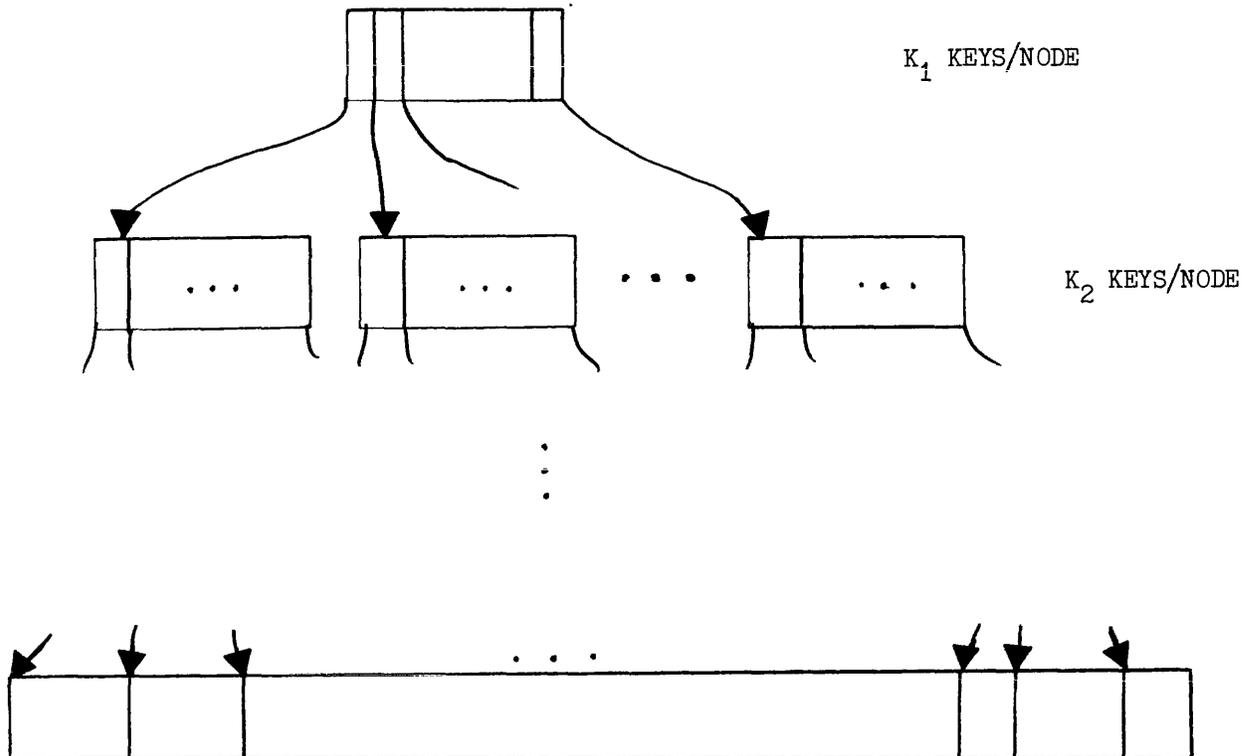


Figure 7—A multi-level tree which requires searching at each node. The structure works well for a file in secondary storage, but is no better than a binary search when keys reside in main memory.

that typical data are no more skewed with respect to a uniform distribution than a spelling dictionary. If this is the case, trie-binary searching should perform well. There is also no reason to test the letters in left-to-right order. Testing in other orders might produce a better distribution for English. Finally, if a hashed method is considered, choice of an appropriate hash function needs to be considered.

REFERENCES

- [COFF70] Coffman, E. and J. Eve, "File Structures Using Hashing Functions," *Comm. ACM* Vol. 13, No. 7, July 1970, pp. 427-436.
- [FRED60] Fredkin, E., "Trie Memory," *Comm. ACM* Vol. 3, No. 9, September 1960, pp. 490-499.
- [GHOS69] Ghosh, S. and M. Senko, "File Organization: On the Selection of Random Access Index Points for Sequential Files," *J. ACM*, Vol. 16, No. 4, October 1969, pp. 569-479.
- [KERN78] Kernighan, B., M. Lesk and J. Ossanna, "UNIX Time Sharing System: Document Preparation," *The Bell System Technical Journal*, Vol. 57, No. 6, July-August 1978, pp. 2115-2135.
- [KNUT73] Knuth, D., *The Art of Computer Programming, Vol 3: Sorting and Searching*, Addison Wesley, 1973.
- [MORR75] Morris, R. and L. Cherry, "Computer Detection of Typographical Errors," *IEEE Trans. on Professional Communication* PC-18 March 1975, pp. 54-56.
- [YAO76] Yao, A. and F. Yao, "The Complexity of Searching an Ordered Random Table," *Proc. 17th IEEE Symposium on Foundations of Computer Science* October 1976, pp. 173-177.

New indices for bibliographic data and their applications

by YAHIKO KAMBAYASHI, SHUZO YAJIMA, OSAMU KONISHI and TAKAKI HAYASHI

Kyoto University
Kyoto, Japan

INTRODUCTION

The KWIC Index is a widely-used tool for finding desired paper titles since it is simple yet very powerful. So in a bibliography of some specific area, the KWIC Index of Titles, Author Index and Subject Index are normally used. These indices are, however, not always sufficient for finding mutually-related papers. In this paper new indices for bibliographic data will be presented together with their applications.

The RS Index (Reference Structure Index) handles reference citation structure and is suitable for detecting influential papers, getting research trends and finding mutually-related research fields. A basic algorithm and improved algorithms have been developed, each of which requires memory only proportional to the number of papers to be handled (i.e. minimum). Functions of these algorithms are explained and example outputs are shown (see Figures 1, 2, 9 and 10).

The MULTI-KWIC has a capability of automatic detection of key phrases by finding maximum common subsequences contained in the paper titles. Using these key phrases, the following two kinds of indices are generated: (1) A MULTI-KWIC list (a linear list of titles ordered by one key phrase and one keyword, as in Figure 6), and (2) a year-distribution table (for each key phrase a table which shows identifications of papers classified by year is prepared, as in Figure 8).

These new indices are applied to 1600 papers in relational database and related areas which have been collected by the authors. The result shows their usefulness in bibliographic data-handling. A personal information system based on these indices has been also developed.

REFERENCE STRUCTURE INDEX

In order to treat a reference citation structure of bibliography, the following problems must be considered:

1. Large amount of input data is required, since for each paper the information on its references must be supplied.
2. Reference citation relationships among papers are usually represented by a directed graph. It is, however, very difficult to print out a directed graph.

3. Usually a large amount of memory space and large software are required to print out a directed graph.
4. It is necessary to print out a directed graph which is human-oriented (i.e., the information contained in the graph is understood easily by looking at the graph).

In our implementation of the RS Index, these problems are solved by the following methods:

1. Reference information is shortened by the use of the identification code (ID for short) representing the paper. The ID is constructed from the names of the authors and the publication date. It can be easily constructed and the maximum length is 10.
2. Usually we need to know a set of papers which have a direct or a transitive citation relationship with a given paper. In such a case, we can use a tree expansion of a directed graph. By this reason, the RS Index only treats tree outputs, which are easy to understand. This output tree is called an RS tree.
3. Even if we restrict the outputs to trees only, usually the required space for output buffer is more than $O(n)$, where n is the number of papers contained in the output tree and $O(n)$ denotes that the value is proportional to n . We have developed a procedure which utilizes a push-down stack. $O(n)$ is the minimum storage space for memorizing the data, and the stack itself normally requires $O(\log n)$ space.
4. An RS tree is printed from left to right. When the RS tree reaches at the right end of the output paper, successive trees are printed after the main RS tree. So we can print the tree whose maximum level is arbitrary. Since the width of the tree is also arbitrary, essentially the tree of the arbitrary size can be printed (the size of the tree is actually restricted by that of push-down stack).
5. When the number of papers in the RS Index is large, the RS tree spreads widely. In such a case it may be difficult to get useful information from the RS tree. A procedure to produce compact RS trees without changing their structure is developed as well as a procedure with tree trimming and ordering facilities. These facilities are also realized by algorithms based on pushdown stacks, thus we don't need separate procedures for these facilities.

We have developed a PL/I program for the RS Index, which consists of approximately 800 steps.

Identification codes for papers must be simple as well as human oriented. We use a character string $oooo\Phi\Pi\Pi\Delta\Delta\chi$ of length ten as an ID.

$oooo\Phi$: (*author field*) It consists of first four characters of the first author's name ($oooo$) and first character of the second author's name (Φ). When the first author's name is shorter than four and there is a second author, - (hyphen) is inserted. If only the first author is shown with *et al.*, * is used.

$\Pi\Pi\Delta\Delta$: (*date field*) Year ($\Pi\Pi$) and month ($\Delta\Delta$) of the publication are shown. If not known, ? is used.

χ : (*extension field*) It is optionally used to distinguish the papers which have the same name and date fields. It is taken from the first letter of the title of the paper except articles. If two distinct papers have the same extension, the next unused character in the alphabetical order is assigned.

There are two kinds of RS Indices; one uses "A refers to B" and the other uses "A is referred to by B". For simplicity, only the "A refers to B" relationship is considered in this paper to explain the algorithm.

Basic algorithm for the RS index

This algorithm needs only one-line output buffer and has a simple recursive formulation, which shows why a stack is needed.

Call display ($P, 0$), where P is the root paper and procedure display is defined as follows:

```

procedure display ( $p$ : Paper,  $i$ : Integer);
  begin  $L \leftarrow$  list of papers referred to by  $p$ ;
  for  $x$  in  $L$  do;
    begin display ( $x, i+1$ );
    if  $x$  not last in  $L$  then newline;
  end;
  print  $p$  at level  $i$ ;
end

```

In an actual program, in order to add many additional facilities easily the iterative procedure is adopted using a push-down stack.¹ The above algorithm is simple since it does not need to store the output position of each ID. The lines to indicate reference relationships among papers can be drawn by providing a bit for each level of the tree. In actual program, if there exists a paper appearing more than once, after first appearance only "TO $i**j$ " is printed if the paper is first printed at the j -th position of the i -th level. This algorithm needs constant output buffer, and the required size for the stack is $O(n)$, which is in the worst case, where n is the number of nodes in the tree. If each paper has more than one reference (it is normally satisfied), the size is $O(\log n)$.

In order to use the RS Index practically the following two problems must be solved.

1. If there exist two papers A and B such that A and B refer to (or transitively refer to) B and A , respectively, then the basic algorithm fails.
2. When the number of references increases, it may be difficult to see the whole figure produced by the basic algorithm, because it spreads widely (see Reference 1 for the output of the basic algorithm). Figure 1 is an example generated by an improved algorithm.

In order to handle Problem 2 some useful facilities are prepared such as ordering facilities and filtering facilities.

1. Elementary facilities
 - a. Output of keywords, comments for each paper.
 - b. Output of information about connection between papers. These are printed on edges.
 - c. Designation of the width of the print-out by restricting maximum level of the RS tree.
 - d. Process of pattern matching for IDs. In the case of specifying a paper, we may not know when the paper was published even though we know who wrote it. For this reason, if we specify "COD-D*****," for example, the RS trees can be produced whose root paper's first author is Codd.
2. Ordering of references

We have adopted the following orderings from the practical viewpoint:

- a. To arrange IDs in alphabetical order by author's name.
 - b. To arrange IDs in order of published date.
 - c. To arrange IDs in order of labels of edges, which means the relationship or connection strength between a pair of papers.
 - d. To arrange IDs in order of the path length from the root. In an RS tree, in order not to print out redundant subtrees, if there exists a paper appearing in the tree more than once, "TO $i**j$ " is printed after the first appearance. This means that the maximum path from the root is not always expressed explicitly. We can position papers in the RS tree more clearly if the maximum path is output explicitly.
3. Tree-trimming facilities
 - a. To eliminate IDs expressed such as "TO $i**j$." This is available when looking over what papers are printed.
 - b. To print out the papers which are cited more than N times, where N is a threshold value.
 - c. To delete the transitively connected edges from the RS Index.
 - d. To print out IDs whose authors and publishing date are specific.
 - e. To eliminate IDs by the label of edges.

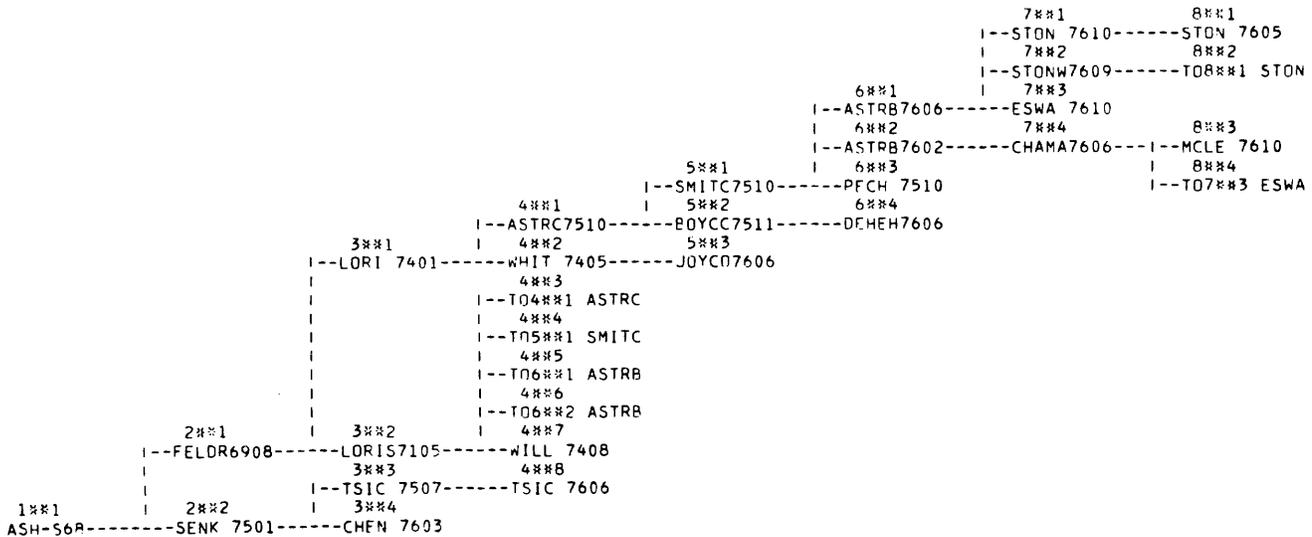


Figure 2—The RS Index obtained by deleting transitively connected edges from Figure 1.

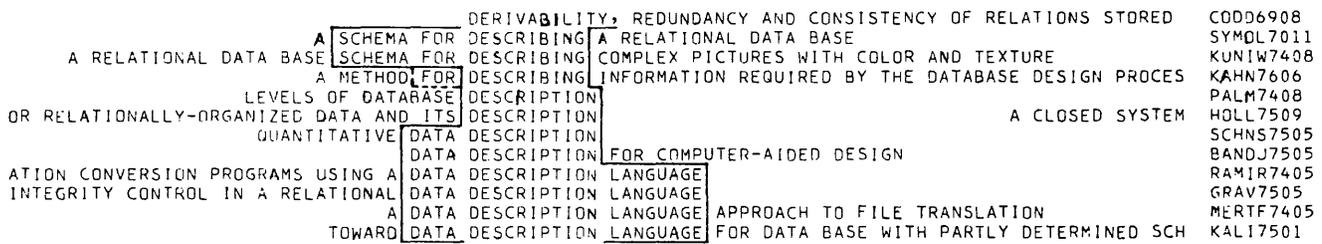


Figure 3—An improved KWIC list.

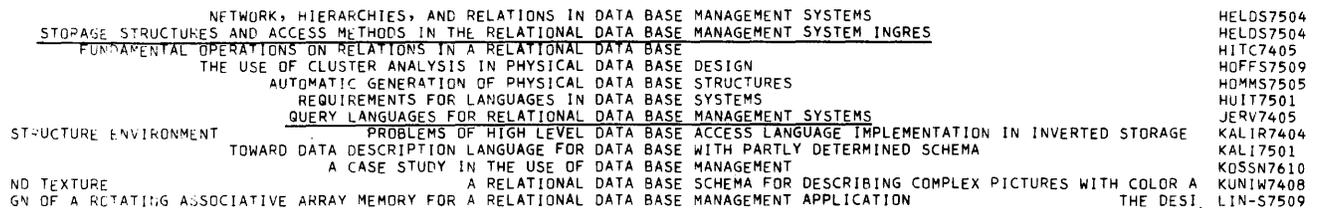


Figure 4—Conventional KWIC list.

of the titles. In another version a KWIC list is produced by considering the keyword and words after the keyword. The total length of the subsentence (the keyword and the words after the keyword) is fixed and thus a sorting facility can be used to produce the output. This version, however, still has

the following problems:

1. Since titles are not ordered by the words before the keyword, there are still cases when two titles with a common key phrase are not printed adjacently.
2. Because of the inflections of words, similar key phrases may be printed separately.

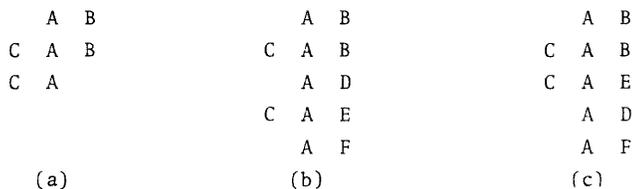


Figure 5—Rearrangement by the consecutive retrieval property.

In our version of KWIC the following methods are used to solve these problems:

1. For each word only the first six characters are used for the ordering information.
2. Words before the keyword are considered as well as the keyword itself and words after the keyword.

```

HILL7509          #DATA LANGUAGE: THE ACCESS LANGUAGE OF THE DATACOMPUTER DATA
KALIR7404        STRUCTURE ENV+ #PROBLEMS OF HIGH LEVEL DATA BASE ACCESS LANGUAGE IMPLEMENTATION IN INVERTED STORAGE DATA
CARLK7606        L DATA BASE SYSTEM #A GENERALIZED ACCESS PATH MODEL AND ITSAPPLICATION TO A RELATIONA DATA
SENKA73          #DATA STRUCTURES AND ACCESSING IN DATA BASE SYSTEMS DATA
SENKA73          #DATA STRUCTURES AND ACCESSING IN DATA BASE SYSTEMS DATA
ALMA7211        #SPECIFICATIONS IN A DATA INDEPENDENT ACCESSING MODEL DATA
ASTRA7211        #CONCEPTS OF A DATA INDEPENDENT ACCESSING MODEL DATA
ASTRG7405        ATH SELECTION ALGORITHM FOR THE DATA INDEPENDENT ACCESSING MODEL (DIAM) #A SEARCH P DATA
SCHN7606        #A RELATIONAL VIEW OF THE DATA INDEPENDENT ACCESSING MODEL DATA
YAO7703         #AN ATTRIBUTE BASED MODEL FOR DATABASE ACCESS COST ANALYSIS DATABASE
YAO-7704         #APPROXIMATIING BLOCK ACCESSES IN DATABASE ORGANIZATIONS DATABASE
GEY-M7509        #KEYWORD ACCESS TO A MASS STORAGE DEVICE AT THE RECORD LEVEL DEVICF
SENKA7609        N THE PHYSICAL DEVICE LEVEL: A GENERAL MODEL FOR ACCESS METHODS #DIAM II AND LEVELS OF ABSTRACTIO DEVICF
    
```

Figure 6—Multi-KWIC list.

Approach 1 is employed since a dictionary is not required. By our experience in using 750 titles there are only three cases when different key phrases are regarded as the same ones. For the purpose of handling this kind of error, an editing facility is prepared in the program. In order to handle the problem of ordering by words before and after the keyword the following approach is used:

1. First, a primitive version of a KWIC list is prepared.
2. Let $f_6(w)$ be the operation of extracting first six characters from w (when the length of w is less than six, blank characters are added and the length is adjusted to six; w can be a null string—a string of length 0) and let W be the keyword in the list prepared in Step 1. For each subsentence $w_4w_5W w_1w_2w_3$ of the title, calculate $f_6(W)$, $f_6(w_1)$, $f_6(w_2)$, $f_6(w_3)$, $f_6(w_4)$ and $f_6(w_5)$. Here w_1, w_2, w_3 are words after the keyword and w_4, w_5 are words before the keyword.
3. Sort the titles by $f_6(W) \circ f_6(w_1) \circ f_6(w_2) \circ f_6(w_3) \circ f_6(w_4) \circ f_6(w_5)$, where \circ denotes the concatenation of the words.

The output of the above procedure gives an improved KWIC list. For further improvement the consecutive retrieval property is used. The consecutive retrieval property is introduced by Ghosh⁴ for organizing an efficient file. (For further information see the list of papers prepared by Lipski, currently of the Coordinate Science Laboratory, University of Illinois⁵). For example, consider the titles containing key phrases AB, CA, CAB where A is the keyword. The ordering shown in Figure 5(a) is appropriate. By this method the list shown in Figure 5(b) is rearranged and the list in Figure 5(c) is obtained.

Using this result key phrases consisting of less than seven words can be extracted from a set of titles. The program has a facility to extract key phrases contained in $M(\cong N)$ for a given threshold value $N(\cong 2)$. By this method key phrases of the form of " w_1 of w_2 " or " w_1 and w_2 " can be extracted. Extraction of such key phrases is not possible by the commonly used extraction method whereby phrases are selected from words between conjunctions and prepositions.

By this method sometimes improper key phrases are extracted. Editing facility for eliminating such key phrases is also prepared. Since key phrases contained in many titles are not useful for characterizing the paper, key phrases contained in $M(N_1 \cong M \cong N_2)$ titles can be used for paper characterization, where N_1 and N_2 are the user-determined threshold values. There will be, however, papers without

any characterizing key phrases and thus thesauri may be required for full automatic classification of papers.

The extracted key phrases are used to generate a MULTI-KWIC list. In the list titles with one common key phrase and one common keyword are adjacently printed. A

1	A B C	E , F
2	A	H
3	A B	F , H
4	A B C D	F
5	B C D	E , R
6	B C	R , S

(a)

1	A B C	E
3	A B	F
1	A B C	F
4	A B C D	F
2	A	H
3	A B	H
5	B C D	E
6	B C	R
5	B C D	R
6	B C	S

(b)

Figure 7—Algorithm for the Multi-KWIC Index.

CONCEPTUAL SCHEMA

FREQ. 7

--1970>	--1970--	--1971--	--1972--	--1973--	--1974--	--1975--	--1976--	--1977--	--1978--
						TSIC 7507	BENCB7601	BILLN7701	
						TSIC 7509	MELT 7609	BUBE 7708	
								NIJS 7701	

RELATIONS

FREQ. 18

--1970>	--1970--	--1971--	--1972--	--1973--	--1974--	--1975--	--1976--	--1977--	--1978--
CODD 6908				CODD 7306	HITC 7405	GOTL 7505	BERN 7612	ABEERF7704	
					MELT 7405	HALLT7501	HALLO7601	BEERF7708	
					TITM 7404	LIENT7509	LOZI 7609	FURTK7708	
						VALL 7503		HUNTS7703	
								RISS 7712	
								TODD 7708	

Figure 8—Year-distribution table.

Double-KWIC list³ is also known as a list with two keys. The major differences are (1) a MULTI-KWIC list has a capability of automatic key phrase extraction and (2) the output format of MULTI-KWIC is similar to conventional KWIC so the both outputs can be combined.

An example of the MULTI-KWIC list is shown in Figure 6, where the titles containing "access" and "data" are listed. The procedure used for the MULTI-KWIC list is explained in Figure 7. Assume that we have a set of titles having key phrases and keywords shown in Figure 7(a) (here, the first line shows that the identification number is 1 and the title contains one key phrase *ABC* and two keywords *E* and *F*). The priority of sorting is determined as (1) the keyword part of the key phrase, (2) the keyword and (3), the remaining part of the key phrase. The result is shown in Figure 7(b).

Figure 8 shows a year-distribution table which will be prepared for each extracted key phrase (here, 'relations' and 'conceptual schema'). If a paper contains two key phrases W_1 and W_2 , where W_2 is a proper substring of W_1 , then the paper is classified to key phrase W_1 category. By this table activity transition of some specific area will be observed.

APPLICATIONS

Figures 9 and 10 show RS Indices taken from the result of application to papers of relational database and related areas, which are collected by the authors (it is an extension of Codd's bibliography⁹). Figure 9 shows the RS tree for the referred-to-by relationship whose root is Abrial's paper which discusses data semantics. This RS tree presents many papers concerned with some kinds of data models and conceptual schema. For example, there are the DIAM II model by Senko (SENK7601, SENK7501), conceptual schema by Tsichritzis (TSIC7507, TSIC7606), DBMS architecture of

the next generation by Nijssen (NIJS7601), criterion of conceptual schema by Kent shown in KENT7609, entity-relationship model by Chen (CHEN7603), and so on. Figure 10 is a descendant tree whose root is the paper of Armstrong. This paper is concerned with axiomatization for functional dependency in relational databases. Some of the descendants of this paper are one of Bernstein, which discusses a synthetic design of relational databases, and one of Fadous, which introduces the algorithm for finding candidate keys efficiently. The papers written by Zaniolo, Fagin and Beeri are also printed, which discuss multi-valued dependency or generalizations of functional dependency.

Another application of indices introduced in this paper is a personal information system. The system has been developed under the LABOLINK network⁷ using the model M-190 computer of Fujitsu at the Data Processing Center, Kyoto University. Papers are accessed by a combination of keywords, authors and years. The system has the following specific features:

1. For selecting appropriate keywords, a key phrase KWIC table is shown (see Figure 11). It is a KWIC list of key phrases which are automatically detected by the MULTI-KWIC.
2. For each specified key phrase a year-distribution table (Figure 8) can be displayed.
3. For each paper an RS Index for the paper can be generated.

The system is examined by JICST (Japan Information Center of Science and Technology) bibliographic data tapes as well as the papers of relational database areas collected by us (about 1,600 papers) and the effectiveness of the system is proved. These facilities are to be merged with the relational-model-based research information system currently under development.⁸

1**1	2**1	3**1	4**1	5**1	3**1
ABRI 7404	SENK 7601	RUCH 7601	GROTV7601	RUCH 76	
			INFORMATION PROCESSING		
		3**2			
		--GROTV76	4**1		
		3**3	4**2		
		--FALK 7601	--GROTV76	4**1	
2**2		3**4	4**3		
--SENK 7501		--TSIC 7507	--TSIC 7606		
		FEATURES OF A CONCEPTU AL SCHEMA			
	2**3	3**5			
--RUCH 76	3**1	--SENK 76	2**1		
2**4		3**6	4**4		
--MIJAP7601		--NIJS 7601	--RUCH 76	3**1	
DATA BASE S TRUCTURES		ARCHITECTUR E FOR THE N EXT GENERAT ION DATABAS E			
	2**5		4**5	5**2	
--MACH 7601			--MIJAP76-2**4	--GROTV76	4**1
INTEGRITY C ONSTRAINTS					
2**6		4**6	5**3	6**1	
--KERSK7609		--MACH 76	2**5	--ADIBD7601	--MIJAP76
DATA MODELS				LOGICAL DAT A BASE DESI GN	2**4
	2**7		4**7		6**2
--KENT 7609		--KENT 76	2**7		--KERSK76
CONCEPTUAL MODEL					2**6
2**8		4**8			
--KALI 7501		--GROTV76	4**1		
DATA DESCRI PTION LANGU AGE					
		4**9			
		--FALK 76	3**3		
		4**10	5**4		
		--BRACP7601	--KERSK76	2**6	
		4**11	5**5		
	3**7	--MIJAP76	2**4	--GROTV76	4**1
	--MOULR7601				
	CONCEPTUAL MODEL				
	3**8	4**12			
	--KERR 7509	--KERSK76	2**6		
		4**13			
		--FALK 76	3**3		
	3**9	4**14	5**6		
	--CHEN 7603	--YAO 7703	--YAO- 7704		
	VIEW OF DAT A	MODEL FOR D ATABASE	DATABASE OR GANIZATIONS		
	3**10	4**15	5**7		
	--BRACP76	4**10	--WEBE 7601	--RUCH 76	3**1
			INTEGRITY C ONSTRAINTS		
	2**9	3**11	4**16	5**8	
--HUIT 7501		--MACH 76	2**5	--KERSK76	2**6
DATA BASE S YSTEMS				--GROTV76	4**1
			4**17		
			--KENT 76	2**7	
		3**12	4**18		
	--BENCB7601	--RUCH 76	3**1		
	CONCEPTUAL SCHEMA				
	2**10	3**13	4**19		
--HALLO7601		--KENT 76	2**7	--KERSK76	2**6
2**11		3**14	4**20		
--FALK 76	3**3	--GROTV76	4**1	--GROTV76	4**1
2**12		3**15	4**21		
--BRACP76	4**10	--FALK 76	3**3	--FALK 76	3**3
2**13					
--BENCB76	3**12				
2**14					
--ADIBD76	5**3				

Figure 9—The RS Index for ABRI7404.

```

ARMS 7408
1**1      2**1      3**1
ARMS 7408---|--ZANI 7607-----BEERF7704
| ANALYSIS AN    MULTIVALUED
| D DESIGN      DEPENDENCI
|               ES IN DATAB
|               ASE RELATIO
|               NS
|
| 2**2
|--FAGI 7607
| MULTIVALUED
| DEPENDENCI
| ES
|
| 2**3      3**2
|--FADOF7505---|--ZANI 76 2**1
| 2**4      | 3**3      4**1
|--BERN 7510  |--BERN 75-2**4---FAGI 76 2**2
| FUNCTIONAL
| DEPENDENCIE
| S
| 2**5
|--BEERF77 3**1
| 2**6      3**4      4**2      5**1      6**1
|--ADIBD7601---|--MIJAP7601---|--GRUTV7601-----RUCH 7601-----GROTV76 4**2
| LOGICAL DAT  | DATA BASE S  | INFORMATION
| A BASE DESI  | TRUCTURES   | PROCESSING
| GN          |           |
|           | 3**5      | 4**3
|           |--KERSK7609  |--AUIAD76 2**6
|           DATA MODELS

```

Figure 10—The RS Index for ARMS7408.

	DATA	60
	LARGE DATA	4
	LEVEL DATA	3
	NETWORK DATA	3
	VIEW OF DATA	4
	SEMANTICS OF DATA	3
	RELATIONAL DATA	9
	DATA BASE	75
	SHARED DATA BASE	5
	LOGICAL DATA BASE	3
	RELATIONAL DATA BASE	24
	MODEL FOR DATA BASE	3
	DISTRIBUTED DATA BASE	6
	DATA BASE DESIGN	8
	DATA BASE ENVIRONMENT	3
	DATA BASE MANAGEMENT	23
	RELATIONAL DATA BASE MANAGEMENT	6
	GENERALIZED DATA BASE MANAGEMENT	3
	DATA BASE MANAGEMENT SYSTEM	12
	DATA BASE MANAGEMENT SYSTEMS	4
	DATA BASE STRUCTURES	3
	DATA BASE SYSTEM	22
	RELATIONAL DATA BASE SYSTEM	12
	DATA BASE SYSTEMS	18
	DATA BASES	35
	LARGE DATA BASES	4
	RELATIONAL DATA BASES	7

Figure 11—Key phrase KWIC table.

ACKNOWLEDGMENT

The authors are indebted to Mr. Katsumi Tanaka, Mr. Le Viet Chung and Mr. Narao Nakatsu for their useful suggestions and comments. This work is supported in part by the Science Foundation Grant of the Ministry of Education, Science and Culture of Japan.

REFERENCES

1. Kambayashi, Y., T. Hayashi, Y. Tanaka and S. Yajima, "A Linear Storage Space Algorithm for a Reference Structure Index," *Information Processing Letters*, Vol. 7, No. 2, February 1978, pp. 66-71.
2. Hayashi, T., Y. Kambayashi and S. Yajima, "RS Index—A New Indexing Method Using Reference Structure of Bibliography," *SIGEC Record*, The Institute of Electronics and Communication Engineers of Japan, EC78-14, June 1978.
3. Petrarca, A. E. and W. M. Lay, "The Double-KWIC Coordinate Index," *Journal of Chemical Documentation*, Vol. 9, No. 6, 1969.
4. Ghosh, S. P., "File Organization: The Consecutive Retrieval Property," *CACM*, Vol. 15, No. 8, September 1972, pp. 802-808.
5. Tanaka, K., Y. Kambayashi and S. Yajima, "Organization of Quasi-Consecutive Retrieval Files," *Information Systems*, Pergamon Press (to appear).
6. Lipski, W., Jr., "Consecutive 1's Property and Related Topics: Recent Results," *Technical Report*, Institute of Computer Science, Polish Academy of Science, Warsaw Poland (Dr. Lipski is currently with the Coordinated Science Laboratory, University of Illinois.), October 1977.
7. Yajima, S., Y. Kambayashi, S. Yoshida and K. Iwama, "LABOLINK: An Optically-Linked Laboratory Computer Network," *IEEE Computer*, Vol. 10, No. 11, Nov. 1977, pp. 53-59.
8. Kambayashi, Y., K. Tanaka and S. Yajima, "A Relational Data Language with Simplified Binary Relation Handling Capability," *Proc. of the Third International Conference on Very Large Data Bases*, October 1977, pp. 338-350.
9. Codd, E. F., "Relational Data Base Management: A Bibliography," IBM Research Laboratory San Jose, California, August 1975, (also appeared in D. D. Chamberlin's paper entitled "Relational Data-Base Management Systems," *ACM Computing Surveys*, Vol. 8, No. 1, March 1976, pp. 43-66.).

Visual inspection of metal surfaces

by J. L. MUNDY

General Electric Company
Schenectady, New York

INTRODUCTION

The majority of applications of automatic visual inspection have been the case in which a high contrast image can be obtained. This will result from object silhouettes¹ and high contrast reflectivity changes as in printed text. In these cases, the image can usually be successfully segmented by a threshold operation^{2,3} leading to a two-level or binary image.

The cases that lead to difficulty at present involve the use of reflected light. Here one is faced with shadows and highly variable reflected intensity. This is the case for metal surfaces where the reflected intensity is a strong function of illumination and viewing direction. On the other hand, the inspection of metal surfaces represents an important domain of applications. Of particular interest is the detection of small surface defects such as nicks and scratches.

In order to implement automatic computer inspection of metal surfaces, optical and illumination means must be provided that provide high contrast images of surface defects. In addition, there must be a relationship between image intensity and surface profile. This paper will discuss a number of theoretical aspects of the scattering of light from metal surfaces. This provides a basis for computer modeling of the metal surface. The theory is based on earlier work by Beckmann⁴ and Horn.⁵ The method has been tested experimentally and several examples will be demonstrated.

SCATTERING THEORY

Several phenomena play a role in the distribution of light scattered from a metallic surface. The most important effects are 1) the variations of the surface normal and 2) shadowing. The variation of surface normal generally occurs on two scales. A fine scale variation is present that represents the basic surface roughness. In the case of surface defects, there exists a more gradual variation corresponding to the surface deformation associated with the defect. The combined variation is illustrated in Figure 1. In the case of shadowing, portions of the surface are occluded by variation in the surface due to defects. This condition is shown in Figure 2 for a crater or pit type defect.

The theoretical situation is best developed for the case of surface normal variations in the presence of random fine

scale surface height variations. Beckmann and Spizzichino⁶ have explored this case extensively for various scales of surface roughness. In this paper we will only consider the case where the surface is rough compared to the wavelength of light. The reflection coefficient for scattered power in this case is given as

$$\langle \rho \rho \rangle^* = \frac{\pi R^2}{4kA^2 \cos^2 \theta_1} \frac{V^4}{V_z^4} \left(\frac{T}{\sigma} \right)^2 \exp \left\{ \frac{-V_{xy}^2}{4V_x^2} (T/\sigma)^2 \right\}$$

where

R=Reflection coefficient of an equivalent smooth surface.

$k=2\pi/\lambda$ (λ —wavelength of illumination)

A=Area of illuminated surface

$\vec{V}=\vec{k}_1\vec{k}_2$

V_x =Component of V along the surface normal

V_{xy} =Component of V perpendicular to surface normal

\vec{k}_1 =Incident wave vector

\vec{k}_2 =Reflected wave vector

T=Correlation distance of surface roughness

σ =Surface height variance.

The coordinate system and scattering vectors are defined in Figure 3.

It will prove useful to transform this notation to that introduced by Horn. This notation is defined in Figure 4. It follows that

$$-\vec{k}_1 \cdot \vec{k}_2 = k^2 \cos g$$

$$\begin{aligned} V^2 &= \vec{k}_1^2 + \vec{k}_2^2 - 2\vec{k}_1 \cdot \vec{k}_2 \\ &= 2k^2(1 + \cos g) \end{aligned}$$

also

$$V_z = k(\cos i + \cos e).$$

Horn defines the I, E, G as the cosines of the angles i, e, g respectively. From the consideration above,

$$V^2 = 2k^2(1 + G)$$

$$V_z^2 = k^2(1 + E)$$

* This appears as Equation 59, p. 88 of Reference 4.

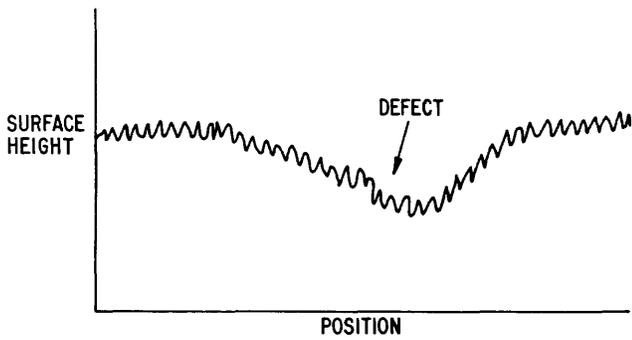


Figure 1—The variation in surface height for a rough surface in the vicinity of a surface defect.

Noting that

$$V_{xy}^2 = V^2 - V_z^2$$

we finally obtain

$$\langle \rho \rho^* \rangle E_0^2 = \alpha \left(\frac{(1+G)^2}{(1+E)^4} \right) e^{-\beta[2(1-G)/(1+E)^2 - 1]} \quad (2)$$

$$\alpha = \frac{R^2 A}{\pi r_0^2} \beta$$

$$\beta = \frac{T^2}{4\sigma^2}$$

Here E_0^2 is proportional to the power scattered by a smooth plane of area A . This is given by

$$E_0^2 = \frac{k^2 A^2 I^2}{4\pi^2 r_0^2}$$

where r_0 is the distance from the observer to the plane. Thus the result in (2) would be proportional to the scattered power from the rough surface. This form is suitable for interpretation and comparison with experiment.

CASE I—SOURCE AT OBSERVER

If the illumination source is colimated (unidirectional) and located on the axis of the observation, then $I=E$ inde-

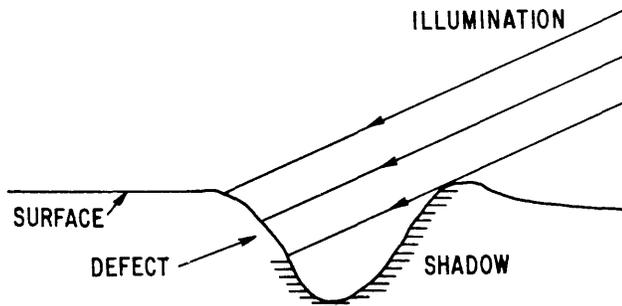


Figure 2—The shadowing of the surface due to a defect.

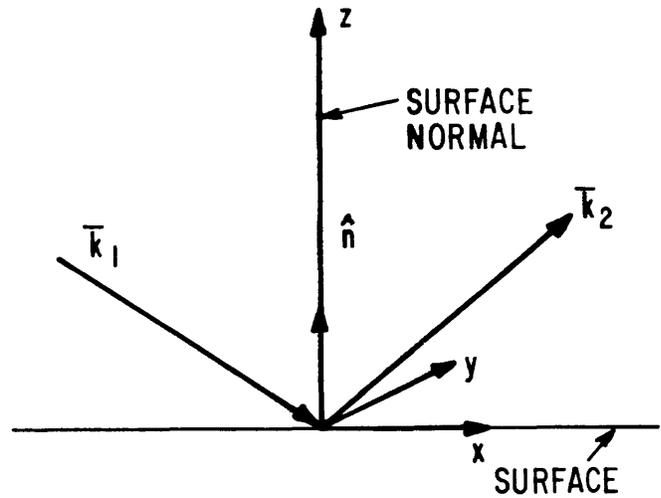


Figure 3—The scattering geometry according to Beckmann.

pendent of the direction of the surface normal. In this case we find

$$P = \langle \rho \rho^* \rangle E_0^2 = \frac{\alpha}{4 I^4} e^{-\beta[(1/I^2)-1]} \quad (3)$$

This is a particularly simple form which can be easily interpreted. The first observation is that β depends only on surface roughness. As the surface becomes rougher β decreases. The exponential term in (3) dominates the behavior of P and, for large β , will lead to a rapid fall off in intensity for $I \neq 1$. P is maximum for $I=1$ which corresponds to the specular reflection condition. Note also that surface reflectivity only affects α and thus not the angular distribution. Also the peak power is proportional to β , which is to be expected, since smoother surfaces concentrate more power into the specular direction.

This expression was tested for a number of surface roughnesses found from a selection of metal industrial parts. The

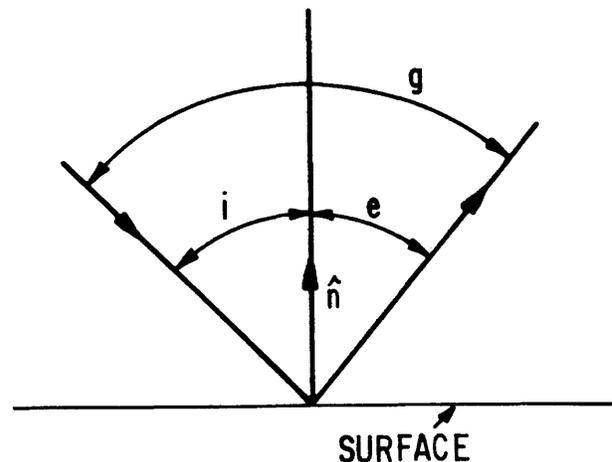


Figure 4—The scattering geometry according to Horn.

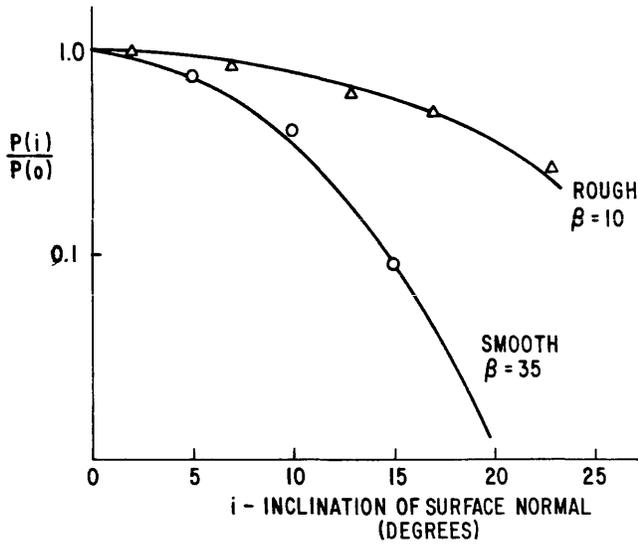


Figure 5—The experimental and theoretical variation of reflected power with surface normal inclination. The theory in Expression 3 is shown as a solid line.

surface was rotated relative to the optical viewing axis and the reflected power on axis is given in Figure 5 for two surfaces. In the same figure a best fit of Expression 3 is shown as solid lines. The agreement is quite satisfactory.

CASE II—GRAZING AND NORMAL ILLUMINATION

From the previous development, it can be seen that the slope of metal surfaces can be deduced from reflectivity

measurements. This assumes that both reflectivity and surface roughness are known. The former condition is not easily obtained in practice. Industrial parts are usually dirty and reflectivity can vary rapidly over the surface.

In order to obtain more information it is necessary to provide an additional direction of illumination. The arrangement shown in Figure 6 provides two nearly orthogonal directions. To see the relationship between the power due to each illumination direction consider (2) for each case. It is assumed that the direction of view is along the normal illumination direction.



a)

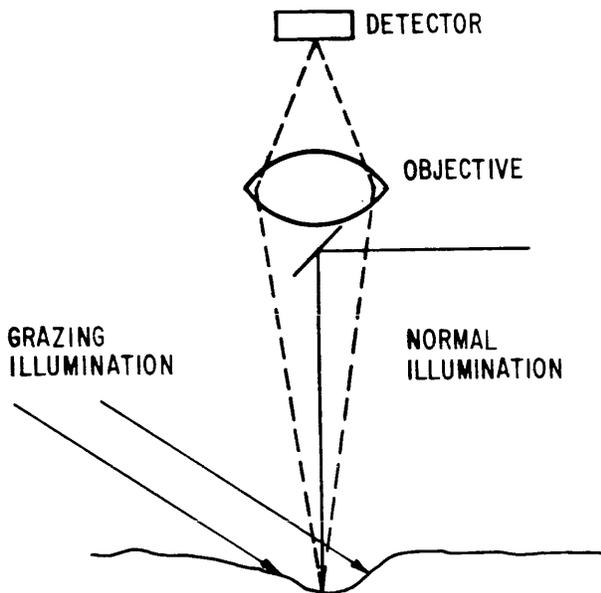
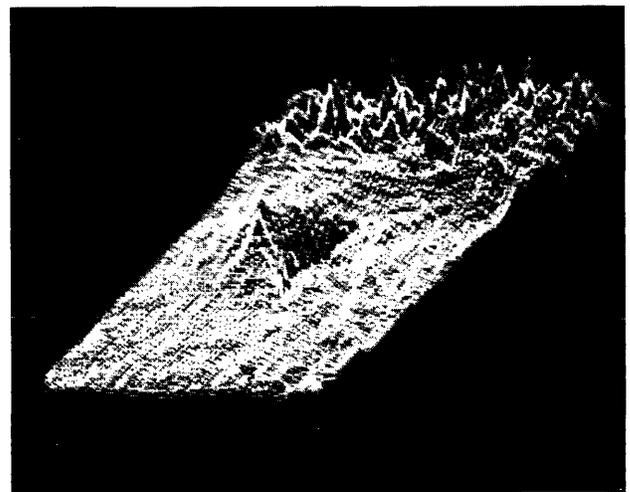


Figure 6—An optical configuration for obtaining two directions of illumination.



b)

Figure 7—(a) A photomicrograph of a surface nick and (b) the resulting S array shown in perspective. The signal variations near the top of the array are due to an unilluminated region.

a) Normal illumination

This is just the previous case, i.e.

$$I_n = E_n$$

$$G_n = I$$

$$P_n = \frac{\alpha}{4} \frac{1}{I_n^4} e^{-\beta(1/I_n^2)^{-1}}$$

b) Grazing illumination

$$G_n \approx 0$$

$$I_g = \sqrt{1 - E_g^2}$$

$$E_g = E_n = I_n$$

So in terms of I_n

$$P_g = \alpha \frac{1}{(\sqrt{1 - I_n^2} + I_n)^4} e^{-e(1/I_n + (1 - I_n^2)^{-1})}$$

If we assume that the surface normal does not vary greatly from the direction of normal illumination,

$$I_n \sim 1.$$

Thus

$$P_g \approx \frac{\alpha}{(I_n)^4} e^{-\beta(1/I_n^2)^{-1}}$$

Now taking the ratio of power due to grazing illumination and normal illumination we have,

$$\frac{P_g}{P_n} = 4e^{-\beta(1/I_n^2)^{-1}}$$

Taking the logarithm of this ratio and dropping subscripts we have

$$S = \ln \left(\frac{P_g}{P_n} \right) = \ln 4 - \beta/I_n^2 \tag{4}$$

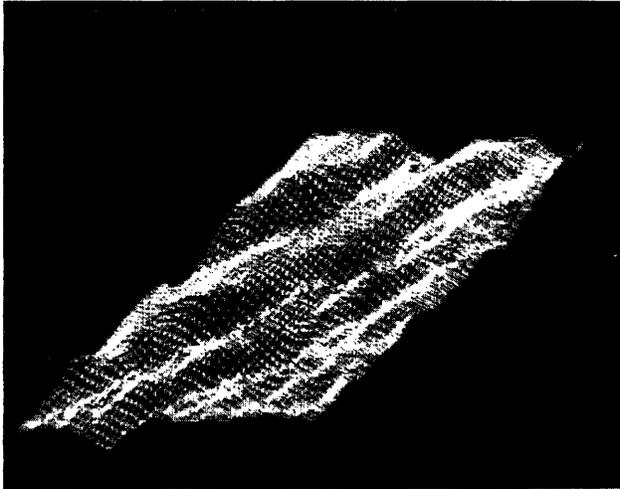


Figure 8b—The perspective of the corresponding S array.

Note that this quantity depends only on surface roughness and slope and not surface reflectivity. It is a reasonable assumption that roughness is constant over a region larger than the size of defects that are to be detected.

A number of surface defects on metal surfaces were imaged using the illumination scheme in Figure 6. Two images were obtained for each case, one for normal illumination, one with grazing light. The quantity S in Expression 4 was obtained by digitizing the images and performing the indicated calculations.

The first case consists of the nick shown in Figure 7a. The resulting S array is shown in perspective in Figure 7b. The nick appears near the center of the array with good contrast. The mountainous peaks near the top of the array were due to random sensor noise since that region was not illuminated.

A more striking example is the series of scratches (~.005" wide) shown in Figure 8a. The S array is shown in Figure

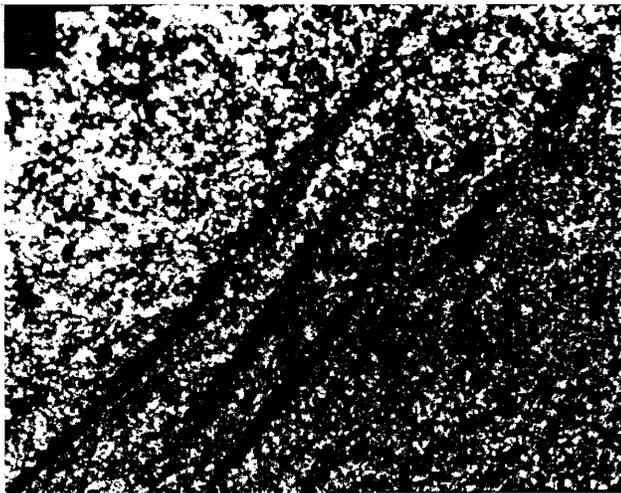
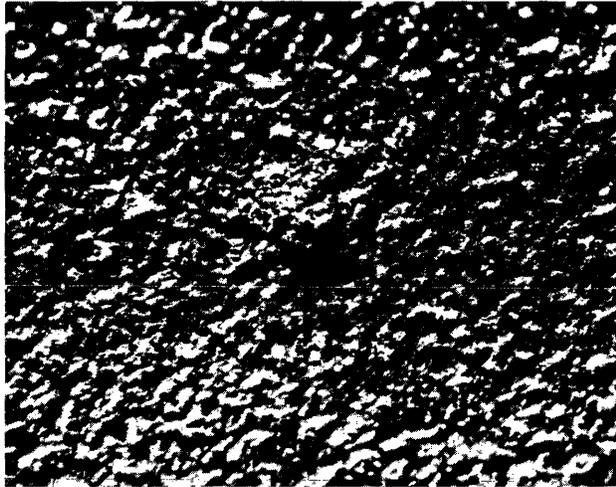


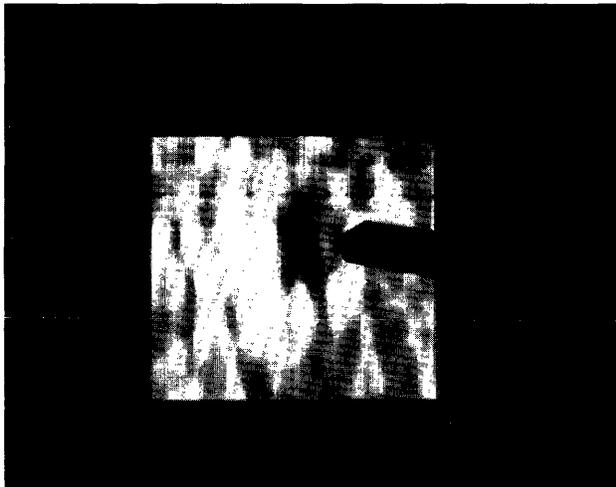
Figure 8a—A photomicrograph of a series of .005-inch-wide scratches. Note the large relative surface roughness.



Figure 8c—A binary image of the S array produced by thresholding.



a)



b)

Figure 9—(a) A photomicrograph of a small pit. (b) The resulting S array as a grey-level image to illustrate the contrast.

8b. The signal has enough contrast to allow a reasonable segmentation of the scratches by thresholding the S array as shown in Figure 8c.

As a final example consider the pit ($\sim .010''$) shown in Figure 9a. The resulting S array is shown as a grey-level image in Figure 9b. The main contribution to S in this case is the occlusion of the grazing illumination. There is good contrast between the pit and surrounding metal even though the surface roughness is on a scale comparable to the defect.

CONCLUSIONS

By considering the theory of scattering from rough surfaces, it has been possible to derive an illumination scheme and method of image analysis that results in good contrast and detectability of surface defects.

It is also noted that the scattering ratio, S, is sensitive to surface occlusions and shadowing. Thus we have a result that provides good contrast for most surface defects. The main drawback to the approach is the necessity to provide two directions of illumination.

ACKNOWLEDGMENTS

I am indebted to P. Beaudet for suggesting the relationship of Beckmann's result to Horn's notation. Also to T. Cipolla for his assistance in obtaining some of the experimental results.

REFERENCES

1. Mundy, J. L., and R. E. Joynson, "Automatic Visual Inspection Using Syntactic Analysis," *Proc. IEEE Conference on Pattern Recognition and Image Processing*, 1977.
2. Rosenfeld, A., *Picture Processing by Computer*, Academic Press, New York, 1969.
3. Weska, J., et al, "A Threshold Selection Technique," *IEEE Trans. on Comp.*, Vol. C-23, 1974.
4. Beckmann, P., and A. Spizzichino, "The Scattering of Electromagnetic Waves from Rough Surfaces," Pergamon Press, New York, 1968.
5. Winston, P., ed., *The Psychology of Computer Vision*, McGraw-Hill, New York, 1975.
6. Ibid, Beckmann.

Monitoring the earth's resources from space— Can you really identify crops by satellite?

by DAVID LANDGREBE

Purdue University
West Lafayette, Indiana

INTRODUCTION

Of the most important questions facing society today, those near the top of the list include the status and future of the world's food supply, its environmental quality and its sources of energy. These questions have increasingly been before the general public in recent years and have for an even longer period of time, been of concern to the world's thinkers, planners and policymakers. It had long been recognized that after all, our earth and its resources are finite and that as society continues to grow we must find better ways to manage these finite resources.

A primary need for good management is up-to-date and accurate information on the status of the resources to be managed. Thus it was that early in the last decade the possibility of using aerospace technology for accumulating better information about the current conditions of agriculture, the earth's resources and man's environment began to be examined. It seemed clear that to monitor resources directly associated with the land, the best vantage point would be above the land looking down upon it.

The need to be higher simply resulted from the need to see more. In practical terms this immediately implies the gathering of larger and larger quantities of data, and the computer was immediately suggested therefore, as somehow an important tool. But how? There now exists a first generation answer to this question. The program of systematic and step-by-step research which led to it, will first be briefly outlined. After description of some typical examples of its use, we will examine the directions being taken to devise an even more effective second generation solution.

THE FIRST LAYER OF TECHNOLOGY

Figure 1 shows a view of a small portion of the earth's surface as seen from space. This image was made from data from the Landsat-2 multi-spectral scanner. It is a simulated, color-infrared image of a region in north central Indiana, 100 nautical miles on a side. Such an image could only be imagined in the early 1960s as no one had seen it at that time. But even then the immensity of the problem was readily apparent. To place a sensor capable of producing such an

image appeared challenging but possible, but scenes such as this would present a real challenge to analyze by computer. The human interpreter can immediately recognize a major river generally flowing in a southwesterly direction across the scene and with a little additional information can identify forest lands in the river bottoms as compared to lands which have been prepared for crops in a great portion of the rest of the image. The more obvious features of a major city, Indianapolis, Indiana are apparent in the southeastern portion of the image; however, a number of smaller cities are barely even identifiable. To see how by computer analysis one would be able to answer such questions as "How many acres of wheat are there in this scene and what will their yield be?" or "Where are there serious problems of soil erosion, resulting not only in stream pollution but also loss of valuable agricultural land?" or "What is the extent and condition of various classes of urban land use and how are they changing?" surely represented a great challenge.

In addition, thinking of this scene in terms of the data volume to produce it and therefore the processing capability which would be needed to analyze it also revealed another challenge. This image consists of about 7.5 million digital pixels in each of four spectral bands. Each of these pixels may have any of 64 shades of gray. This amounts to 180 million bits for only one look at a 185×185 km (100×100 n. miles) area. Landsat collects such images at a rate of more than two a minute.

Such a situation seemed made to measure for the emerging field of pattern recognition to tackle. Figure 2 shows the now classical view of the organization of a pattern recognition device. A series of measurements are taken of the scene by the receptor, which in this case corresponds to the spaceborne sensor system, and these measurements are turned over to a classification algorithm. Realizing the great complexity of the scene it was recognized as especially important to devise a receptor which was very efficient in representing the needed scene characteristics in a manageable number of features.

For it to become possible to measure crop acreages in data such as shown in Figure 1 it was decided to rely for these features upon the spectral characteristics of individual pixels.¹ Figure 3 shows in conceptual form the spectral characteristics of three types of land cover. The features to be

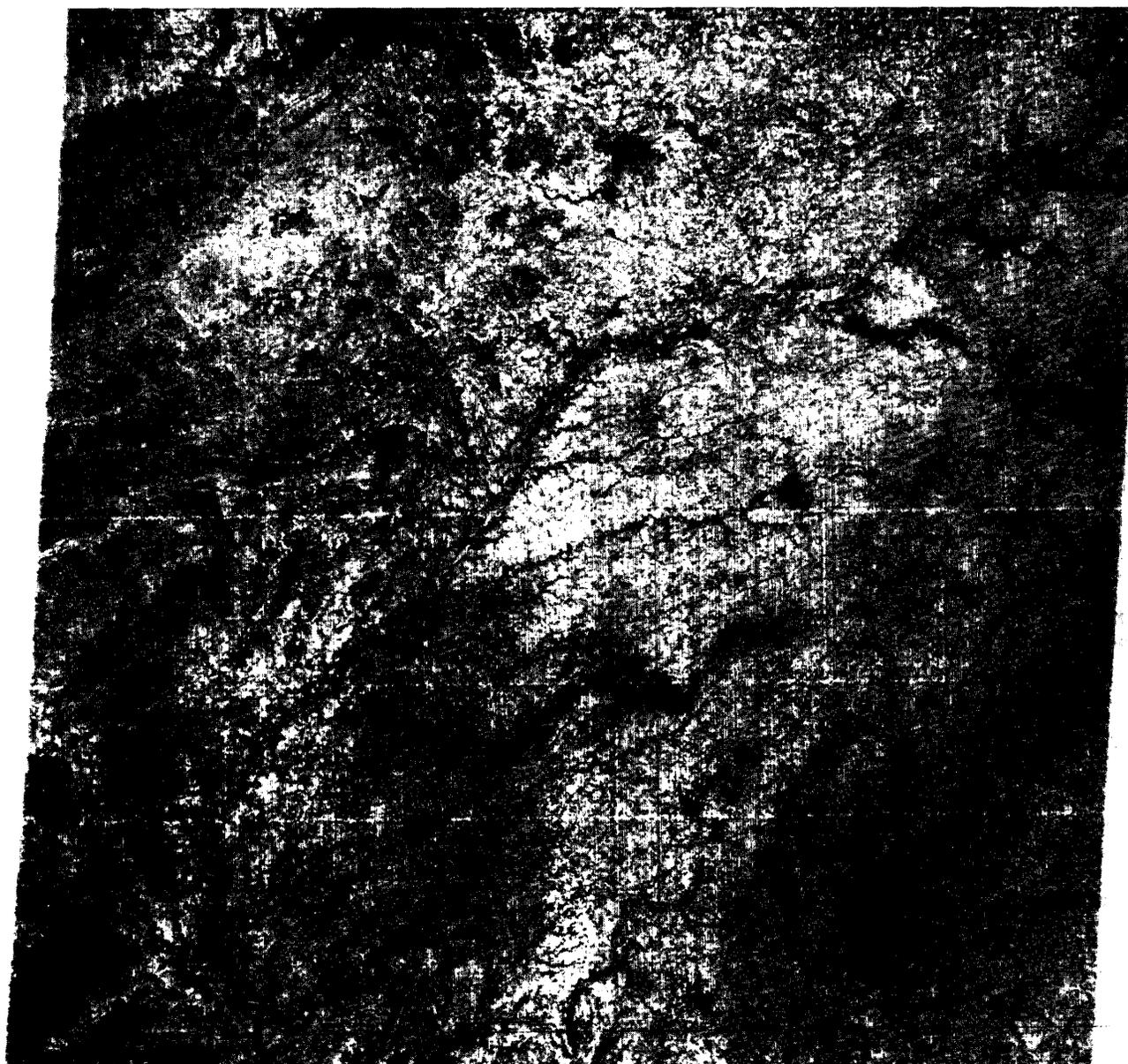


Figure 1—A scene of north central Indiana from the multispectral scanner of Landsat-2. (Original in color.)

measured would be the relative spectral response in each of the several wave bands. These responses would then be represented in a multivariate form as implied by the two-dimensional plot in Figure 3. The pattern classifier is trained by segmenting this multivariate space into an exhaustive list of nonoverlapping regions, each region corresponding to one informational class.

The choice of this spectral approach puts primary emphasis on the spectral characteristic of the measurement as compared to the image characteristics. For this reason a multispectral scanner, as opposed to a camera, was needed for the sensor so that the precision of the wave band energy measurement could be high and regions of the spectrum in

addition to the visible and the near infrared would be accessible.

The results of the first test of this concept are shown in Figure 4.^{2,3} On the left is an air photo of an agricultural area with the crop type present in each field indicated by a letter symbol. On the right is the result of using a maximum likelihood classifier on four bands of data in a pixel by pixel fashion. Based on this and similar early results obtained in 1966 there followed a period of two or three years of intensive research to define methods for data collection, pre-processing, classifier design, training procedures and the like.

By 1969 with the launch of the Apollo 9 spacecraft the

technology was ready for its first test on space data. The Landsat-1 spacecraft then referred to as ERTS-A, was already in the planning and design stages. Apollo IX carried aboard it an experiment known as S065. This experiment consisted of a bank of 4 boresited cameras with film and filter combinations to simulate the expected Landsat data. After exposure and developing of the film each of the 4 images from the different parts of the spectrum of the same scene were scanned then precisely registered to one another. This was accomplished by computer implemented precision image registration algorithms which had also been devised during the previous research period. Since these algorithms were capable of registration to subpixel accuracy, data simulating multispectral scanner data was available. Promising results were obtained from pattern recognition results in applications associated with agricultural crops, general land use and geologic recognizance mapping among others.⁴

A second major test of this emerging technology occurred in 1971 as a result of a major episodic event in the corn belt. Southern corn leaf blight emerged during the 1970 growing season, and though it arrived late enough in the season that devastating damage did not occur, it appeared that a catastrophe could occur during the 1971 season before new strains of seed corn which would be resistant to the blight could be produced. Thus during the winter of 1970-71 a major effort to monitor the corn crop during the 1971 season was planned. The effort involved overflight of some 220 1×10 mile segments located over the seven states of the corn belt on a bi-weekly basis. All segments were overflown by a high altitude aircraft carrying photographic cameras; 30 of the segments in western Indiana were also overflown by a low altitude aircraft carrying a multispectral scanner. The results showed that not only could crop species be distinguished accurately but that the condition of a single crop (corn) could be successfully subdivided into subcategories based on the degree of blight infestation.⁵ This proved to be true using both the more well established manual image interpretation methods and the newer computer implemented mul-

tispectral techniques. In the latter case however, the results were demonstrably more precise and objective.

Thus by the time of the launch of Landsat-1 in July 1972 a new technology had been well researched and tested and was ready for exposure to the potential user community. This exposure began with the studies of some more than 400 principal investigators in the U.S. and around the world who for the first few years of Landsat-1's life tested out both image-oriented and computer-implemented algorithms for extracting information from earth-orbital views of the earth.⁶⁻⁸ Since that time Landsat-2 and Landsat-3, launched in January 1975 and March 1978 respectively with nearly identical sensor systems to that of Landsat-1, have been used. Landsat is now an essentially operational system in that data is routinely collected and placed in the public domain. It is routinely available to anyone from any nation. Table I gives the location of Landsat ground stations in addition to the three operated by the U.S.

The technology represented by this satellite series and computer-aided processing techniques associated with it is indeed now quite broad touching essentially all application fields which deal with land resources.⁹⁻¹¹ Machine-implemented techniques for processing the data for improvement of its geometric properties to high cartographic standards are now not uncommon. The registration of two frames of data of the same scene collected at different times is now also being accomplished at a number of locations and to pixel and in some cases subpixel accuracies. A large number of pattern recognition algorithms implemented on both special purpose and general purpose hardware are being used routinely and are commercially available.

EXAMPLE APPLICATIONS

The precise definition of operational is an elusive one, however clearly routine use of these methods is now beginning to occur in the fields of agricultural crops and soils

PATTERN RECOGNITION: Scheme for Multivariant Analysis of Physically Observable Data

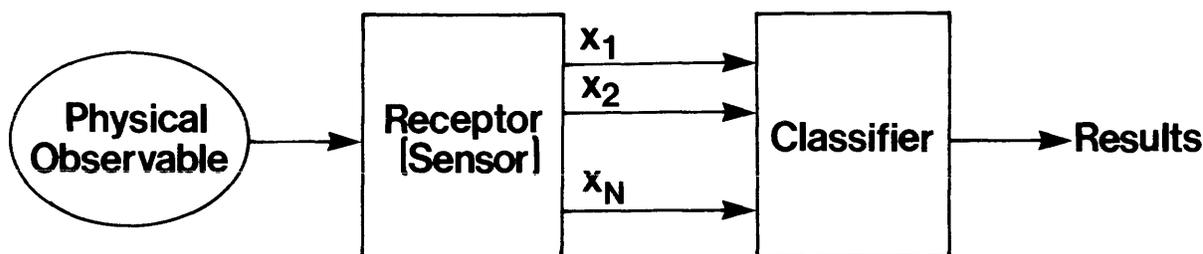


Figure 2—The organization of a pattern recognition device.

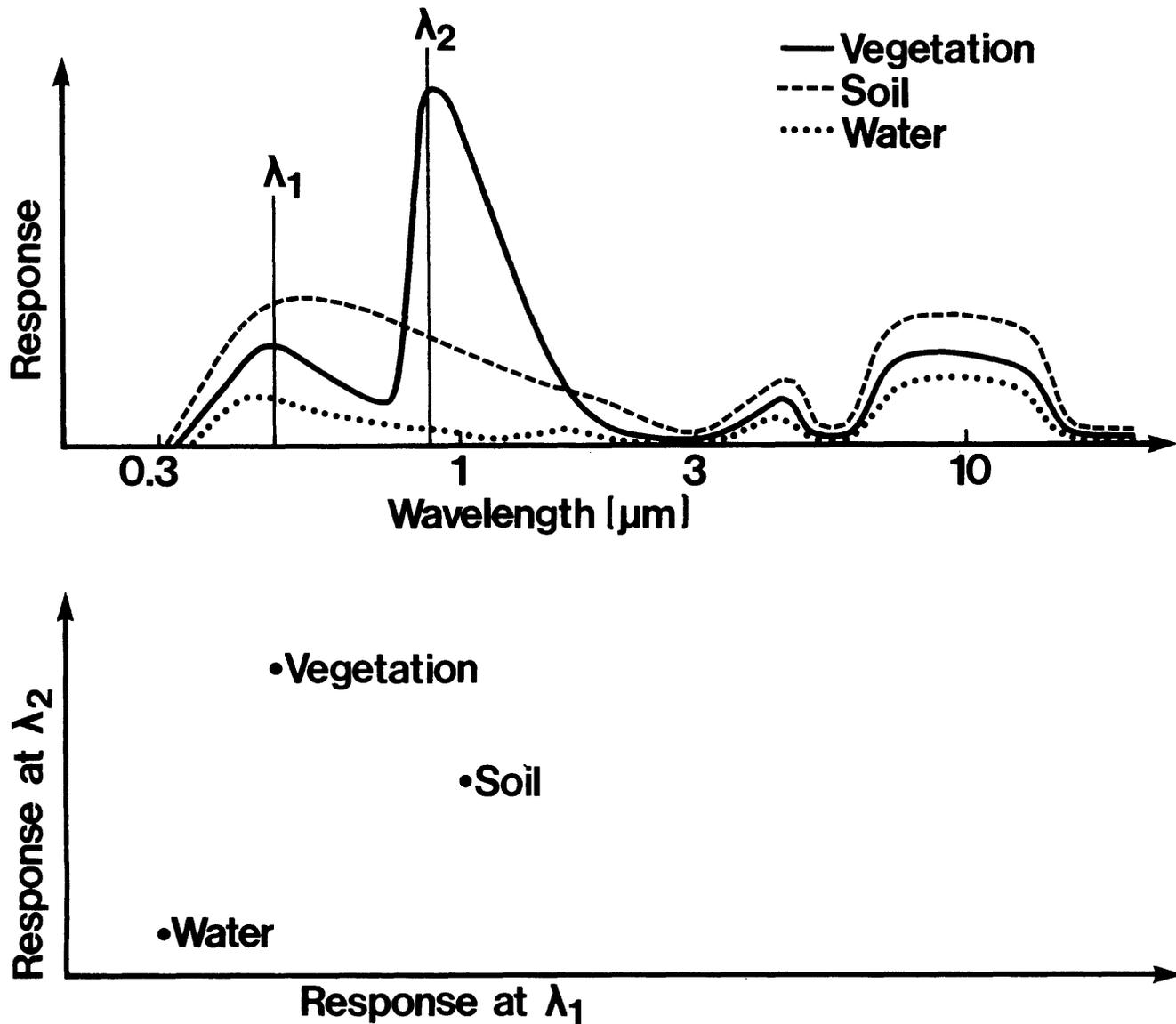


Figure 3—The representation of spectral characteristics in two-dimensional space.

TABLE I.—Location of Non-U.S. Landsat Ground Stations

Country	Date Station Began Receiving Data
Argentina	1980*
Australia	1980*
Brazil	1973
Canada	
Prince Albert	1972
Shoe Cove	1977
India	1979*
Iran	1978*
Italy (ESA)	1975
Japan	1978*
Sweden (ESA)	1978*

* Anticipated date. Source - NASA/Goddard Space Flight Center, June 1, 1978.

mapping, forest cover and condition assessment, geology and mineral prospecting, rangeland assessment, general land cover mapping especially associated with the mapping of the urban and near urban regions. We will discuss only a few examples to illustrate this technology, especially that portion with it associated with computer processing.

Land Cover Mapping Examples. One of the earlier large scale applications of Landsat data came about as a result of the need to remedy problems of pollution of the Great Lakes. The International Joint Commission of the U.S. and Canada set about to determine the precise causes of the problem. It established several reference groups associated with sources of the pollution, one of which was the reference group on land use. This reference group was concerned with the introduction of undesirable materials into the Great Lakes which enter them through the various streams of the

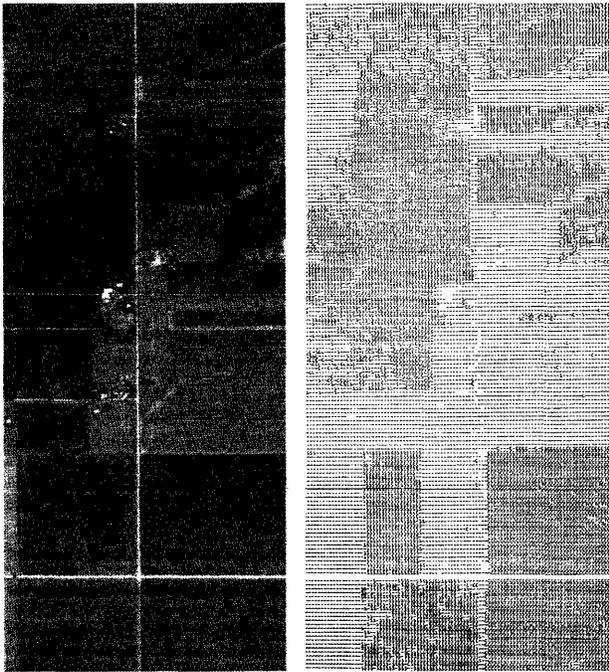


Figure 4—An early result of crop species classification from multispectral data. (Original in color.)

watershed and which result from the manner in which the land of the Great Lakes watershed is being utilized. To begin this work it was necessary to study the relationship of the land use to the possible pollutants which might be carried by these streams. It was immediately found that no up-to-date land use maps of the region existed. Thus the first step would be the construction of such maps. More specifically land use maps of each of the 192 counties were needed on a county-by-county basis. It was desired to know the type of land use in each four- to five-acre plot of counties. Furthermore, while the requirements on map classification accuracy were modest, the time available was short being less than one year. Such maps were produced in 1973 at a cost of approximately \$1,000 per county. The processing required involved selection of the proper Landsat frame, geometric correction of the data to reasonable cartographic precision, and delineation in the data of the county boundaries. This was followed by derivation of appropriate classifier training statistics and classification of the county using a maximum

likelihood Gaussian classifier. Maps were then produced in a color coded form showing the various land cover classes. Place names and boundaries were later overlaid onto the maps using conventional (non-computer) techniques and the intention was to interpret the required land use knowledge from these land cover maps by conventional means. Tables giving the aerial proportion of each county in each land cover class were also produced.

Work has continued since that time to further refine the techniques and adapt them to the needs of user agencies. In 1978 the U.S. Geologic Survey produced and published a land use map of the Washington, D.C. urban area using essentially these same techniques.¹² This map which is for sale by the U.S. Geologic Survey at Reston, Virginia (map I-858-E) is to a scale of 1:100,000; it contains a color coding showing the classes given in Table II. Supplied along with this map are quantitative tabulations of land use classes in each 5×5 km UTM grid cell.

An even more detailed land use map has been prepared and published as a demonstration product by the U.S. Department of Interior, the Pacific Northwest Regional Commission and the National Aeronautics and Space Administration.¹³ This map of the land cover type of the Puget Sound Region in the state of Washington is at a scale of 1:250,000 and shows 21 land cover types as listed in Table III.

An Example in Forestry. The field of forest lands management is another important application area; to illustrate the use of earth observational data in this field we will consider briefly the management of commercial forest lands. In this case the need is for (1) complete, accurate, and current data as a basis for both immediate and long range management planning, (2) information within the time span required for the management decisions, and (3) augmentation of the difficult task of data collection by increasing the efficiency of ground sampling procedures while maintaining the scope and precision of the data collected.

Active forest management practices were begun in the United States at the turn of the century but have become greatly intensified in the post World War II years. They have in this period become increasingly quantitative in nature and quite sophisticated. The problem is one of choosing the correct management actions to take on forest lands which may total millions of acres but which may require different management decisions on tracks as small as a few tens or hundreds of acres. Management decisions required are both of a technical nature (plantation and fertilization schemes, harvesting schedules, etc.) and an administrative

TABLE II.—Land Use Classes of Computer-Derived U.S.G.S. Map of Washington, D.C. Urban Area (and number of spectral classes)¹²

Build Up Area	Transitional	Open Space
Commercial, industrial, & services (3) —includes bare rock	Disturbed land (3), cover in transition —includes extractive industry	Agriculture (3) Unimproved open space (3) —includes forest land and brush land
Parking, paved surfaces (2) —includes special runway	Improved open space (1) —includes golf course, cemetery, grass	Clouds (1) Cloud shadows (1) —includes some wetland
Residential, older (1) —with maturing landscape		Water (3)
Residential, newer (5) —with less mature landscaping		

TABLE III.—Land Cover Classes Discernable from Landsat MSS Data in a Mapping of the Puget Sound Region¹³

Residential	Clear Cut	Water
Wooded Residential	Brush	Turbid Water
Mobile Homes	Deciduous	Wetland
Commerical/Industrial	Mixed Forest	Barren
Pavement	Immature Conifer	Ice/Glacial Debris
Crop Land	Second Growth Conifer	Snow
Grass/Pasture	Old Growth Conifer	Shadow

nature (decisions on buying, selling or leasing of lands, contracting for timber, etc.). Time intervals between decisions and results may be as long as 30 to 40 years and as short as a few days or weeks.

Variables to be monitored include (1) quantitative timber values in terms of commercial units of value (cubic feet, board feet, tons, etc.), (2) stand structure and condition including stand composition by species, cover type, timber size and condition, age classes, density, topographic position, competing vegetation, past cultural activity, etc. and (3) the dynamic response of the stand in time, both to natural and cultural factors. A manager must know not only the conditions at the present time but how well and in what fashion a stand has responded to past management practices and natural environmental conditions.

Traditionally, data for forest management has been acquired by manual ground observations. This can, of course, be a very expensive (and disagreeable) process to carry out. Sites must be suitably located over the land to be managed according to a suitably designed statistically sampling plan. These sites have to be visited by ground sampling teams periodically. The work may be slow and difficult to carry out because of terrain difficulties. It was natural in this case for the field to turn to remote sensing techniques, first from aircraft and now spacecraft, as a means to increase the efficiency of data collection, to increase its objectivity, and to realize its availability on a more timely basis.

Even if it were not possible from remote observations to increase the accuracy of data collection, one would anticipate that it should be possible to improve the timeliness and especially the efficiency of ground data collection. For example, the number of ground sites needed in a sampling scheme depends on how well they are located relative to the stratification of parameters to be measured. The synoptic view from space provides a very effective vantage point for sensing this scene stratification and therefore can lead to improved efficiency in dispatching ground teams.

The techniques of machine processing of Landsat MSS data have been shown over the last few years to very clearly be capable of providing the needed stratification and identification capabilities in an objective and timely manner. Even in the rugged terrain of the U.S. Rocky Mountain region where the high degree of terrain relief causes variations in illumination due to shadowing and slope angle effects, it has been possible to convincingly demonstrate the ability to distinguish between deciduous and coniferous stands and to some extent to distinguish between species or species groupings.¹⁴⁻¹⁶ These capabilities are precisely the

ones needed to more efficiently assign ground observations teams and to provide accurate aerial determinations for use in multistage statistical sampling procedures. These techniques have been and are being adapted very rapidly to the specific requirements of forest information systems operated over both public and private land holdings. The St. Regis paper company, Southern Timberlands Division, for example has a very active program at the present time to incorporate such Landsat techniques for managing its 1.7 million acres of land owned or controlled in the states of Florida, Georgia, Alabama, Mississippi, and Louisiana. St. Regis had some time ago seen the need for objective management schemes and had devised a number of computer implemented management models. The development of these models gave the corporation the ability to rapidly make large numbers of decisions of a forest lands management nature. This, in turn, very greatly increased the need for input data on quantitative timber values, stand structure and condition and dynamic temporal response, the very variables which the Landsat technology can readily produce. This situation is not atypical of that which the manager of both commercial and public lands finds himself in today.

A Food Commodity Production Forecast Example. As a third example of the application of computer technology in remote sensing we will cite an experiment known as LACIE, the Large Area Crop Inventory Experiment.¹⁷ LACIE was a proof of concept experiment conducted between 1974 and 1978 by NASA, USDA, and NOAA in which the major wheat production areas of the world were monitored by Landsat in order to obtain continual estimates of the acreage, yield, and production of wheat. The goal of the program so far as accuracy is concerned was that the at-harvest estimates were to be within 10 percent of the true estimate at the national level nine years out of ten. This goal, if achieved in an operational system, would be a significant improvement in the current ability to be able to estimate at-harvest production. An important feature of LACIE for the application of existing remote sensing technology was a self-imposed constraint against the use of observations from the ground to identify wheat. This restriction was imposed on all geographic areas, both the U.S. and foreign, to insure the development of a technology applicable to regions inaccessible to observations from the ground.

The procedure used involved a statistical sampling scheme in which 5×6 mile segments located in each subregion were used to estimate the wheat acreage in that subregion. Weather station reports of the subregion were used with a yield regression model to estimate the subregion yield. The total wheat production for each subregion is then obtained as the product of the available wheat hectareage times the yield for that subregion. The production forecasts for all subregions are then summed to obtain the national level forecast.

During the course of the LACIE program about 15,000 such 5×6 mile data sets from more than 2600 sample segments in five major global crop regions and meteorological data from more than 1500 reporting stations were used. During at least one growing season of the program, monthly reports on area, yield, and production estimates on wheat

were generated for the U.S. Great Plains, Canada, and the U.S.S.R. Exploratory analysis was carried out for segments in five other countries; Argentina, Australia, Brazil, India, The Peoples Republic of China.¹⁷

Based upon a very extensive effort to measure, evaluate, and report the results of LACIE, it is clear that the ability to monitor wheat production by multispectral means has been demonstrated. Figure 5 shows an example LACIE result for the U.S.S.R.¹⁷ The solid curve of this figure shows the monthly estimate reported by the project. These production forecasts were generated and released to USDA in Washington, D.C. the day prior to the corresponding public release by the USDA Foreign Agricultural Service (FAS).

The FAS forecasts are also shown for comparison. The third variable graphed in the figure are LACIE recomputed estimates. The recomputed estimates are the seasonal forecasts obtained from the LACIE system after correction of two Landsat data problems encountered during that year of operation; these problems were of a type typical of first time system operation and would not be expected to occur in an operational system. Also indicated in this figure is the 1977 production figure of 92 million metric tons officially released by the U.S.S.R. in January 1978. The ability of this system to produce an accurate estimate but also to produce an estimate of this accuracy so early in the season is indeed impressive.

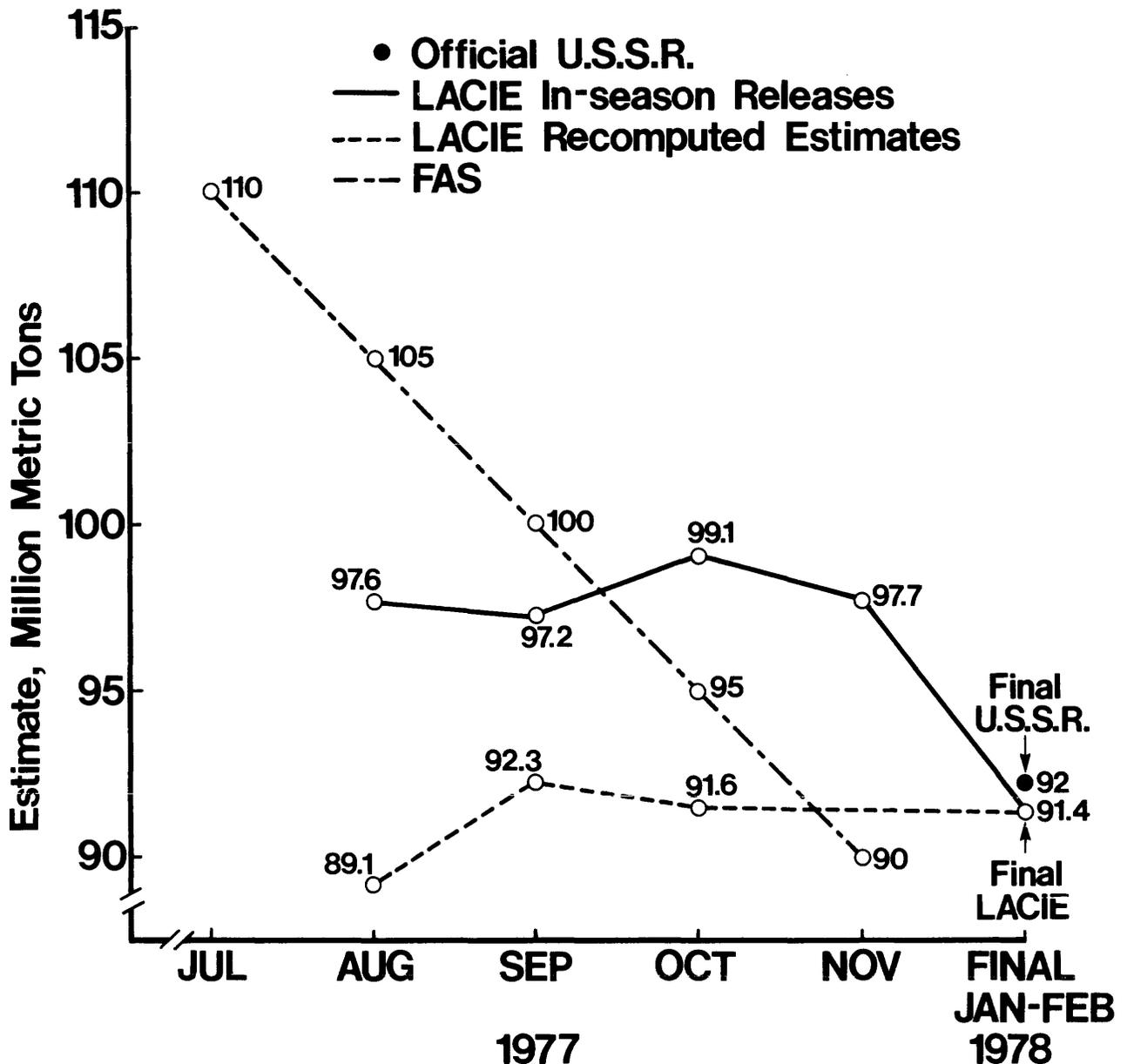


Figure 5—LACIE Phase III USSR total wheat production results for 1977 compared to FAS and official USSR estimates.¹⁷

The need for such data grows stronger as the degree of international interdependence increases. Though this interdependence has always been growing the growth has accelerated greatly in the last few years. For example in 1950-52 the U.S. Balance of Trade in the agricultural area was at approximately minus one billion dollars. By 1960-62 the figure was approximately plus one billion dollars. By 1970 the figure had increased to approximately 1.5 billion and by 1975 the same figure exceeded 12½ billion. Clearly these figures show the increasing importance to the U.S. of early and accurate estimates not only of the projected U.S. production but that of its trading partners around the world as well.

These three examples were selected not only to sample the spectrum of different uses as related to different disciplines but also to illustrate the differing types of information needs which the technology is now able to supply. In the case of the land cover mapping example the need was for highly accurate categorization of land cover type displayed in a map format of high cartographic quality; ground observations were readily obtainable. In the second example the information required was not in map but statistical form. In addition in this case the remote observations were to be used to reduce the amount of ground observations which had been previously necessary. In the third case again statistical information rather than map output was required and no ground observations were possible at all. In all three cases an improvement in an existing data source was the objective rather than the production of a wholly new type of data.

RESEARCH DIRECTIONS FOR A SECOND LAYER OF TECHNOLOGY

These techniques are flowing rapidly into routine use in a series of application areas which is so large and diversified as to defy listing. Given the continued availability of data of the type now being returned to the earth, the technology seems destined to mature to a position of considerable importance. However, in addition to this maturing of the current technology significant new advances are in the offing. A new series of Landsat satellites is now under construction. The first will be launched in 1981. In addition to the current type multispectral scanner (MSS) it will carry a more advanced scanner known as thematic mapper (TM). Table IV provides a comparison of the specifications of thematic mapper as compared to MSS. It is seen that thematic mapper will have a larger number of bands of greater spectral resolution covering not only the visible and near IR but the middle IR and thermal region as well. Note also the significant improvement in spatial resolution and the signal-to-noise ratio of the data as reflected by the 8 bit data providing 256 shades of gray per band as compared to the 6 bit data of MSS providing 64 shades of gray. The data thus provided will make possible deeper penetration into the hierarchy of classes of earth cover. For example, as compared to the identification of wheat using MSS in the LACIE experiment the thematic mapper should make possible crop species

TABLE IV.—A Comparison of some Parameters of MSS, the Multispectral Scanner Aboard the Early Landsat Satellites with Thematic Mapper (TM) to Be Aboard Landsat D in 1981

Scanner Parameter	MSS	TM
Spectral Bands	4: .50- .60 μm	7: .45- .52 μm
	.60- .70 μm	.52- .60 μm
	.70- .80 μm	.63- .69 μm
	.80-1.1 μm	.76- .90 μm
		1.55-1.75 μm
		2.08-2.35 μm
		10.4 -12.5 μm
Instantaneous Spatial Field of View	80 m	30 m
Data System Precision	6 bit	8 bit

identification and condition estimation approaching that achieved with the airborne scanner used in the 1971 corn blight watch experiment. In this latter case, corn, soybeans, and other crops typical of the U.S. Corn Belt were identified with high accuracy throughout the growing season.

A second area of progress in the research laboratory which is leading to an advancement in applications capability is in the use of various types of ancillary data geographically associated with Landsat data. Data bases constructed by registering Landsat observations at different times through the season together with such variables as elevation, slope, aspect, radar return, political boundaries, land ownership data, soil type, etc. will significantly widen the number of applications by greatly increasing the number of land cover classes to which the data can be divided into.

On the other hand, the construction of such new data sets will greatly increase the amount of stored data which must be accessible and will place increased demands upon analysis procedures of higher and higher sophistication. It is in these areas techniques associated with the construction, storage, and accessing of more complex data sets and the creation of more sophisticated analysis procedures capable of achieving the full advantage which such data sets provide that an enhanced research effort is needed at the present time.

Though it is usually the case that advancements are first made in data collection, in this field it is most important that a balance be achieved in the development of the state-of-the-science of data collection and information extraction. These in turn require a continued effort to increase our understanding of the scene and just which attributes in the scene are information bearing and also in the problems of the user community and which techniques are needed to interface the information extraction techniques of this emerging technology with bonafide user needs.

REFERENCES

1. Swain, P. H., and S. M. Davis, *Remote Sensing: The Quantitative Approach*, McGraw Hill International Book Co., 1978.
2. Landgrebe, D. A., and Staff, "Automatic Identification and Classification of Wheat by Remote Sensing," Purdue University Agricultural Experiment Station Research Progress Report 279, LARS Information Note 021567, 1967.

3. Fu, K. S., D. A. Landgrebe, and T. L. Phillips, "Information Processing of Remotely Sensed Agricultural Data," *Proceedings of the IEEE*, Vol. 57, No. 4, April 1969, pp. 639-653.
4. Anuta, P. E., and R. B. MacDonald, "Crop Surveys from Multiband Satellite Photography Using Digital Techniques," *Remote Sensing of the Environment*, Vol. 2, No. 1, 1971, pp. 53-67.
5. MacDonald, R. B., et al, "Results of the 1971 Corn Blight Watch Experiment," *Proceedings of the Eighth International Symposium on Remote Sensing of the Environment*, The University of Michigan, Ann Arbor, Michigan, Vol. 1, 1972, pp. 157-189.
6. *Proceedings of the Earth Resources Technology Satellite-1 Symposium*, NASA/Goddard Space Flight Center, Greenbelt, Maryland, X-650-73-10, September 29, 1972.
7. *Proceedings of the Symposium on Significant Results Obtained from ERTS-1*, NASA/Goddard Space Flight Center, Greenbelt, Maryland, NASA Special Publication SP-327, March 5-9, 1973.
8. *Proceedings of the Third Earth Resources Technology Satellite-1 Symposium*, Vol. 1, NASA Special Publication SP-351, Vol. 2, NASA Special Publication SP-356, Vol. 3 NASA Special Publication SP-357, NASA/Goddard Space Flight Center, Greenbelt, Maryland, December 10-14, 1973.
9. Lintz, J. Jr., and D. S. Simonett, eds., *Remote Sensing of the Environment*, Addison-Wesley Publishing Company, Inc., 1976.
10. Sabins, F. F., Jr., *Remote Sensing Principals and Interpretation*, W. H. Freeman and Company, 1978.
11. Reeves, R. G., *Manual of Remote Sensing*, American Society of Photogrammetry, Falls Church, VA, 1975.
12. Gaydos, L., and J. R. Wray, Land Cover Map from Landsat 1973, with Place Names, (Map I-858-E) and with Census Tracts (Map I-858-F), 1978, for sale by U.S. Geological Survey, Reston, VA 22092.
13. Gaydos, L., and W. L. Newland, "Inventory of Land Use and Land Cover of the Puget Sound Region Using Landsat Digital Data," *Journal of Research of the U.S. Geological Survey*, Vol. 6, No. 6, Nov-Dec 1978, pp. 807-814.
14. Hoffer, R. M., and Staff, "Natural Resource Mapping in Mountainous Terrain by Computer Analysis of ERTS-1 Satellite Data," *Agricultural Experiment Station Research Bulletin 919*, Purdue University, West Lafayette, Indiana, 1975, 124 pp.
15. Dodge, A. G., and E. S. Bryant, "Forest Type Mapping with Satellite Data," *Journal Forestry*, Vol. 74, No. 8, 1976, pp. 526-531.
16. Williams, D. L., and G. F. Haver, "Forest Land Management by Satellite: Landsat-Derived Information as Input to a Forest Inventory System," Interlab Project #75-1, NASA Goddard Space Flight Center, Silver Spring, Maryland, 1976, 36 pp.
17. MacDonald, R. B. and F. G. Hall, "LACIE: An Experiment in Global Crop Forecasting," *Proceedings of the LACIE Symposium*, NASA Johnson Space Center, Houston, Texas, October 1978, Document No. JSC-14551.

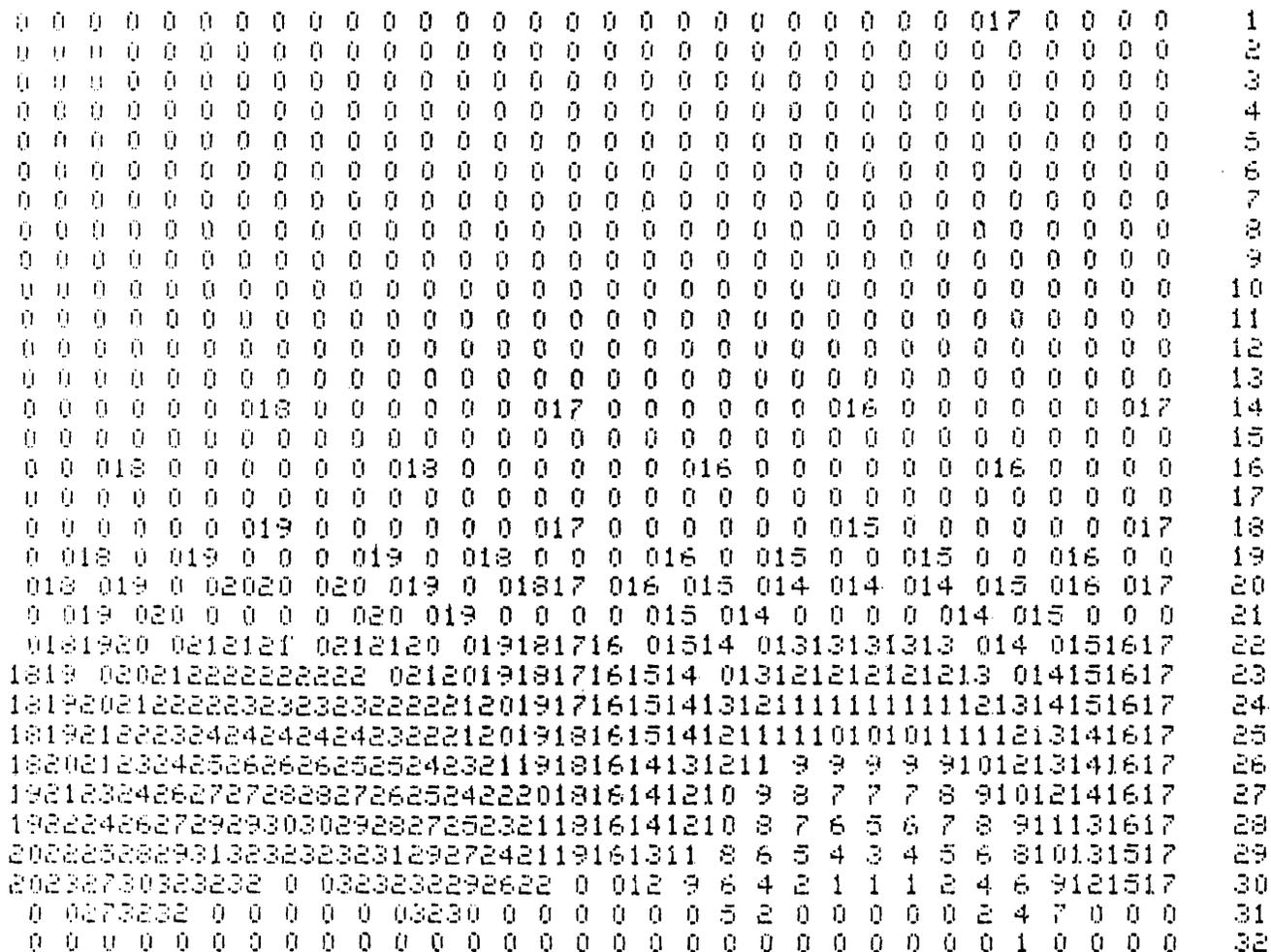


Figure 2

- Shape—the distribution of gray level values of the object.

Often, only the shape is relevant, i.e., a match is sought between the test image and the reference image even if they differ in translation, orientation and scale. It is desirable in these cases to transform the image pair in ways that are independent of these irrelevant geometric features.

Shape detection has been a topic of interest to the digital image pattern recognition community for many years,⁸ and it remains a prominent topic of discussion in the current literature.^{1,2,5} Two recent survey articles^{3,4} document the popularity of the topic.

The Power Spectrum of an image is well known to be independent of translation. The Mellin Transform has been shown⁹ to be scale-independent. The Polar-Cartesian (POLCAR) Transform¹⁰ described in the following, converts rotation into translation. The Power Spectrum of the POLCAR Transform is then independent of rotation. Hence, a combination of these performed successively will allow shape to

be matched to shape, independent of translation, rotation and scale. Mention of this approach is found in the literature:^{6,7} this paper explores this approach in more detail. Listings of the FORTRAN programs used in this study are available from the author on request.

POLCAR TRANSFORM (32x32)

In a NXN image, let I be the row index and J be the column index with (I, J)=(1, 1) in the upper left hand corner. Let

$$S=(N/2)+1.01 \tag{1}$$

$$X=J-S \tag{2}$$

$$Y=-I+S \tag{3}$$

$$R=\text{SQRT}(X \times X + Y \times Y) + .1 \tag{4}$$

$$T=\text{arctan}(Y/X) + 3.14159 \tag{5}$$

Then for $N=32$, the X axis is originated at 17.01 and is positive to the right while the Y axis is originated at 17.01 and is positive upward in the I, J system. R ranges, for integer I, J, between 0.11 and 22.74 while T ranges between 0 and 6.2832. Let

$$IR = \text{INT}(\text{LOG}(R) * 13.155 + 14.2) \quad (6)$$

$$ITH = \text{INT}(T * 5. + 1.) \quad (7)$$

then IR is an index ranging between 1 and 32 and ITH is an index ranging between 1 and 32. Let IR be a row index and ITH be a column index with $(IR, ITH) \doteq (1, 1)$ in the upper left corner of a 32×32 array. This array is termed the POLCAR transform of the input array: $G(IR, ITH) = G(I, J)$.

An input array of horizontal lines is shown in Figure 1; the geometric features of the POLCAR Transform of Figure 1 is shown in Figure 2. Figure 3 shows the isolines of Figure 2. Figure 4 shows the full POLCAR Transform of Figure 1, including gray level amplification as a function of R squared.

Asterisks indicate format overflow for values greater than 99.

A set of vertical lines are shown in Figure 5; the geometric features of the POLCAR Transform of Figure 5 are shown in Figure 6 with the isolines drawn in in Figure 7. Note that Figure 7 resembles Figure 3, shifted eight spaces to the right corresponding to a 90-degree rotation of the input array.

Figure 8 shows the array of IR values as a function of I, J; Figure 9 shows the isolines of Figure 8. The array of ITH values as a function of I, J is shown in Figure 10. The isolines of Figure 10 are shown in Figure 11.

The FORTRAN subroutine implementing the POLCAR Transform is given in Table I. It is intended that the input array, IMAG, is the Power Spectrum of a 32×32 image, rearranged to emulate an optical power spectrum so that the zero frequency component is position at (17, 17). Setting, the origin at (17.01, 17.01) assures that R is strictly positive.

The form of IR is

$$IR = H \log R + K.$$

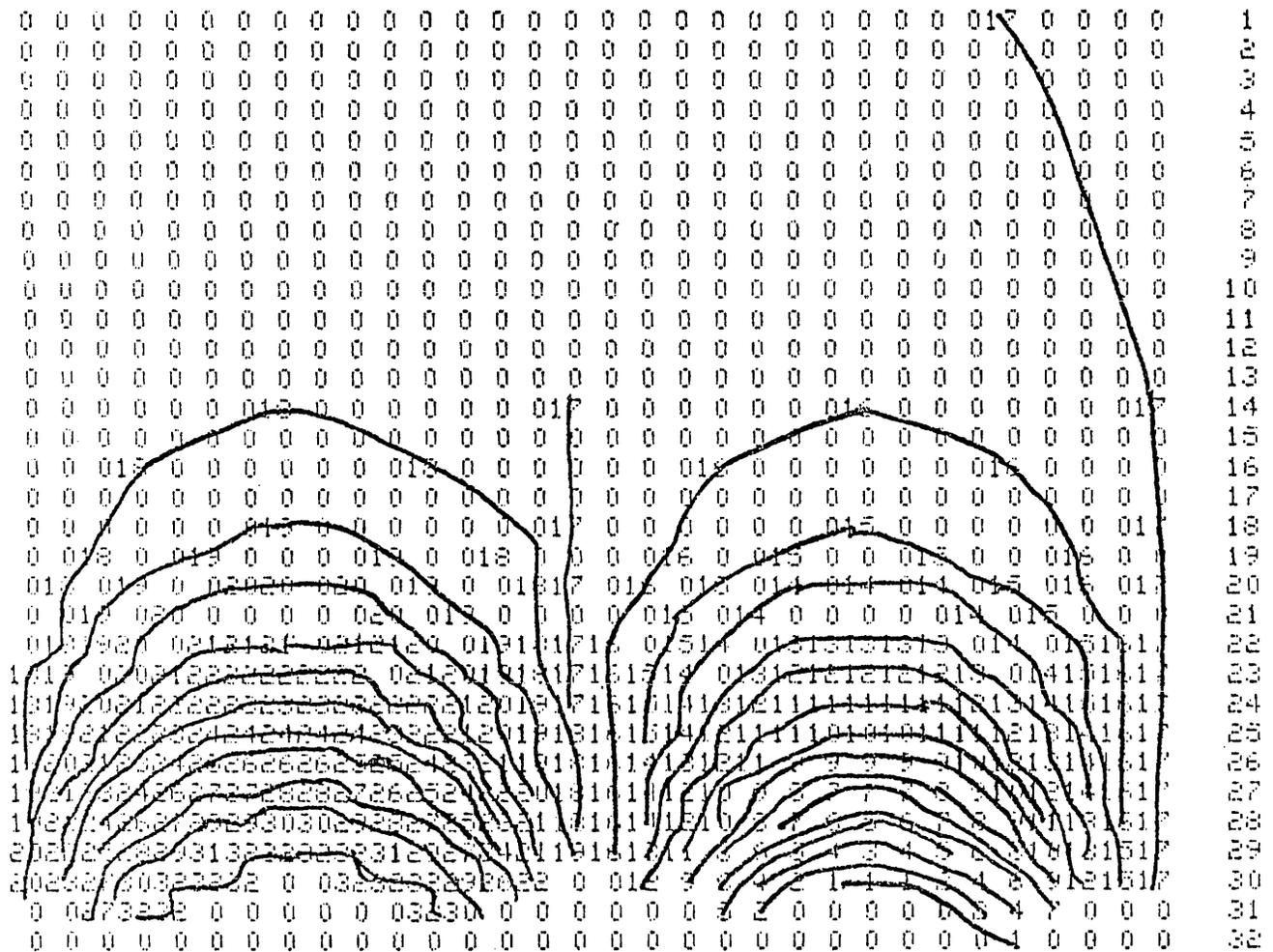


Figure 3

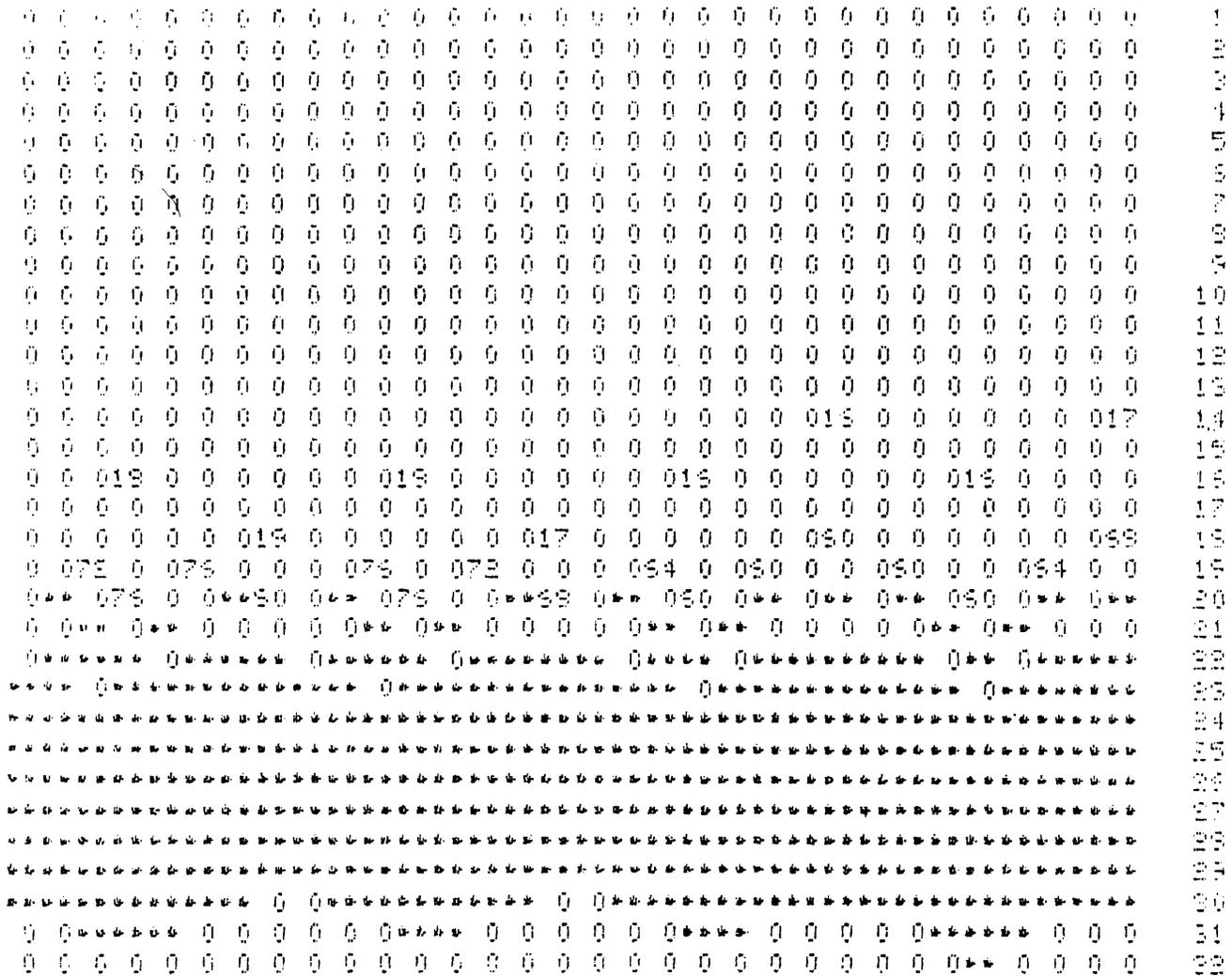


Figure 4

H and K are chosen so that the minimum value of IR is 1 and the maximum value of IR is 32. The log is employed to provide the scale insensitivity of the Mellin Transform.

The gray-level values are amplified by a factor of R squared since in the frequency domain this is comparable to taking a second derivative of the spatial domain input image.

SHAPE DETECTION ALGORITHM

A block diagram of a shape detection algorithm incorporating the POLCAR Transform is shown in Table II. Two images, I1 and I2, are input. Their Power Spectra are obtained using a Fast Fourier Transform subroutine (FFT). The output of FFT has the zero frequency component at (1, 1); by analogy with optical power spectra, a rearrangement of the frequency components with the zero frequency component at (17, 17) is desired. The output, aside from various intermediate displays, is a value of QMAX(I1, I2). This is used to obtain M(I1, I2).

DISCUSSION

Sixteen binary 32x32 arrays containing simple objects were studied. There are essentially six types of objects categorized by shape in the set of test images:

- A. Solid rectangles, 2.6:1 aspect ratio; I1, I2, I5, I6
- B. Solid right isosceles triangles; I3, I4, I10
- C. Solid rectangles, 3.0:1 aspect ratio; I7, I8, I9
- D. Solid circles; I11, I12
- E. Hollow right isosceles triangles; I13, I14
- F. Hollow rectangles, 3.0:1 aspect ratio; I15, I16

The values of M obtained by comparing various image pairs using the SHAPE program are shown in Table III. High values indicate a good match, low values, a poor match.

I6 is simply I2 with gray values multiplied by 4; the M values in Table III for I6 as expected are virtually identical

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	2
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	3
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	4
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	5
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	6
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	7
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	8
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	9
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	10
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	11
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	12
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	13
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	15
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	17
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	18
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	19
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	20
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	21
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	22
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	23
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	24
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	25
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	26
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	27
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	28
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	29
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	30
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	31
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	32

Figure 5

to those for I2. Hence, I6 need not be analyzed as a distinct object for these purposes. For the 15 distinct objects, 105 cross comparisons are possible. It is expected that objects with shapes of the same type should have high M values. Naturally, all objects have somewhat the same shape as all other objects. Hence, M can take on a continuous set of values from zero to one. For discrimination purposes a threshold T can be specified; here T=0.080 is chosen, that is, M is considered high if M is greater than or equal to T.

Tables IV through IX exhibit subsets of Table III for the six object types. Of the 12 comparisons shown, 11 detections succeed, but one falls below the threshold T, that for I1 vs. I5. This comparison involves a scale factor of 2, together with a 90 degree rotation and a translated centroid.

There are, other than these expected high M values, 36 cross object high M values. Most of these are not too disturbing. Twenty are rectangles matched to rectangles but solid vs. hollow or of differing aspect ratio. Five are triangles matched to triangles, solid vs. hollow.

Seven spurious high M values match triangles with circles; of these seven, 5 are solid triangles, 2 are hollow triangles. Discrimination is shown to be unreliable in these 32x32 format cases.

TABLE I

```

SUBROUTINE POLCAR (IMAG,N)
DIMENSION IMAG(32,32),IA(32,32)
XX=FLOAT(N)/2.+1.01
DO 1 I=1,N
DO 1 K=1,N
1 IA(I,K)=0
DO 8 I=1,N
Y=-FLOAT(I)+XX
DO 8 J=1,N
X=FLOAT(J)-XX
R=X*X+Y*Y
R=FLOAT(R)
R=R+.1
IR=ALOG10(R)*13.155+14.2
TH=ATAN2(Y,X)+3.14159
ITH=5*TH+1.
IF (IR.LT.1.OR.ITH.LT.1) GO TO 8
KR=R-.1
IMAG(I,J)=IMAG(I,J)+KR*KR
IA(IR,ITH)=IMAG(I,J)
8 CONTINUE
DO 7 I=1,N
DO 7 J=1,N
7 IMAG(I,J)=IA(I,J)
RETURN
END
    
```

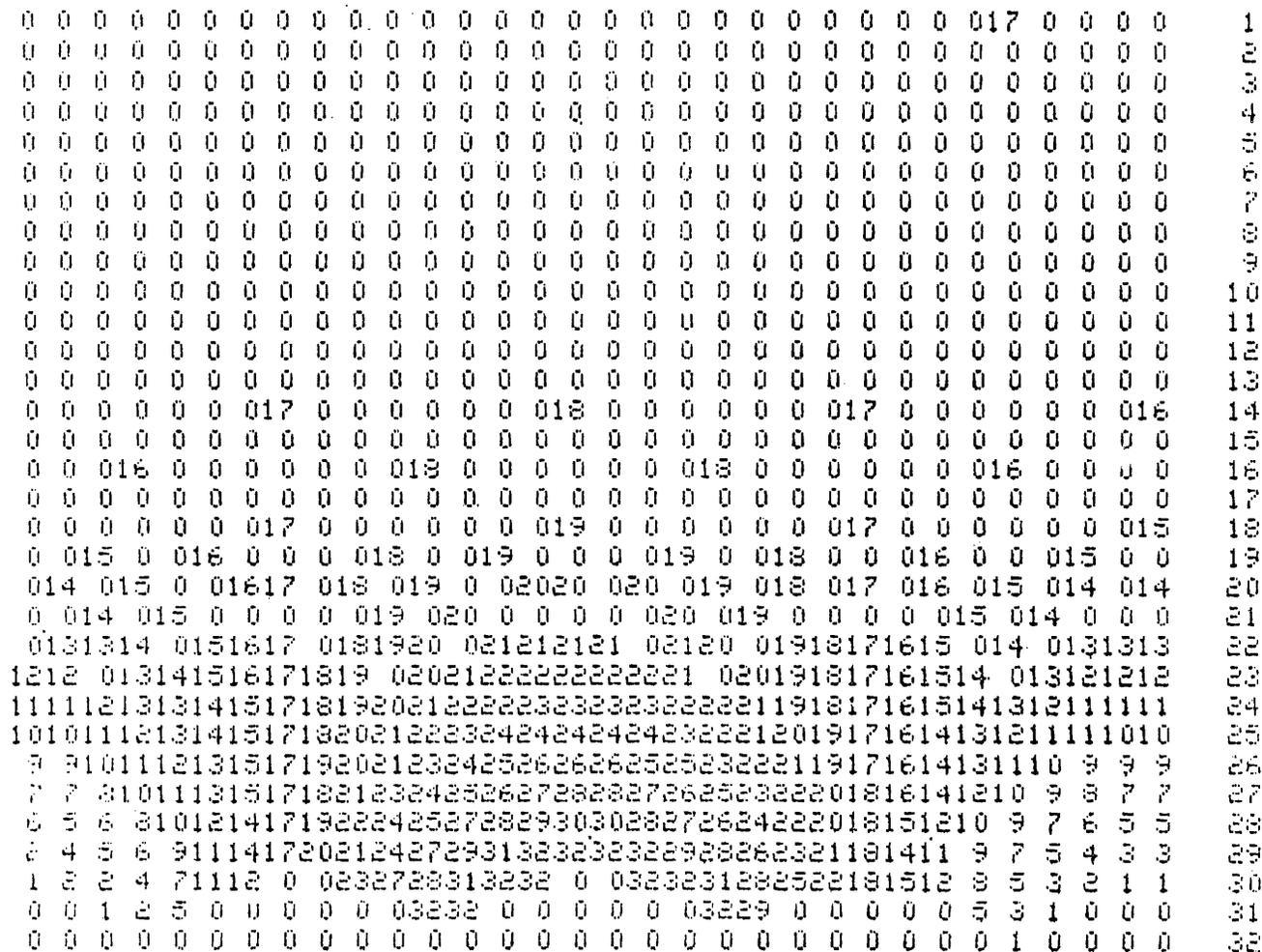


Figure 6

The four remaining anomalous high M values erroneously match triangles and rectangles. The other 57 M values of Table III are all less than T.

arrays measuring 64x64. The FORTRAN SHAPE program described previously was used with minor modification in this 64x64 study. These modifications are:

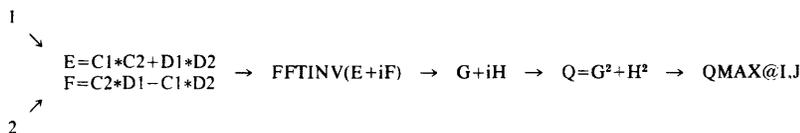
FORMAT SIZE (64x64)

The digital image shape detection research for binary images of size 32x32 has been extended to address image

1. Substitute 64 for every occurrence of 32 in the main program and all subroutines except FOURT.

TABLE II

I1	→	FFT	→	A1+iB1	→	A1 ² +B1 ²	→	REARRANGE	→	POLCAR	→	FFT	→	C1+iD1	→	1
I2	→	FFT	→	A2+iB2	→	A2 ² +B2 ²	→	REARRANGE	→	POLCAR	→	FFT	→	C2+iD2	→	2



$$M = \frac{QMAX(I1, I2) * QMAX(I1, I2)}{QMAX(I1, I1) * QMAX(I2, I2)}$$

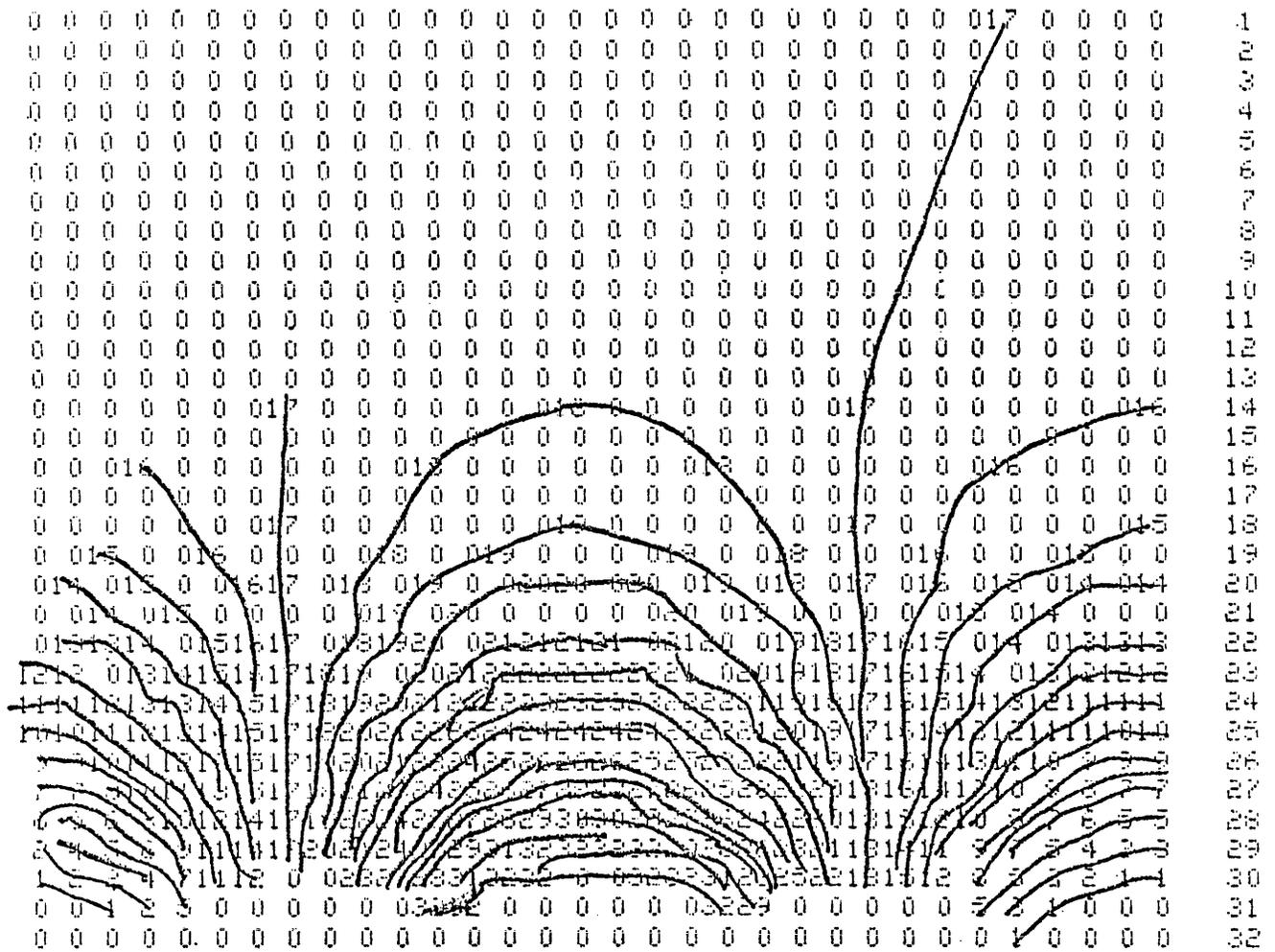


Figure 7

2. In the POLCAR subroutine, change to

$$IR=ALOG10(R)*24.0+24.1$$

$$ITH=10.*TH+1.$$

3. Delete array output displays to reduce wall time.

Sixteen binary 64x64 arrays containing simple objects were studied. The values of M obtained by comparing various (64x64) image pairs using the SHAPE program are shown in the accompanying table, Table X.

There are essentially five types of objects categorized by

TABLE III

	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15	I16
I1	1.000															
I2	0.103	1.000														
I3	0.018	0.078	1.000													
I4	0.020	0.057	0.797	1.000												
I5	0.068	0.344	0.087	0.071	1.000											
I6	0.102	0.999	0.078	0.057	0.346	1.000										
I7	0.131	0.925	0.071	0.057	0.317	0.923	1.000									
I8	0.310	0.177	0.048	0.039	0.122	0.176	0.198	1.000								
I9	0.154	0.237	0.029	0.048	0.089	0.237	0.278	0.080	1.000							
I10	0.026	0.043	0.461	0.347	0.056	0.044	0.037	0.061	0.021	1.000						
I11	0.018	0.078	0.091	0.108	0.035		0.060	0.031	0.053	0.078	1.000					
I12	0.010	0.040	0.132	0.136	0.041		0.035	0.015	0.028	0.137	0.276	1.000				
I13	0.032	0.056	0.327	0.328	0.048		0.055	0.076	0.176	0.100	0.053	0.086	1.000			
I14	0.036	0.024	0.077	0.083	0.020		0.027	0.033	0.041	0.239	0.047	0.087	0.264	1.000		
I15	0.215	0.307	0.021	0.023	0.045		0.350	0.270	0.253	0.018	0.033	0.029	0.101	0.066	1.000	
I16	0.087	0.116	0.029	0.042	0.207		0.130	0.143	0.109	0.012	0.015	0.013	0.136	0.036	0.233	1.000

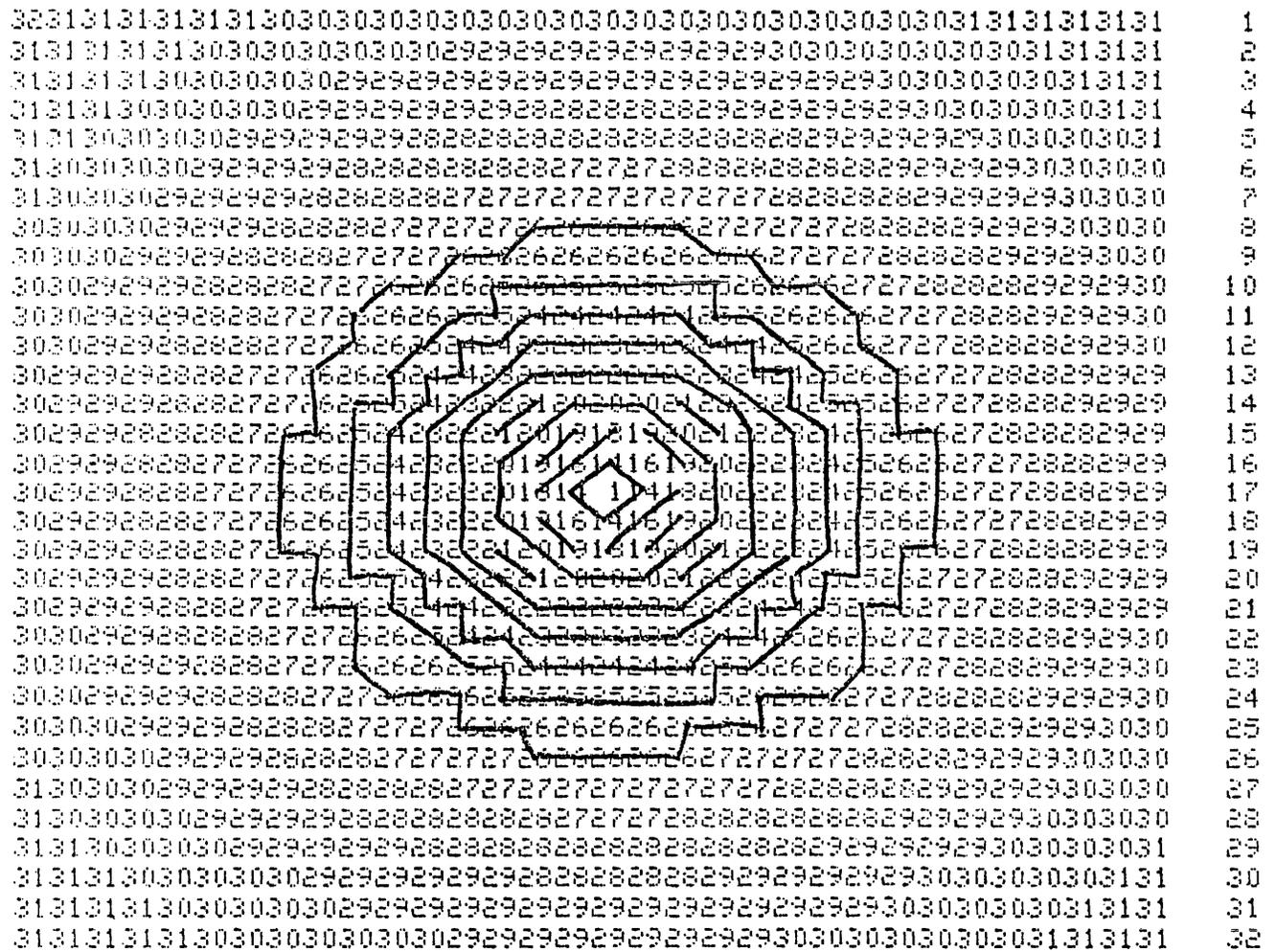


Figure 9

TABLE X

	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15	I16
I1	1.000															
I2	0.229	1.000														
I3	0.025	0.045	1.000													
I4	0.021	0.064	0.201	1.000												
I5	0.032	0.031	0.032	0.033	1.000											
I6	0.050	0.055	0.008	0.008	0.011	1.000										
I7	0.028	0.025	0.204	0.059	0.164	0.010	1.000									
I8	0.051	0.051	0.005	0.005	0.011	0.076	0.006	1.000								
I9	0.205	0.217	0.190	0.106	0.028	0.035	0.037	0.035	1.000							
I10	0.039	0.145	0.106	0.311	0.022	0.002	0.057	0.003	0.039	1.000						
I11	0.806	0.162	0.032	0.027	0.038	0.045	0.034	0.046	0.216	0.038	1.000					
I12	0.038	0.018	0.066	0.231	0.154	0.009	0.232	0.006	0.021	0.150	0.038	1.000				
I13	0.054	0.052	0.030	0.031	0.123	0.008	0.125	0.003	0.015	0.028	0.060	0.107	1.000			
I14	0.007	0.025	0.081	0.109	0.025	0.003	0.045	0.001	0.036	0.174	0.010	0.050	0.113	1.000		
I15	0.038	0.041	0.003	0.002	0.005	0.169	0.004	0.103	0.028	0.004	0.033	0.002	0.003	0.002	1.000	
I16	0.006	0.005	0.004	0.003	0.002	0.003	0.003	0.007	0.003	0.004	0.004	0.002	0.027	0.015	0.005	1.000

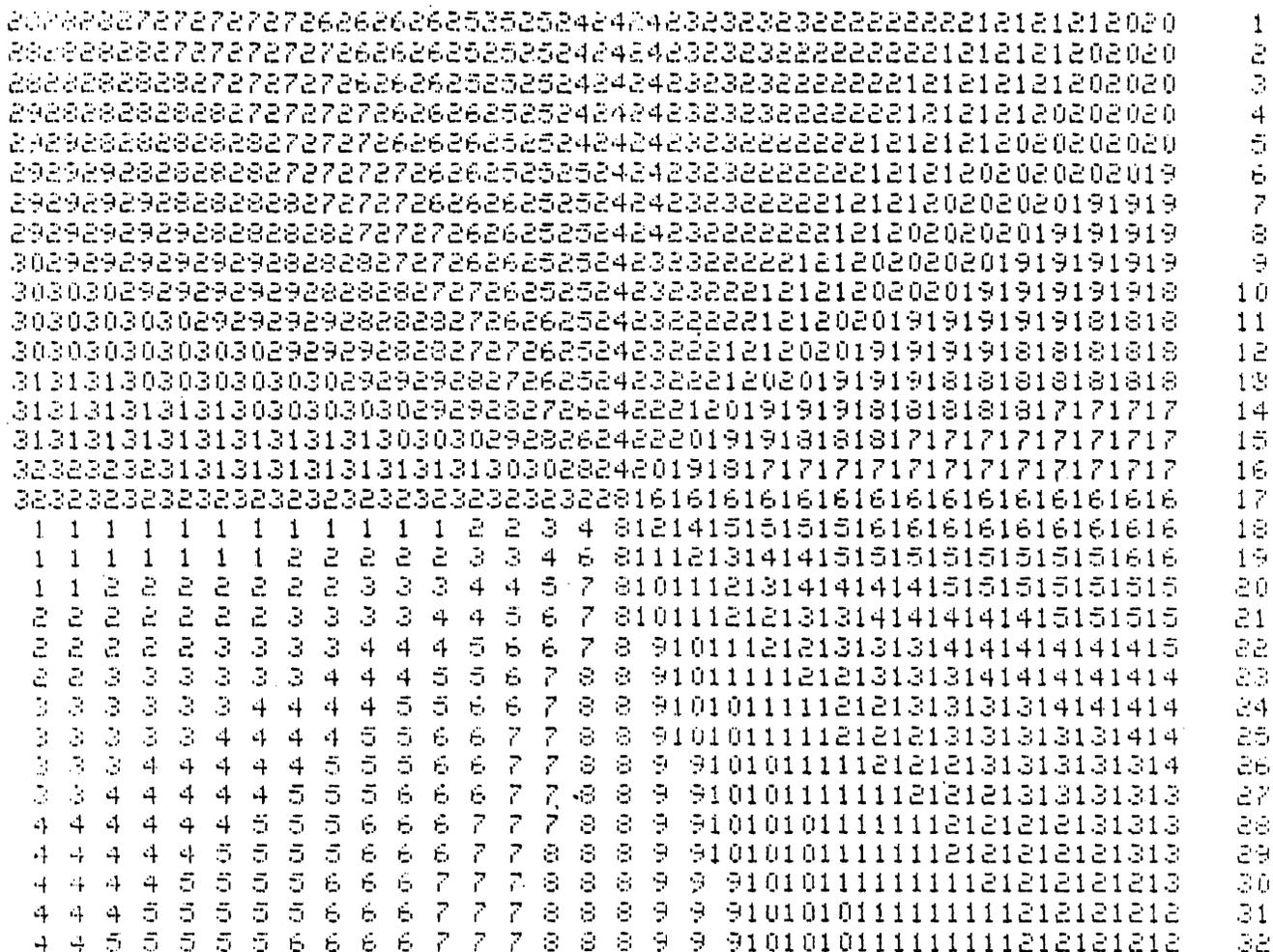


Figure 10

shape in the set of test images:

- A. Solid rectangles, 3.0:1 aspect ratio: I3, I4, I5, I7, I10, I12
- B. Solid right isosceles triangles: I1, I2, I9, I11
- C. Solid circles: I6, I8, I15
- D. Unequal arm crosses: I13, I14
- E. Random noise: I16

TABLE XI—Object A

	I3	I4	I5	I7	I10	I12
I3	1.000					
I4	0.201	1.000				
I5	0.032	0.033	1.000			
I7	0.204	0.059	0.164	1.000		
I10	0.106	0.311	0.022	0.057	1.000	
I12	0.066	0.231	0.154	0.232	0.150	1.000

TABLE XII—Object B

	I1	I2	I9	I11
I1	1.000			
I2	0.229	1.000		
I9	0.205	0.217	1.000	
I11	0.806	0.162	0.216	1.000

For the distinct objects, 120 cross comparisons are possible. It is noted that in contrast to the 32x32 study, the average M value is lower by half. The mean of the 105 entry M32 table is 0.122; the mean of the 120 entry M64 table is 0.061. Accordingly, a lower threshold for discrimination purposes can be selected; here T=0.056 is selected, where T=0.080 was used in the 32x32 case.

Tables XI through XIV exhibit subsets of Table X for four of the five object types. No table is needed for object type E since, as expected, none of the other objects produce a high M value when compared with I16, random noise.

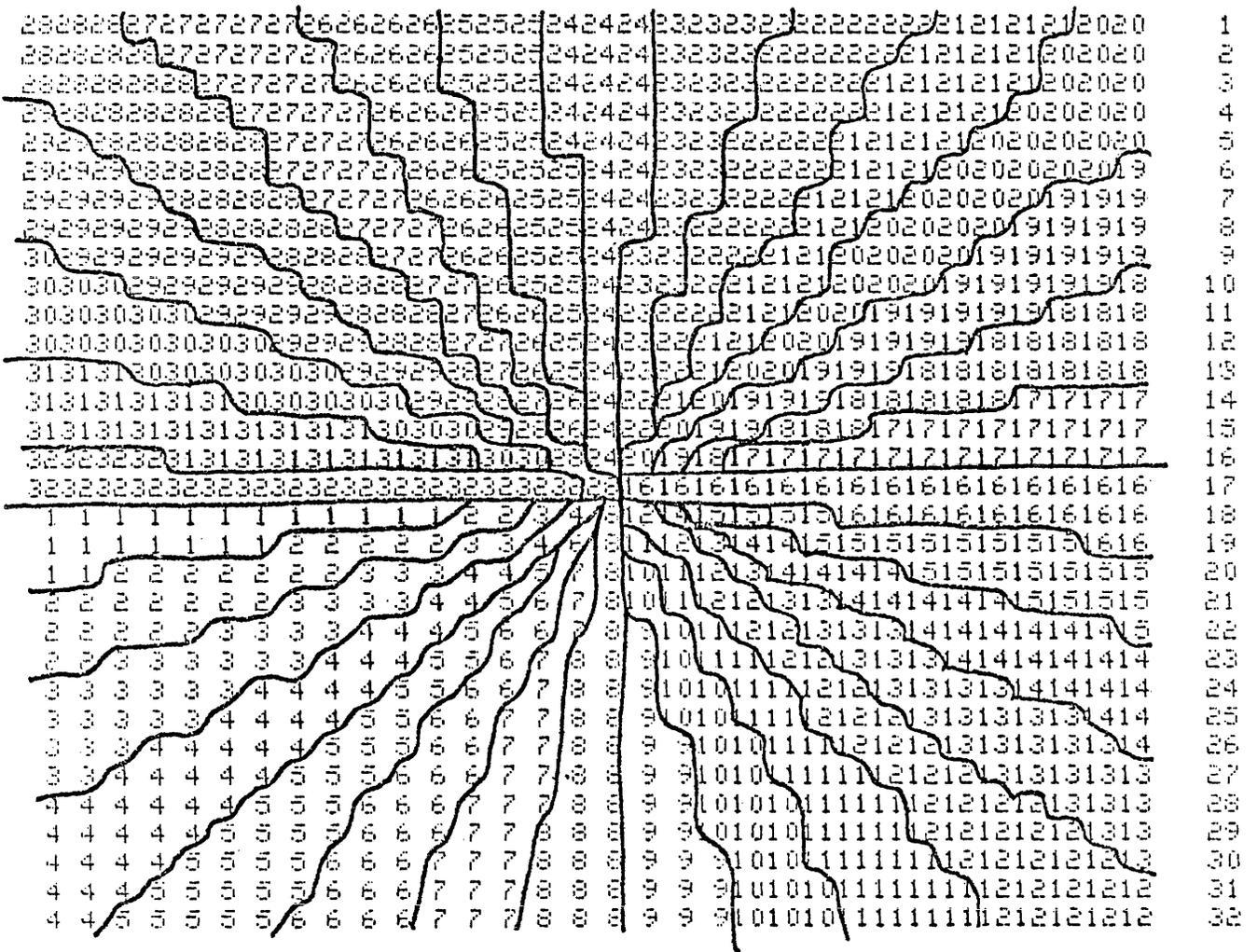


Figure 11

Of the 25 comparisons shown, 22 succeed, but three fall below the threshold T. The three failures are all for I5. I5 is a vertical rectangle and matches are found for the other vertical rectangles, I7 and I12, but matches are not found for the horizontal rectangles I3 and I4, nor is a match found for the 45 degree rectangles, I10. The other vertical rectangles do find matches with the horizontal rectangles and the 45 degree rectangle.

There are, other than the 25 expected high M values, 11 cross object high M comparisons. Of these 11 false alarms, seven are for object type D. I13 is mistaken for vertical rectangles three times and once for a triangle; I14 is confused

twice for horizontal rectangles and once for the 45-degree rectangle. Clearly this algorithm is not suited for discriminating unequal arm crosses from other simple objects. The other four false alarms are triangle/rectangle mismatches.

It is satisfying to note that the circle/triangle discrimination problem encountered in the 32x32 case has in the 64x64 case gone away. All circles are detected, no circles are false alarmed. The other 84 M values of Table X are all less than T.

No attempt at minimizing execution time was made in programming. Execution time for the 32x32 SHAPE Program on a PDP-10 computer operating under the TENEX

TABLE XIII—Object C

	I6	I8	I15
I6	1.000		
I8	0.076	1.000	
I15	0.169	0.103	1.000

TABLE XIV—Object D

	I13	I14
I13	1.000	
I14	0.113	1.000

operating system is approximately 17 seconds; the 64×64 SHAPE Program executes in approximately 23 seconds.

CONCLUSION

The performance of the SHAPE algorithm on these simple objects continues to be encouraging; additional investigation is intended.

ACKNOWLEDGMENT

Gratitude is expressed to Charles Sheffield who suggested the POLAR-Cartesian Transform for research.

REFERENCES

1. Agrawala, Ashok K., and Ashok V. Kulkarni, "A Sequential Approach to the Extraction of SHAPE Features," *Computer Graphics and Image Processing*, Vol. 6, No. 6, Dec. 1977, p. 538.
2. Wong, Robert Y., and Ernest L. Hall, "Scene Matching with Invariant Moments," *Computer Graphics and Image Processing*, Vol. 8, No. 1, Aug. 1978, p. 16.
3. Pavlidis, Theodosios, "A Review of Algorithms for SHAPE Analysis," *Computer Graphics and Image Processing*, Vol. 7, No. 2, April 1978, p. 243.
4. Rosenfeld, Azriel, "Picture Processing: 1977," *Computer Graphics and Image Processing*, Vol. 7, No. 2, April 1978, p. 211.
5. Danielsson, Per-Erik, "A New Shape Factor," *Computer Graphics and Image Processing*, Vol. 7, No. 2, April 1978, p. 292.
6. Rosenfeld, Azriel, and Avinash C. Kak, *Digital Picture Processing*, Academic Press, 1976, p. 416.
7. Brousil, James K., and David R. Smith, "A Threshold Logic Network for SHAPE Invariance," *IEEE Transactions on Electronic Computers*, Vol. EC-16, No. 6, Dec. 1967, p. 818.
8. Hawkins, J. K., et al., "Automatic SHAPE Detection for Programmed Terrain Classification," *Proceedings, Society of Photographic Instrumentation Engineers*, Boston, 1966, p. XVI-1.
9. Huang, G. C., et al., "Pattern Recognition by Mellin Transform," *EIA/AIPR Symposium*, University of Maryland, April 1975.
10. Hord, R. M., and Sheffield, Charles, "The POLAR-Cartesian Picture Transform," Earth Satellite Corporation Technical Report, May 1975.

PM⁴—A reconfigurable multiprocessor system for pattern recognition and image processing*

by FAYÉ A. BRIGGS, KING-SUN FU, KAI HWANG and JANAK H. PATEL

Purdue University
West Lafayette, Indiana

MOTIVATION AND OBJECTIVES

Pictorial information is often described by digitized arrays, syntactic (and semantic) strings and high-dimensional trees or graphs. The analysis and extraction of meaningful information for pictorial patterns by digital computers is called *pictorial pattern analysis*. Pattern analysis tasks require a wide variety of processing techniques and mathematical tools. In most machine intelligence systems, large computers are employed to process pictorial information. Because most image processing tasks require only repetitive Boolean operations or simple arithmetic operations defined over extremely large arrays of picture elements (pixels),¹ the use of large computers with rigidly structured sequential or parallel processors may result in intolerable waste of resources.² For example, the array-structured ILLIAC IV³ and STARAN⁴ are efficient for processing fixed-length vectors, but are very inefficient for mixed scalar and vector operations, due to the fact that multiple instruction streams do not exist simultaneously in these supercomputers.

In the application domain, explosive amounts of pictorial information need to be processed. For example, a single frame of LANDSAT imagery contains 30 million bytes of information and it takes 13 such images to cover the state of Alabama. What is even more demanding is that an entirely new set of imageries is produced for the entire earth surface every nine days. The conventional parallel computers are not tailored for such large-scale image processing. A computer system is demanded to maximize the utilization of parallelism embedded in repetitive image operations. A simple example may help to quantitatively justify such a need. Suppose that we are interested in performing the texture analysis of an image with size 500×500 pixels. A 10×10 pixel window size is selected. Assume that, on the average, ten assembly instructions are required to perform one texture analysis (neighborhood) operation. It then requires $500 \times 500 \times 10 \times 10 \times 10 = 2.5 \times 10^8$ instructions to analyze the whole image. For a computer system with one MIPS, it will take $2.5 \times 10^8 / 10^6 = 250$ sec. = 4.17 min. to perform each texture analysis operation on the whole image. An increase of

machine speed to 100 MIPS will reduce the time required to perform one texture analysis operation to 2.5 seconds. Similar examples can be easily found in cluster analysis and statistical classification of high-dimensional pattern recognition problems.

To meet the needs of the 80s or beyond, a versatile computer system must be able to execute more than 100 MIPS with a memory bandwidth of 256 megabytes or greater. With the rapidly growing IC technology, it is now possible to consider the use of a large number of microprocessors as the processing elements of a computer system for pattern recognition. This system will derive its high performance by the multiplicity of processing elements and the high level of concurrency of processing.

In this paper, we report a powerful computer system that is currently under development at the Advanced Automation Research Laboratory (AARL) of Purdue University. The system consists of hundreds of LSI bit-slice microprocessors with a large number of shared memory modules and flexible interconnection networks for efficient image processing and pattern recognition applications. The system is designed to be able to reconfigure its resources under system control to assume four different operation modes—SIMD, MIMD,² multiple SIMD and distributed mixed modes. Fast interactive I/O and large image data base are incorporated into the system. Cost effective system architecture and wide range of applications are the main development concerns.

An overview of various existing special computer architectures for pattern information processing can be found in Fu.⁵ The system presented here offers a relatively new architectural configuration with high application flexibility and high system throughput at only a moderate system cost.

THE PM⁴ SYSTEM ARCHITECTURE

The architecture of the Purdue Multi-mode Multimicroprocessor (PM⁴) system grew from a consideration of existing system organizations like the C.mmp⁶ ILLIAC IV and Cm*.⁷ We wanted a system that would reconfigure itself to execute MIMD or SIMD processes. In fact, the flexibility was extended so that the system can be partitioned into groups of processors which may be assigned to different

* This work is partially supported by the National Science Foundation Grant ENG 78-16970.

SIMD processes. Hence, multiple SIMD (MSIMD) and MIMD processes can be in execution concurrently. Moreover, since we wanted dynamic system reconfiguration of resources and high level of concurrency of processes, each processor would be designed to handle such system requirements. The reconfiguration is mostly software controlled. This architecture differs from the restructurable computer system proposed in Reference 8 or the partitionable multiprocessor system discussed in Reference 9 in many aspects, as will be seen shortly. The architecture of the PM⁴ was configured by considering some of the major problems involving multiprocessor systems as discussed in Reference 10.

The basic components of the PM⁴ consist of *N* identical Processor-Memory Units (PMU), *K* identical Vector Control Units (VCU), a three-level hierarchical memory connected by a set of interconnection networks and memory management units. Figure 1 shows a block diagram of the PM⁴. We

will give a brief description of each of the individual components and their interrelationship in the system.

The *Vector Control Units* (VCU) are used in the SIMD mode of operation. Each of these units has a microprocessor, a local memory (LM) which is managed by its own *Local Memory Management Unit* (LMMU) as shown in Figure 2a. The vector control instructions and program of an SIMD process are loaded into the VCU local memory prior to execution. When the SIMD process is ready-to-run, the VCU broadcasts instruction to all of the *Processor-Memory Units* (PMU) that are assigned to the SIMD process. The VCU may also send permutation function commands to the *Interprocessor Communication Network* (IPCIN) to permute the data in a group of PMUs. Furthermore, the VCU has the ability to *mask* or disable PMUs so that only the active or unmasked PMUs execute the broadcasted instructions.

Each of the PMUs consists of three functional units,

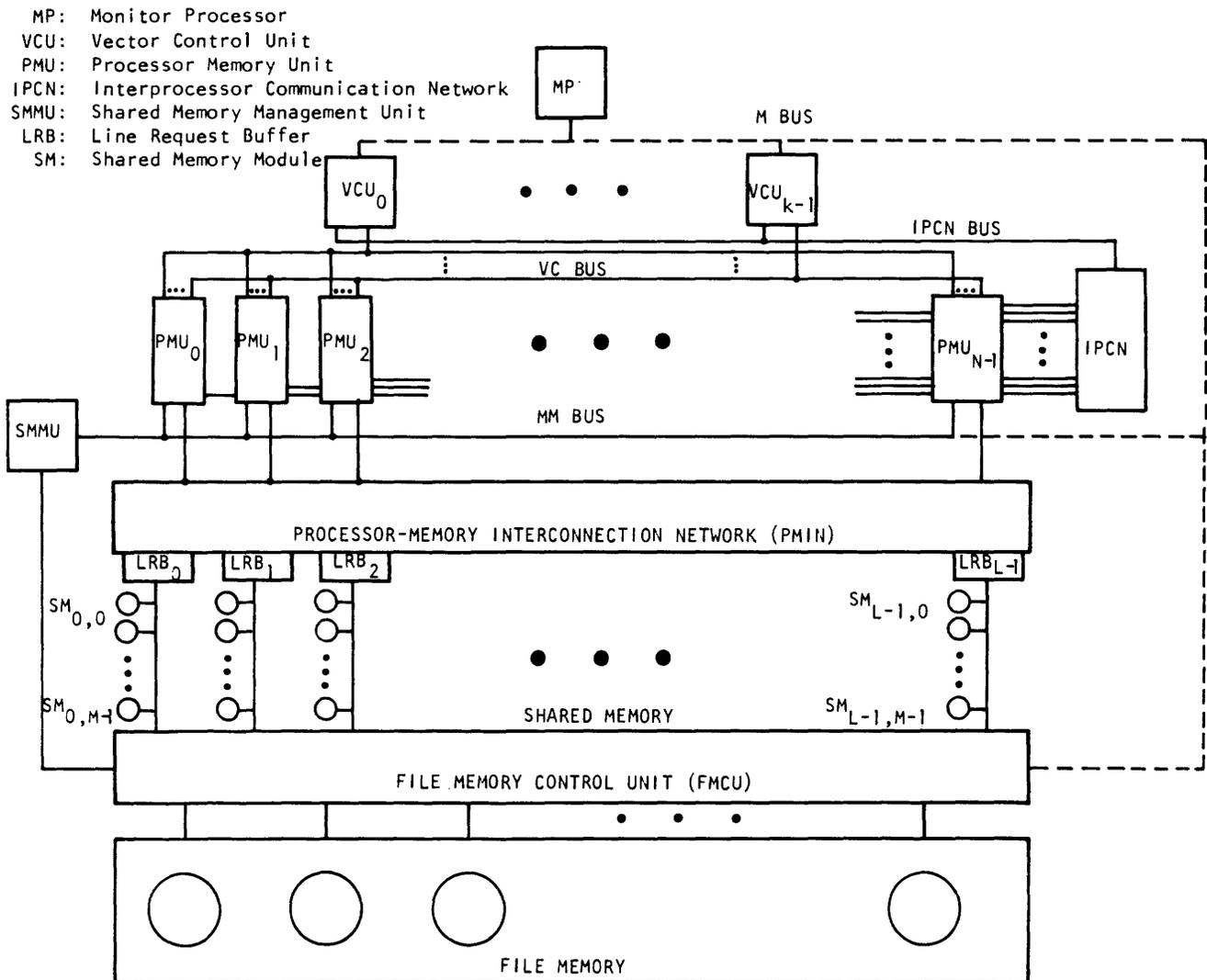


Figure 1—The PM⁴ architecture.

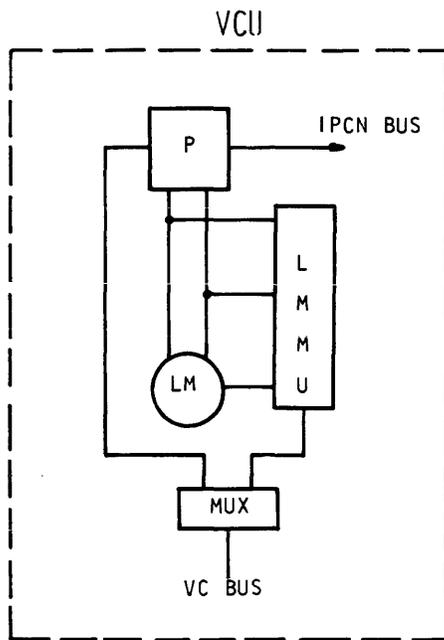


Figure 2a—Details of Vector Control Unit (VCU).

namely, a microprocessor (P), a local memory (LM) and a local memory management unit (LMMU) as shown in Figure 2b. The LMMU in the PMU is similar to that in the VCU. Each local memory, which acts as a cache memory to its associated processor, is interleaved to allow it to meet the speed requirements of the high speed LMMU.

Both VCU and PMU operate under a virtual memory system and hence have hardware facilities to map virtual addresses to physical addresses. We have decided to implement each processor so that it can directly address all of shared memory.

The LMMU of the PMU is also used to load and unload the local memory of a PMU. Furthermore, it can also act as a channel to transfer a block of shared memory to any VCU memory that is associated with that PMU. Each LMMU in a PMU or VCU handles the page replacement policy for its local memory. In both cases, the transfer may be initiated by a command from the processor of the PMU to its LMMU. A multiplexer has been conveniently located between each PMU and the Vector Control busses to switch the signal path from any VCU to either the processor (P) or the LMMU of the PMU. Hence, the program for an SIMD process may be transferred from shared memory to the VCU's local memory through the LMMU of an assigned PMU. Moreover, during the execution of the SIMD process, the multiplexer can route the broadcasted instruction from a VCU to the PMU.

The Interprocessor Communications Network (IPCN), is used to implement permutation functions needed during execution of SIMD processes. This network permits concurrent permutation of data from multiple SIMD processes which are assigned to distinct subsets of PMUs to be performed simultaneously. The IPCN is controlled by the

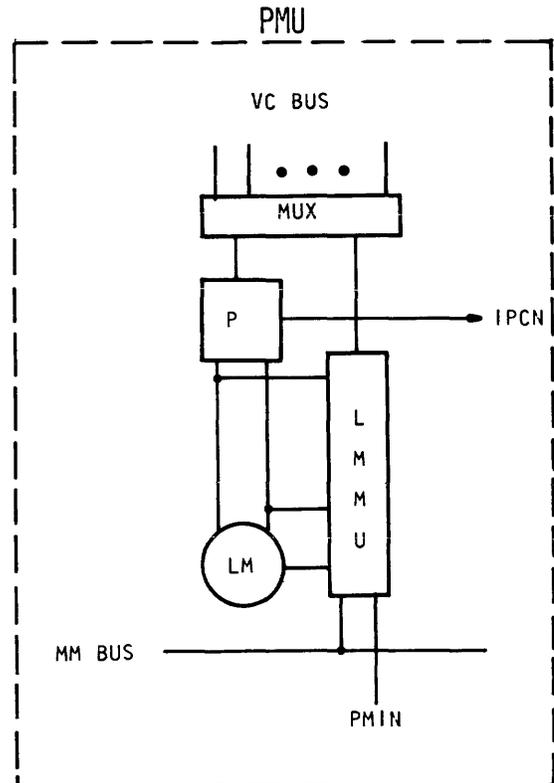


Figure 2b—Details of Processor Memory Unit (PMU).

VCUs over a time-shared bus and contains its own internal conflict resolution logic.

The *Shared Memory Management Unit* (SMMU) is connected to each LMMU of a PCU via the Memory Management (MM) Bus. The SMMU acts to control the use of the shared memory by communicating with each LMMU or the *File Management Control Unit* (FMCU) and effecting appropriate page replacement policy in the shared memory.

The *Processor Memory Interconnection Network* (PMIN) is used to transfer information between the shared memory and the LMMUs. Transfers are made in a *burst-mode* on this network. Hence, once a path through the network has been established, it is held (for the most part) until the transaction is completed. Further discussion on the PMIN is given in a later section.

The file memory control unit (FMCU) controls the transfer of information between the shared memory and the file memory. We defer the discussion of the shared memory to a later section.

For performance measurement purposes, we have incorporated a monitor processor (MP) to monitor the activities of the various modules of the system as shown in Figure 1. The information collected will be used to determine the operating characteristics of the system.

We have incorporated fault-tolerance capabilities into the architecture of the PM⁴ by modularizing the system structure. This, for example, may permit a PMU or VCU to be logically isolated from the rest of the system once a fault is

detected in the unit. The logical isolation of a unit will permit its diagnosis to be carried out without appreciably affecting system performance.

CHARACTERISTICS OF THE MICROPROCESSORS

The processors used in the multiprocessor system must have certain desirable characteristics in order to handle the multi-modal requirements of the PM⁴ system architecture. We have investigated the characteristics of existing LSI microprocessors such as the LSI 11, Intel 8086, Z8000 and Motorola 68000 only to find that they do not meet either our operating system requirements or they fall a bit short on our speed requirements. We will discuss some of the processor characteristics we find desirable in our system configuration. We feel that such processor requirements can be attained by using bit-slice microprocessors.

For swift reconfiguration of system resources, a processor must be capable of holding more than one active process state. Hence, the VCU and PMU are multiprogrammed. The degree of multiprogramming in the PMU is tentatively chosen to be four. Hence, the processor of a PMU should consist of four register arrays. Each register array may be used to hold the state of an active process. For example, one array may be assigned to the kernel of an operating system while the other three are assigned to user MIMD and SIMD processes. A register array which is designated for an SIMD process when the PMU is allocated to a VCU will retain the state of the SIMD process until the PMU is deallocated. The coexistence of SIMD and MIMD processes in a PMU will permit the switching of a current SIMD process to an active MIMD process, and vice versa, efficiently. Further studies will be needed to determine the degree of multiprogramming in the VCU which will be required to maintain a high level of concurrency efficiently.

Each processor of a VCU or PMU has a Status Output Register (SOR), the contents of which can be read by any other processor. Traps and interrupts are also needed to handle fault and communication problems. For example, a page fault from either VCU or PMU should cause a page fault trap which will abort the execution of the current instruction. The instruction may be re-executed when the page fault condition is resolved. In the case when the trap condition occurs in a PMU which is assigned to a VCU for an SIMD process, the VCU is signalled to suspend its instruction broadcasting. Furthermore, all PMUs in that group will be interrupted to suspend the current SIMD process and switch to a ready-to-run MIMD process until the SIMD process is awakened. The context switching of processes can be performed, in this case, simply by modifying a *Current Process Pointer* (CPP) register in the PMU to point to a ready-to-run MIMD process whose process state is resident in a register set of the PMU.

Note that when an SIMD process is suspended due to, say, a page fault, the PMU group is still allocated to the suspended SIMD process. When the VCU is ready to resume its suspended process, it may do this by broadcasting an instruction to the allocated PMU group. If the PMU is

within an instruction cycle of a current MIMD process when the vector instruction is broadcasted, it sets an internal VIP (Vector Instruction Present) flip-flop. Hence the broadcasted instruction may be asynchronous with respect to a group of PMUs. The VIP flipflop is checked at the end of the instruction cycle if the PMU is allocated to an SIMD process. If the VIP is set, the CPP register may be modified to point to the state vector of the resumed SIMD process which is resident in the processor of the PMUs. Further, it puts the processor in the instruction fetch state in order to receive the broadcasted instruction.

When a processor of a PMU is executing broadcasted instructions in the SIMD mode, an instruction completion signal is sent to the associated VCU at the end of each instruction cycle. This will permit the VCU to broadcast the next instruction to the PMUs. In general, instruction fetch and execution may be overlapped in both the VCU and PMU by prefetching the instructions in both SIMD and MIMD modes. We are currently investigating some other characteristics of the processors.

MEMORY HIERARCHY

The memory hierarchy consists of three levels of memory. The highest level of memory are the local memories in the Vector Control Unit (VCU) and the Processor-Memory Unit (PMU). The next level is the shared memory that is shared by all the processors in the system. The lowest level is the file memory which is essential for the data base. Generally, the higher the level of the memory, the faster is its speed, the higher is its cost per byte and the smaller is its capacity. Transfer of information between adjacent levels of memory in the hierarchy is entirely controlled by activities in the first level. The first level in this case consists of the set of VCUs and the set of PMUs. However, this does not imply that the memory access times for the local memory in the VCU is identical to that in the PMU.

One of the advantages of the hierarchical memory organization is that the working set¹¹ of a process accumulates rapidly in the fastest level. Hence, accesses to memory words in a process are completed at nearly the speed of the local memories, but the total cost of the storage system approaches that of the lowest level. Another advantage is that the mechanism which effects the transfer of pages between adjacent levels of memory can be readily implemented with very little intervention by the operating system.

The local memory of a PMU or VCU acts as a "cache" memory to its local processor. The wide usage of cache memories has shown that a process' memory references tend to cluster in a small portion of its address space in a space-time window.^{12,13} Each local memory is logically partitioned into several pages which is large enough to hold the working sets of the several active processes. The local memory also has the appropriate hardware to keep track of the usage of each page. This information may be collected by the Local Memory Management Unit (LMMU) and used to implement the page replacement policy.

The shared memory uses a buffered version of the L-M

memory organization studied by Briggs and Davidson.¹⁴ This memory consists of l memory lines and m memory modules on each line. A line refers to a bus within the shared memory. Each memory module has an address and data latch so that the address cycle (hold-time), a , is much shorter than the memory cycle time, c . Hence, if $a=1$ and $c=4$, four memory requests can be in different stages of the service concurrently on the same line, thereby increasing the memory bandwidth without increasing the cost of the PMIN. Multiple-access conflicts which occur when simultaneous (parallel) requests reference the same line are resolved in the PMIN as discussed in the next section.

In our studies, we assume that the number of lines, l , is equal to the number of parallel PMUs, N , so that the cost of the processor memory interconnection network (PMIN) will be kept to a minimum with respect to l .

Each line has a Line Request Buffer (LRB) which buffers and resolves the conflicts of the memory requests made to modules on the same line. The LRB may subsequently issue the request to a referenced memory module which is idle, and initiate a return path through the PMIN for the referenced data. A memory request for a page can be made to the LRB which will eventually generate the series of sequential accesses required to retrieve or store the requested block. Furthermore, the LRB may be interrupted by the File Memory Control Unit (FMCU) when it performs DMA (direct memory access) transfers between shared memory and file memory.

The performance of the L-M organization was discussed in Reference 15 for the nonbuffered requests with random address references. The analysis of the buffered requests is currently being investigated, but intuitively the implementation of block transfers on such an organization would result in a higher memory bandwidth than discussed in Reference 15.

The file memory is a very large data base and backup for the programs and data in the system.

INTERCONNECTION NETWORKS

Several communication paths exist between different components of the PM⁴ system. A glance at the block diagram of Figure 1 shows the explicit connections between vector control units (VCUs) and the processors. Other principal connections shown simply as black boxes are inter-processor communication network (IPCN), processor-memory interconnection network (PMIN), and the implied connection in the file-memory control unit (FMCU). Of these three networks the connection in the FMCU is the simplest and the least demanding of all. For these reasons, a single high-speed time-shared bus is chosen as the communication path between the shared memory and the file memory due to the slow transfer rates of the file memory. The other two networks, namely, PMIN and IPCN, are quite complex and if their design is not properly chosen these networks can either become the bottleneck of the system or they can become the most expensive parts of the whole system.

For processor-memory interconnection (PMIN) we have

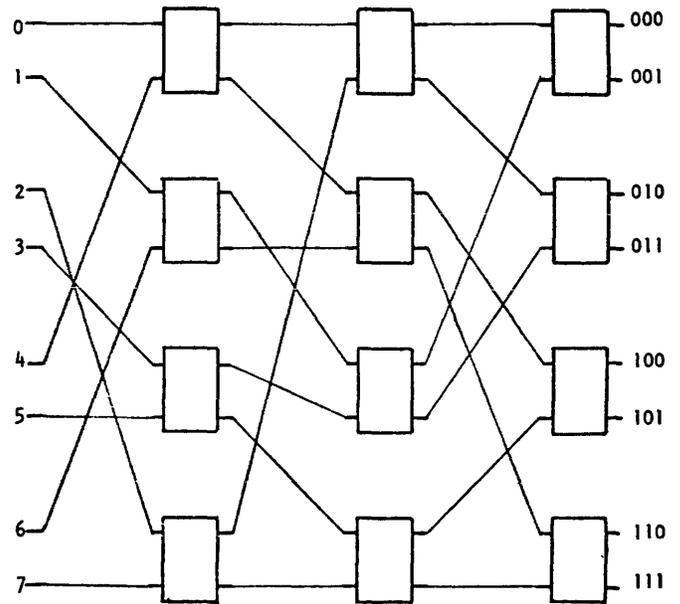


Figure 3—An 8x8 delta network.

investigated the delta networks.¹⁶ These networks are easy to design and control. The networks use 2x2 crossbars as the basic building blocks. The logic for arbitration between conflicting requests is distributed throughout the network. A connection between a processor, LMMU, and a shared-memory module is established at the request of the processor which sends the address of the requested module on the control lines. This address acts as the pathfinder through the network and the path is established locally at each 2x2 crossbar module. Each module requires a single bit from the address to establish a path, thus the control is completely distributed. An example of an 8x8 delta network is shown in Figure 3. The 2x2 modules used are sketched in Figure 4. The complexity of a $N \times N$ delta network grows as $N \log_2 N$

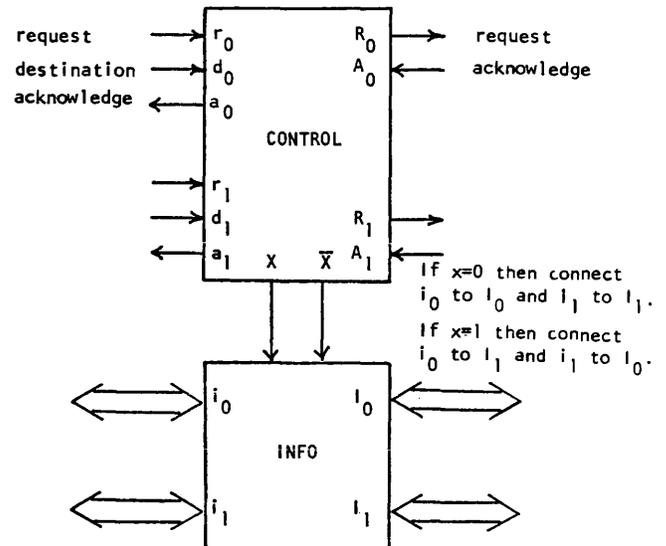


Figure 4—Details of 2x2 modules for delta networks.

as opposed to N^2 for a full crossbar. However, the bandwidth of the delta network, assuming completely random requests, is not substantially less than that of crossbars. For example, for a delta network of size 256×256 , the expected bandwidth is 77 requests per memory cycle; for a full crossbar of the same size the bandwidth is 162; however, the crossbar costs about 20 times as much as delta. Once a path between a processor and a memory module is established, the words can be transferred at a continuous rate without any conflict. Thus, every subsequent word transfer does not suffer the initial delay to establish a path, and, therefore, the effectiveness of a delta network is higher in the block transfer mode than in single word transfer mode. We will study the use of delta networks to do block transfers between the processor memory and the shared memory, where a block may consist of 64, 128, 256 or 512 bytes.

The interprocessor communication network (IPCN) is still under investigation. When it is finally designed, it will have the following characteristics.

The network will be of low cost, recirculating type. The design will be such that the permutations most frequently used in an SIMD environment can be generated in a single pass through the network. Other less frequently used permutations may require several passes. Furthermore, the network will be partitionable in fixed-size blocks so that several small size independent SIMD operations may be executed in parallel. For example, a network of size 64×64 can be partitioned into four networks of size 16×16 , and each of these networks will be partitionable into two networks of size 8×8 . To reduce the cost and delay, arbitrary partitions will not be implemented. Again for cost reasons, the network will not necessarily be of the same dimension as that of PM^4 system. For example, if we build the system with 256 processors, we may have an IPCN of size 64×64 . The design and cost performance trade-offs of the IPCN will be reported at a later date.

PARALLEL PROGRAMMING LANGUAGES

In order to have an operational PM^4 system, a multiple mode operating system must be developed for the multiprocessor system. The resident UNIX system in PDP 11/45 is currently under extensive revision at Purdue to handle the following four operation modes of PM^4 . Special high-level programming languages for parallel processing need to be developed, most probably by extension of the C-programming language or the concurrent PASCAL¹⁷ or APL. The parallel programming language will provide the user the power to exploit the full capacity of the system. This will include vector operations (SIMD mode), and two or more distinct vector operations in parallel (Multiple SIMD mode). MIMD mode will permit concurrent execution of several scalar processes and distributed mixed mode allows part of the PM^4 system to operate as an SIMD computer and part as an MIMD computer. The user will not be burdened with the layout of the vectors in the memory, or the allocation, deallocation and synchronization of processors.

The four fundamental operation modes of PM^4 and the

corresponding user programming requirements are briefly described as follows:

1. *SIMD Mode*—Vector instructions with *Single Instruction Stream and Multiple Data Streams* (SIMD) must be explicitly declared by user's programs. The compiler is responsible for the layout of vectors and the VCU is responsible for broadcasting the instructions. The VCU executes control or non-vectorized instructions without passing them to the PMUs. In this mode, each SIMD vector statement is executed in parallel, but subsequent vector statements are executed sequentially. In other words, no multiple SIMD statements can be simultaneously executed as demonstrated.

Example 1: Consider the use of a 32-processor PM^4 system for SIMD operations.

```
Begin
Integer Vector A, B, C [0:31];
Real Vector X, Y, Z [0:31];
Integer I, J;
:
A ← I*B + C;
X ← Y + (Z/2);
:
End
```

2. *Multiple SIMD Mode*—In this mode, multiple number of SIMD operations are executed in parallel. With a 64-processor system, typical Multiple SIMD instructions may assume the following form. In this example, A, B, and C can be considered to be arrays of 128 vectors each, where a vector has 16 elements. Similarly X, Y and Z are arrays of vectors with 32 elements in each vector. The notation $X[I,*]$ signifies a vector: $X[I,0], X[I,1], \dots, X[I,31]$.

Example 2:

```
Begin
Integer Vector A, B, C [0:127, 0:15];
Real Vector X, Y, Z [0:127, 0:31];
:
Parbegin
For I=0 until 127 do
A[I,*] ← B[I,*] + C[I,*],
For J=0 until 127 do
X[J,*] ← Y[J,*] + Z[J,*],
:
Parend
End
```

The two vector processes between the *parbegin* (parallel begin) and *parend* (parallel end) may be executed simultaneously by two VCUs in this mode.

3. *MIMD Mode*—*Multiple Instruction streams and Multiple Data streams* (MIMD) operations are the most generalized parallel programs. Each individual instruction stream must have a sequence of scalar operations. These parallel processes may be interdependent. System deadlock would be a major problem to be solved for MIMD operations. Vector instructions may not ap-

pear in strict MIMD mode, but may appear in the mixed mode to be described in 4.

Example 3:

```
Parbegin
  Subprocess 1,
  Subprocess 2,
  :
  Subprocess n,
Parend
```

4. *Distributive Mixed Mode*—In this mode, SIMD vector instructions and parallel MIMD processes are simultaneously executed as declared by the following statements.

Example 4:

```
Parbegin
  A ← B + C,
  X ← Y * Z,
  Subprocess 1,
  :
  Subprocess n,
Parend
```

} SIMD mode

} MIMD mode

The above operation modes are only the fundamental ones to be implemented. There are many combinations of the above modes. Only after we implement the basic modes can we challenge the implementation of more sophisticated operation modes to upgrade the system throughput and enhance its flexibility.

Special system control instructions must be developed to make the above operations possible. Listed below are several typical system command instructions that may be implemented in the system.

1. INITIALIZE—Set the program counters of allocated processors to specific values.
2. SYNCHRONIZE—Put the allocated PMUs in the WAIT or FETCH state.
3. Vector issue, mask, routing, etc.
4. Memory management, interrupts and I/O commands, etc.

For special parallel-processing applications, such as multiple-frame image processing or pattern classification, special programming or query languages must be developed to handle the very large scale data bases. There always exists a trade-off between the complexity of user programming language and the operating system capabilities.

OPERATING SYSTEM REQUIREMENTS

The operating strategy for the PM⁴ system has to be decided from the following choices:

1. *Multiprogramming* versus *uniprogramming* on vector control and processor-memory units.
2. *Distributed* versus *dedicated* processor operating system.

Based on the architectural features of PM⁴, a *Distributed Multiprogramming Operating System* (DMOS) is under development for the PM⁴ machine. The DMOS system will be developed based on operating system design trade-offs of existing MIMD machines such as the C.mmp Hydra,¹⁸ the CM* operating system.¹⁹ We will also consider the incorporation of some of the operating system aspects of multiple SIMD machines proposed by Nutt for the MAP system²⁰ and the DIMO for the DAMP system discussed in Hwang and Ni.²¹

The DMOS is to handle all the four operation modes described in the preceding section with emphasis on multiple SIMD and MIMD modes. We have considered the case in which the operating system is distributed over all PMUs. Each PMU contains a local kernel operating system which resides partly in its local memory and partly in shared memory. This kernel will be used for scheduling of MIMD and SIMD processes as well as supervising the execution of MIMD and component vector processes.

In the DMOS system, the processes are scheduled to processors on the basis of processor availability and its workload. In addition, the availability of other system resources demanded by the processes will influence the process scheduling algorithm. However, we have investigated a vector or SIMD process scheduling procedure that is flexible. In this case a PMU, which is a temporary *master* scheduler, schedules the vector process to an available VCU. In the following illustration we assume that the VCUs are uniprogrammed systems and the PMUs are preemptible. Hence a vector process always has a higher priority over a user's MIMD process in being assigned to a PMU.

Let us assume that a vector process which was allocated to a $(VCU_i, \mathcal{S}_{j,k})$ pair has just been completed. $\mathcal{S}_{j,k}$ is a set of 2^k consecutively numbered PMUs starting at PMU_p , where $p = j \cdot 2^k$, for $j=0,1,2, \dots$ and k is a nonnegative integer. For example, $\mathcal{S}_{4,3} = \{PMU_{32}, PMU_{33}, \dots, PMU_{40}\}$. Furthermore, assume that there exists a $PMU_m \in \mathcal{S}_{j,k}$ which is a master for the $\mathcal{S}_{j,k}$ group of PMUs. The completion of the old vector process causes PMU_m to search the vector process queues (*VPQis*) (which may reside in SMMU) for a *schedulable process*. A vector process of vector size, V , is schedulable if it is independent or if all the dependencies for that process have been satisfied. The schedulable process is scheduled by the PMU_m to VCU_i if there exists a set, $\mathcal{S}_{\alpha,\beta}$ of PMUs which is not currently allocated to an SIMD process and consists of 2^β PMUs such that $\beta = \lceil \log_2 V \rceil$. If the vector process is scheduled to VCU_i , PMU_m requests allocation of processors in $\mathcal{S}_{\alpha,\beta}$ for the new SIMD process by transmitting the request signal to the processor via the MM bus as shown in Figure 2. Of course, such a scheduling process would be implemented as a *critical section*.²²

The PMU_m also gives the processor in $\mathcal{S}_{\alpha,\beta}$ the necessary information to load-in the multiple data streams required for the SIMD process into their respective local memories. While these PMUs are being loaded with their data, the PMU_m may also initiate the loading of the program segment of the SIMD process through its LMMU to the VCU_i 's local memory. After the loading of VCU_i is complete, the PMU_m

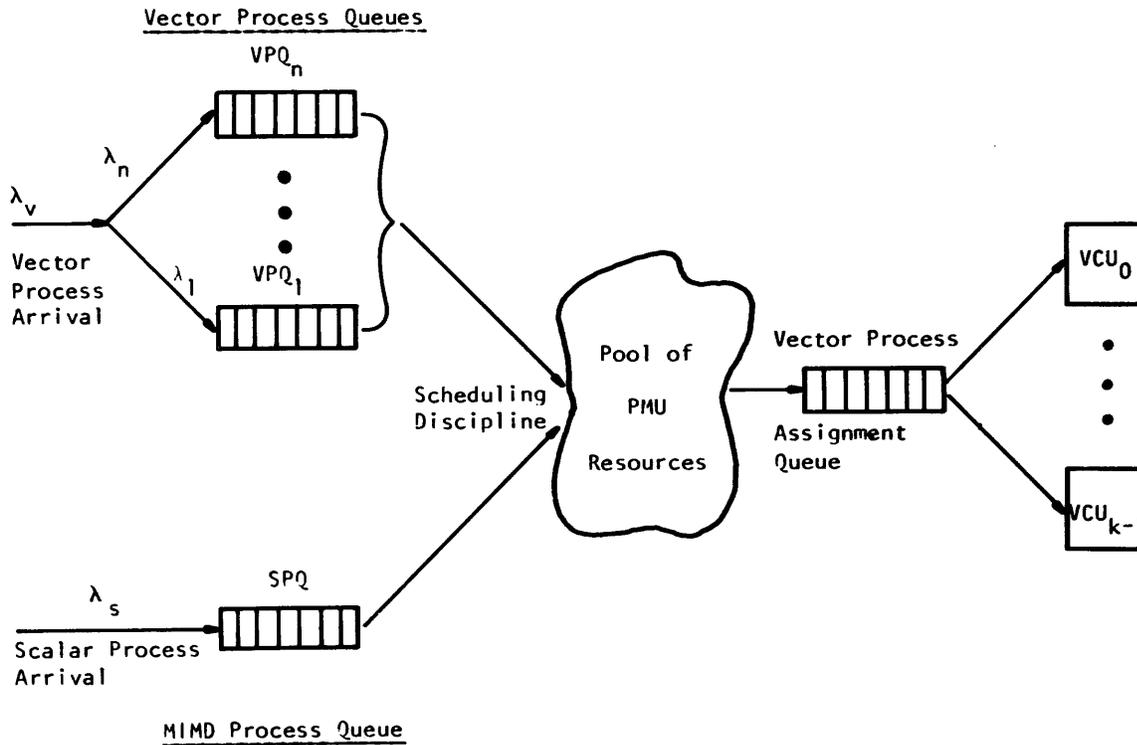


Figure 5—Queuing model for performance evaluation of scheduling disciplines.

may assign any PMU_n in $\mathcal{S}_{\alpha,\beta}$ as the new master of $\mathcal{S}_{\alpha,\beta}$. Henceforth, the current master, PMU_n , will coordinate and monitor the activities of VCU_i and handle page fault traps from VCU_i . Notice that the loading of multiple data streams in $\mathcal{S}_{\alpha,\beta}$ may be overlapped with MIMD process execution in these processors. Now the new pair ($VCU_i, \mathcal{S}_{\alpha,\beta}$) is allocated to execute the new SIMD process.

In case a vector process is not scheduled because of the unavailability of an appropriate sized \mathcal{S} group of PMUs, the master of the next set of PMUs released may check the vector process queue for the schedulability of a vector process. However, if the VP queues are empty, any PMU not currently assigned to an SIMD process may periodically check the VP queues for a process. Alternately, if an empty VP queue becomes non-empty it may signal PMUs not assigned to an SIMD process for service.

Figure 5 shows a typical queuing model which may be used to study the performance of various scheduling disciplines for SIMD and MIMD processes in the PM⁴. In this diagram, VPQ_i is a queue which buffers vector processes that require a set of 2^i PMUs for their execution. SPQ is a queue which buffers MIMD processes. Each process may have tags that will indicate the dependency of the process on another.

Control and scalar instructions in an SIMD process are executed directly by the VCU with no need to broadcast them to the PMU. However, some information may be broadcasted to the PMU during such executions to inform the PMU of current activities in the VCU. During the execution of a sequence of vector instructions, the VCU may

issue a MASKing instruction to select the necessary subset of PMUs among the PMU group allocated to the VCU. Only the masked (enabled) PMUs will execute the broadcasted instructions while the remaining allocated PMUs can continue executing their resident MIMD processes. A DEALLOCATE or RELEASE instruction is needed to release part or all of the allocated PMUs when the SIMD process or subprocess is completed.

Figure 6 illustrates a simplified state transition diagram for each PMU assuming the VCU is operated in uniprogramming mode. States ϕ , K, V and correspond to the PMU being idle, executing a kernel process, vector process or an

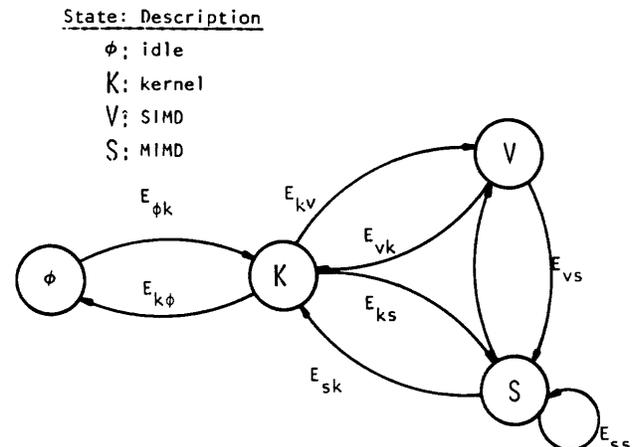


Figure 6—State transitions in PMU for SIMD and MIMD modes.

MIMD or Scalar process respectively. The events which trigger the state transitions are listed below.

- E_{ϕ_K} —Arrival of a new process
- $E_{K\phi}$ —Departure of a last process in PMU
- E_{KV} —Vector process initiation on allocated PMU
- E_{VK} —Trap condition in SIMD process
- E_{KS} —MIMD process scheduled
- E_{SK} —Trap condition in MIMD process
- E_{VS} —Suspended SIMD process causes process switch to MIMD
- E_{SV} —Process switch to ready-to-run SIMD process
- E_{SS} —Process switch from one MIMD process to another

It is expected that the utilization factors of the PMUs will

be high if we assume that deadlock problems are eliminated in this multi-mode multiprogrammed operating system.

APPLICATION AREAS AND COMPUTATIONAL TASKS

The PM⁴ system was designed for the following applications. Both statistical and syntactic methods^{23,24} are to be used in image enhancement, feature extraction, picture segmentation, pattern recognition and scene analysis.

1. Industrial automation (automatic assembly and inspection)
2. Medical diagnosis of X-ray pictures and cytology analysis

TABLE I.—Computational Tasks Required for Various Application Problems in Intelligent Systems

Image Processing & Pattern Classification Problems Typical Computational Tasks	Image Registration	Image Restoration	Image Enhancement	Image Coding	Image Segmentation	Feature Extraction	Pattern Classification (statistical)	Clustering Techniques	Parsing of string or Tree Languages	Error Correcting Parsing	Graph Matching	Computer Graphics
Solving Linear System of Equations	X	X					X	X				X
Solving Nonlinear System of Equations		X					X					
Matrix Multiplication and Inversion	X		X		X		X				X	X
Polynomial Evaluation		X		X	X		X	X				
Nonlinear Mapping		X	X									
Interpolation Methods	X	X										
Iterative Relaxation Methods		X			X			X				
Transformation & Convolution Methods	X	X	X	X		X						
Linear and Non-linear Programming		X					X					
Searching & Sorting			X		X	X		X	X	X	X	
Counting, Backtracking & Estimating						X		X	X	X	X	
Graph Operations					X			X	X		X	X

3. Remote sensing of LANDSAT pictures
4. Automated cartography and stereo compilation
5. Target identification and change detection
6. Computer vision and three-dimensional scene analysis
7. Recognition of human faces, fingerprints and hand-written characters
8. Speech recognition and understanding
9. Pollution control, archaeology and socio-economics

Typical computational tasks associated with above application problems are summarized in Table I. Both numerical and combinatorial (syntactic) algorithms need to be efficiently implemented. An illustrative example is given here to show the advantages of using parallel processing in syntactic pattern recognition and image analysis.

It has been demonstrated that tree languages are efficient in describing and analyzing two-dimensional pictorial patterns.²⁵ Application examples include classification of bubble chamber events,²⁶ fingerprint identification,²⁷ texture analysis²⁸ and recognition of objects of LANDSAT images.^{29,30} In order to effectively analyze noisy and distorted images, the use of error-correcting tree automata has been suggested.^{30,31} The price to pay for the error-correcting capability in image analysis is the increase of computation time. However, with a parallel processing computer system, such an increase of computation time can be easily reduced. An SIMD parallel parsing algorithm for tree languages has been recently proposed.³² Computer simulations based on the analysis of five tree languages $L(G_1)$, $L(G_{22})$, $L(G_{34})$, $L(G_{38})$, $L(G_{68})$ (one for highway recognition is a LANDSAT image and four for texture analysis, with a window size of 9×9 pixels) have produced the interesting results shown in Figure 7. Furthermore, since all the windows in an image are independent in this problem, they can certainly be proc-

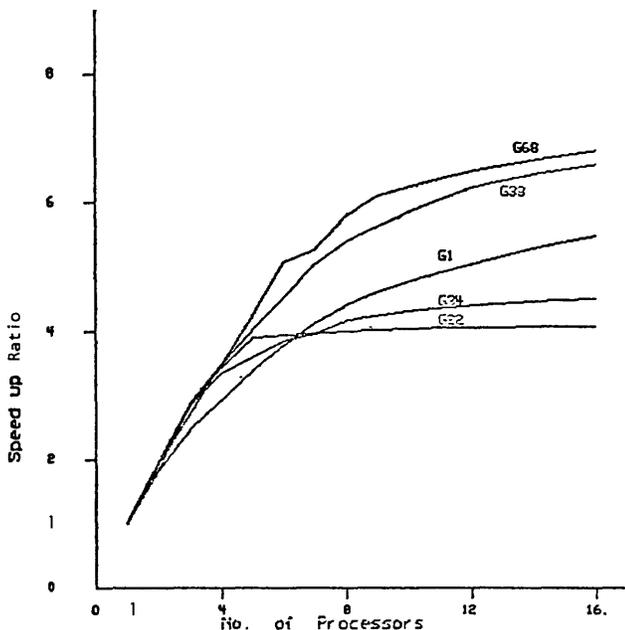


Figure 7—Speed-up for a 9×9 window.

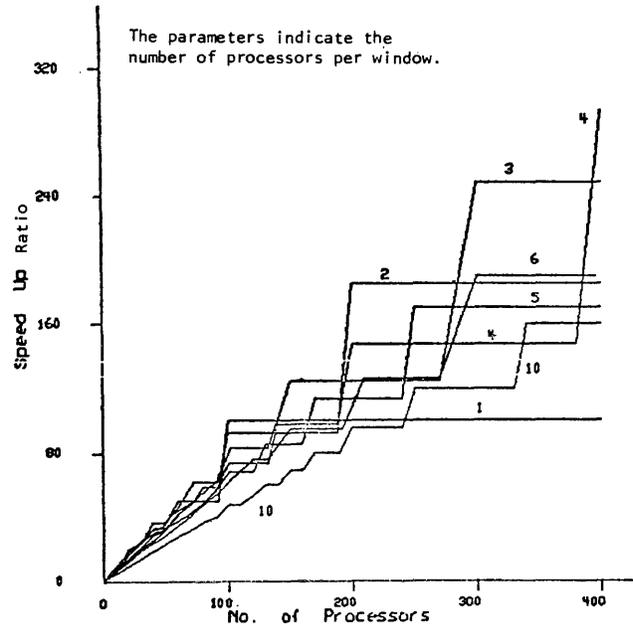


Figure 8—Speed-up for 100 windows.

essed in parallel to improve the processing speed. With 100 window patterns processed in parallel, the speed-up of highway recognition ($L(G_1)$) result is shown in Figure 8. It is noticed from Figures 7 and 8 that either the parallel parsing of tree languages or the parallel processing of image windows would result in significant speed-up of computation time in image analysis and recognition.

ACKNOWLEDGMENT

We would like to extend our appreciation to Richard W. Bishop for his direct contributions to the PM⁴ architecture.

REFERENCES

1. Fu, K. S., and R. Rosenfeld, "Pattern Recognition and Image Processing," *IEEE Trans. Comput.*, Vol. C-25, 1976, pp. 1336-1346.
2. Stone, H. S. (ed.), *Introduction to Computer Architecture*, Science Research Assn., Inc., Chicago, Illinois, 1975.
3. Barnes, G., et al., "The ILLIAC IV Computer," *IEEE Trans. Comput.*, Vol. C-17, August 1968, pp. 746-757.
4. Thurber, K. J., *Large Scale Computer Architecture*, Hayden Book Co., 1976.
5. Fu, K. S., "Special Computer Architectures for Pattern Recognition and Image Processing," *Proc. 1978 National Computer Conference*, pp. 1003-1013.
6. Wulf, W. A., and C. G. Bell, "C.mmp—A multi-mini-processor," *AFIPS Conf. Proc.*, Vol. 41, Pt. II, FJCC 1972, pp. 765-777.
7. Swan, R. J., S. H. Fuller, and D. P. Siewiovek, "Cm*—A modular, multi-microprocessor," *Proc. 1977 National Comp. Conf.*, pp. 637-644.
8. Reddi, S. S., and E. A. Feustel, "A Restructurable Computer System," *IEEE Trans. Comput.*, Vol. C-27, January 1978, pp. 1-20.
9. Bogdanowicz, J. F., "Preliminary Design of a Partitionable Multi-microprogrammable Microprocessor System for Image Processing," *Tech. Report. EE 77 12*, School of Electrical Engineering, Purdue University, Nov. 1977.

10. Fuller, S. H., et al, "Multi-microprocessors: An Overview and Working Examples," *Proc. IEEE*, Vol. 66, No. 2, Feb. 1978, pp. 216-228.
11. Denning, P. J., "Virtual Memory," *Comput. Surveys*, Vol. 2, No. 3, September 1970, pp. 153-189.
12. Meade, R. M., "On Memory System Design," *Fall Joint Comput. Conf.*, 1970, pp. 33-43.
13. Strecker, W. D., "Cache Memories for PDP-11 Family Computers," *3rd Annual Symp. Comput. Arch.*, January 1976, pp. 155-158.
14. Briggs, F. A., and E. S. Davidson, "Organization of Semiconductor Memories for Parallel-Pipelined Processors," *IEEE Trans. Comput.*, Vol. C-26, No. 2, pp. 162-169, February 1977.
15. Briggs, F. A., "Performance of Memory Configurations for Parallel-Pipelined Computers," *5th Annual Symp. Comput. Arch.*, April 1978, pp. 202-209.
16. Patel, J. H., "Design and Analysis of Processor-Memory Interconnections for Multiprocessors," *Tech. Report EE 78-40*, School of Elec. Engr., Purdue University, October 1978.
17. Brinch Hansen, P., "The Programming Language Concurrent Pascal," *IEEE Trans. Software Eng.*, Vol. S-1, No. 2, pp. 199-207, June 1975.
18. Wulf, W., et al, "HYDRA: The Kernel of a Multiprocessor Operating System," *CACM*, Vol. 17, No. 6, pp. 337-345, June 1974.
19. Jones, A. K., et al, "Software Management of Cm*—A Distributed Multiprocessor," *National Comput. Conf.*, 1977, pp. 657-663.
20. Nutt, G. J., "A Parallel Processor Operating System Comparison," *IEEE Trans. on Software Eng.*, Vol. SE-3, pp. 467-475, November 1977.
21. Hwang, K., and L. M. Ni, "Modeling and Analysis of Multiple SIMD/SISD Computer Systems," *Proc. of the Third International Computer Symposium*, Taipei, Taiwan, China, December 1978.
22. Habermann, A. N., *Introduction to Operating System Design*, Science Research Assn., Inc., Chicago, Illinois, 1976.
23. Fu, K. S., *Sequential Methods in Pattern Recognition*, Academic Press, 1968.
24. Fu, K. S., *Syntactic Methods in Pattern Recognition*, Academic Press, 1974.
25. Fu, K. S., "Tree Languages and Syntactic Pattern Recognition," *Pattern Recognition and Artificial Intelligence*, C. H. Chen (ed.), Academic Press, 1977.
26. Fu, K. S., and B. K. Bhargava, "Tree Systems for Syntactic Pattern Recognition," *IEEE Trans. on Computers*, Vol. C-22, December 1973.
27. Moayer, B., K. S. Fu, "A Tree System Approach for Fingerprint Pattern Recognition," *IEEE Trans. on Computers*, Vol. C-25, March 1976.
28. Lu, S. Y., and K. S. Fu, "A Syntactic Approach to Texture Analysis," *Computer Graphics and Image Processing*, Vol. 7, June 1978.
29. Li, R. Y., and K. S. Fu, "Tree System Approach for LANDSAT Data Interpretation," *Proc. Symposium on Machine Processing of Remotely Sensed Data*, 1976, West Lafayette, Indiana.
30. Lu, S. Y. and K. S. Fu, "Structure-Preserved Error-Correcting Tree Automata for Syntactic Pattern Recognition," *Proc. 1976 IEEE Conference on Decision and Control*, Clearwater Beach, Florida.
31. Lu, S. Y., and K. S. Fu, "Error-Correcting Tree Automata for Syntactic Pattern Recognition," *IEEE Trans. on Computers*, Vol. C-27, November 1978.
32. Chang, N. S.; and K. S. Fu, "Parallel Parsing of Tree Languages for Syntactic Pattern Recognition," *Proc. 1978 IEEE Computer Society Conference on Pattern Recognition and Image Processing*, May 31-June 2, Chicago, Illinois.

Transportable image-processing software*

by R. G. HAMLET and A. ROSENFELD

University of Maryland
College Park, Maryland

INTRODUCTION

The computations of image processing, like those of many technical disciplines, require substantial programs to perform. These programs are often organized into "packages" with the intent of making them easy for the (computer) novice to use. Access to a package of programs is an important resource, since its creation is beyond the capabilities of all but a few research groups. Unfortunately, while packages are invaluable, they could often be improved in the following ways:

1. They could be easier to use. The intellectual task of communication with the package is too difficult, the commands too peculiar and errors too easy to make. When things go wrong, very little help is available.
2. They could make more efficient use of the underlying machine and its operating system. A package may use very sophisticated algorithms for its discipline-oriented operations, while at the same time using the most cumbersome mechanisms for controlling the resources of the machine. Its authors are seldom systems programming experts.
3. They could be easier to move from machine to machine. In the process of getting the package to work at all, many peculiarities of the programming language (in its local implementation) and the local system become entwined in the code and getting it to run elsewhere may be difficult or impossible.
4. They could be easier to understand, modify and extend. To add a new routine or alter the behavior of an existing one may not be too difficult for the program's author, but for others it may be impossible. If many changes are made independently, combining them without conflict is difficult.

Improvements in package programs must resolve the conflict between quality (Items 1, 2 and 4) and transportability (Item 3). Here, "transportability" is to mean more than the ability to export software from a development site to many others. We have in mind a software system moving freely

among research groups using a variety of machines, in which modifications arise simultaneously in several places. In this situation the structure of the programming package is all-important—it must be rigid enough to support changes that conform to the style of the original; yet, changes must be easy to make. In the research environment no single group can long afford to maintain and support a large, changing system, so the system must be so constructed as to take care of itself.

Although the scheme presented here applies to many kinds of packages, it is designed to support image processing. From a systems point of view, this means that the computations of the package are characterized by a short interchange of control information with a human user, which determines the amount of resources needed (and these vary greatly), followed by operations that are either input-output limited, or in which there is little overlap between input-output and computation. This rough characterization is used to decide the compromise between quality and transportability.

In the sections to follow, we assume that transportability is a requirement, then attempt to find a way to attain it with the smallest loss of quality. The second section considers the programming language to be used; the third section deals with the operating system and program organization is the subject of the fourth section.

PROGRAMMING LANGUAGE

Even a cursory survey of existing computers shows that FORTRAN is the only programming language with a standard, widespread implementation. FORTRAN is widely implemented partly because it is already so popular, but also because it was designed to fit the von Neumann architecture, still in widespread use. Most FORTRAN implementations "extend" the ANSI standard of 1966¹ in some (nonstandard) way; of course, these extensions are not transportable.

The deficiencies of FORTRAN are widely recognized, but they are largely the other side of the transportability coin—the language allows no control of computer resources; except for the ability to write arithmetic formulas, it is not very high-level and the facilities for separating, protecting and centralizing information are minimal. If a modern language designed with software engineering in mind were

* This work was partially supported by National Science Foundation Grant MCS 77-18719.

widely available, its programs could be moved more easily than FORTRAN programs can be. For example, a language like Alphard² produces programs that are easy to move. The kicker is that Alphard compilers (when there are some) can be expected to be very difficult to implement on a variety of machines, since the compiler (and its run-time support) must make up the gap between the machines and the transportable programs.

The history of digital computers certainly shows that it is foolish to await the rapid spread of good ideas—somehow the bad ideas wind up cheaper—so there seems little danger in making FORTRAN work properly rather than waiting for (say) Pascal.³ Furthermore, an ambitious language may never spread to the range of machines on which FORTRAN already exists. Major vendors will have to deal with better languages; many mini- and micro-processor systems may never support them. We thus consider how FORTRAN can be tamed and transported.

RATFOR⁴ is the most popular version of structured FORTRAN. It has the virtues of a published definition and a partial inverse processor.⁵ Perhaps it should have been designed with one less iteration construct and one more conditional construct, but its wide acceptance more than compensates for such matters of taste. If we specify RATFOR as the implementation language for a transportable package, we must consider the transportability of RATFOR itself. Although the preprocessor exists for many machines, that is not sufficient. Minor variations result from conflicts between the definition⁴ and its presentation in a text⁶ and from an unfortunate choice of delimiting characters unavailable on many machines. The solution is evidently to transport RATFOR along with the package, and the techniques for doing so are well developed.⁷ RATFOR is written in RATFOR, and once any preprocessor exists, a pure FORTRAN version is available, for use on any machine. (It is tempting to use this mechanism to extend RATFOR, for example, to include a fancy macroprocessor;⁸ we judge that the departure from the published standard is not worth the power gained.)

Even with a universal RATFOR available, there may be difficulties in transporting programs and difficulties in writing them, because RATFOR is a "permissive" translator—most of a source is never examined, but simply passed along to FORTRAN. There are three difficulties:

1. Errors in the source are first detected by the FORTRAN compiler, and difficult to relate back to RATFOR.
2. Nothing prevents the FORTRAN imbedded in the RATFOR structure from being machine-dependent, so that although it gets through compilation on a machine, it will not run properly.
3. RATFOR makes no attempt to eliminate a number of legal but error-prone constructs in FORTRAN, notably involving undeclared variable names and inconsistencies in usage. These usually result in strange run-time behavior.

Difficulty (1) is more annoying than fundamental; the oth-

ers can be eliminated without compromising transportability, by a mechanism similar to preprocessing—the RATFOR source can be checked for problems before being translated, compiled and run. The PFORT verifier⁹ is a tool of this kind that attacks Problem 2—it checks that the FORTRAN does not go outside the 1966 ANSI subset and that certain machine-dependent tricks are not used.

There is one transportability problem of any word-oriented language that PFORT makes no attempt to solve, that of numeric precision in the presence of different word sizes and arithmetic algorithms. Techniques have been devised to attack this difficulty,¹⁰ but for the purposes of many packages, it is sufficient to trust the mathematical subroutine library of the target computer.

Problem 3 remains. For all that programmers try to keep usage consistent, mistakes are easy to make. In FORTRAN, a well-meaning programmer cannot see if his intentions were carried out. Since most of the errors that we want to detect occur across the boundaries of separate compilations, checks must operate on the complete package of subprograms as a single source. It is convenient to distribute a package as a single file on magnetic tape, so the preprocessor that goes with it should divide the routines for separate compilation and checking can then be done on the composite source. The following seems a minimal set of operations, and its implementation is no more difficult than building an identifier scanner coupled with a symbol table of usages:

- a. Check for declaration of all variables and observance of conventions in variable names. (The latter is necessary to avoid conflict in libraries.)
- b. Check for consistency across subprograms—argument counts and types, COMMON sizes and types, etc. More restrictive conventions can be enforced here; for example, one can forbid potential side effects.
- c. Prepare a cross-reference table for all symbols of the composite program, listing usage and location in the source. (This can be helpful in decoding FORTRAN-generated error messages.)

Writing in RATFOR according to a set of conventions, then checking for those conventions before preprocessing and compilation, is almost as pleasant as programming in a modern language. On many systems it is more cumbersome, since several programs are involved in a sequence, but the payoff in debugging time saved is excellent.

OPERATING SYSTEM INTERFACE

FORTRAN's deficiencies as a low-level language—its inability to get at this or that machine feature—have always been supplied by a "few little assembler routines." As operating systems have grown and excluded regular programs from direct manipulation of shared resources, the most valuable machine instructions have become the system service calls. Since shared resources are scarce, a program that makes intelligent use of operating system service calls can

run more efficiently than one that does not. Two obvious examples are

1. A program that calculates an optimal memory allocation will give better response at less cost than one which runs with the largest space it might ever need.
2. A program that knows its pattern of record requests on a file can seek these records more efficiently than can any standard access method.

Proper support services are available in almost every system, but in varied form. For transportability we need a standard FORTRAN-callable interface to system functions. This interface must be kept so small that its implementation on a new machine is easy. At the same time, there is a need to make operating system services easy to use, and to tailor them to the application. These conflicting needs can be resolved by separating the interface into two parts—a “kernel” and a “surround.”

In the kernel we seek the bare minimum of code, which we expect to be machine-dependent. The kernel is to be a collection of entry points that transmit essential system functions outward without regard for convenience of use. This kernel is always too large to best serve transportability, partly because there are many needed features. Part of its size results from seeking a set of features common to all systems—the smallest set may not be implementable in some cases. This factor also works against the quality of the kernel—it tends to mimic the worst system on which it must be implemented rather than the best.

Outside the kernel we disguise its awkward properties with another level of interface, the “surround.” The surround contains only machine-independent FORTRAN code. It is therefore appropriate to make its routines easy to use and not worry about their extent. The surround has the special property that although its calling sequences are fixed, and it is viewed as a part of the operating system interface, its code may be juggled in package conversion. In contrast, the kernel routines *require* modification to implement their standard calling sequences; in the package code outside of the surround the code is movable, and the calling sequences themselves are subject to alteration. It can happen that on some particular system one of the surround routines is easy to rewrite as a direct system call, with important advantages in efficiency. So long as the entry sequence is not changed this is encouraged, but no package user *needs* to make the change and it has no effect should the package be retransported.

Functions supported by the kernel

A detailed description of the interface kernel, with FORTRAN calling sequences and implementation hints for many machines, is presented in Reference 11; here we only indicate the necessary functions.

Random-access file operations form the heart of the kernel. It must be possible to create mass storage files, manipulate their names and protections and read or write them in

arbitrary-sized blocks in a true random-access fashion. The input-output operations themselves, and the file formats, should be at the lowest level the operating system provides, to minimize memory and processing overhead. Thus it is important that the operations move data directly to/from FORTRAN arrays without invisible buffering; where possible, the operations should be started and the calling FORTRAN routine permitted to continue, waiting for completion only when necessary. It is common to provide routines of this kind for FORTRAN use and the implementation is straightforward.

FORTRAN provides no memory control facilities. In image processing it is often desirable to calculate memory space required for a given picture (particularly for input-output buffers). The usual implementation of this scheme in FORTRAN uses “get” and “put” routines to move blocks of words out of and into dynamically allocated space. This is unsatisfactory because the overhead is high whenever the elements are addressed in small groups. It is much better to provide a single array whose addressing is efficient and which can grow and shrink as needed; in almost every system it is possible to place such an array in memory so that it can indeed change its real size.

Process control is needed in the kernel to support the open-ended programming techniques suggested in the sixth section. The minimum facility required is the ability for an executing program to “call down” another as its replacement, without the overhead of more than an input operation. In some systems implementation is difficult, but a variety of tricks exist.

The final portion of the kernel is concerned with user communication, in the form of cosmetic features and error control. Most systems can provide information such as the date and time. Of more importance are parameters such as the best record sizes to use for disk operations and the precision/storage capacities of machine words. Run-time information should include resource usage and limitations, particularly for memory. The more a package can find out about what is really happening in the underlying system, the better it can communicate with its human users about problems encountered in execution and the more efficient it can be. Error control is also important. Most errors are unexpected in the sense that they appear as failures at a very low level, and are then communicated up to be processed by the package code. The kernel must see to it that *all* errors are in fact handled in this way, and in some systems that can be very difficult, since the error appears first as an asynchronous interrupt. Something similar to PL/1's ON unit can usually be arranged, leading to cumbersome but complete control.

Functions supported by the surround

Serial file operations can be easily built on the random-access ones of the kernel and there is seldom any reason to rewrite these outside the machine-independent FORTRAN versions, since these can be better adapted to a package's needs than the usual serial routines of either FORTRAN or

most systems. For example, it is easy to specify a buffered scan through a file in which (say) every tenth record is actually read; or, a file can be reblocked to take advantage of the availability of large buffers.

The surround can also be used to eliminate functions from the kernel that cannot be accomplished on all machines. For example, if immediate-return input-output operations are impossible, placing the entries in the surround allows implementation of the starting operation as "start and wait" and waiting as "no operation."

Most interactive communication with package users can employ the FORTRAN formatted input-output package. The memory overhead of the format-scanner routines is high, however, so formatted i-o can be eliminated by including some functions in the surround.

Experience with two systems

An operating-system interface has been implemented on two very different systems. The first is Univac 1100 Exec 8, an "old" system. The second is PDP-11 UNIX, which is about as "new" as operating systems come. The design philosophies of these two are also almost opposite—the Univac is very low-level, compensating for its deficiencies with large library packages; UNIX is designed to support high-level programs.

A technique designed to reduce the implementation effort was used for the kernel. On each machine a routine was written in assembler to provide FORTRAN access to the necessary system calls. For example, on Exec 8, one such call is for programmatic execution of a control statement; under UNIX one is for direct execution of another program. Neither system has the other's service; their different services are needed for a "change to new program" function of the kernel. Once these basic services are available, the interface routine to employ them is written in FORTRAN. The code is peculiar to one machine, but it is often easy to adapt to another. In the example, 30 lines of FORTRAN are common to both systems, the Univac routine has 20 extra lines setting up its peculiar system call, while for UNIX this takes only one line.

In another example, the input-output part of the kernel keeps a table of open files and their characteristics. The

format is different for the two systems, but the code that uses the table is exactly the same.

Code characteristics of the complete kernel are summarized in Table I.

Because the interface routines are largely independent of each other, they can be tested individually. An interesting point is that the test driver is machine-independent, so it can be distributed with a package to help the systems programmer who must convert the kernel. Debugging is further aided by the existence of the package itself. It may contain bugs, but when one of its working features goes wrong, it tends to point to an error in the kernel.

OPEN-ENDED SOFTWARE

The package software that is to be built in FORTRAN on the operating system interface can be expected to undergo almost constant modification, to fit the changing needs of a community of research users. Under this stress it is easy to imagine a "good" package turning into many disparate "bad" packages as the code is modified by many unskilled hands. Insult is added to injury when even the most slipshod changes are hard to make, requiring extensive study of the existing code, then extensive debugging. The internal structure of the package must protect against such changes.

Careful adherence to programming and documentation standards are often offered as solutions to the problems of program modification. But it is observed that not everyone will follow standards and not everyone who tries, succeeds. The bad easily drives out the good. The only structure that will preserve itself is one that is easier to work within than to violate.

The primary technique for structuring software to encourage change is that of centralization and information hiding.¹² Operations should be confined to a single place in the code and encapsulated with access to just the information needed to perform properly. For example, in a package program, the user interface is a candidate for this treatment. If all interactive communication is confined to a collection of routines, rigidly bounded by a well defined interface across which the information passes to other parts of the software, several advantages are gained:

1. It is easy to learn to employ the communication routines, because only the stable interface need be mastered.
2. Collecting the code in one place makes it easier to study, and if it is modified, the modification applies uniformly to all parts of the system.
3. When the centralized code is not in use, it may be possible to get rid of it, reducing the memory overhead.

There are two common methods of circumscribing operations in software. The operation may be implemented as a separate, stand-alone program, linked to other programs only by files. Or, the operation may be implemented as a collection of sub-routines and data structures within a program, communicating with the rest of the program through parameters and global data. In the first mechanism separa-

TABLE I

<i>Property</i>	<i>Exec 8</i>	<i>UNIX</i>
Systems programmer experience (excluding learning assembler)	5 years	5 hours
Assembler "service caller"		
Number of system calls	50	15
Code instructions (excluding dispatch tables)	80	100
Hours to design, code	15	10
FORTRAN Routines		
Number of interface entries	11	11
Support function routines	20	14
Total RATFOR statements	700	530
Hours to initially design, code	30	—
Hours to convert	—	10

tion is easy to enforce, but it is less easy to provide support functions for the independent programs; in the second mechanism support routines are readily available and the problem is to preserve the separation of the parts. Both mechanisms are valuable in an image-processing package.

Communicating independent programs

Of course, any two programs can "cooperate" by interchanging information in files. The user interface of a package is a good example. One program is the collection of routines that interact with the human user, and another is devoted to actually processing the information so obtained. Communication between the programs is through a file of commands/results. The separation is perfect in the sense that the processing program does not interact with the user, and the user routines do not process the information they receive. Furthermore, memory overhead is handled perfectly by this organization—the scanner, command tables, etc. are entirely gone once processing starts, leaving only a standardized command, already checked so that little error recovery code need be part of the processing. The payment for this ideal separation comes when one of the independent programs is changed—how can the changes be taken into account by the other programs without modifying them also? In the example, suppose the behavior of one processing program is changed. How can this be automatically reflected in the user dialogue? Similarly, how can the communications program really check input commands when it does not know exactly what the processing program intends to do with them?

These problems can be solved by arranging another level of communication between the independent programs. Each can notify the others of its capabilities in an initialization run, resulting in the creation of a kind of "configuration" file. This file records what programs exist, what operations they perform and describes the commands for those operations. When a change takes place it is only necessary to repeat the initialization run to have its effect felt throughout the collection of programs. This organization suggests another independent program function, that of explaining the system's capabilities and operation. A "help" program would make use of the configuration file to explain difficulties and provide on-line documentation, with this high-overhead operation entirely divorced from all "working" programs, yet necessarily up-to-date.

The two essentials for cooperation of independent programs are the ability of one to invoke another (and itself be reinvoked to inspect the results), and the definition of processing tasks in a format that can be concisely described in a configuration file. The operating system interface provides the former and the latter is a natural consequence of any command language that can be formally described.

Linked sub-routine organization

The primary reason for encapsulating package operations as complete, independent programs rather than as loadable

overlays is that overlaying is done very differently on different machines, and is usually a high-overhead process. Nevertheless, connecting groups of sub-routines by conventional linking is often a better organization than that of separate programs. In particular, this organization is essential for cascaded processing such as neighborhood operations on an image. In this situation the separate-program organization would lead to as many passes through the image file as there are operations, while subroutines called in sequence would require only one pass. The question for linked sub-routines is how we can centralize support functions and make it easy to add sub-routines and integrate them with existing ones.

There is no difficulty in passing information about sub-routine capabilities across program boundaries—the description in the configuration file can be broken down by routine within program. Rather, the problem is that existing routines interact in an intricate pattern, and a new or altered routine must be allowed to participate without its author mastering very much of the complex code environment. We illustrate how this can be done for the composition of neighborhood operations.

Neighborhood operations in cascade can be performed on a very restricted portion of an image. There is always a "window" that, moving serially through a file, contains all the pixels needed to perform one step of the composite operation. Some operations in the cascade make use of the results of others as well as data from this window, but it is straightforward to arrange row buffers so that all of the necessary data is present at once. Control flow is more complex, since one operation (or sequence of operations) may have to be repeated before the next can proceed. An open-ended package requires that the routines which participate in the sequence be written as unit operations, transforming a fixed number of rows of input into a similar output.

To add a routine requires no understanding of the complex driver mechanism that adjusts buffer pointers and sequences the operations, but only the understanding of the parameter conventions for receiving unit input and delivering unit output. The existing routines are selected and called through tables, which must be updated to reflect changes. This organization is as flexible and easy to extend as might be imaged. The routine-name table is the basis for building the configuration file, so an independent program interacting with users knows which neighborhood operations are available, and what parameters each requires. Any sequence of these operations can be passed to the program in which the unit-operation routines are driven by the same table. The user-communication program can even determine memory requirements, and break the sequence up with intermediate files if not enough core is available for the necessary window.

PRESENT AND FUTURE PLANS

Our immediate goal is the production of an image-processing package design that is both transportable in the wide sense described in the first section, and of high quality. As

a practical demonstration of this design we imagine the following pieces of software:

1. Operating system interface kernels for several machines, along with documentation describing the implementation on a new machine. Operating system interface surround.
2. Preprocessors for RATFOR and for extended syntax checking of transportable code, themselves transportable.
3. An independent user-interface module capable of checking commands for and invoking any number of processing routines, their operations described and driven by tables. This module will include "help" information driven by the same tables.
4. A sample processing module implementing neighborhood operations with almost arbitrary cascade and parallel capabilities.

Once this design has proved itself, it will be appropriate to extend it to a full-blown image processing package by the addition of other processing modules. This task will be less difficult than the complete creation of a package, but it is not easy. The major advantage is that once the effort has been expended, nothing like it should ever be needed again—the package should be adaptable to new situations at very low cost.

REFERENCES

1. American National Standards Institute FORTRAN, X3.9-1966, New York, 1966.
2. Wulf, W. A., R. L. London and M. Shaw. "An introduction to the construction and verification of Alghard programs," *IEEE Trans. on Soft. Eng.*, Vol. SE-2, 1976, pp. 253-265.
3. Wirth, N., "The design of a Pascal compiler," *Software—Prac. & Exp.*, Vol. 1, 1971, pp. 309-333.
4. Kernighan, B. W., "RATFOR—a preprocessor for a rational fortran," *Software—Prac. & Exp.*, Vol. 5, 1975, pp. 395-406.
5. Baker, B. S., "An algorithm for structuring flowcharts," *J. ACM*, Vol. 24, 1977, pp. 98-120.
6. Kernighan, B. W., and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.
7. Poole, P. C., and W. M. Waite, "Portability and adaptability (sic)," *Advanced Course on Software Engineering*, F. L. Bauer (ed.), Springer-Verlag, 1973, pp. 183-277.
8. Munn, R. J., and J. M. Stewart, "RATMAC: Kernighan and Plauger's FORTRAN Programming Language," University of Maryland, Computer Science Technical Report TR-675, July 1978.
9. Ryder, B. G., "The PFORT verifier," *Software—Prac. & Exp.*, Vol. 4, 1974, pp. 359-377.
10. Boyle, J. M., and K. W. Dritz, "An automated programming system to facilitate the development of quality mathematical software," *Information Processing '74*, J. Rosenfeld (ed.), North-Holland, 1974, pp. 542-546.
11. Hamlet, R. G., and R. M. Haralick, "Transportable package software," University of Maryland, Computer Science Technical Report TR-706, October 1978.
12. Parnas, D. L., "A technique for software module specification," *Comm. ACM*, Vol. 15, 1972, pp. 330-336.

A data-handling mechanics of on-line text editing system with efficient secondary storage access

by SAKTI PRAMANIK

*Indiana University-Purdue University
Indianapolis, Indiana*

and

EDGAR T. IRONS

*Yale University
New Haven, Connecticut*

INTRODUCTION

The data-handling algorithm in many current editors² uses a text which is a continuous string of characters. In this technique the characters are moved directly from their input source to the text string by expanding it in the core buffer. The disadvantage of such existing systems is due to the core buffer constraint. The buffer can be extended to the auxiliary storage but will result in less efficient auxiliary storage accessing.

The data-handling algorithm presented here uses a text which consists of several substrings. Initially the text consists of only one substring which is the original version of the text. This text resides in the consecutive blocks of the scroll area of the tape; the characters carry smoothly across the blocks boundaries, producing a continuous string. No writing is permitted on this scroll area during the editing process. The characters which are inserted into the text for editorial changes are moved directly into a core buffer, producing new substrings. The substrings are logically interconnected in a linear fashion through a map. The map, in fact, contains pointers to all the substrings in the text.

DESCRIPTION OF THE EDITING PROCESS

The editing process starts with the original version of the text residing on the tape unit. Initially the map will contain pointers to this text string. An entry consists of the location of the string in the storage and its length, i.e., the number of characters in it. Figure 1 below, for example, shows that the pointer which corresponds to the string of characters "ABCDEF," consists of two integers, L_A , the address of the string starting with the character A, and the length 6. The string A through F is assumed to reside on the tape unit.

Suppose an input string "xyz" is to be inserted between the character positions D and E of the string as shown in

Figure 2. Instead of physically splicing in this string between D and E, it is moved into a buffer located in main storage. The entire string now consists of characters in the following sequence:

1. Characters ABCD on tape
2. Characters XYZ in core buffer storage
3. Characters EF on tape

We have given the name "substring" to each of these smaller strings (e.g., "ABCD") and the map of Figure 1 is updated to contain three pointers to identify each of the above three substrings. The first entry in the map (Figure 2) consists of the location of the first character A of the substring "ABCD" and its length, 4. The second entry contains the location of the first character x of the substring "xyz" and its length 3. The third entry consists of the location of the character E of the string "EF" and its length 2. The location of "E" is L_A+4 because it is four character positions to the right of the character A in the string "ABCDEF." A delete command simply updates the map. If the character "Z," for example, is deleted the length field of the second entry in Figure 2 is changed from value 3 to 2.

When the input buffer is full an overflow area is needed to save the characters currently being input. Instead of this overflow area the buffer can also be emptied onto a disk storage. In the latter case, however, the most recently input characters will no longer remain in the main storage; two input buffers can be used to eliminate this problem. One difficulty of transferring substrings onto the disk is that the substring locations in the map have to be updated. To avoid this update logical addresses are used for the substrings. Several of these substrings are stored in fixed size page. A substring location is now given by the page number and the offset of the substring in the page. To handle all characters uniformly, the initial text on the tape is assumed to be split into fixed length substrings each fitting into separate pages.

Address	Length
L _A	6

Figure 1—Map for the character string "ABCDEF"

Paging scheme is simple here because the content of a page is not directly updated when they are on the tape or on disk; this is done logically through the map.

The direction of data flow between the storage mediums is shown in Figure 3 below. The flow of data from disk and tape into core is required to regenerate any part of the text.

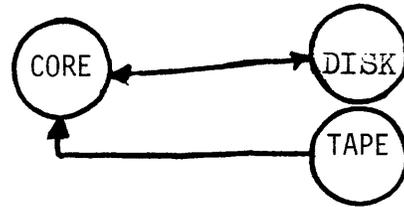


Figure 3—Data flow between storage devices

GROWTH OF THE MAP

The growth in the number of pointers in a map is due to the splitting of the substrings by the edit operations into more substrings of smaller sizes. The number of substrings created for each edit operation depends on the type of edit function (i.e., insert, delete and replace) and the editorial point. Figure 4 below shows the number of new entries created by each edit function. Some of the factors that influence the growth rate of the map are the type of editing task the user is performing, the familiarity of the user with the editor, and the type of edit functions available on the editor. Figure 5 below shows the growth rate of the maps on an experimental editor described in Appendix 1.

BOUND ON THE MAP SIZE

The size of a map depends on the growth rate of the pointers in the map and the time a user spends in editing the file. Let Y_i be the growth rate of the pointers in the ith map buffer. Then at anytime the total number of pointers in the buffer is less than or equal to t. Y_i, where, t is the total time spend in editing. The total number of pointers for editing N files is given by

$$t \cdot \sum_{i=1}^N y_i \quad (\text{for pooled buffer})$$

A B C D E F
 ↑
 / \ / \
 x y z

Address	Length
L _A	4
L _x	3
L _{A+4}	2

Figure 2—Updated map

The probability of overflow for a buffer size of N.α is as follows:

$$\Pr \left(\sum_{i=1}^N y_i \geq \frac{N \cdot \alpha}{t} \right)$$

where α is a constant.

Let {y_i} be a set of statistically independent, identically distributed random variables. Then by Markov inequality the bound is given by the relation

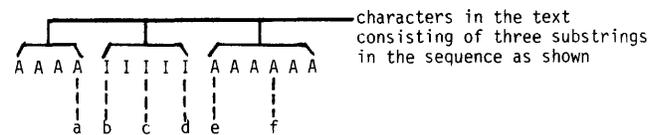
$$\Pr \left(\sum_{i=1}^N y_i \geq \frac{N \cdot \alpha}{t} \right) \leq \frac{t \cdot E_y}{\alpha}$$

where E_y is the expected value of the random variable y. In direct editing system the bound on the size of the buffer is computed as follows:

Let F_i be the size of the ith file. Then the storage requirement for N files is given by

$$\sum_{i=1}^N F_i$$

So the probability of buffer overflow for an in-core buffer



Note: A: characters on auxiliary storage
I: characters in the input buffer
a,b,c,d,e, and f denotes the editorial point.
Insertions are made preceding the character

Editorial Point	Insert	Delete	Replace
a	2	0	1
b	1	0	0
c	2	1*	0
d	2	0	0
e	1	0	1
f	2	1*	2

*Delete function does not produce any new substring if it is the end character of the substring to be deleted

Figure 4—Number of pointers created for each edit function

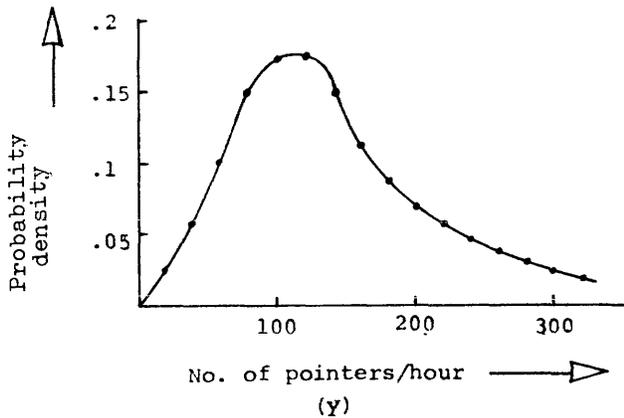


Figure 5—Probability density function of the growth rate of the pointers in the map

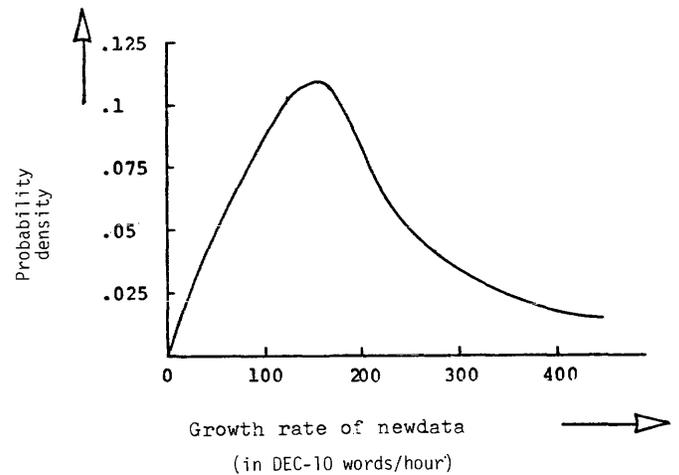


Figure 7—Probability density function of the growth rate of new data

size of $N \cdot \alpha$ is bounded by

$$\Pr \left(\sum_{i=1}^N F_i \geq N \cdot \alpha \right) \leq \frac{E_F}{\alpha}$$

where E_F is the expected value of the variable F . From Figures 5 and 6 one can compute the value of $t \cdot E_y$ and E_F to be 310 and 1140 DEC-10 words respectively. A session time of two hours is assumed. Thus $E_F \gg t \cdot E_y$ which implies that the map requires much less core storage than the direct editing system.

GROWTH RATE OF INPUT CHARACTERS

Figure 7 shows the growth rate of input characters per user in the experimental editing system. A typical editing session of two hours has been assumed. The bound on the growth rate, of course, is determined by the typing speed of the user. Each user in the experiment was using CRT terminals to debug and run the scientific programs. The actual

growth rate is much smaller than the speed at which a user can type. In fact, users working at display terminals appear to spend much of their time displaying different parts of the text and only occasionally making an editorial correction. Figure 8 below shows the overflow rate for different input buffer sizes. This is computed on the basis of the growth rate of input characters in Figure 7.

PERFORMANCE COMPARISON OF MAP EDITING WITH A DIRECT-EDITING SYSTEM

One commonly used data structure for a direct-editing system is a continuous string of characters stored in the logically contiguous blocks of the tape unit.² Here, a block is simply the transfer unit between the tape and the main storage. Locating any part of the string on the tape unit, however, takes longer time because the blocks are accessed sequentially. This problem is eliminated by storing the entire text in random access core storage. But the typical constraint in the size of core storage often allows only a part of

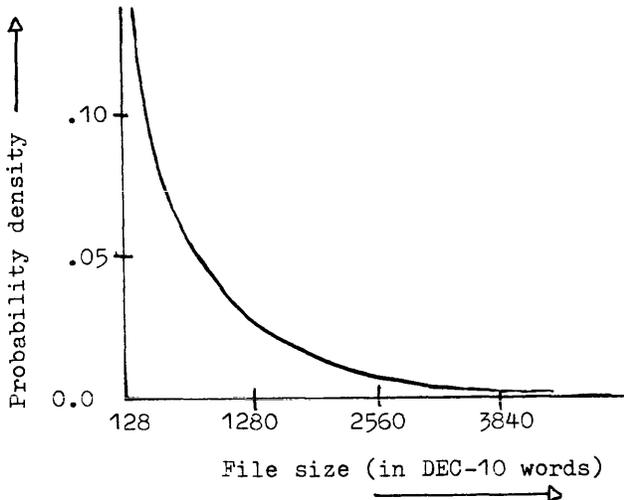


Figure 6—Probability density function of file size

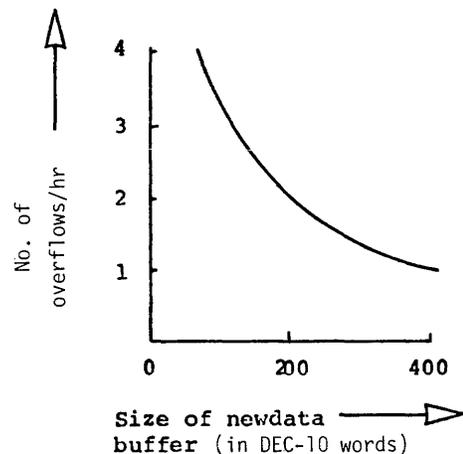


Figure 8—Overflow rate of the new data buffer

the text to remain in it while the rest is resident on the tape unit. Whenever the data to be edited or displayed is not in the core, the new data is read in from the tape unit by scrolling through the text. Many times this process requires the re-writing of data from the core to the same tape unit. These write operations slow down the scrolling process because the sequence of read and write operations involves reversing the tape motion. This reversal occurs more frequently when core buffer sizes are small. Writing is completely eliminated in map editing, thus considerably improving the access speed of the tape drive.

In a direct-editing system the insert or delete operation causes the character strings to expand or contract within the core buffer. This character movement can be reduced to some extent by inserting sparingly null characters in the text string¹. But the reduction of character movements by this method will depend on the nature of the programmer's editing pattern. In map editing these character movements in core is completely eliminated. In many tape systems writing in place is not permitted because positioning the tape at a precise location is not possible; here direct-editing is impossible. Usually, tapes are completely rewritten for making any update changes.

CONCLUSION

The increased processing time in this editing system is due to a considerable amount of fragmentation of the text. The substrings need to be remapped, sometimes, into one continuous string for improved editor performance. This remapping is done by copying the substrings into a scratch file on the disk.

APPENDIX 1

Experimental Editor

This is a CRT-oriented text-editor implemented on a PDP-10 time-sharing system at Yale University. When the user is editing a file, his CRT screen appears to be a window on the target file. This window can be moved up or down to display any part of the file on the screen. A cursor on the screen is used to point to the place where the editor is to perform an editing function such as inserting and deleting text. In most non-CRT editors this is accomplished by typing a line number or the surrounding text (context), neither of which is as convenient or natural as being able to directly use the cursor. This editor works in the user mode under the standard PDP-10 operating system and uses its file structure.

The general format of the edit commands is as follows:

(Enter) Parameter (Edit-function)

The angular bracket indicates a key on the terminal keyboard. (Enter), for example, is a key on the terminal which when pushed echos a `)` character on the terminal to indicate

that the system is in parameter mode. Any text now typed in goes into a parameter buffer without affecting the file. Parameter mode is terminated when any other function key is typed in and the parameters typed in are associated with that function. The command to delete next seven lines, for example, will be as follows:

(Enter) 7 (delete-lines)

The following commands are available on this editor:

File functions

These functions specify the target file to be edited.

(Enter) File-name (Make-File) Creates a new file whose name is file-name and gets it ready for inputting text.
 (Enter) File-name (Set-file) Gets an already existing file ready for editing

Screen movement functions

These functions move the window represented by the CRT screen with respect to the file.

(Enter) N (+Lines) Moves the window N lines forward in the file
 (Enter) N (-Lines) Moves the window N lines backward in the file
 (Enter) N (+Pages) Moves the window N pages forward in the file (a page is defined as 20 lines—the size of the screen)
 (Enter) N (-Pages) Moves N pages backward

Cursor positioning functions

These functions move the cursor around within the window without moving the window with respect to the file. Up, down, left and right cursor keys are grouped together on the keyboard and have a built-in auto-repeat which makes it convenient to position the cursor rapidly at any desired place on the screen. Echo to the CRT terminal is generated at monitor level by the teletype service routine. This ensures instantaneous response and the editor may not process the functions till somewhat later. These four function keys do not need any parameter.

Editing functions

These functions are used to update the text on the screen.

(Enter) N (Insert-space) Opens N spaces by expanding the text

<Enter> N <Delete-space> Deletes N characters from the text
<Enter> N <Insert-line> Inserts N blank lines in the text
<Enter> N <Delete-line> Deletes N lines from the text
<Enter> N <Pick> Picks N lines from the text and saves it in the pick-buffer
<PUT> Does not require any parameter. It puts the content of the pick-buffer into the text.

In all of the previous cases the editorial point is indicated by the location of the cursor.

REFERENCES

1. Weiner, P., I. Shingh, D. J. Mostow and E. T. Irons, "A CRT-Based Text Editing System," *Research Report #19*, Computer Science Department, Yale University, April 1973.
2. Wilkes, A., "An on-line algorithm for manipulating long character strings," *IEEE Transactions on Computers*, November, 1970.

How do we best control the flow of electronic information across sovereign borders?

by PETER SAFIRSTEIN

Data Processing Management Association
Arlington, Virginia

When all is said and done, the United States must answer one basic question: How do we best control the flow of electronic information across sovereign borders?

Essentially the question is one of power politics. Information is an entity which must be viewed as a form of power. When one considers that 50-60 percent of European domestic records are processed by American companies,¹ it becomes rather apparent that the United States subsequently exerts a great deal of power *vis-a-vis* Europe. Our "power," or ability to influence, does not end in Europe though, but extends rather significantly into the Third and Fourth Worlds as well, thus making our basic question a global one.

Electronic technology, by providing the means to accumulate, store, change and transmit information on an unprecedented scale,² has recently emerged as an issue of great importance as well as controversy. The complex myriad of issues involves personal data, economic data, financial data, statistical data, etc. and affects our daily lives through television, telephones, satellites, etc. More specifically, electronic technology's uses can be seen today in the airlines reservations network (SITA), the international banking network (SWIFT), creditworthiness evaluation data bases usually situated in the United States, global satellites providing remote sensing, corporation electronic information systems used in management controls for production, marketing, personnel, capital expenses and investment.³ These examples represent just a few of the controversial issues which governments are beginning to realize that they must confront.

Confrontation has emerged as a necessity for one overriding reason directly related to the recent emergence of electronic technology. That one concern is economic power. Europe has recognized that the uncontrolled flow of data creates hardships for its economy. European political pundits are frightened by the prospect of being cut off from vital data that is stored in foreign data bases. Hence, advisers warn of the dangers of dependence and advocate the creation of domestic data bases. Europeans are quickly developing their own data processing capabilities, yet they remain still far behind the Americans for primarily economic rea-

sons. It should come as no surprise then, that European governments feel an incentive to make their data processing industry competitive with that of the United States. In order to make a domestic data base network more competitive with the United States alternative, Europeans are implementing non-tariff barriers that take on the very appealing appearance of privacy legislation designed ostensibly for the protection of the principles of individual rights and national sovereignty.

It has been stated repeatedly that the principle questions involved in the transborder data flow controversy revolve around the issues of privacy and sovereignty. To view the controversy in terms of these areas is to beg the question. The essence of the dispute is, as stated above, *economics*. What we are dealing with is a question of economic domination which dwarfs the privacy and sovereignty questions to a status deemed secondary at best.

Privacy and sovereignty are relative objectives within the framework of international affairs. It may be well and good to espouse a policy that fosters absolute privacy and total sovereignty; however, a realistic appraisal must encounter the notion that in a world of interdependence and tradeoffs, there can exist only limited privacy as well as limited sovereignty. The days have passed when we could have viewed both individuals and nations as islands unto themselves.

Policies of privacy and sovereignty gain importance, though as convenient diplomatic tools to facilitate economic gains. That is precisely how Europe is using them.

Sweden, France, and West Germany have already implemented restrictive privacy laws and five other Western European nations are considering similar action. These restrictive laws are said to be the basis for the confrontation. Of course both sides want to have the economic edge and it is clear that the United States' business community is considerably opposed to any non-tariff barriers that might effectively exclude or hinder the opportunities within the European community. On the other hand, France, for example, does not want to be dependent upon the United States for raw data. France considers its present situation to be troublesome. William Fishman of the National Telecommunications and Information Administration noted France's pre-

dicament in a speech before the 4th International Conference on Computer and Communications in Japan: "at a recent OECD Symposium in Vienna . . . an official of the French government noted with alarm that France relies for a good deal of its macroeconomic planning on economic models, data bases and associated expertise located in the United States."⁴ The dispute is a clear one, with both sides defending vital national interests.

Yet, it is the author's opinion that Europe should not be the dominant area of concern for U.S. policymakers and businessmen. The teachings of history indicate that the United States and Europe have been in competitive situations before the postwar era and that reasonable statesmen have followed prudent guidelines of diplomacy toward resolution of the differences. Because of the interdependent status of the Western developed nations and the complex nature of the entire array of policy linkages, a compromise is inevitable.

The main domain of concern should be the Third World, yet the literature on the transborder data flow controversy focuses primarily on Europe. This is an unfortunate situation, for unlike the European case, we cannot depend upon reasonable men and prudent statecraft to win the day for us in the Third World.

The Third World is an area of significance to the United States (as is the Fourth World) because U.S. based multinationals have expanded into this area in large numbers over the past years so that they may take advantage of the wealth of natural resources and the plethora of cheap labor that exists in those regions. This considerable investment, though always at a risk, is now further threatened by the clamorings of the Third and Fourth World for a new economic order. In regard to information specifically, the major conflicts revolve around the precept of equity. Ali Shumo, of the Sudanese government, was referring to the upcoming World Administrative Radio Conference (WARC) where nations will reevaluate the allocation of the radio spectrum, when he stated to representatives of industrial nations, "You have 90 percent of the spectrum and 10 percent of the population. We have 90 percent of the population and 10 percent of the spectrum. We want your share."⁵

At the recent Spin Conference (Intergovernmental Conference on Strategies and Policies for Informatics) held in Torremolinas, Spain, Benin and Tunisia submitted a recommendation calling for aid in assisting developing countries to obtain "access to information located in national and international data banks in the more advanced countries." The impetus behind the motion is obviously a strive for equity. At that same conference, Bolivia, Morocco and Tunisia submitted a resolution recommending that "countries which have developed substantial data base capabilities should provide in their programs for the use of scientific and technological information resources by all interested countries."⁶ It is rather apparent that these nations have realized the "power" that information holds.

What makes negotiations with Third and Fourth World nations so difficult is their perception of capitalism and colonialism (with all its distasteful connotations) as synonymous. Hence, leaders of the nations can achieve great do-

mestic popularity and support by challenging the developed nations.

Historically, consider the example of Nasser of Egypt. It was said that his power and influence rest on his ability to symbolize Arab nationalism as an idea and as a political force. As he walks on the world stage, millions of Arabs see him playing the role they would like to play and doing the things they would like to do. . . . When he challenges the great powers and takes daring risks in the name of Arab rights and dignity, and gets away with it, the Arab masses feel an emotional lift and a satisfaction that no Arab leader has given them within memory.⁷

Stated simply, Nasser achieved immediate political consensus by maintaining an obstreperous foreign policy. His death was heavily mourned by the masses despite the fact that his failure to provide substantive leadership left the Egyptian economy and standard of living in shambles. It is quite conceivable for Third and Fourth World nations to neglect their nations' standard of living status for immediate political support. This becomes much clearer when one takes note of a major paradox of our times: there is a political disincentive for widespread economic improvements. This is so for two reasons: 1) When Third World nations increasingly move toward modernity, their population inevitably succumbs to discontent due to the principle of rising expectations. Citizens of a nation are told that political and economic subservience to technologically superior nations is necessary and should be tolerated, despite its distaste, for economic gains. When these economic gains are not immediately forthcoming to the populace due to a lack of an industrial base, there is political upheaval and popular support will sway to the leader that promises to throw the "exploiters" out. 2) The second economic reality is also unpleasant. Industrialized nations will insist that underdeveloped nations use the former's technology to industrialize, yet the situation will quickly emerge into one where everyone realizes that the more they, the indigent population, work and increase their production, the more prices in the marketplace will fall resulting in economic displacement.

Having stated these principles; how is this Third and Fourth World discourse relevant to the information community? It is relevant on a number of levels: 1) Industry will be faced with difficulties at times in exporting hardware and software to these areas, 2) these nations could conceivably pass laws making it difficult for multinationals to transmit vital data into and out of the country and 3) these nations could unite and sabotage multilateral conferences such as WARC.

In the final analysis, Europe and the United States will achieve agreement that is satisfactory because the economics of the situation mandate it. As was pointed out, the Third and Fourth Worlds are not easily impressed with compromises that will strengthen them economically. This makes dealing with the Third and Fourth World more difficult. It has become clear that economic assistance does not provide the United States with the leverage that it did with regard to different regions in different times.

These political and economic realities affect everyone, but the information community is right at the core of the dispute.

for the information community carries great weight in political and economic planning. Inevitably, the solution must be comprehensive and this is the major criticism that the author has of the present attempts to deal with the transborder data flow problems. Our present approach is a fragmented one that carries with it devastating consequences. Few people are looking at this problem *in toto*; rather they maintain specialists who are keyed into certain areas of controversy.

While it is true that the United States lacks a unified and coherent national information policy which would give legitimacy to a definition of information; it cannot be denied that recent major conferences have scheduled information policy questions on their agendas. It is in that regard that they are all somewhat similar. The conferences, however, take on a fragmented appearance:

- A UNESCO meeting in Paris in October, 1978, where a topic for discussion revolved around a government's right to control news generated and reported within its own borders.
- A World Administrative Radio Conference next September where Third and Fourth World nations will seek a far larger share of radio frequencies now used by the United States and other developed nations.
- A United Nations Committee meeting next spring and summer where many countries will push a resolution requiring TV broadcasters of one nation to get advance clearance from another nation before sending programs there directly via broadcast satellites.⁸
- The United Nations Conference on Science and Technology for Development meeting will take place next year where developed and developing nations will discuss the linkages involved between science and technology applications and the world economic order.
- The United Nations Conference on the Peaceful Uses of Outer Space will meet to discuss the implications of remote sensing. That is, one nation's surveillance of another nation's vital resources.
- OECD will meet to discuss the transborder data flow problem.

Further adding to the disparate attempts to come to grips with the problem:

- The President has ordered a review of the recommendations of the Privacy Protection Study Commission which contains transborder data flow considerations.
- The State Department Advisory Committee on Transnational Enterprises has set up a sub-group on data flows.⁹

This listing of the conferences dealing with information policy questions is not complete. When one considers how disjointed our approach to the transborder data flow problem is, one realizes that our method is dangerous as well as impractical. Undoubtedly, the United States will be pressed for concessions at each of these conferences. Should the United States concede certain holdings at each of these

discussions, as is expected, it will be confronted with a cumulative effect of astronomical proportions. The United States must negotiate from a position of strength which will not be facilitated in a framework in which the United States is pitted against hostile entities negotiating on a piecemeal basis.

What is desperately needed is a joining of public and private sector forces in a comprehensive, international conference where all the relevant issues will be discussed by all interested nations. This conference should be held in lieu of the list of conferences mentioned before. The United States should seek internationally acceptable parameters as to what information entails. Furthermore, the United States should seek, similar to the U.N. Law of the Sea Conference, a weighted vote so that bloc voting by the developing nations will lose some of its effectiveness.

For the United States to achieve its aims, three factors loom important:

1. Confronted by a demanding Third and Fourth World, the United States and Europe need to unite on the common ground that they have. While their interests, as explained previously, are presently not compatible, a quick resolution of their differences will strengthen their position in such a conference. Despite their problems, Europe and the United States should realize that they are closer to each other than they are to the LDC's and that a united position can only serve to strengthen their hand.
2. The United States and Europe need to take advantage of changes within the Third and Fourth Worlds. Some of the leaders of the developing nations have established such a firm political foundation in their lands that they can afford to be friendly toward the developed nations and the multinational corporations. The dichotomy between a Sadat of Egypt and a Qaddafi of Libya serves an analytically useful purpose. Both these lands are designated as developing states, yet no reasonable man can lump these two states together and call them identical. Sadat reflects a slow but changing nature toward prudent judgment within the Third and Fourth Worlds whereas Qaddafi represents the uncompromising element that is just as much a reality today as well. The key for the United States and Europe deals with their ability to put the developing nations at odds with each other. A weighted industrial nation vote linked with a divided Third and Fourth Worlds vote does make the prospects for success of American objectives seem encouraging.
3. Within the context of a comprehensive conference, the developed nations are afforded the luxury of tradeoffs and compromises that go into the formation of a unified and coherent international information policy. The comprehensive nature of the conference will tactically provide negotiators with linkages that are not subjects for discussion within a more limited framework. This will serve to guard against sacrificing elements on all issues which would only serve to weaken the United States' position in the world.

This comprehensive conference is a realistic solution. The position of the United States should be that it will negotiate on its own terms at a conference of its choosing. This it can do now, because it is U.S. possessions that the others want. A United States refusal to attend WARC with a comprehensive conference offered as an alternative would serve to signal to those Third and Fourth World nations that the United States is ready to take a firm stand.

A more limited comprehensive conference was given the support of Representative Goldwater in his Joint Resolution 1141 offered in the 95th Congress. Unfortunately, Mr. Goldwater's resolution is internally inconsistent. He claims:

Whereas communications and information transactions and associated activities are of major and growing economic importance to industrialized societies and developing societies which require rapid and reliable information transmission and processing systems to effectively advance commerce and governmental responsibilities; and whereas a growing divergence in national laws, regulations, and practices which impose special conditions, preferential rates, tariffs and technical standards, taxation policies and licensing, reporting, and disclosure practices, may threaten fair commercial competition and may jeopardize the widest sharing and utilization of information and knowledge made possible by electronic technologies; and whereas such a divergence threatens international cooperation and harmony: Now therefore be it resolved that the President of the United States shall 1) convene an international conference on Communication and Information not later than January 1, 1980, to which governments of the PRINCIPAL INDUSTRIALIZED NATIONS (emphasis added) will be invited to designate official delegations to attend. . .¹⁰

Mr. Goldwater explicitly states that developing societies are integral to communications and information transac-

tions, but then he calls for a conference that would not contain even all industrialized states, but only the *principal* industrialized nations.

That is an unacceptable solution. The conference that should be held must invite all interested parties. While the position that the United States takes should be strong, there still is a need to settle the dispute with all interested actors. Failure to do so would simply further the animosity that is felt between the developed and developing nations.

How do we best control the flow of electronic information across sovereign borders? Only when all sovereign states are involved in a major conference that supersedes the present fragmented approach.

REFERENCES

1. Tourtellot, Jonathan B., "A World Information War?," *European Communications* Jan.-Feb. 1978, p. 14.
2. Gotlieb, Allan, Charles Dalfen and Kenneth Katz, "The Transborder Transfer of Information By Communications and Computer Systems: Issues and Approaches to Guiding Principles," *American Journal of International Law*, Vol. 68, No. 2, April 1974, p. 229.
3. "International Data Flow: Shall We Have International Cooperation," Address by Oswald H. Ganley, Deputy Asst. Secretary of State for Technology at the University of Washington Conference on Communications, Dec. 12, 1977.
4. Fishman, William, *4th International Conference on Computer and Communications*, Sept. 26-29, 1977, Kyoto, Japan.
5. Burnham, David, "U.S. Is Worried by World Efforts to Curtail Info.," *New York Times*, Feb. 26, 1978.
6. *Computerworld*, Sept. 11, 1978, p. 4.
7. Cremeans, Charles D., *The Arabs and The World*, New York, Praeger, 1963, p. 25.
8. Otten, Alan L., "Controlling Global Information," *Wall Street Journal*, Sept. 7, 1978.
9. On-Line pamphlet advertising Conference in New York, 28-30 Nov.
10. Joint Resolution 1141.

Privacy and security in transnational data processing systems

by REIN TURN

California State University, Northridge
Northridge, California

and

TRW Defense and Space Systems Group
Redondo Beach, California

INTRODUCTION

Transnational data processing systems are international value-added, public, special-interest community, or private computer-communications networks that operate computers in several countries and provide computing services to users in these and other countries. Examples of such systems are the North American-based Telenet, Infonet, Mark III, Tymnet, and Datapac, and the European developmental Euronet and the Nordic Data Network. Among computer networks operated by international communities are SITA, an airlines reservation system, and SWIFT, a worldwide interbank financial telecommunications network. In development is the European Informatics Network (EIN) which will provide information services and data base access to users in the countries of the European Economic Community (EEC). Finally, private computer-communications systems are operated over international data carrier networks by multinational corporations in many countries.

To date, transnational data flows in the various types of computer communications networks have proceeded relatively freely among most of the industrialized countries in the world, subject only to economic and technical considerations, but these flows have not been balanced. Most of the international data processing services are offered by vendors that are located in a single country—the United States. Multinational corporations, likewise, tend to be headquartered in only a few countries. As a result, data are flowing to these countries for processing and storage from the countries that subscribe to the services of international data processing systems or that contain subsidiaries of multinational corporations. Thus, organizations in public and private sectors in many “computer-poor” countries depend heavily on vendors of data processing services, data processing industry, and computer communications networks located in and operated from abroad. It is not surprising, therefore, that a number of concerns over this situation have surfaced in countries from where such trans-

national data flows originate:

1. The possible erosion of the sovereignty of a country when large amounts of data about its economy, citizens, or government operations are transmitted abroad for processing or storage in data bases. Increased vulnerability to disruption of access to these data, and the lack of control over data processed and stored abroad can put a country in a position of significant dependency on foreign data processing services.¹
2. The increased complexity and technical and procedural difficulties in ensuring data security and maintaining accountability when data are transmitted in networks that may span several countries, may involve several transmission link technologies, and may be operated by several organizations that may be headquartered in different countries. As discussed later, international conventions regarding communications further complicate the situation.
3. The possible erosion of privacy rights provided to individuals in their home country when identifiable personal information about them is transmitted to be processed and/or stored in countries where privacy protection requirements are weaker. Thus, the possibility of “data havens” arises, where personal data about individuals may be maintained and used in ways that are violating the privacy protection requirements that exist in their home countries.
4. Potentially adverse effects on the development or continued existence of native data processing expertise and industry in countries that utilize foreign data processing services. The principal reasons for using foreign data processing services are economy, as compared to services available in the home country, and unavailability of the desired data processing resources or capabilities in the home country.²

These problems and the possibility that restrictive meas-

ures may be taken by data-exporting countries to alleviate them, are causing concern among vendors of transnational data processing services, multinational corporations and users of international computer communication systems that (1) free data flows (i.e., subject only to economic considerations and not hindered by the so-called "non-tariff" barriers) may be constrained by national laws, (2) international business, commerce and information exchanges may be severely discouraged when the necessary transnational data flows are continuously at risk of being curtailed by the countries involved, and (3) protectionist policies regarding international data processing systems and services will be instituted by data-exporting countries.

Extending privacy rights of individuals to countries where personal data about them may be processed and stored, and providing security to all data in transnational computer communication networks are two of the important questions that are currently in the focus of attention of national policy-making bodies and international organizations. The purpose of this paper is to examine the technical and procedural considerations that arise.

PRIVACY PROTECTION REQUIREMENTS

In the context of automated personal data record-keeping systems, the term "privacy" is used to refer to certain rights of individuals *vis-à-vis* collection, processing, storage, dissemination and use in decision-making of personal data about them. In Europe, the term "data protection" is used in the same sense, but the difference in terminology is causing some confusion. Privacy protection is achieved when these rights are granted, corresponding requirements are placed on record-keeping organizations, and compliance is enforced. Privacy protection takes on a transnational scope when individuals can exercise their privacy rights in other countries where personal data about them are handled or maintained.

National privacy protection laws

Privacy protection legislation has been enacted in several countries. In the United States, the Privacy Act of 1974 places privacy protection requirements on record-keeping agencies of the Federal Government.³ In the private sector, legislation has been enacted to provide privacy protection in the areas of credit reporting, education, bank records and in emerging EFT systems. Several states have enacted their own privacy protection and information practices laws.⁴ In Canada, privacy protection requirements are placed on agencies of the federal government by the Canadian Human Rights Act of 1977.⁵ In Europe, several countries have enacted privacy protection legislation, starting with Sweden in 1973 and followed by the Federal Republic of Germany, France, Norway, Austria and Denmark in 1977 and 1978.^{6,7}

The important dimensions of a privacy protection law include the following: (1) The scope of coverage (e.g.,

public sector, private sector, both, or various subsections of them); (2) data subjects covered (physical persons, associations of physical persons, legal persons); (3) types of record-keeping systems covered (automated, manual); (4) privacy rights granted to individuals; (5) privacy protection requirements placed on record-keepers; (6) mechanisms and authorities for enforcement (licensing, self-regulation, national commissions); and (7) systems of penalties. There are strong similarities between, but also considerable differences in, these dimensions between the laws enacted in the United States and in Canada, on one hand, and those enacted or pending in Europe, on the other hand. These differences can lead to problems in attempts to implement privacy protection requirements on an international scale.

To date, privacy protection legislation in North America has followed an area-by-area approach in the scope of coverage—laws have been enacted to regulate record-keeping by federal governments, other laws have been enacted by state or province legislatures, and additional laws provide for privacy protection in selected parts of the private sector. In Europe, however, privacy protection laws tend to be based on an omnibus approach—the same law and the same requirements apply to record-keeping organizations in both the government and the private sector. There are differences also in other dimensions, as depicted in Table I.

Privacy rights and requirements

Despite some of the differences previously described, privacy rights granted to individuals by the national privacy protection laws tend to be remarkably similar, and so are the corresponding requirements placed on record-keeping organizations. This is due to the use of essentially the same principles of privacy protection—the Code of Fair Information Practices as first stated in a U.S. government committee report in 1973⁸ and augmented by the Privacy Protection Study Commission in 1977,⁹ and the Council of Europe's resolutions on individual rights of privacy.^{10,11} Collectively, these principles can be stated as follows:

1. *Openness.* There must be no personal data record-keeping systems whose very existence is secret, and there must be a policy of openness about any organization's record-keeping policies, practices and systems.
2. *Individual access.* There must be a way for individuals to find out what personal data about them are on record and how they are used, and to see and make copies of those data.
3. *Individual participation.* There must be a way for individuals to correct or amend personal data about themselves.
4. *Collection limitation.* There must exist limits on types of personal data organizations may collect about individuals, and restrictions on the manner in which they collect these data.
5. *Use limitation.* There must be a way for individuals to

TABLE I—Features of National Privacy Protection Laws

Country	Scope	Covered data subjects	Enforcement mechanism	Explicit trans-national data flow requirements
United States	Federal gov. Some states Parts of private sector	Citizens Certain aliens*	Self-enforcement	No
Sweden	Both public and private sectors	All residents	Data Inspection Board	Yes
Federal Republic of Germany	Both sectors	All residents	Data Protection Commissioner	No
France	Both sectors	All residents	National Commission on Informatics and Liberties	Yes
Norway	Both sectors	All residents Associations	Data Surveillance System	Yes
Denmark	Both sectors (separate laws)	All residents	Register Board	Yes
Austria	Both sectors	All residents Legal persons	Data Inspection Board	Yes
Canada	Federal gov. Some provinces Some parts of private sector	Citizens Certain aliens*	Privacy Commissioner	No

* Aliens admitted for legal residence

prevent the personal data collected about themselves for one purpose being used for other purposes without their prior knowledge or consent.

6. *Disclosure limitation.* There must exist limits on external disclosures of personal data about individuals that record-keeping organizations may make, and there must exist legally enforceable confidentiality obligations of record-keeping organizations with respect to the use and disclosure of identifiable personal data.
7. *Accountability.* Record-keeping organizations must be accountable for their personal data record-keeping policies, practices and systems.

In privacy protection laws these principles are stated as rights that individuals have *vis-à-vis* record-keeping organizations, and requirements that the latter must comply with. As is to be expected, the national privacy protection legislation in each country tends to state the requirements and provide for compliance in a manner that is consistent with its existing legal system, traditional approach to establishing regulatory mechanisms and cultural setting. For example, some countries require licensing of record-keeping systems prior to permitting operation (e.g., Sweden), while other countries (e.g., the United States) rely on a corrective approach—those record-keeping organizations that fail to comply can be subject to court proceedings and penalties.

In general, national laws require that record-keeping organizations implement the following types of procedures and safeguards:

1. Inform individuals and the public in general about the existence and details of all record-keeping systems.

2. Notify individuals about existence of personal data records about them.
3. Establish procedures and facilities where individuals can inspect their own records; make records available in comprehensible form; establish procedures for reviewing challenges to data quality; provide means for correction and inclusion of rebuttal statements; and establish mechanisms for notifying prior recipients of disputed records of any corrections or additions.
4. Refrain from using data for new purposes unless explicitly permitted by law or by individuals concerned; establish procedures for requesting permission.
5. Keep accountings of all disclosures to external organizations such that the data could be traced (e.g., for sending corrections).
6. Establish procedures and means to ensure that personal data are collected by lawful means, and that they are appropriate and relevant to the purposes for which they are collected, and that they are maintained accurately, completely, and are up-to-date.
7. Maintain confidentiality of personal data by permitting access by only those personnel who need them to carry out their job functions.
8. Conform with security standards that afford reasonable protection to the installation, programs and data against accidental or willful loss or destruction, and against unauthorized access, alteration, or transfer.

A guiding principle in implementing privacy protection requirements by record-keeping organizations should be "easy access" by individuals to personal data about them—by mail or at locations close to their residences, at convenient times, without need to justify why they would like to

inspect their records, with the least amount of red tape in identifying themselves, and with the record-keeper, rather than the individual, charged with the burden of proof that the challenged data are actually correct or relevant.

International aspects

Any extension of one country's privacy rights to some other country to follow exported personal data involves the possibility that privacy rights in the "host" country are weaker, totally absent, or incompatible in some manner. For example, compared to some of the European countries, privacy protection in the United States at present is weaker, since only a small part of the private sector is covered, only citizens or aliens admitted for permanent residence are covered, and compliance is on a self-enforcement basis. Consequently, non-tariff barriers to international data flows and transnational data processing can arise when countries with strong privacy protection laws restrict data exports to those with weaker laws. While privacy protection should relate only to personal information about individuals, its scope can be extended to data that are indirectly related to individuals. Laws that also cover legal persons (e.g., corporations) have, of course, a much wider scope and can be used to restrict export of nearly all data.

The potential legal obstacles and conflicts in providing privacy protection on a transnational scale have prompted several international bodies to study the problem and develop solutions.¹² Since 1977 the Council of Europe is developing an International Data Protection Convention to establish a minimum set of privacy protection principles for all people in countries that ratify the convention (possibly the 21 countries in Europe, but others are also encouraged to join) without regard to their nationality, but permitting each signatory country to choose its own method of implementation.¹³ Several procedural considerations in the Convention are still unresolved, such as the question of which country has jurisdiction, and what international mechanism is best for enforcement.

Another activity toward harmonization of national privacy protection laws is taking place in the Organization for Economic Cooperation and Development (OECD) in France. OECD membership includes a larger community of countries (European countries, U.S., Canada, Japan, Australia, New Zealand) and, thus, could achieve harmonization on a larger scale. At the present time OECD is drafting a set of guidelines for establishing the basic rules governing transborder data flow and protection of individual privacy in transnational data processing systems.¹⁴ Finally, also concerned with privacy protection on an international scale and the associated concerns in transnational data flows are the European Economic Community (EEC) and the Intergovernmental Bureau for Informatics (IBI).^{15,16} The latter represents many of the so-called "third world" countries which, as data exporters, have many serious concerns regarding their position *vis-à-vis* the developed countries.

The recent conference on Strategy and Policies for Informatics (SPIN) brought out many of these concerns.^{16,17}

IMPLEMENTATION IN TRANSNATIONAL SYSTEMS

Implementation of privacy protection requirements in transnational data flow situations can be discussed with the help of a set of simple models. The following basic elements will be used in these models:

X—An organization in a "home country," *A*, that collects, maintains and uses personal data on individuals residing in *A*, subject to privacy protection laws and regulations that may be in force in *A*.

Y—An organization in a foreign "host country," *B*, that collects, receives, processes, stores, and/or uses, as the case may be, personal data about residents of *A*, subject to applicable privacy protection laws and regulations of *B*.

In each of the following data-flow situations, the basic goals from the point of view of privacy protection authorities in the home country *A* are to ensure that (1) individuals involved be able to continue exercising their privacy rights granted in *A*, (2) data quality and integrity continues to be maintained, (3) misuses of the data, as defined in *A*, are prevented, and (4) confidentiality and security of the data are maintained.

Transnational service bureaus

A common situation in transnational data flows is the case where *Y* is an international vendor of data processing services with computers and data bases in country *B*, as well as in several other countries, that services subscribers in several countries. *Y* may provide processing services only, or provide both processing and long-term storage of the subscribers' programs and data. A record-keeping organization *X* in its home country *A* may contract for *Y*'s services because of competitive economic advantage or because *Y* may offer resources or capabilities not otherwise obtainable.

The responsibility for complying with any privacy protection requirements in its home country, *A*, naturally belongs to *X*, independently of where the data themselves are processed and stored. Thus, in the public notices about its record-keeping operations, *X* must reveal information on any uses of foreign data processing services, and satisfy other requirements regarding the rights of its data subjects to inspect, correct and amend their records. *X* must also assure data quality (relevance, correctness, completeness and currency) prior to transmitting these data abroad. In this regard, it would be preposterous to require data subjects in country *A* to deal directly with the vendor *Y* in country *B* when they want to exercise their privacy rights, or to require that *Y* establish the data quality (e.g., to

assure completeness when engaging in data collection activities in country *A*), in the first place.

In contracting with *Y* for data processing or storage services, *X* must be assured that data accuracy is not diminished while in *Y*'s system, and that confidentiality and security are maintained so as to prevent any unauthorized disclosures or uses while in *Y*'s custody. For this purpose *Y* must implement appropriate procedural and technical safeguards in its computer systems and data communication networks. In general, it is not likely that *Y* would clandestinely copy and retain its customer's programs and data—it would not stay in business very long. Even then, confidentiality and security requirements, and prohibitions about retaining data or programs should be explicitly stated in the contract between *X* and *Y*. Any violations could then be handled in ways commonly used in the case of international contracts.

A more difficult situation arises when *X* deliberately attempts to evade privacy protection requirements in its home country, *A*, by maintaining illicit data abroad. For example, *X* may be a credit reporting bureau that collects personal information items not permitted in *A*. Here it is unreasonable to require *Y* to police the data processing requests of its subscribers and to make determinations whether or not they violate laws in the subscriber's home countries. To protect itself against unknowingly being an accomplice to activities illicit in *A*, *Y* may require that *X* certify in the contract with *Y* that its data processing activities are not illicit. In general, *Y* should not be held responsible for the uses of the data it processes or stores, but it should be obliged to inform privacy protection authorities of suspected violations.

Shared data banks

In a different situation, *Y*, the data processing organization abroad, may operate a data bank of personal information and make information available to an international community of subscribers who also contribute personal data on individuals in their respective countries. An example would be an international credit reporting bureau. *X* could be an international chain of department stores that gives credit to purchasers in many countries that, as a part of its contract with *Y*, sends credit status information on its customers in country *A* and in other countries. The individual records in *Y* are likely to contain personal information from multiple sources.

In the case of a similar system within a given country, privacy protection laws in that country give rights to individuals to examine their records, request correction, etc. directly at the record-keeping organization (e.g., the credit reporting bureau). In the transnational case the situation is more complicated. Again, it is not reasonable to require individuals to interact directly with the organization *Y* abroad. Enforcement of privacy rights afforded to individuals in *A* whose personal data are in *Y*'s records can also be done only in indirect ways. This is a good example of a

case in which international harmonization and cooperation between privacy protection authorities would pay off—each country would provide at least an agreed-upon level of privacy protection to everyone whose personal data are processed or stored in the country and, thus, all record-keeping organizations would be made to comply to essentially similar laws. In that case, any organization *X* in *A* that subscribes to *Y*'s services could be required to act as an interface between individuals in *A* and *Y*, and *Y* would be required by privacy laws and authorities in its own country to make their records available at *X*'s facilities and treat their requests with the same attention as it would treat those by individuals in its own country.

Maintenance of data confidentiality and security, and taking steps to assure that the existing data quality does not diminish in *Y*'s systems would be the procedural and technical responsibility of *Y*, but the quality of the data provided by subscribers *X* would be the latter's responsibility. Implementation and enforcement of this could be based on affixing unforgeable digital signatures to all data items to indicate their origins.¹⁸ Techniques for generating such signatures through cryptographic methods have recently been developed.¹⁹

Multinational corporations

For the case in which *X* is a subsidiary or an operating unit in country *A* of a multinational corporation *Y* that is headquartered in country *B*, *X* is likely to send to *Y* personal data on at least some of its employees who are also residents (citizens) of *A*. Unlike in the previous two models, these data are likely to be used abroad to make decisions about the individuals involved, not just merely to be processed and stored there. *X* must necessarily comply with any privacy protection requirements that are in force in *A*, and, thus, its employees would have access to their employment records to the extent specified by those laws. They would also have to be informed about personal data files maintained on them at the headquarters, *Y*, in country *B*. Again, *X* may have to serve as the interface between these individuals and the corporate headquarters (which it would do anyway as an element of corporate hierarchy) and be responsible for compliance with privacy protection laws in *A*. However, in this case it would not be unreasonable for the employees concerned to interact directly with *Y*, too.

To arrange for access to personal data about them at *Y* in accordance with their privacy rights in *A* when such rights are not available in the headquarter's country *B*, the privacy protection authorities in *A* may have to make special arrangements with *Y*. In the limit, they may prohibit transmission of personal data from *X* to *Y*, or recommend other regulatory restrictions on the operation of the multinational in country *A*. Of course, any such actions would involve analyses of tradeoffs of benefits to *A* of the multinational's operations in *A* as against possible limitations of

the privacy rights of its residents employed by the multinational.

Data collection abroad

There are at least two cases where data about residents of *A* are acquired by an organization in *B* directly without the role of some organization *X* in *A*: (1) in *B*, personal data on visiting residents of *A* are collected by some organization *Y* (e.g., the customs office or the police), and (2) in country *A*, employees of an organization *Y* located in country *B* are collecting personal data on residents of *A* (e.g., salesmen of a business firm). In both cases it is necessary to find out that such activities are taking place and, if legitimate, attempt to arrange privacy protection available in *A* to include personal data on residents of *A* maintained by these organizations.

In the first case, privacy protection to residents of *A* depends entirely on privacy protection laws that may be in force in *B*, or on bi- or multilateral agreements that may have been made between *A*, *B*, and other countries in this regard. Under such agreements, participating countries could compile lists of organizations in their countries that collect personal data about visitors from abroad, make these lists public on a reciprocal basis with other countries, and permit the basic privacy rights of access and request for correction to be exercised. Thus a certain balance could be established whereby international travelers would be assured that data are not collected about them for purposes of harassment or other actions against them when they are abroad or upon their return home.

Not much can be done in the second case when data collection in *A* in behalf of some organization *Y* in *B* is done clandestinely. When such collection is in the open, however, the privacy protection authority in *A* can insist that this be permitted to continue only if privacy rights in country *A* be extended to cover these data when maintained by the organization *Y* in country *B*. In some cases such collection may be prohibited entirely, as in the so-called *Reader's Digest* case in Sweden where the Swedish Data Inspection Board objected to the establishment of a data base in England on nearly all households in Sweden.

TECHNOLOGICAL CONSIDERATIONS

Technical concerns in implementing privacy protection requirements in both national and transnational data systems arise mainly in the areas of data security, maintenance of data quality, design and maintenance of data bases of personal information, auditing, and enforcement of compliance. In each of these areas there exist considerable technical capabilities, but also certain shortcomings of the state of the art. These must be kept in mind when privacy protection requirements are specified. Any problems that may arise in implementing national privacy protection requirements are likely to be amplified in transnational data systems where, unless perfect harmonization of privacy

protection laws is achieved, different countries are likely to have different requirements for data quality, confidentiality, etc., which may have to be reflected in the system design and operation.

Data security

Privacy protection principles and associated requirements that limit the use and disclosure of personal data, and that establish data quality and accountability responsibilities imply the use of data security techniques and safeguards. In general, data security encompasses the procedural and technical means for reducing the risk of unauthorized data disclosure, distribution, modification, or use, and of any physical damage to the computer communication system. Case histories of computer crime, attacks against computers by terrorists, and uses of computers to defraud their owners or the public at large underscore the reality of these risks. Indeed, concerns of the users of transnational data processing services over continuing availability and integrity of these services are not entirely unfounded.

After a decade of concern over data security and research and development of technological safeguards, there is now available a variety of techniques for implementing physical security, access controls within software and communications security:

1. Except for international standards, physical security techniques are well in hand for protecting computer installations against natural disasters, preventing unauthorized access, providing safe data storage, and setting up backup data bases, processing facilities, and communications links.²⁰
2. Software security techniques deal with protecting programs and data, and especially the operating system software against unauthorized access or modification due to hardware malfunctions of deliberate attempts. While software security cannot yet be guaranteed, the "security kernel" approach is promising and is being pursued actively.²¹
3. Techniques for unique, unforgeable identification and authentication of users have made considerable progress; the development of new approaches to digital signatures are particularly relevant in the context of transnational data systems.
4. New developments in cryptographic techniques, such as the Data Encryption Standard²² and the proposed public-key cryptosystems^{23,24} are making communications security easier and less costly to implement.

The use of encryption in transnational data communications is subject to more than just technological and economic considerations that complicate options for achieving effective communications security:

1. While international standards for technical aspects of data communication are being developed, no stan-

dards are available for data security in communications systems.

2. Existing international agreements, such as the International Telecommunications Convention of Malaga-Torremolinos, recognize the "sovereign right of each country to regulate its telecommunications," and reserve the right to monitor any communications and their contents. Thus the use of encryption in transnational data systems depends on whether or not governments involved insist on access to the keys or prohibit the use of encryption entirely in their communications systems. The use of satellite communications can reduce some of the severity of this problem, however.
3. On the international scale, there is no uniformity of legal prohibition against interception or diversion by private parties of data transmitted in telecommunications systems.
4. There is a wide variation in the technical characteristics and quality of the telecommunications links in various countries that may be involved in transnational data flows and, correspondingly, there are wide variations in their vulnerabilities from the data security point of view.

Among the state-of-the-art shortcomings that must be taken into account when formulating privacy protection requirements that depend on implementing data security techniques in various subsystems of a data network are the following:²⁵

1. Absolute security is not yet achievable in any automated, multi-user, resource-sharing data processing system or computer network, since it is not yet feasible to prove correctness of its operating system's design and implementation, nor can the hardware be guaranteed to be free of design flaws.
2. Physical security of a computer communication network, even though all techniques are well known, cannot be guaranteed against sophisticated penetration or overpowering attacks.
3. Personnel trustworthiness cannot be reliably predicted, assured, or maintained.
4. Currently-proposed encryption techniques appear to offer high levels of resistance against attempts to "break" them, but this has not been proven against massive analytical or trial-and-error attacks based on advanced computer technology.
5. The "confinement problem," leakage of sensitive data from a protected system using some externally observable system variable (e.g., the execution time of a program), has not been adequately solved and, thus, represents another security vulnerability.

Even a reasonable amount of security may be difficult to achieve in a provable manner in large and complex systems. Risk analysis methodology and techniques have not yet been developed to levels where they can be used with confidence since adequate guidelines on how to satisfy a

specified security requirement are yet to be developed,²⁶ and threat detection and monitoring techniques are not adequate for detection of covert penetration attempts while they are in progress. These shortcomings are multiplied in complex, transnational telecommunication systems. In statistical data bases that contain personal data in aggregate form, it is still possible to compromise such data by associating the summary data with identifiable individuals on the basis of other data that may be available and the context of particular situations.²⁷

Finally, a system of security safeguards is effective only when it is correctly designed and implemented, operates correctly, and is constantly monitored for lapses in performance. In a transnational computer communication system this implies standards, access to equipment that may be located in other countries, ability to audit the system's operation and, if parts of the system are under the control of third parties, cooperation from them when corrective actions must be taken. While many of the necessary procedures can be established within a contractual framework between the parties involved, certain aspects of enforcing these contracts may require effective international agreements and support.

Data base systems

The design of personal information data bases is strongly affected by privacy protection requirements regarding data quality, confidentiality and security, auditing and accounting. In general, it will be quite difficult and costly to restructure existing data bases to incorporate the additional data fields and data traceability capabilities that are implied. The design of new data base systems that incorporate these requirements is a much simpler matter.

Maintenance of data quality is an important privacy protection requirement in any personal data record-keeping system since these data are the principal and sometimes the only representation of the individual for decision-making purposes. Thus, these data must be (1) appropriate and relevant for the purposes for which they are used, (2) accurate, complete, and up-to-date, and (3) obtained by fair and lawful means. Determination of the relevance and appropriateness of personal data items to be collected is an important policy matter. Inclusion of certain data items and exclusion of others, and the time periods of retention can significantly affect the fairness of decisions. Some of the national privacy protection laws (including the Privacy Act of 1974) contain restrictions on the types of data items that may be collected, and the Council of Europe's draft Convention has proposed others, but more work needs to be done in this area, especially from the point of view of transnational data flows.

Achieving and maintaining accuracy, timeliness and completeness in personal data items that have been determined to be relevant is mainly a procedural and technical matter. It may be necessary to include in records additional data fields to indicate the age of various data items, to provide for appending any rebuttal statements and to provide for

tracking data disseminations such that corrections can be propagated to recipients as well as into backup files. Access limitation requirements imply establishing a selective access capability in the data base system. Special programs, possibly in several languages, may have to be written for translating data codes into natural language statements comprehensible to individuals who may wish to inspect their records. Associated with the above are various accounting requirements to (1) permit furnishing to individuals records of all uses made of their personal data, (2) permit propagation of corrections and rebuttal statements to prior recipients and (3) provide information for audits and compliance verification. While these data base additions are relatively simple to include in newly-designed data bases, retrofitting them into existing data bases is a much more difficult and costlier task.

Auditing and compliance verification

Auditing of a transnational data processing system for compliance with privacy protection requirements, and verifying that no one involved is taking actions to circumvent required safeguards, is a difficult technical problem. Indeed, there are several reasons why assuring total compliance may be technologically infeasible, especially in transnational data systems:

1. Auditing of the system software and its operation cannot be done with high confidence due to the complexity of tracing system's transactions, and the difficulty of understanding the software. Proving software correctness by formal means is infeasible for all but very small systems, although advances have been made in proving correctness of software design relative to specifications.²⁸
2. It is not possible to prove that the system's hardware, software or applications programs have not been specified and designed to covertly circumvent privacy protection requirements, or to mislead the auditors. Clever techniques could be used to hide deception, such as the use of "shadow systems" where auditors may find complete compliance, while illicit data are maintained in other parts of the system inaccessible to the auditors.

Thus, in general, it is not possible at the present time to verify positively without full cooperation of the record-keeping organization being audited that its automated record-keeping systems are in full compliance with privacy protection requirements. That is, it is not possible to prove that an organization does not maintain secret record-keeping systems or illicit data, that it permits individuals to inspect all data about them, that it is not engaged in clandestine data exchanges with other organizations, or that it is not using personal data in its system for purposes that have not been publicly announced.

Technical solutions to the auditing and compliance verification problems are now being developed²⁹ but they are

likely to be complex and costly. For example, it may be necessary to design and install tamper-proof accounting systems similar to the flight recorders now used in commercial aircraft. In the interim, it may be necessary to depend on the expectation that technically-motivated employees of record-keeping systems, especially those of transnational data processing systems, will inform privacy protection agencies or the media of suspected violations.

CONCLUDING REMARKS

Implementation of privacy protection and data security requirements in transnational data systems involves numerous unanswered questions about policies, procedures and technical means. From a brief examination of different types of transnational data processing situations it appears that individual privacy rights are easiest to exercise when organizations that send personal data abroad are made responsible for interfacing with the individuals involved. Organizations abroad that perform only the processing and data storage functions have to be responsible for maintaining data quality at the original level, and assuring data confidentiality and security.

Techniques now exist for satisfying most of the data security requirements, but shortcomings still exist, and their effective use in transnational computer communication systems may be constrained by the lack of international standards and by the policies of the countries involved. No technical problems appear to arise in designing new data base systems that include provisions for meeting privacy protection requirements, but retrofitting old ones may range from excessively costly to downright technically infeasible. No technological means are available to assure total compliance with privacy protection requirements.

Computer technology is advancing rapidly and new capabilities for record-keeping in national and transnational systems are certain to emerge. Some of these may enhance avoidance or circumvention of privacy protection requirements. Hence it is important that technological advances in record-keeping be monitored continually and the adequacy of existing privacy protection requirements be periodically reevaluated.

REFERENCES

1. *The Vulnerability of the Computerized Society*. Ministry of Defense, Stockholm, Sweden, 1978.
2. Carroll, J. M., "The Problem of Transnational Data Flow," *Policy Issues in Data Protection and Privacy*, OECD Informatics Studies No. 10, Paris, 1976, pp. 201-207.
3. *Privacy Act of 1974*, Title 5, U.S. Code, Section 552a (Public Law 93-579), 1974.
4. Smith, R. E., *Compilation of State and Federal Privacy Laws*. Privacy Journal, Washington, D.C., 1978.
5. *Canadian Human Rights Act*, Ottawa, Canada, July 14, 1977.
6. Wilk, C. (ed.), *Selected Foreign National Data Protection Laws and Bills*, OT Special Publication 78-19, Office of Telecommunications, U.S. Department of Commerce, Washington, D.C., March 1978.
7. Hondius, F. W., *Emerging Data Protection in Europe*. North-Holland/American Elsevier, Amsterdam, 1975.

8. *Records, Computers and the Rights of Citizens*, U. S. Department of Health, Education and Welfare, Washington, D.C., July 1973.
9. *Personal Privacy in an Information Society*, Report of the Privacy Protection Study Commission, Washington, D.C., July 1977.
10. *On the Protection of the Privacy of Individuals Vis-à-Vis Electronic Data Banks in the Private Sector*, Resolution (73)22, Council of Europe, Strassbourg, France, September 26, 1973.
11. *On the Protection of the Privacy of Individuals Vis-à-Vis Electronic Data Banks in the Public Sector*, Resolution (74)29, Council of Europe, Strassbourg, France, September 30, 1974.
12. Stadler, G., "Options for Privacy Protection in International Data Flows," *Data Regulation, European and Third World Realities*, Online Conferences Ltd., Uxbridge, England, November 1978, pp. 37-57.
13. Hondius, F. W., "The Work of the Council of Europe in the Area of Data Protection," *Data Regulation, European and Third World Realities*, Online Conferences Ltd., Uxbridge, England, November 1978, pp. 59-69.
14. Gassman, H. P., "The Activities of the OECD in the Field of Transnational Data Regulation," *Data Regulation, European and Third World Realities*, Online Conferences Ltd., Uxbridge, England, November 1978, pp. 177-184.
15. Carmody, F., "The Work of the European Community in the Area of Computers and Privacy," *Data Regulation, European and Third World Realities*, Online Conferences Ltd., Uxbridge, England, November 1978, pp. 105-112.
16. Bernasconi, F. A., "Informatics Integral to a New International Economic and Information Order," *Data Regulation, European and Third World Realities*, Online Conferences Ltd., Uxbridge, England, November 1978, pp. 113-121.
17. Korloff, G., "The New World Information Order," *Data Regulation, European and Third World Realities*, Online Conferences Ltd., Uxbridge, England, November 1976, pp. 195-206.
18. Norman, A. R. D., "A Scheme for Regulating Trans-Border Data Flows," *Data Regulation, European and Third World Realities*, Online Conferences Ltd., Uxbridge, England, November 1978, pp. 123-134.
19. Rivest, R. L., A. Shamir and L. Adleman, "A Method of Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, February 1978, pp. 120-126.
20. *Physical Security and Risk Management*, FIPS Publication No. 41, National Bureau of Standards, Washington, D.C., June 1974.
21. Popek, G. J., and C. S. Kline, "Issues in Kernel Design," *AFIPS Conference Proceedings Volume 47, 1978 NCC*, June 1978, pp. 1079-1086.
22. Morris, R., "The Data Encryption Standard—Retrospective and Prospects," *IEEE Communications Society Magazine*, November 1978, pp. 11-19.
23. Diffie, W., and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, November 1976, pp. 644-654.
24. Hellman, M. E., "An Overview of Public Key Cryptosystems," *IEEE Communications Society Magazine*, November 1978, pp. 24-32.
25. Turn, R., "Technical Aspects of Privacy Protection," *Proceedings, COMPSAC 78*, November 1978, pp. 229-234.
26. Glaseman, S., R. Turn, and R. S. Gaines, "Problem Areas in Computer Security Risk Assessment," *AFIPS Conference Proceedings, Vol. 46, 1977 NCC*, June 1977, pp. 905-910.
27. Denning, D. E., "Are Statistical Data Bases Secure?," *AFIPS Conference Proceedings, Vol. 47, 1978 NCC*, June 1978, pp. 525-530.
28. Feiertag, R. J., and P. G. Neumann, "The Foundations of A Provably-Secure Operating System (PSOS)," *AFIPS Conference Proceedings, Vol. 48, 1979 NCC*.
29. Ruthberg, Z. (Ed.), *Audit and Evaluation of Computer Security*, SP 500-19, National Bureau of Standards, Washington, D.C., 1977.

A modular approach to computer security risk management

by ROBERT P. CAMPBELL and GERALD A. SANDS

*Headquarters, Department of the Army
Washington, D.C.*

INTRODUCTION

Risk management is concerned with the identification, measurement, control and minimization of impact of uncertain events upon organizations that depend upon automated operations.¹¹ It is an analytical process with a large number of variables, many of which are unique to the environment under consideration. For this reason, there has not yet been developed within general state-of-the-art a single methodology broadly applicable to all risk management environments.^{1,10}

The proposed risk management model is designed to provide a framework responsive to the needs of most environments. To derive this model, many risk assessment schemes were reviewed and analyzed for their ability to satisfy a broad range of needs.^{3-7,9} While none were found which could meet these criteria, their different philosophies and schemes provided excellent background for development of the present model. Additionally, experts from industry were consulted and many security surveys, tests, evaluations and audits were analyzed for useful methods and procedures.

It is generally accepted within industry and the federal government that, given today's state-of-the-art in data processing and security, there does not now exist the capability to completely assure the security of sensitive automated operations. It is also agreed that the best method of controlling risk and achieving optimum security is through the use of some type of risk analysis or assessment process which provides decision-makers sufficient information to enable them to make informed judgments regarding the relative risks to these sensitive automated operations.²

Risk management is a management responsibility, involving all levels of management dependent upon sensitive automated operations.¹ These managers need to play an active role in the risk management process, making decisions regarding impact of risks upon support vital to operations of their organizations. Data processing has traditionally been a support function. However, it has become so indispensable and so much a part of overall business operations as to be virtually inseparable. As part of the risk management process, senior management must accurately ascertain its degree of reliance upon the data processing function.

Current risk management schemes are based almost exclusively on assessing the impact of uncertain events upon the data processing function alone, when in fact, losses

suffered through compromise of sensitive information, unauthorized exposure of personal information, or misappropriation of resources handled by automated systems are not borne by the data processing function. These losses are sustained by the owners of the data, the functional proponents and users of the system who, through senior management at various levels, must be represented in and dominate the decision-making process. Given the risk management responsibility, the data processing function (since it does not absorb the true loss) can reasonably be expected to accept the risk rather than divert resources from the data processing budget to counter security weaknesses.

Similarly, senior managers should carefully assess the impact upon the data processing function of attempting to implement necessary security improvements within existing data processing resource levels. In most instances, because of previous inattention to data processing security needs, considerable resource investment will be needed initially to bring operations up to minimum security standards. Past experience has shown that failure to supplement data processing resource levels for basic security needs has resulted in (a) minimal implementation of security improvements, or (b) considerable degradation of operational programs due to significant diversion of data processing resources. Managers should review these potential impacts and consider an initial supplement of data processing resource levels so as to eliminate undesirable contention for resources.

The specific features desired in a risk management model include the following:

1. It should be modularly structured so that discrete activities and tasks can be described and independently conducted, where appropriate. Some 40 separate modules have thus been identified in the accompanying model as a result of decomposition of basic activities.
2. It should be hierarchical, so that it can be applied in varying depths and degrees based upon the sensitivity and size of the data processing environment being assessed.
3. It should be iterative, logically allowing the entire process or a part of the process to be reinitiated or repeated as necessary.
4. It should prescribe, for each major step or activity, the participants, their responsibilities and decision-making authorities, with particular emphasis upon the involve-

ment of system users, proponents and senior management.

- 5. It should *not* require all factors to be reduced to quantitative terms. State-of-the-art is such that all factors cannot be reduced to discrete dollars and probabilities. Experience has shown that, except in highly specific situations, attempts to fully quantify all factors usually produce misleading results. Instead, managers and decision-makers must be involved in making qualitative judgments.

THE RISK MANAGEMENT MODEL (RMM)

The model described herein decomposes into sufficient detail to allow depth of analysis to vary with the specific nature of the problem. The less sensitive operation will require lesser analysis, while the more sensitive will require considerably more extensive analysis. The RMM (Figure 1) is composed of eight basic steps—Value Analysis, Threat Identification/Analysis, Vulnerability Analysis, Risk Analysis, Risk Assessment, Management Decision, Control Implementation and Effectiveness Review.

Value analysis (1.0)

The purpose is to determine the relative value of a facility or operation and its components for the purpose of evaluating its susceptibility to exploitation. The objective is to use value analysis to achieve an understanding of the likelihood that a particular facility, the information handled by that facility, and/or the function performed by that facility would be singled out for exploitation.

The value analysis process is based upon the following analytical procedures as illustrated in Figure 2:

- 1. Determine sensitivity of information handled (1.1).
- 2. Determine mission impact of loss or denial of support (1.2).
- 3. Estimate the asset value of automated resources providing support (1.3).

Determine sensitivity of information handled (1.1)

The basic analysis of sensitivity should begin at the application level. The objective is to relate each application (including data base manipulated) to a sensitivity level based upon the most sensitive type of data processed (e.g., privacy, asset/resource, proprietary). This analysis provides the framework for subsequent analysis (Task 1.2), so its detail and accuracy are important.

Information collected at the application level should be aggregated at the subsystem, system and finally at the data processing activity level in order to accurately assess the types of sensitive information being handled and the nature of processing performed on that information. The cumulative value of this analysis lies in the factual data compiled about application software and the data bases being manipulated. This should reveal the sensitivities of the data bases and the products derived therefrom. The bulk of this information should be collected from within the data processing functional area of assessment by system proponents and user organizations.

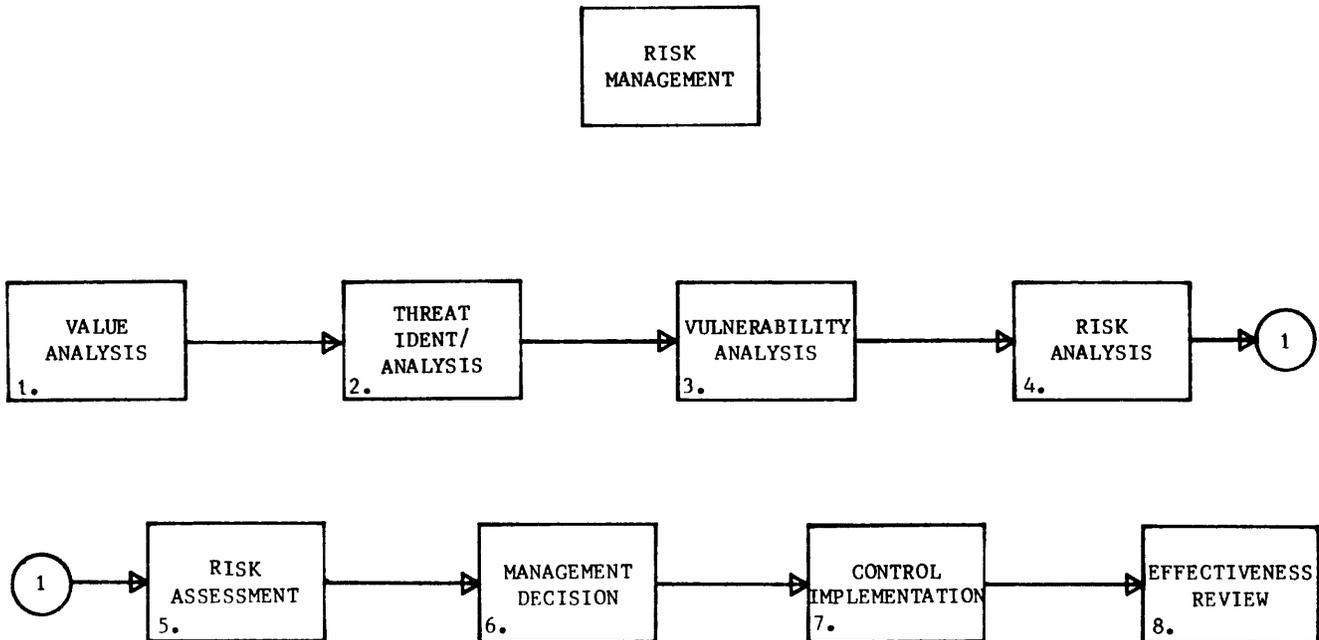


Figure 1—The risk management model.

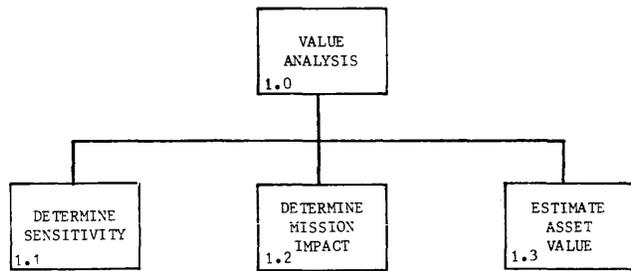


Figure 2—The value analysis process.

Determine mission impact of loss or denial of support (1.2)

Analysis should be performed of the general impact upon the mission of the organizations supported in case automation support capability is partially or totally lost for varying periods of time. This includes determination in as much depth as necessary (i.e., application, subsystem, or system level) the impact of any loss of automated support. Impact could be categorized as "Negate," "Major Impact," "Moderate Impact," "Some Impact", and "Little or None." If useful, period of time of loss can be quantitatively estimated for each degree of impact (i.e., one-day loss of support would be of "some impact," whereas loss of support for seven days can have "major impact").

Judgment as to the impact should be made by management officials that use the product of the application. If the users are not the sponsoring organization, the judgment of the sponsoring organization should also be solicited. Judgments on impact of loss should be rendered by higher-level management officials as applications are aggregated into subsystem, subsystems into systems and impact upon entirety of supported organizations is adjudged. Final review and judgments of impact should be made by the heads of organizations supported.

For purposes of value analysis, it is not intended that in-depth study and analysis be conducted into the specific losses and into quantification of losses. Such analyses will be conducted during later tasks. At this juncture, all that is required are subjective statements by management-level officials about the importance of automated operations to the mission of supported organizations.

Estimate the asset value of automated resources providing support (1.3)

While the risk assessment process should be oriented primarily toward the impact of loss upon supported organizations, the asset value of automation support resources is often so high as to merit consideration in this process. Consequently, fixed assets such as physical facilities, equipment, furnishings and other ADP-related assets (add equipment, supplies, software and documentation) should also be valued at either cost or replacement value, whichever is more realistic. The following should be determined in terms

of the dollar value of:

Physical Facility

- Building/Rooms
 - Office equipment and furnishings
 - Special equipment (fire extinguishers, degaussers, microfiche viewer)
 - Utilities (electricity, air conditioning, heat)
- Subtotal _____

ADP Equipment and Supplies

- CPU and memory units
 - Peripheral storage devices
 - Input/Output equipment
 - Specialized devices (off line plotters, COM)
 - Communication devices (controllers, modems)
 - Storage media (mag tapes, disk packs)
 - Remote I/O devices
- Subtotal _____

Software (cost or lease—use estimated purchase price. Do not include applications software unless purchased or leased)

- Systems software
 - Utility software
 - Other software (programming aids)
- Subtotal _____
Total _____

Nature of Supporting ADP Operations

- Estimated hours of processing (annually)
- Number of applications
- Size of data base manipulated

Threat identification/analysis (2.0)

The purpose is to identify threat agents as they relate to the particular facility or operation, and the manner by which they may be manifested. A threat is manifested by a threat agent using a specific technique, methodology, or spontaneous occurrence to produce an undesired effect upon a facility, operation, or system. Threats may be *actual*, in which case there is documented evidence of a threat or class of threats, or they may be *postulated*, based upon an assumed capability for which there is no hard evidence. Research and investigation of threats should be conducted jointly by facility personnel, security personnel, local fire department officials, and others able to contribute to this analysis. Figure 3 illustrates this process.

Identify threat agents (2.1)

Threat agents are considered to be environmental factors (tornado, hurricane, earthquake, flood, fire, rain, etc.), au-

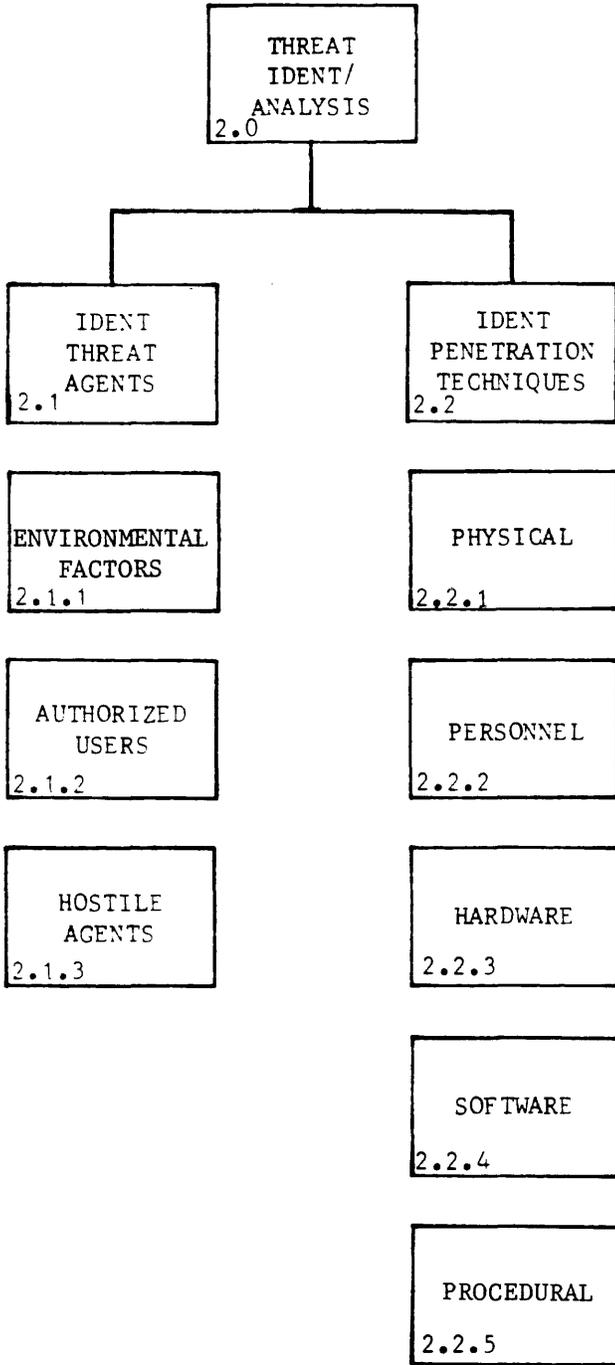


Figure 3—Threat analysis process.

thorized users (programmers, operators, customer) and "hostile" agents (anyone not an authorized user). The purpose of this step is to identify those threat agents likely to affect this specific facility or operation by reviewing generic threat agents, documenting evidence of those agents pertinent to the specific situation and/or drawing inference as to the seriousness of the threat they pose to the facility or operation.

Environmental factors (2.1.1)

Some areas of the country are more prone to certain environmental influences than others. Managers should be aware, for example, of the types of natural disasters likely to occur in their area. Some types of disasters, such as fire, are not geographically dependent, while others, such as tornadoes and floods, can be anticipated on a more regular basis in specific areas.

Chapter Two, Federal Information Processing Standards Publication 31 (FIPS Pub 31), June 1974, a U.S. Department of Commerce, National Bureau of Standards publication on Computer Security in ADP Operations, deals with the subject, "Anticipating Natural Disasters." Areas of the country particularly susceptible to the various types of disasters are clearly identified, and methods of preparing for these disasters are discussed.

In addition to natural disasters, appropriate concern should be directed towards the threat of mechanical and electrical equipment failure and curtailment of electrical power. They can easily disrupt operations as effectively as other more hostile types of threats.

Authorized users (2.1.2)

Authorized users and ADP personnel engaged in support operations can be considered as potential threats when they exceed their privileges and authorities and thus affect the ability of the system to perform its mission. Personnel granted access to systems or occupying positions of special trust and having the capability or opportunity to abuse their access authorities, privileges, or trusts will have to be considered as potential threats during this evaluation.

Hostile agents (2.1.3)

"Hostile" agents could be anyone, not an authorized user of the system, who by design, attempts to interrupt the productivity of the system or operation either overtly or covertly. Overt methods could include outright acts of sabotage affecting hardware and associated equipment, or more subtle efforts of destruction which could be accomplished through the manipulation of software, both systems and application.

Identify penetration techniques (2.2)

A threat agent mounts an attack against a facility or operation using a single or a group of specific techniques, methods, or spontaneous occurrences. A useful means of categorizing these techniques is according to the five sub-disciplines of automation security—physical, personnel, hardware, software and procedural security. For example, a threat agent mounting an attack against a facility by trying to subvert access controls would be using a physical penetration technique.

Physical (2.2.1)

Physical penetration implies use of a physical means to gain entry into the restricted areas housing the ADP system or any of its components. It could be a building, compound, room, or any other area designated as part of the ADP site.

Personnel (2.2.2)

Penetration techniques and methods generally deal with the subverting of personnel authorized some degree of access and privilege regarding a system, either as users or "operators." (As used here, "operators" are considered to be anyone involved in the operation of the system—analysts, programmers, operators, tape librarians, input/output schedulers, maintenance, custodial personnel, or the like.) They can be recruited by a threat agent and used to penetrate the system, operation or facility, or can themselves become disaffected or motivated to mount an attack. Highly technical functions performed with minimum effective supervision or control represent the types of circumstances under which an attack may be launched. For example, the vendor's hardware maintenance personnel are equipped with detailed technical knowledge and sophisticated diagnostic tools that can be used to penetrate the system. Generally, no one in the user organization has the technical knowledge base (a vulnerability) to effectively review and affirm the actions of the customer engineer.

Hardware (2.2.3)

Attacks can be mounted against the hardware for the purpose of using the hardware as a means of subverting or denying use of the system. A physical attack against the equipment, a "bug" implanted within the hardware or an attack against the supporting utilities are means of subverting the system by using the characteristics of the hardware. Hardware, as used in this category, generally includes any piece of equipment which is part of, or comprises the system, i.e., the mainframe, peripherals, communications controllers, or modems. It also includes indirect system support equipment, such as power supplies, air conditioning systems, backup power, water supplies and the like.

Software (2.2.4)

Software penetration techniques can be directed against systems software, applications programs, or utility routines. Software attacks can range from discrete alterations, subtly imposed for purposes of compromising the system, to less discrete changes, intended to produce catastrophic results such as destruction of data or important systems features.

Procedural (2.2.5)

Authorized users or "hostile" agents can effect procedural penetration due to lack or inadequacy of controls, or

failure to adhere to existing controls. Examples of penetrations include former employees retaining and using valid passwords, unauthorized personnel picking up output, users "browsing" without being detected due to failure to diligently check audit trails, and personnel removing material from the computer room when not authorized.

Vulnerability analysis (3.0)

The purpose is to identify possible weaknesses existing in the defenses of a facility, system, or operation. Through analysis of identified weaknesses and weighting each on the basis of exploitability, an appreciation of the likelihood that a threat agent will mount an attack to exploit a specific weakness or series of weaknesses will be achieved. Vulnerabilities are weaknesses in our defensive mechanisms, exposing that which we are trying to protect. Vulnerabilities, like threats, are also causative factors. Vulnerabilities are generally under our control and can thus be modified to limit the effectiveness of an attack. Figure 4 illustrates the vulnerability analysis process.

Identification of vulnerabilities (3.1)

Weaknesses or flaws in the design, implementation, or operation of the security controls of a facility, system, or operation must be identified, whether through analysis of the security controls alone, or as causal factors directly related to a previously identified threat. Vulnerabilities can be classified and their relationships to threats more easily identified using the same basic categorizations used for threats.

Weighting of vulnerabilities (3.2)

Vulnerabilities just identified should be considered in relation to one another and arrayed according to adjudged seriousness and potential degree of exploitability. These

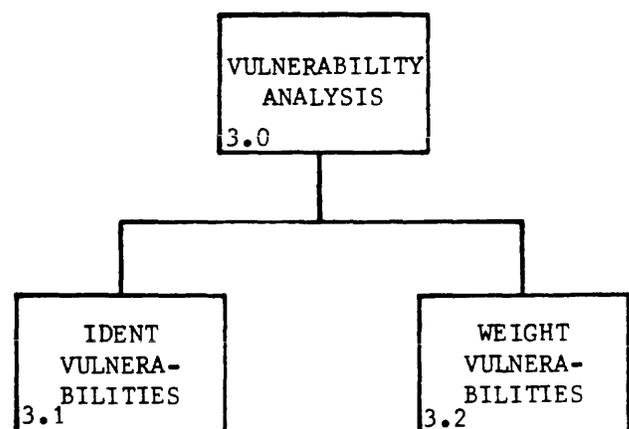


Figure 4—Vulnerability analysis.

judgments are made without consideration of the threat (threat relationship will be evaluated later) and based upon the experience of the appropriate management-level official. Not all vulnerabilities need to be weighted; general categorizations can be used (e.g., extremely serious, serious, minor). Weighting, at this point, should be pursued only to the depth and degree necessary to inform data processing management of the vulnerabilities and elicit management opinion as to their relative exploitability. It is not necessary that discrete values be assigned to each vulnerability or that probabilities and potential dollar losses be computed during this step. Informed judgments by data processing or other knowledgeable management-level official are desired and adequate.

Risk analysis (4.0)

The purpose is to identify specific undesirable events through analysis of the possible impacts of previously identified threats and vulnerabilities. The primary objective of this step is to determine the effects upon the system, facility, or operation caused by the interaction of threats and vulnerabilities. This is the "effect" portion of the "cause-effect" relationship. Identification of these relationships is extremely important to future analysis and documentation of impact, identification of countermeasures and cost/benefit analysis. Undesirable events are generally categorized as unauthorized disclosure of information, unauthorized manipulation of information, unauthorized use and denial of service, as depicted in Figure 5.

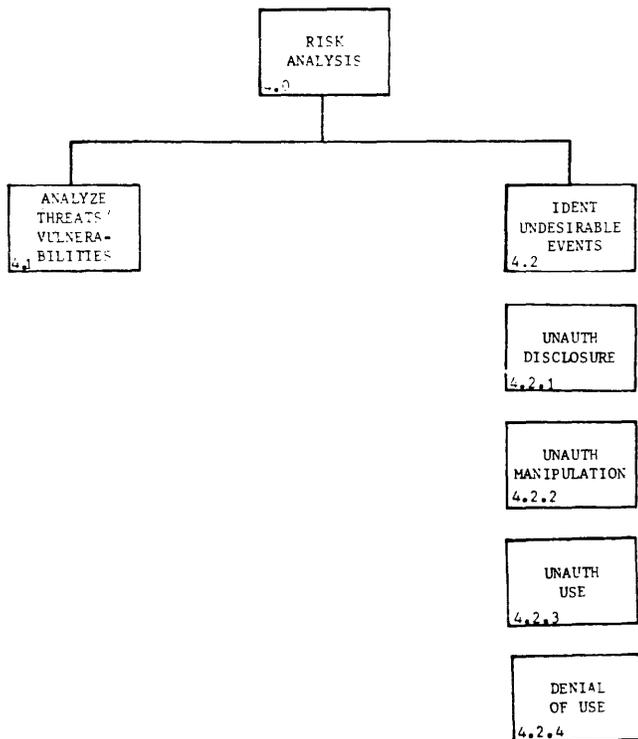


Figure 5—Risk analysis.

Analyze threats/vulnerabilities (4.1)

In this step, the formal relationships between threats and vulnerabilities are documented. The threat agent, the specific techniques or methods available to mount an attack and the vulnerabilities to be exploited (as previously identified), are now formally related and their relationship documented. The following two examples illustrate such relationships:

a. *Case 1*

Threat Agent—Authorized User.

Attack Methodology/Technique—Mounts an attack on the systems software by scavenging through memory workspace searching for passwords or other sensitive data.

Vulnerability Exploited—Residual data in memory working space is not erased, thus allowing sensitive information to be accessed by subsequent users not authorized such access; or

b. *Case 2*

Threat Agent—Hostile Agent.

Attack Methodology/Techniques—(a) Accesses remote terminal by bypassing physical security features; (b) Uses an authorized password, found on printout discarded in trash and unsecured User's Manual to access system.

Vulnerabilities Exploited—(a) Inadequate physical security of remote terminal area (area unguarded; simple key lock on door; no alarms or motion detectors); (b) Terminal software not disabled during non-duty periods when use is not authorized; (c) User's Manual left unsecured, revealing detailed sign-on procedures; (d) System software fails to suppress or block out password during sign-on/validation routine; (e) Sensitive trash left unsecured/unattended after duty hours.

As noted in Case 1, the relationship between threats and vulnerabilities may be linear or, as in Case 2, the relationships may be more complex.

Identification of undesirable events (4.2)

An undesirable event is considered to be a potential occurrence resulting from the activity of a threat agent exploiting a vulnerability. Whereas threats and vulnerabilities are causative factors, the undesirable events identified herein are the effects of these factors. Undesirable events are generally categorized according to effect upon the system, facility or operation:

- a. Unauthorized disclosure of information.
- b. Unauthorized manipulation of information.
- c. Unauthorized use.
- d. Denial of service or use.

In Step 4.1, the Case 1 situation describing the authorized user as the threat agent scavenging through working memory

covered only the causative threat/vulnerability factors. In this step, the effects must now be described. Continuing with the Case 1 scenario, the effect could be compromise of passwords and thus possible compromise of sensitive data accessed through use of those passwords.

Unauthorized disclosure of information (4.2.1)

Unauthorized disclosure includes access, either deliberate or inadvertent, to any type of sensitive information for which the individual has not been authorized access or does not have a need-to-know. In this context, sensitive information includes not only national defense, privacy, asset/resource, or proprietary information, but also critical system or operational information (passwords, system software, sensitive control processes) which should not be disclosed.

Unauthorized manipulation of information (4.2.2)

This category implies not only accessing information, but also manipulating and/or modifying it to destroy the integrity of the original information. Generally, a much more serious effect than unauthorized access, unauthorized manipulation is extremely difficult to detect and can insidiously deceive users or processors and contaminate other information. Unauthorized manipulation can also be a vehicle in fraud, and may take the form of creating unauthorized accounts and/or records.

Unauthorized use of information (4.2.3)

This category involves an authorized user using information for other than its intended purpose. Examples include using contractual or proprietary information for personal gain or using privacy information, such as home addresses and phone numbers, to solicit business.

Denial of service or use (4.2.4)

This category encompasses a wide variety of effects intended to partially or completely prevent use of the system

or its features caused by technologies ranging from blowing up the computer, destroying its source of power, causing the system to crash, or overloading the computer with tasks so as to make it sluggish and unresponsive. The primary variable in denial of service or use is the consistency of the effect lasting from micro-seconds to days or weeks. Denial of service or use may be either deliberate or inadvertent.

Risk assessment (5.0)

The purpose is to evaluate identified risks to determine their relative impacts upon the facility, the information handled, the processing performed, the support being provided and the mission accomplishment of the organizations being supported. The primary objective is to assess the severity of the identified risks and weigh the likelihood of occurrence so that they may be ranked according to degree of acceptability or unacceptability. This risk assessment task is the single most important activity in the Risk Management Model since it summarizes all previous risk analysis activities and presents these findings to appropriate levels of management for their review and evaluation. It is during this step that the senior management officials of the supported organizations are able to qualitatively confirm their reliance upon automated support for accomplishment of their mission. It is at this point also that the appropriateness of the assigned sensitivity designation should be reviewed and action initiated to change it, if necessary. The Risk Assessment Process (Figure 6) includes the sub-tasks of Assignment of Event Weights, Determination of Relative Impacts, Estimation of Likelihood and Ranking According to Acceptability/Unacceptability.

Assignments of event weights (5.1)

This task relates to the assignment of individual event weights indicating the severity of the effects of the undesirable events identified in the risk analysis process. This step is common to all risk analysis schemes and usually involves

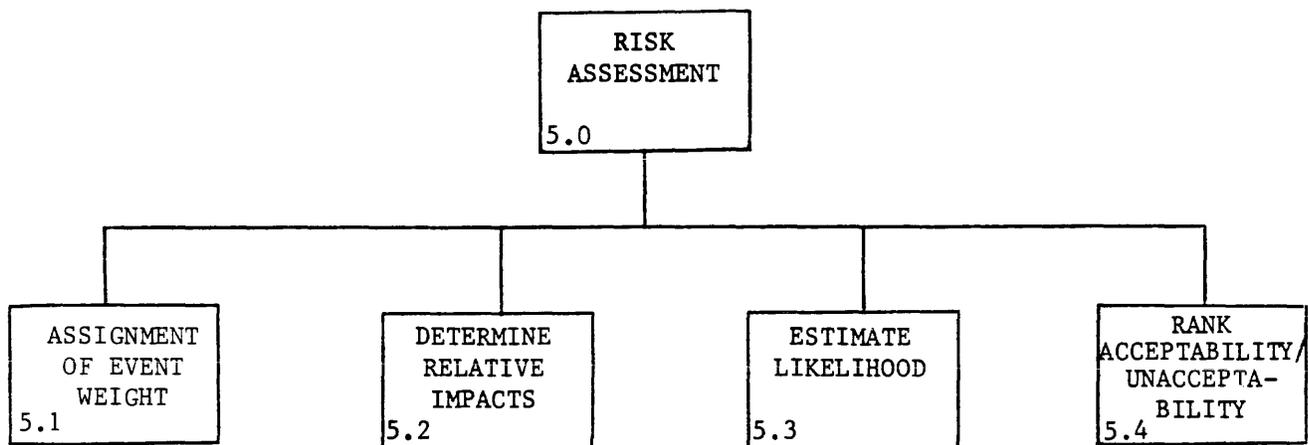


Figure 6—Risk assessment process.

some metric based upon probability of occurrence and estimated annualized dollar loss. As stated previously, use of quantitative probability and dollar-based metrics are generally not appropriate for the total measurement of impact upon the information being handled, the processing being performed, or the mission accomplishment of the supported organizations. Such metrics may have some utility in determining impact upon the physical facility. It is recommended that individual event severities be adjudged qualitatively by management-level officials using descriptive adjectives or a "fuzzy metric" and, at the most, a simple 0-10 or 0-100 rating scale. The success of this activity is dependent upon the quality of judgments made by management officials after they have had the benefit of all previous analysis. It is appropriate for replacement costs of physical facilities, equipment, supplies, data, or software to be estimated and entered into the decision-making process during this step.

Determine relative impacts (5.2)

This task formally assesses, for the first time, the relative impacts of all identified risks. The objective is to evaluate each in relation to all others to develop an appreciation for their relative severities. All quantitative and qualitative data collected thus far is evaluated and events scaled according to severity of impact. Again, qualitative descriptions such as "severe," "moderate," "light," possibly supported by a simple 0-10 or 0-100 rating scale, applied by the appropriate evaluators and/or managers, are the key inputs to this process. The end product is a table of undesirable events with qualitative descriptors of severity assigned.

Estimate likelihood (5.3)

This task requires analysis of individual undesirable events, and their interrelationship with other such events, to derive an estimate of the likelihood that each event will occur. Again, the application of qualitative descriptors, of "fuzzy metrics," such as "high," "moderate," or "low," to describe the possibility of occurrence is required. Previous weights and estimates assigned to threats and vulner-

abilities must be considered in arriving at these determinations. The end product is a table of undesirable events with qualitative descriptors of likelihood assigned.

Rank according to acceptability/unacceptability (5.4)

This task considers all previous analysis in arriving at an assessment of the relative degree of acceptability/unacceptability of each undesirable event. Qualitative descriptors, such as "unacceptable," "marginally acceptable," "acceptable," possibly supported by a simple 0-10 or 0-100 rating scale, and based upon the informed judgments of management officials, are the key inputs to this process. These officials must, for each undesirable event, consider the impact (Step 5.2) and the likelihood (Step 5.3) in order to adjudge the degree of acceptability/unacceptability. The end product of this step is a listing of undesirable events ranked according to acceptability/unacceptability, which will then form the basis of action in identifying and implementing countermeasures. It is in this step that senior management officials will be called upon to make informal judgments as to the effect of undesirable events upon performance of mission and thus assess their dependency upon this automated support.

Management decision (6.0)

The purpose is to evaluate identified risks according to degree of acceptability/unacceptability and, in consideration of the nature of the threats and vulnerabilities as they relate to risks, to identify and select countermeasures to effectively reduce the risk. The Management Decision Process (Figure 7) requires the use of management techniques such as cost/benefit analysis to determine those countermeasures which appear most effective and offer the best return on resource investment. Top management involvement is required to judge which countermeasures should be implemented and the priority for implementation. The Management Decision Process includes the sub-tasks of Risk Review, Identification of Countermeasures, Evaluation of Countermeasures and Selection of Countermeasures.

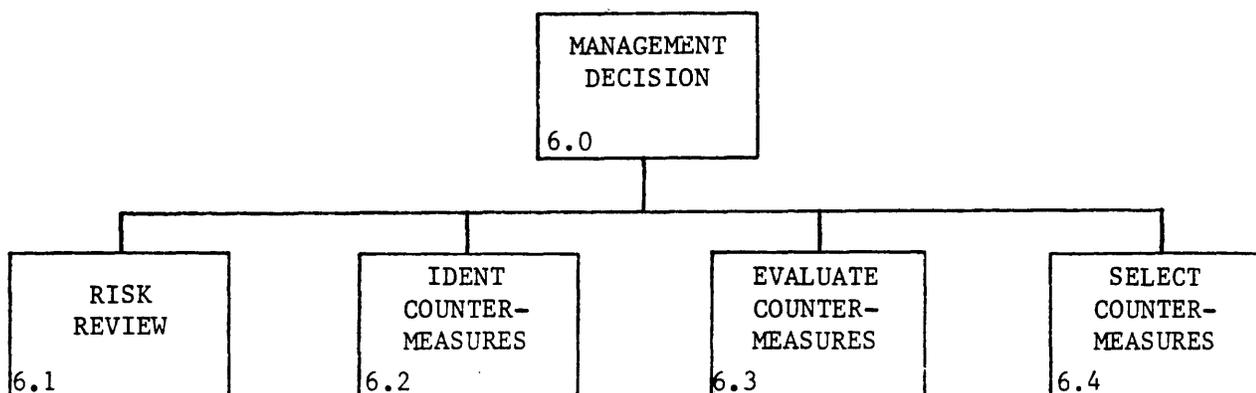


Figure 7—Management decision process.

Risk review (6.1)

This task requires the critical review of previously identified risks and related threats and vulnerabilities. It is a review and revalidation of the cause-effect relationships, previously identified in Steps 2, 3, and 4 of this Risk Management Model, for the purpose of determining which of the cause-effect factors would be most favorably influenced by counteraction. In attempting to alter the cause-effect relationship, countermeasures may be applied to either threats or vulnerabilities. However, countermeasures are usually applied against vulnerabilities since they are most directly under our control and are the easiest and most cost-effective to influence. Such traditional responses as the use of security guards, card-entry access control devices, or sophisticated password systems are directed toward correcting a weakness or vulnerability. In some very limited situations, the threat may also be reduced by taking counteraction; for example, denying the authorization to use the system to a person who fails to follow approved procedures or who abuses the system. The results of this task will be an initial indication of where to apply emphasis in development of countermeasures.

Identification of countermeasures (6.2)

In this task, the threats and vulnerabilities associated with a specific risk are analyzed to determine the best means of countering them. Specific actions are recommended which will lessen or minimize their impact. As pointed out previously, threats rarely can be cost-effectively countered but this course of action should not be discounted without some consideration. In this analysis, countermeasures should also be evaluated for effect upon other causal factors and these effects entered into the considerations. It is rare for a single countermeasure not to impact upon multiple events, so these relationships must be carefully explored so as to fully define their effect. Analysts must be careful that a countermeasure does not create a more serious vulnerability than that which it is intended to correct. The classic example of this situation is the implementation of software security measures requiring multiple, lengthy, passwords which end up being written down in wallets or displayed on walls and provide an easily accessible key to the system and data bases. Whenever possible, this task should present alternative measures for evaluation in the next step. To provide continuity with threat agent attack methodologies/penetration techniques (Step 2.2) and identification of vulnerabilities (Step 3.1), countermeasures should be identified and similarly categorized (e.g., physical, personnel, hardware, software, and procedural security).

Evaluation of countermeasures (6.3)

This step focuses upon alternative countermeasures, evaluating the relative costs and benefits of each in relation to the tangible and intangible losses which each is trying to

prevent. Using such techniques as cost/benefit analysis to display cost offset or detail cost avoidances attributable to a specific countermeasure, this step is intended to provide the informational base for subsequent decision-making by management-level officials. Full life-cycle costs and offset savings should be documented for consideration in the decision-making process. It should be noted that formal cost/benefit analysis techniques are inadequate for detailing intangible losses and benefits of avoidance of a loss, and therefore should be used only as a guide for decision-makers. No countermeasure should be dismissed by analysts based upon cost/benefit analysis alone, but rather should be presented to management-level officials for consideration in light of intangible benefits. Analysts will identify these intangible benefits as part of the evaluation process.

Selection of countermeasures (6.4)

This task is concerned with the gathering of pertinent information regarding proposed countermeasures in order that management may determine those that appear to be most effective and offer the best return on resource investment. Management will decide which should be implemented and the priority for implementation. While all efforts thus far have been directed towards minimization of risk, or "risk avoidance," it is here that management can decide to establish full or partial alternate or back-up capability (e.g., continuity of operations plans) or, if the risk potential is large enough, to no longer entrust the storage and handling of that information to automated processing.

Control implementation (7.0)

The purpose is to develop and execute a plan to implement those countermeasures required to improve the security and provide an acceptable degree of risk to senior management. This process requires the development of a comprehensive plan, the acquiring of management-level approvals necessary to commence implementation of the plan, the actual implementation of the selected countermeasures and testing of the effectiveness of selected countermeasures once implemented. The Control Implementation process (Figure 8) includes the sub-tasks of Development of a Plan, Approval of Plan, Implementation of Countermeasures and Test and Evaluation of Countermeasures.

Development of a plan (7.1)

To develop a plan to implement required countermeasures it will be necessary to establish priorities for their implementation. Generally, countermeasures should be implemented according to the severity of the undesirable effect being countered, as determined during preceding analysis. Using this as the basic criterion, other influences can be brought into consideration (e.g., ease of implementation, fiscal restrictions, personnel constraints, timing limitations).

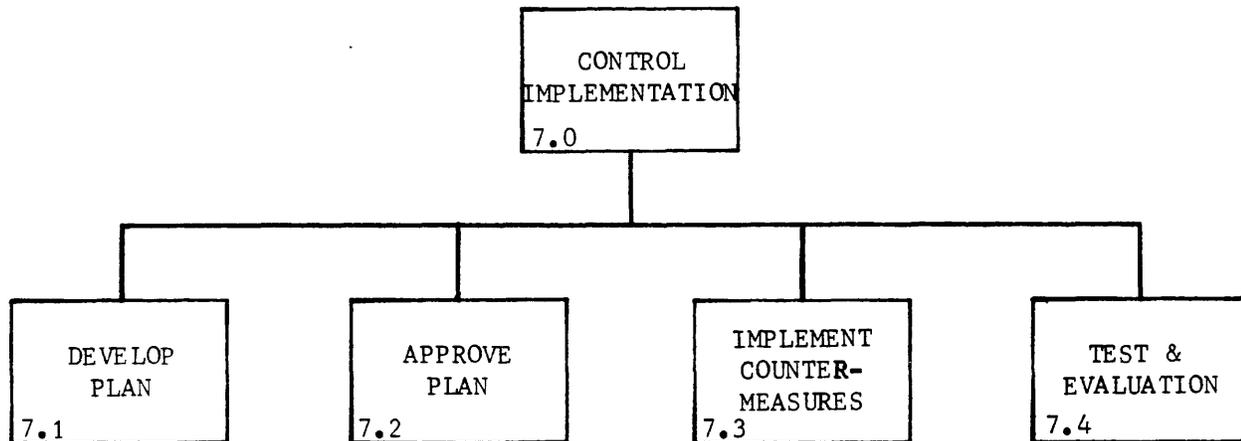


Figure 8—Control implementation.

Each countermeasure must be examined, both individually and collectively, to preclude duplication. The plan should identify specific resource requirements and the timeframe in which these resources will be required so that it may be used as the basis for input to planning, budgeting and other resource justification documents.

Approval of plan (7.2)

Once developed, the plan must be reviewed and approved by senior management. In this step, senior management must be given the opportunity to validate or change the priority of implementation based upon previous assessment of the risk. The level of management which controls the approval or allocation of resources must exercise decision authority on the plan so that it reflects their priorities and has their support. Senior management will be given the opportunity to assess the full resource impacts and make judgements as to how to best respond to resource requirements. It is important to give these resource requirements separate visibility rather than to attempt to bury them in normal data processing operating budgets. The decision can be made by management whether to handle the plan as a separate major program; to assign an allocated, "charge-back" type cost to users; or to include them in the data processing budget. Once these decisions have been made, formal justification can be made in budget and other resource management documents.

Implementation of countermeasures (7.3)

Once the planning documents have been completed action can commence on implementation of countermeasures. Some will be procedural or of such small dollar impact that they can and should be implemented immediately, while others have already been implemented during the risk assessment because of the severity of the security deficiency. Others will involve long-term, multi-year projects, such as the construction of an entirely new faculty. During imple-

mentation, it is imperative that the integrity of the plan be preserved and no significant deviations allowed without conscious management decision. The analytical risk assessment process should have resulted in the selection of countermeasures to support interlocking and mutually supportive defensive barriers for the operation. Arbitrary alteration of planned improvements will inevitably disrupt the cohesiveness of security features and lead to less than optimum results.

Test and evaluation of countermeasures (7.4)

Sensitive systems with stringent security requirements should have formal test and evaluation of significant countermeasures immediately prior to or during initial implementation. The purpose of test and evaluation is to ascertain, with reasonable assurance, that the proposed countermeasure will produce the desired effect and will not result in undesirable side effects. Testing can be supplemented by a formal review and evaluation period during the initial implementation phase during which the effect of the countermeasure is carefully monitored. In some cases, such as the implementation of a security front-end or a secure telecommunications monitor, it would be appropriate to constitute a formal test team, develop a test scenario and conduct a formal documented test of the effect of the countermeasure.

Effectiveness review (8.0)

The purpose is to periodically review the effectiveness of security controls and to assess the impact of those planned and currently being implemented in order to determine that they do what they were intended to do and have not created additional vulnerabilities. A formal audit of effectiveness should be conducted at a frequency determined by the sensitivity of the facility and its operations. As a result of the audit, management can make the determination whether or not to reinitiate the entire risk management process. Figure 9 illustrates this activity.

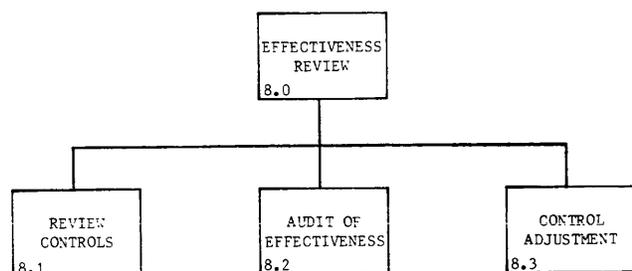


Figure 9—Effectiveness review.

Review controls (8.1)

Management is responsible for designing and implementing security controls. At this stage in the process, management must review all previously implemented controls to assure proper placement and operation. Particular attention should be paid to new controls and their integration into the overall security system.

Audit of effectiveness (8.2)

If deemed appropriate, full and formal review by outside experts should be conducted to determine the overall effectiveness of security controls. This can be accomplished by formal security survey teams and internal auditors. Audit is an important part of the risk management process, providing independent review and analysis for senior management.

Control adjustment (8.3)

This step in the process deals with the "fine-tuning" of the overall security system to address deficiencies and gaps found in the review and audit steps. Based upon the outcome of the effectiveness review, this step may lead to reinitiation of the total risk management process.

CONCLUSIONS

Most risk analysis/assessment schemes fall prey to the challenge of precise quantification of impact in terms of probabilities and dollar losses, preferring to be "exactly wrong rather than approximately right." To their detriment, these schemes focus primarily upon impact upon the data processing function (e.g., loss of or damage to equipment, cost to regenerate a data base, or accomplish a missed processing) rather than the effect upon the total business system, the real loser and bearer of risk. Perhaps the greatest weakness of these schemes is that they keep the risk analysis/assessment process a strictly data processing problem, inevitably victimized by the "conflict of interest" between security and productivity goals, whether real or merely perceived. This model attempts to address these problems and many of those raised by Turn, Gaines and Glaseman¹⁰ while providing the foundation for further work in this area.

REFERENCES

1. Army Regulation 380-380, "Automated Systems Security," October 1977.
2. Campbell, R. P., and H. R. Aaron. "Managing the Computer Security Problem," *Security Management*, November 1977.
3. Courtney, Robert H., Jr. "Security Risk Assessment in Electronic Data Processing System," *AFIPS Conference Proceedings*, Vol. 46, 1977 National Computer Conference.
4. Hoffman, L. J., E. H. Michelman, and D. Clements. "SECURATE—Security Evaluation and Analysis Using Fuzzy Metrics," *AFIPS Conference Proceedings*, Vol. 47, 1978 National Computer Conference.
5. Martin, J., "Security, Accuracy, and Privacy in Computer System," Prentice-Hall, Englewood Cliffs, New Jersey 1973.
6. McKenzie, R., and Z. Ruthberg. "Audit and Evaluation of Computer Security," *NBS Special Publication 500-19*, October 1977.
7. NBS FIPS Publication 31, "Guidelines for Automatic Data Processing Physical Security and Risk Management," June 1974.
8. NBS FIPS Publication 39, "Glossary for Computer Systems Security," February 1976.
9. Reed, Susan K., "Automatic Data Processing Risk Assessment," *NBSIR 77-1228*, March 1977.
10. Turn, R., R. S. Gaines and S. Glaseman, "Problem Areas in Computer Security Assessment," *AFIPS Conference Proceedings*, Vol. 46, 1977 National Computer Conference.
11. US General Accounting Office, "Managers Need to Provide Better Protection for Federal Automatic Data Processing Facilities," Report Number FGMSD 76-40, May 10, 1976.

The design and operation of public-key cryptosystems

by ERIC H. MICHELMAN

Intel Corporation
Santa Clara, California

INTRODUCTION

Recently, there has been a major advance in the area of communications security—that of a practical way to implement public-key cryptosystems (PKCS).

Public-key cryptosystems make use of a method for encryption and decryption in which the encryption key is different from the decryption key. Not only are the keys different, but revealing *one* doesn't provide any useful help in determining the *other*. A particular value for one key does of course "set" the value for the other, but for practical purposes, the other key is impossible to determine without additional information.

One major implication of this scheme is that encryption keys can be public; literally anyone can have access to them without threatening the security of encrypted communications. The decryption key is kept private; there is never any need for anyone to communicate his decryption key to anyone else. This eliminates the need for a secret transferral of keys, as is the case with conventional encryption methods. In systems using conventional methods, before two parties can communicate, they must agree on a key to be used for both the encryption and decryption. This key must be kept secret, as well, or someone who intercepts a message will be able to read and/or modify it. This agreement on a key is generally either very expensive and time-consuming or relatively insecure.

PKCS largely eliminate this problem. There is still a need for reliable transferral of the public keys, to be discussed, but it is minimal and, not requiring secrecy, may be done inexpensively.

The second major implication of PKCS is that it is possible to "sign" messages in a way that is unforgeable but easily verifiable. In other words, John can send Mary a "signed" message which she can prove came from him, even though the content and the encryption key may be public knowledge. This can be accomplished because the keys can be used in either order; encrypting with the private key and then decrypting with the public key produces the original message. To sign a message, John first encrypts the message with his private key (yes, his decryption key), then encrypts the result with Mary's public key (her encryption key). Mary now has a doubly-encrypted message which she first decrypts using her private decryption key, then decrypts again

using John's public key (his encryption key) to arrive at the English message. Since only John knows his private key, only John can create an encoded message which produces English when his public key is applied to it.

The idea of PKCS was first publicized in an article by Diffie and Hellman of Stanford.¹ In this article, they discussed the advantages that could be obtained from an algorithm which allowed the implementation of PKCS. They did not, however, suggest a specific algorithm. Such an algorithm was first published in April 1977 by Profs. Ronald Rivest, Adi Shamir and Len Adleman of MIT. This algorithm (the RSA algorithm) is based on the computational difficulty of factoring large numbers. This paper focuses on the use of that algorithm in particular.

THE RSA ALGORITHM

As described in the paper by Rivest, Shamir, and Adleman,⁶ the encryption and decryption operations are quite straightforward.

The encryption and decryption algorithms

Both the encryption key and the decryption key are composed of a pair of numbers, (e, n) and (d, n) respectively. To encrypt a message, the message must first be represented as an integer between zero and $n-1$. This can be done in any way that is convenient, such as stringing together the characters' ASCII representations. Long messages can be broken up into a series of blocks of this size, with each block treated as an individual message. The purpose of this is to put the message into numeric form as required by the encryption procedure.

The encryption is performed by simply raising the message to the e -th power modulo n . That is, the encrypted message C (for ciphertext) is obtained by raising the message M to the power e , and then taking as C the remainder obtained by dividing M^e by n . Decryption is similar to encryption. The d key is used where the e key is used for encryption.

Stated as formulas, the encryption and decryption algorithms are:

Encryption: $C=M^e(mod n)$ for a message M
 Decryption: $M=C^d(mod n)$ for a ciphertext C

The encryption key, then, is the pair of integers (e,n) and the decryption key is the pair of integers (d,n) .

Choosing the encryption and decryption keys

The first step is to generate two very large primes, p and q . To compute n , simply multiply p and q :

$$n=p \times q$$

The d key is a large random number which is prime relative to $(p-1) \times (q-1)$. This implies that

$$gcd(d, (p-1)(q-1))=1$$

where gcd means greatest common divisor.

Finally, the e key is computed from p , q , and d to be the "multiplicative inverse" of d modulo $(p-1) \times (q-1)$:

$$e \cdot d = 1 (mod (p-1)(q-1))$$

Although n will be revealed publicly, it will be virtually impossible for others to determine p and q by factoring n . This is due to the tremendous computational difficulty involved in factoring very large numbers. The size of the keys involved are typically, but not necessarily, about 50 to 200 decimal digits for p and q and about 100-200 decimal digits for n . Procedures for generating p, q, n, e , and d are presented in the Rivest paper.

Theoretical security of the RSA algorithm

In terms of theoretical security, using the fastest factoring algorithm known to the authors of the Rivest paper, the amount of time necessary to break the encryption scheme by factoring the n parameter is given in Table I. Rivest makes the claim that any method of breaking the scheme by mathematical analysis must be at least as difficult as factoring n as it would provide the factors of n . This table, taken directly from that paper, shows the time needed to factor n on a computer which performs one operation per microsecond.

There does exist the possibility that a faster factoring algorithm will be discovered. In that case, the time needed to crack the system with a given key length will decrease accordingly. However, it is likely that any improvement in the factoring algorithm will produce a marginal rather than a dramatic improvement, although that would be relative to

TABLE I

Digits in n	Number of Operations	Time
50	1.4×10^{10}	3.9 hours
75	9.0×10^{12}	104 days
100	2.3×10^{15}	74 years
200	1.2×10^{28}	3.8×10^8 years
300	1.5×10^{29}	4.9×10^{15} years
500	1.3×10^{39}	4.2×10^{25} years

the key length in use. An improvement of a factor of ten, for example, would be a matter of concern with a key length of 50, but not with a key length of 200. It should be noted, however, that this has not been proved to be a difficult problem. It has been worked on extensively by mathematicians, though, and it is generally believed that there is no easy solution possible.

KEY MANAGEMENT AND ADMINISTRATIVE SECURITY

With respect to key management and administrative security, PKCS compare very favorably to conventional systems. In systems where the encryption key is the same as the decryption key, two parties must agree on a key prior to sending messages. This secure transmittal of keys is generally an expensive and time-consuming process.

In establishing a common key in conventional systems, there is always a tradeoff between security and expense—the easier and cheaper ways to transfer the keys, such as by telephone or mail, are relatively insecure. The more secure methods, such as a trusted carrier, are expensive, especially over long distances. Often these systems are not as secure as they should be because of the expense involved and the tradeoff present between expense and security. The risk exposure may be greater than the cost of the desired security; however as is often the case in computer security, the risk of possible security failures in the future is preferable to the manager in charge than the immediate outflow of dollars necessary to provide the desired level of security.

In fact, relying only on the competence and trustworthiness of others in a security system introduces a great deal of risk. This is a primary problem with conventional encryption systems—even using a courier to transmit keys is only as secure as the courier, unless special precautions are taken such as using tamper-proof, locked containers to transport the keys.

With PKCS one still needs to be careful, particularly concerning the authenticity of encryption keys that are transmitted over the communication lines. *It is important to note that there is little security in a system in which people happily broadcast their encryption keys for the other users as often and whenever they like.* With such a system there is nothing to prevent an intruder from tapping into the communications lines and substituting his own encryption key for the legitimate keys sent by users. Other users receiving these encryption keys will then unwittingly send messages encrypted with the intruder's key. The intruder can then read the messages (having tapped into the communication lines), retransmit them with the intended user's real encryption key (which the intruder had intercepted), and even alter the message and then retransmit it. None of this would be detectable by the users.

The suggested remedy for this problem is to have a trusted "system controller" keep track of all the users' encryption keys. With an encryption/decryption key pair of its own, the system controller can send users signed messages indicating what the encryption keys of other users are. As an impostor could not forge the system controller's signature, the users

can be assured of the authenticity of encryption keys received in this manner.

Implementing this scheme requires that each user be able to transmit his encryption key to the system controller in a reliable manner so that the system controller can then relay it to other users. It also requires that each user be sure of the system controller's encryption key so that he can verify the system controller's signature. These keys cannot simply be transmitted over the communications lines for the reason just discussed.

If both keys (the system controller's and the user's) had to be transmitted externally to the system, the requirements would be similar to those with conventional encryption methods. That is, a secure external transferral of keys would be necessary, although with PKCS secrecy would not be required (we are, after all, dealing with just the public keys).

Fortunately, this can be avoided if two conditions can be met. First, the system controller's key needs to be made public knowledge. This shouldn't be very difficult as there is just one for the entire system. Second, there must be an appropriate action available to users if they discover a security problem. This also seems likely; however, if only the first condition is filled, then one can run into the problem of being guaranteed detection of security problems but not being able to do anything about them. These conditions will be further discussed later. Assuming they are filled, users can send their encryption keys to the system controller over the insecure communications lines in the following manner.

To establish a new encryption key, a user first sends it to the system controller. The system controller then signs it and sends it back to that user. As the user knows what the system controller's encryption key is, he can verify the signature on the returned message and thus be sure that the system controller has accurately received the key. If the returned key isn't accurate, then the user knows that there is an active security threat and can take appropriate action.

To prevent an impostor from sending a new key to the system controller purporting to be from someone else, the following restrictions are necessary. First, key changes take effect at prespecified times, perhaps the same time every day, week or month. At a given time before the key changes are scheduled to go into effect, the system controller stops accepting key change messages. It then sends a signed message to each user indicating what it thinks the user's key is for the next period. If there is a discrepancy—either a key was incorrect or the message wasn't received—the user takes appropriate action. All confirmation messages that are sent by the system controller should be date- and time-stamped to prevent an intruder from recording one and playing it back at a later time.

In a physically-insecure communications system, it is not possible to prevent the recording and/or modification of message traffic (presumably in its encrypted form), or even denial of service (perhaps by being as crude as just cutting the line). It is possible, though, to meet the following goals:²

1. Prevention of release of message contents.
2. Detection of message modification.
3. Detection of denial of service.

The procedures just described for use in a PKCS meet these three goals without requiring any secure external key transferrals, either between the users themselves or between users and the system controller (with the exception of the one public encryption key for the system controller).

It might seem that an undeservedly large amount of effort is devoted to changing keys with these procedures. Traditionally, key changes have been very important, though. They limit the potential exposure in case of a compromised key to the time until the next key change. The time that a cryptanalyst has to crack a key while it is still active is limited. Also, often after having given someone else a key, a user wants to revoke access by changing the key (e.g. after an employee goes to work for a competitor). In applications where the information transmitted is sensitive for only a short time, one also has the option of using more frequent key changes in conjunction with shorter keys to decrease the cost of encryption and decryption. While the shorter keys would be easier to break, there would be less time to break them while they are still in use.

The system controller referred to here is expected to be contained in either the central computer or the communications links, as appropriate. Little functionality is required other than maintaining a list of user encryption keys and communicating these to users in a signed form. It is necessary, though, that the integrity of this list be ensured although it need not be kept secret. The system controller's decryption key does need to be kept secret, of course.

The two critical assumptions in this scheme are that the system controller's encryption key can be made public knowledge and that appropriate corrective action would in fact exist if a user found that his line was compromised. As the system controller will presumably have only one key for the entire system, it shouldn't be very expensive to make that key public knowledge with respect to the system's users. This can be accomplished in many ways—via company newspapers, the *Federal Register*, public newspaper ads, etc. The success of this phase will be easily and immediately apparent to the system operators and if there is a problem there will be actions that can be taken, ranging from not changing the system controller's key to shutting down the system, depending on how serious the situation is thought to be.

The problem of being sure that a user will be able to take appropriate action in case of a security problem is a bit stickier. If the user finds that his new key confirmation message hasn't come or showed an incorrect key, he may assume that an impostor has substituted a new key. This means that the user must contact the system controller or (human) operators before the new keys are to go into effect or the other users will start using the impostor's key to send encrypted messages to that user. The seriousness of this situation depends on the time constraints and the physical situation. In some cases it is clear that this wouldn't be a concern at all. Such a case would be a communications system that connects sales offices in American cities with the company's main office and in which keys are changed once a month with a two-week clearing period before key changes go into effect. On the other hand, if an operative in

a hostile country had five minutes to contact his home office, he shouldn't be optimistic. We assume physical force isn't an issue, since by that method an intruder could also gain access to everything via the user's own terminal.

It seems likely that this wouldn't be an either-or constraint in most applications, but more of a limitation on how often the keys can be changed. Two weeks to take corrective action should be long enough to guarantee success in almost all applications, while five minutes would be cutting it close in most applications. Where one can safely draw the line depends on the application. It might very well be that keys can be changed once a day (with a few-hour clearing period for the new keys) in the vast majority of applications, especially if the protocols are largely automated.

Obtaining other users' keys

To communicate with another user, a user obtains the appropriate encryption key from the system controller. This can be done in a number of ways, all relying on the integrity of the system controller's signature to prevent an impostor from posing as the system controller and sending one system user a false encryption key for another user.

One arrangement would be for the system controller to issue signed messages indicating what the encryption key is for any given user. These "certificates" may be traded about among users with the authenticity guaranteed by the system controller's signature. The major disadvantage to this scheme is the problem of outdated certificates when a key is changed.

An alternative similar to certificates is for the system controller to periodically issue a list of the system users' encryption keys. Keys could then easily be changed on new lists, but only when a new list is issued. This sort of arrangement would likely be largely automated within the communication system, with the lists being transmitted as messages signed by the system controller.

Another alternative, but certainly not the last, is for users to request the appropriate key from the system controller prior to communication with other users. This could be a useful adjunct to abbreviated lists, with each user being supplied a list of the most likely communicants and requesting the keys of the others as the need comes up.

SIGNATURES

Uses

There are two primary uses for signatures in communications systems—future verification and immediate identification. Future verification is potentially very important in commercial systems in which users may want to effect a contract over the system. In this sense the signature is just like a written signature: its purpose is to prove at a later date that the sender did in fact author the message. This is valuable and particularly important among mutually suspi-

cious parties, such as two banks executing a contract over communications lines.

Immediate identification is important for a more technical reason. In an electronic communications system, if someone taps into the communications line he can originate a message purporting to be from someone else. However, with the use of PKCS signatures, a receiver of a message can verify the sender's identity. This is important for both mutually suspicious parties and those who are not. In communications between two trusted parties, one still wants to ensure that he cannot be impersonated.

Added cost

As always, nothing is free. There is a potentially-substantial added cost to using the signature capability, depending on how it is implemented. With a hardware implementation, the added cost is either an extra encryption/decryption device on each terminal and communications link or extra time consumed by cycling messages through one encryption device twice. This applies to both sending and receiving messages.

With a software implementation, if the encrypted message is signed as it is, it will require twice the time to encrypt and decrypt a signed message as an unsigned message. There is, however, a way to save time in this operation—that is to sign a compressed version of the message and send the signed compressed version along with the unsigned version. A reduction procedure for this would have to be such that it wouldn't be possible in practice for someone to change part of the message without changing the compressed version.

The need for impartial recording of encryption keys

Unfortunately, there is another operation requirement in terms of future verification: One must also be able to prove what the encryption key was when the message was sent before a disputed signature can be considered proven. A good example of the issue here would be a money transfer system connecting two banks.

It sounds very easy for two banks to send signed messages back and forth, secure in the knowledge that if there is a dispute each can produce the signed messages from the other as proof of contract. However, assuming that the keys change periodically, they must also be able to prove what the other's encryption key was at the time any given message was received. Otherwise a situation could arise in which one bank fabricates a message purporting to be from the other, generates a decryption key to sign the fake message with, and claims that the message was actually received from the other, and, at the time it was received, the corresponding encryption key was the public key in effect for the other bank.

An analogy can easily be drawn between this problem and conventional paper-and-ink signatures. That paper signatures can be used as proof is partly based on the fact that

someone's signature doesn't change over time. In a court of law, then, the signature on a document can be compared to the person's signature at that or any other time. If people changed their signatures frequently, or at all, it would be much harder to prove that someone did in fact sign a document. Similarly, if the keys didn't change in a PKCS this problem wouldn't exist. However, from a security point of view it is desirable to change keys.

The implication of this is that for signatures to be useful for future verification between two mutually suspicious parties, an impartial referee must record the public keys with the time that they are in effect. This would be very easy for the system controller to do, as it needs to have an accurate list of encryption keys at each point in time so it can transmit key information to users. The only extra effort necessary is the retention of the lists.

The question of automatic signatures

It seems that one would like two modes of operation, one in which all messages are signed when requested to by the system controller, and one in which messages are signed only when the user indicates.

Whether or not the signature operation should be user-controlled depends on the system and the purpose of the system. If its purpose is to ensure the identity of the sender, then it is probably more convenient to have the signature operation under system control. An example of this might be water-level sensors in a large dam's floodgate control system. There is an instance in which someone proved able to dial into the control computer in a large dam and have it release all of the floodgates. Similarly, if the water-level sensors indicated that the floodgates should be opened, the computer might well ask for a signature to make sure that a disgruntled employee didn't tap into the line and send the message.

On the other hand, in an electronic bill-paying system, one would probably want the user to retain very explicit control over what gets signed.

The major difference between these two examples seems to be that in the first, the signature is used to ensure proper identification. In the second, it is used in the conventional sense as well—for future verification. Through the signature, the user is presumably committing himself to a contract. Another way of looking at it is that the first case concerns itself with ensuring system integrity and therefore should be under system control. The second case concerns itself with user intent, clearly the domain of the user.

HARDWARE VS. SOFTWARE IMPLEMENTATIONS

The encryption/decryption and key generation functions can be implemented in either hardware or software. A combination is also quite possible, if not often desirable. Presently, hardware implementations are not available, although at least one hardware implementation project is under way

and various companies have given consideration to an IC chip version.

The software programs needed are rather simple and straightforward. With multiple precision routines to work with, they can be on the order of a few dozen lines of code. The multiple precision arithmetic routines are more complicated than the rest of the code, but algorithms and example programs are provided in Knuth's volume on semi-numerical algorithms.³ Complete functional specifications and design instructions for encryption/decryption and key generation procedures are provided in Rivest's paper.⁶ The problem with software implementations is that they seemed to be slow. The speed of software implementations will be explored in depth in the next section.

Until a PKCS chip becomes available (we don't know of any that are planned), the alternative for hardware implementations is the use of microprocessors and discrete ICs. The one hardware implementation project currently known is using ICs. The component cost is expected to come to between \$1900 and \$2000 for one and about \$1200 in quantities of 1000s. For commercial products of this sort, the component cost is usually multiplied by from three to seven to arrive at the selling price for a finished product. This particular implementation is designed to be fast—about 6000 baud—and uses high-quality components. Alternative designs could be done at lower cost. In comparison, a DES (the National Bureau of Standards' Data Encryption Standard) device is available for \$2500 in small quantities and less than \$2000 in large quantities. This particular DES device is faster as well—up to 19.2 Kbaud, depending on mode of operation. DES boards are sold for \$850 in small quantities and less than \$500 in large quantities (these boards operate at 6.7 to 56 Kbaud, depending on mode of operation.)

Microprocessors are now too slow to be used for PKCS applications. This is true for several reasons: the actual instruction speed is slow compared to larger machines, the word size is generally eight bits (necessitating more operations), and they do not have cache memories. On the other hand, microprocessors are continually getting faster, 16-bit microprocessors are becoming commonplace (the Intel 8086 and TI's TMS9900), and cache memories on the chip are expected within a couple of years. So while microprocessors are not presently an adequate vehicle for implementing PKCS, they may well be in a few years. Another interesting possibility is that since message blocks can be processed independently, microprocessor arrays may become a viable alternative. In this case the speed of encrypting one block would be the effective speed for encrypting an entire message.

Software speed analysis

One of the primary considerations in determining the usefulness of PKCS is the cost of using them. In terms of a software implementation, this refers primarily to the time required to encrypt and decrypt messages. This is dependent on the number and types of instructions necessary to carry

out these operations, as well as the speed of the individual computer.

As software implementations do not offer the option of a speed/cost tradeoff, we present here estimates of the time requirements for software encryption and decryption.

The procedure considered for encryption and decryption, which is recommended in Rivest's paper, is called "exponentiation by repeated squaring and multiplication:"

```
FOR I=LOG2(e) TO 1 BY -1
  C=REM ((C*C),n)
  IF ei=1 THEN C=REM ((C*M),n)
END
```

where e_i is the i^{th} bit, starting with the most significant, in the binary representation of e . C is then the encrypted form of the message.

From this procedure, it is clear that the bottleneck in a software implementation will be the multiple precision multiplication and division routines. The multiplication and division routines analyzed here are basically from Knuth.³ A combination is used, in that each operand is assumed to be split into eighths and the basic "high school" algorithm is used in multiplying the individual eighth sections. They are split into eighths to utilize the following technique:

$$A = A_0 + (2^n) * A_1 \text{ Example: } 110010 = 010 + (2^3) * 110$$

$$B = B_0 + (2^n) * B_1$$

$$A * B = A_1 B_1 * (2^{2n} + 2^n) + 2^n (A_1 - A_0) (B_0 - B_1) + (2^n + 1) A_0 B_0$$

Note that only three multiplications were required, whereas in the basic algorithm, if A and B were each more than one computer word long, four multiplications would have been necessary (multiplying by a power of two in a computer is only a shift operation and can be done very quickly; with multiple precision numbers, only keeping track of the decimal point is involved.) Since splitting the operands in half yields only three-fourths the number of multiplications that would be needed otherwise, splitting them into eighths yields $(\frac{3}{4})^3 = \frac{27}{64}$ multiplications. The actual number of sections the operands can be split into depends on the size of the operands. We chose three splits as a representative illustration.

Adding time for overhead to the $\frac{27}{64}$ figure, we will estimate that multiplications are done in half the time estimated by Knuth's basic algorithm (he also describes more advanced algorithms, including the one just discussed, but he only provides detailed time estimates for the basic one.) It should be noted that these time estimates are based on simple, straightforward techniques. We are not suggesting that this is the best that can be done. Optimization techniques and coding tricks would provide a time reduction. On the other hand, these figures do not take into account any system overhead. With the assumptions described just below, it is expected that the system overhead would be small, but there would be some.

Since the encryption/decryption routines are small programs running in tight loops, we assume in these calculations that if a cache memory is available, the instructions will be

found in the cache. Further, since the number of operands, parameters, and counters is also small, we assume that operands are found in the general registers. While both of these assumptions will not always be true, they will be true the great majority of the time. One last assumption is that of the total number of cycles required to perform some function, half will be for executing the instructions and half will be for obtaining operands. Encryption/decryption times are shown for four machines: The PDP 11/45, PDP 11/70, IBM 370/168, and CDC 6400. The times are shown for both an n of 100 and 200 decimal digits, and key sizes of n , $\log_2 n$, and three.

The key size of three refers to actually using three as a key; it is the smallest key that can work. Using a key of less than $\log_2 n$ risks the encrypted message coming out the same as the clear text; that is, no encryption will take place. However, if a message block is filled out through all $\log_2 n$ places, using a key of three will work. Message formatting schemes can be easily devised to ensure that all messages will be encoded; however, there is corresponding overhead.

Note that encryption and decryption of a given message require the use of two different keys, at least one of which will be on the order of size n . This implies that while a small key may be used on one end, providing very fast operation, a large key must be used on the other end, with its correspondingly-higher overhead. Presumably the decryption key will be about size n as it mustn't be guessable. To find the total time involved in encoding and then decoding a message, add the time required to encode with whatever size key is to be used to the time required to decode with the decryption key.

Table II summarizes the execution times (in microseconds) for encrypting a 1000-byte message. Table III provides the number of bits per second that can be encrypted using the different key sizes and machine word sizes: The detailed derivation of these timing figures can be found in Reference 5. In comparison, DES software routines have been claimed for the 370/168 which perform encryption/decryption at rates of 600 Kbits/second.

TABLE II—1000-byte Encryption Times in μsecs

	KEY SIZE		
	3	$\log_2 n$	n
16-BIT MACHINE SIZE (the PDP11)			
200-DECIMAL-DIGIT n SIZE			
without cache store (11/45)	1,826,877	8,563,485	606,821,235
with cache store (11/70)	811,945	3,805,993	269,698,326
100-DECIMAL-DIGIT n SIZE			
without cache store	1,040,185	4,356,207	170,229,626
with cache store	462,304	1,936,092	75,657,611
32-BIT MACHINE SIZE (the 370/168, which has a cache)			
200-DECIMAL-DIGIT n SIZE	44,417	208,223	14,754,988
100-DECIMAL-DIGIT n SIZE	25,629	107,333	4,254,457
60-BIT MACHINE SIZE (the CDC 6400, which has no cache)			
200-DECIMAL-DIGIT n SIZE	112,331	526,597	37,310,797
100-DECIMAL-DIGIT n SIZE	76,799	321,595	12,755,416

TABLE III—Bits Encrypted per Second

	KEY SIZE		
	3	$\log_2 n$	n
16-BIT MACHINE SIZE (the PDP11)			
200-DECIMAL-DIGIT n SIZE			
without cache store (11/45)	4,379	934	13
with cache store (11/70)	9,853	2,102	30
100-DECIMAL-DIGIT n SIZE			
without cache store	7,691	1,836	47
with cache store	17,305	4,132	106
32-BIT MACHINE SIZE (the 370/168, which has a cache)			
200-DECIMAL-DIGIT n SIZE			
	180,111	38,420	542
100-DECIMAL-DIGIT n SIZE			
	312,146	74,534	1,880
60-BIT MACHINE SIZE (the CDC 6400, which has no cache)			
200-DECIMAL-DIGIT n SIZE			
	71,218	15,192	214
100-DECIMAL-DIGIT n SIZE			
	104,168	24,876	627

PKCS/DES HYBRID COMMUNICATIONS SYSTEM

A promising technique for utilizing the respective advantages of both PKCS and DES is a two-level key system using PKCS for the primary key system and DES for the secondary key system.

In a two-level key system, each user has a primary key and a secondary key. The primary key is used solely to encrypt secondary key changes that are communicated between the user and system controller over the communications lines. The secondary key is used to encrypt messages. By using the primary key for encrypting the secondary key, the secondary key can be sent over insecure lines and thus be changed frequently at little cost. With traditional systems, the primary key must be transmitted externally to the communications system. This is the expensive and awkward key agreement problem discussed previously.

A promising technique for utilizing the respective benefits

of both DES and PKCS is implementing a hybrid two-level system. Messages would be encrypted and decrypted using DES. The DES keys would be transmitted over the communications lines encrypted with the PKCS system. This would in effect be a two-level system with PKCS constituting the primary key system and DES constituting the secondary key system.

The advantage of such a system would be that the speed of a DES implementation would be available with the key management cost and security of a PKCS implementation. Note that the primary keys would be changed infrequently, leaving plenty of leeway for the key management technique suggested above. Generating and transmitting a new secondary key would only involve encrypting a 64-bit message. This would require little time in a software implementation. At the same time, DES encryption hardware or software routines would be used for the messages, yielding the encryption/decryption cost of a straight DES system.

REFERENCES

1. Diffie, W., and M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, November 1976.
2. Kent, Stephen T., "Encryption-Based Protection Protocols for Interactive User-Computer Communications," MIT/LCS/TR-162, Laboratory for Computer Science, MIT, Cambridge, MA, May 1976.
3. Knuth, D., *The Art of Computer Programming, Vol. 2. Seminumerical Algorithms*, Addison-Wesley, 1969.
4. Kohnfelder, L., "Towards a Practical Public-Key Cryptosystem," MIT Department of Computer Science, June 1978.
5. Michelman, E., "Security Management in Communications Systems—Everything You've Always Wanted to Know About Public-Key Cryptosystems," Sloan School of Management, MIT, 1978.
6. Rivest, R., A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, February 1978.
7. Rivest, R., "Remarks on a Proposed Cryptanalytic Attack on the MIT Public-Key Cryptosystem," *Cryptologia*, Vol. 2, No. 1, January 1978.
8. Simmons, G., and M. Norris, "Preliminary Comments on the MIT Public-Key Cryptosystem," *Cryptologia*, Vol. 1, No. 4, 1977.

Safeguarding cryptographic keys

by G. R. BLAKLEY

Texas A&M University
College Station, Texas

INTRODUCTION

Certain cryptographic keys, such as a number which makes it possible to compute the secret decoding exponent in an RSA public key cryptosystem,^{1,5} or the system master key and certain other keys in a DES cryptosystem,³ are so important that they present a dilemma. If too many copies are distributed one might go astray. If too few copies are made they might all be destroyed.

A typical cryptosystem will have several volatile copies of an important key in protected memory locations where they will very probably evaporate if any tampering or probing occurs. Since an opponent may be content to disrupt the system by forcing the evaporation of all these copies it is useful to entrust one or more other nonvolatile copies to reliable individuals or secure locations. What must the nonvolatile copies of the keys, or nonvolatile pieces of information from which the keys are reconstructed, be guarded against? The answer is that there are at least three types of incidents:

- An *abnegation incident* is an event after which a nonvolatile piece of information is no longer completely reclaimable by the organization which entrusted it to a guard. There are three main types of abnegation incidents:
 - Destruction* of the nonvolatile piece of information. For example, a person carrying a copy of a number can meet with an unexpected accident, during which the copy is destroyed.
 - Degradation* of the nonvolatile piece of information. For example, a person may lose his copy of the number and, in embarrassment and confusion, produce some other number when asked.
 - Defection* with the nonvolatile information. For example, the person with the copy of the number may divulge it to the opposition and refuse to tell it to the organization which entrusted it to him.
- A *betrayal incident* is an event after which a nonvolatile piece of information is completely known to an opponent of the organization which entrusted it to a guard. Defection, which we have already encountered among abnegation incidents, is one kind of betrayal incident. The other main kind of betrayal incident is
 - Dereliction* with the nonvolatile piece of information,

an act which reveals it to the opposition so as not to be discovered by the organization which entrusted it to the guard, either before or after he has been requested to return it. For example, the person who has the copy of the number can show it to an opponent but still play the part of a faithful guard, and even report the number back correctly when requested.

- A *combination incident* is an abnegation incident which is also a betrayal incident. The main kind of combination incident is defection. The three types of incident are, thus, A, B and C. And the commonest kinds of A, B or C incidents are the four Ds. Note that none of the four Ds need imply malfeasance, misfeasance or even nonfeasance on the part of the guard. But it would be wise to consider such possibilities whenever an incident of any of the three types is detected.

Why was simple loss of the nonvolatile piece of information not included above? The answer is that some types of loss amount essentially to destruction of the nonvolatile piece of information, in the sense that neither the organization that entrusted it to a guard nor any of its opponents is likely to get the piece of information before the encrypted information becomes valueless. For example, the person with the copy of the number was on a Mars flyby which lost contact forever with Earth as it went behind Mars. But if a loss cannot be confidently regarded as a destruction, the proverbial "prudent man," in charge of evaluating this incident for the organization which entrusted the nonvolatile piece of information to a guard, must regard it as a defection. For example, if the person who memorized the number disappeared after a family quarrel the prudent man evaluating the incident must assume that an opponent knows the piece of information in question.

COUNTING AND DISCOUNTING INCIDENTS

There are two principles for counting incidents. The first is Boole's law of inclusion and exclusion. Suppose that an organization issues nonvolatile pieces of information to guards and waits a modest period of time during which incidents occasionally occur. Let a stand for the number of abnegation incidents, b for the number of betrayal incidents and c for the number of combination incidents. The total

number d of incidents is $d=a+b-c$ because a combination incident gets counted twice, once by a and once by b .

The second principle is that incidents are so rare, that the possibility of two separate incidents occurring with the same nonvolatile piece of information is usually dismissed on probabilistic grounds as absurd. A defection is a single incident with two aspects, abnegation and betrayal, so it is not dismissed as too improbable. But the idea that the person who has a copy of the number dies in a plane crash one month after confiding it to an opponent is dismissed as too improbable. Too slavish an adherence to this "second-order improbability" prejudice can lead to ludicrously inappropriate actions, such as that of the statistician who always carries his own bomb on airplanes because it is so improbable that there will be two bombs on the same flight. But it is a good rule of thumb if used properly.

This latter principle implies, among other things, that none of the four numbers a , b , c or $a+b-c$ exceeds the number g of nonvolatile pieces of information entrusted to the g guards.

Suppose an organization chooses in advance the number a of abnegation incidents and the number b of betrayal incidents it feels it must be protected against when entrusting several nonvolatile pieces of key reconstruction information to a set of guards. Each guard gets a different piece of information. The lifetime of this scheme must not be very many months if separate incidents involving the same piece of information are to be ruled out. We know that $c \leq \text{MIN}\{a, b\}$ since a combination incident is both an abnegation incident and a betrayal incident. From the two counting principles above it then follows that

$$a+b-\text{MIN}\{a, b\} \leq d \leq a+b$$

and that $d \leq g$, where g is the number of guards to which the organization entrusts the g nonvolatile pieces of information.

The prudent man, when designing a system of safeguarding key information which is secure from a abnegation incidents as well as b betrayal incidents, must assume that the number c of combination incidents is zero. This means that the maximum number of incidents must be anticipated, since

$$d = a + b - c = a + b - 0$$

in this case. Such a key information safeguarding system must have the property that $a+b+1$ different nonvolatile pieces of key reconstruction information are generated, and given to distinct guards. The key must be reconstructible from any $b+1$ of these pieces (this assumes a abnegation incidents) but there must be no information whatever about the key which can be inferred from knowledge of only b of these pieces (this is protection against b betrayal incidents). This last requirement is unusual. For example, a polynomial of degree b can be reconstructed from its values at $b+1$ points. But already its values at any b points tell a lot about it. It can also be reconstructed from the values of its 0th through b th Taylor coefficients at a point. But already the values of any b of these $b+1$ numbers tell a lot about it. What we are asking for, then, is somewhat counter-intui-

tive. Let us coin a metaphor to describe it. We want to give every one of $a+b+1$ guards a shadow of a different profile of the key, so that the key can be reconstituted in its entirety from any $b+1$ of these shadows. However, somebody who has seen only b such shadows should be completely in the dark, in the very strong sense that any key on the keyring could cast these b shadows when illuminated from b appropriately chosen directions.

Let us look at what happens if $a=b=4$. Then any five of the nine guards have the wherewithal to reconstruct the key. Thus, there is considerable protection against defection and dereliction, since even four of the nine pieces of information are not enough to reveal anything at all about the key to an opponent. There is also protection against destruction. If the four pieces of information belonging to any four guards are destroyed the other five can still be used to reconstruct the key. As to degradation, suppose that six guards give correct reports of the shadows they carry, to return to the metaphor. Then there are six different sets of five guards whose pieces of information can reconstitute the same key. If the other three misreport their shadows then any one of the 120 sets of five guards containing at least one of the misreporting three guards will give a description of the key, but probably all these descriptions will differ among themselves and will also differ from the true value of the key. Thus, the six reports of different sets of five guards which agree are singled out as correct. Of course, if it is possible to tell whether a proffered key is the right one, then it is possible to reconstruct the key when only five guards report correctly. So protection against degradation need not be synonymous with protection against destruction, but they are largely concomitant with each other. In the approach to be discussed it will be assumed that the right key can be recognized when proffered. This assumption is reasonable since lists of plaintext to cryptext pairs can be publicized for testing as a backstop to the simpler test, which is that stored ciphertext messages will probably yield nonsensical diecipherments under a false key.

The rest of the paper describes one way to cast the $a+b+1$ shadows of a key in such a fashion that it can be reconstructed from any $b+1$ of them, but that no b of them tell anything about it whatever. The way this is done is to set up a many-to-many correspondence between keys and one-dimensional vector subspaces of (i.e. lines through the origin of) a finite vector space, F . One key determines a vast collection of lines but one line determines a tiny collection of keys. When an organization has a key to apportion among $a+b+1$ guards, it picks at random one of the lines corresponding to that key. Let us call this line L . Then it picks at random $a+b+1$ vector subspaces of F —the shadows of the key—such that any b or fewer of them intersect in a large vector subspace of F whose various one dimensional vector subspaces lead back to all possible keys with approximately equal probability, but such that any $b+1$ of them intersect in L . Once L has been found there are only a few possible keys which could have given rise to it. Each one is tried against a stored list of plaintext to cryptext pairs and the correct one identified. This is not the first application of projective geometric ideas to problems involving codes.²

SOME PRELIMINARY RESULTS

$a \uparrow b$ is the b th power of a , and $a * b$ is the product of a and b .

- **Lemma 1**—Let a and b be positive integers. Let R be a set with at least $a+b+2$ members. Then there are more than $a+b+2$ subsets of R which consist of $b+1$ objects. There are more than $a+b+1$ subsets of R which consist of b objects. If f and g are two members of R there are more than $a+b$ subsets of $R \setminus \{f\}$ which consist of b objects, and there are at least $a+b$ subsets of $R \setminus \{f, g\}$ which consist of b objects.
- **Lemma 2**—Suppose that a and b are positive integers smaller than z . Let M be a matrix with at most $a+b+2$ rows and at most $b+2$ columns. Then M has at most $(z+1)(2z)$ entries and at most $(z+1) \binom{2z}{z} b+1$ by $b+1$ submatrices. Thus M has fewer than $3z \uparrow 2$ entries and fewer than $4 \uparrow z$ submatrices of size $b+1 = by = b+1$.
- **Lemma 3**—Suppose that $0 < 2Ex < 2Q < 2 < E$. Then $2|(1-x) \uparrow E - (1-Ex)| < Q \uparrow 2$.
- **Lemma 4**—If $4 \leq A < B$ then $\prod(1-j/B) < \prod(1-2/B)$, where the products are over positive integers $j < A$.
- **Lemma 5**—Suppose that A and B are integers and that

$$0 < 2(A-1) \uparrow 2 < 2BQ < 2B < (A-1)B.$$

Then $1-2Q < B! / ((B-A)! * [B \uparrow A]) < 1$, and $1-2Q < ((B-2)/B) \uparrow A < ((B-1)/B) \uparrow A < 1$.

- **Lemma 6**—Suppose that A and B are integers and that

$$0 < 2(A-1) \uparrow 2 < 2BQ < 2B < (A-1)B$$

Suppose that a sample of A points (with replacement) is taken from a population of B points. Then the probability U that all sample points are distinct exceeds $1-2Q$. If two distinguished points of the population are specified in advance the probability V that no sample point is equal to either of them exceeds $1-2Q$. Therefore it follows *a fortiori* that if one or two distinguished population points are specified in advance, then the probability W that none of the points of the sample is equal to any of the distinguished points or to any other point of the sample exceeds $1-4Q$.

• **Lemma 7**—Let p be an odd prime. Let d be a positive integer. Let $S(d, p)$ be the collection of all d by d matrices with entries taken from the field F of integers modulo p . Let v and w be two non-zero members of F . Then there are as many members of $S(d, p)$ with determinant equal to v as there are with determinant equal to w .

• **Lemma 8**—Let p be an odd prime. Let k and n be positive integers. Let $f(p, n, k)$ be the number of k by n matrices over the field F of residue classes modulo p whose rank is less than k . Then $f(p, n, 1) = 1$ and, whenever $2 \leq k \leq n$,

$$f(p, n, k) = p \uparrow [(k-1)(n+1)] + (p \uparrow n - p \uparrow (k-1)) f(p, n, k-1).$$

Consequently,

$$p \uparrow [(k-1)(n+1)] < f(p, n, k) < p \uparrow [(k-1)(n+2)] + (p \uparrow n) f(p, n, k-1)$$

for every integer k such that $2 \leq k \leq n$.

• **Lemma 9**—Let p be an odd prime. Let $f(p, n, k)$ be as in Lemma 8. Then $p \uparrow (n \uparrow 2 - 1) < f(p, n, n) < 2p \uparrow (n \uparrow 2 - 1)$.

- **Theorem 1**—Let p be a prime larger than 6. Let d be a positive integer. Let $S(d, p)$ be the collection of all d by d matrices with entries taken from the field F of integers modulo p . If $v \in F$ let $n(v, d, p)$ be the number of members of $S(d, p)$ whose determinant is equal to v . Suppose that h and g are members of F . Then

$$n(h, d, p) < 3n(g, d, p)$$

$$\text{and } [p \uparrow (n \uparrow 2 - 1)] / 2 < f(p, n, n) < 2p \uparrow (n \uparrow 2 - 1).$$

Thus, all determinants occur approximately equally often. In fact every non-zero field element occurs equally often as the value of the determinant of a member of $S(d, p)$ but zero occurs more often, though not thrice as often.

- **Theorem 2**—Let a , b and p be positive integers. Let M be a matrix with $a+b+2$ rows and $b+2$ columns. Suppose that

$$0 < 2[(a+b+2)(b+1)-1] \uparrow 2 < 2pQ < 2p < [(a+b+2)(b+1)-1]p.$$

Suppose that one position in each row of M is chosen at random, and that that entry is set equal to 1. Suppose that the remaining $(a+b+2)(b+1)$ entries of M are chosen at random (with replacement) from the population of all p residue classes modulo p . Then each of the two events

1. Two entries of M , neither of which is one of the $a+b+2$ entries which were set equal to 1 at the outset, are congruent to each other modulo p
2. An entry of M , other than one of the $a+b+2$ entries which were set equal to 1 at the outset, is congruent to either 0 or 1 modulo p

have probability smaller than $2Q$. Consequently, the probability that neither Event 1 nor Event 2 occurs exceeds $1-4Q$.

It is easy to verify that if $Q = 1/10 \uparrow 7$, and a and b are both smaller than 10, then it suffices to choose any $p > 10 \uparrow 12$ in order to satisfy the hypotheses of Theorem 2. This is the order in which users of the keyguard system will usually proceed. The tiny positive number Q is a measure of the departure from complete randomness of the concealing procedure. The modest-sized positive integer a (resp. b) is the number of abnegation (resp. betrayal) incidents to be guarded against. After deciding on these three safety levels a user must then accept a value of p as large as dictated by the hypotheses of Theorem 2 in order to achieve them. The keyspace will then be chosen to contain at least p keys.

Consider, now, the probabilistic interpretation of Theorem 1. If you choose a member x of the field F of integers modulo p , and choose some d by d matrix M over F at random (by choosing its successive entries at random with replacement from f) then the probability W that $\det(M) = x$ satisfies the inequality $1/2p < W < 2/p$.

We will assume that the manner in which the matrix in

Theorem 2 is chosen (salting each row with a 1 entry) does not do much violence to this conclusion. In other words, we will make the following (unproven but plausible) assumption.

- *Hypothesis 1*—Let p be an odd prime. Let a and b be positive integers. Let M be a matrix with $a+b+2$ rows and $b+2$ columns. Suppose that a position in each row is chosen at random and that that entry is set equal to 1. Suppose that, thereafter, each of the remaining $(a+b+2)(b+1)$ entries is chosen at random from the field F of residue classes modulo p . Suppose that, then, a collection of $b+1$ row indices is chosen at random from the set of all $b+1$ member subsets of the set of all $a+b+2$ row indices. Suppose, finally, that a collection of $b+1$ column indices is chosen at random from the set of all $b+1$ member subsets of the set of all $b+2$ column indices. Let x be a member of F . Let W be the probability that the value of the determinant of the $b+1$ by $b+1$ submatrix S of M corresponding to these row and column indices is equal to x . Then W satisfies the inequality

$$1/2p < W < 2/p.$$

To put matters in a nutshell, a judicious salting of an otherwise randomly chosen matrix with a few entries equal to 1 should not cause the determinants of its large square submatrices to depart from the quite uniform distribution that determinants of completely randomly selected matrices exhibit.

GUARDING KEYS

A key k is a positive integer. A keyset K is a finite set of keys. Let B be the largest member of the keyset K . A reasonably small positive integer z is chosen. On practical grounds z should probably be smaller than 100. Two positive integers a and b smaller than z are chosen. A prime p only slightly smaller than B is found. It would not, in fact, be too expensive to find the largest *pseudoprime* smaller than B and let it be p . A pseudoprime is a large positive integer which satisfies a considerable number of Rabin's (hopefully) stochastically independent necessary⁴ conditions for primality, and can therefore be assumed to be prime with a probability in excess of 0.99999 99999 99999 99999, or even more, if desired. Though p might be composite we shall regard it as prime in the development below. Let F be the field of integers modulo p . Let V be the $b+2$ dimensional vector space over F which consists of all lists (written in the form of rows) of $b+2$ members of F . For every member g of the set G of $a+b+1$ guards we will define a corresponding $b+1$ dimensional vector subspace $V(g)$ of the $b+2$ dimensional vector space V . To each key k there will correspond many lines, through the origin of V , representing k . The organization wishing to entrust k to a set of guards will choose one of these lines at random and call it $L(k)$. When b guards intersect their subspaces the intersection must be at least two-dimensional. Moreover, it will be such that its various one-dimensional vector subspaces represent all members of

F with approximately equal likelihood. But when $b+1$ guards intersect their subspaces the intersection is the line $L(k)$, which does not depend on which $b+1$ guards were chosen. To $L(k)$ there will correspond only $b+2$ possible keys. The candidates can be checked and the key reclaimed. The rest of this section fleshes out this outline.

To begin we pick z and choose positive integers a and b smaller than z . Then we choose a small Q , and thereafter a suitably large p to satisfy the inequalities in the hypotheses of Theorem 2. We construct a matrix M with $a+b+2$ rows and $b+2$ columns as follows. For each row of M we pick an entry at random and set it equal to 1. Next we pick an entry at random in the first row of M and choose its value k at random from F . Then we choose the remaining $(a+b+2)(b+2)-1$ entries of M at random (with replacement) from F . Now we test M for acceptance or rejection. In order to pass the first test M must have only one 1 in each row, it must have no zero entry and no two of its entries can be equal unless they are both equal to 1. Since a and b are non-negative integers smaller than z it follows from Lemma 2 that there are fewer than $3z \uparrow 2$ entries of M . Since p and Q satisfy the inequalities in the hypotheses of Theorem 2, it then follows from Lemma 4 that such a random process will produce a matrix which passes the first test with probability in excess of $1-2Q$. In order to pass the second test M must have no $b+1$ by $b+1$ submatrix whose determinant, calculated in F , is zero, and must have no two $b+1$ by $b+1$ submatrices whose determinants, calculated in F , are equal. There are fewer than $4 \uparrow z$ such submatrices, according to Lemma 2. The foregoing suggests that the random process which produced M will cause it to pass the second test with probability in excess of $1-2Q$. Therefore, it should pass both tests with probability in excess of $1-4Q$. In other words, the process used almost always produces a usable matrix M the first time it is employed. Once a matrix M passes the tests we know from Lemma 1 that we can form more than $a+b+1$ sets of $b+1$ rows of M which contain the first row of M . So we pick $a+b+1$ different sets of $b+1$ rows of M , each of which contains the first row of M . Each such set is linearly independent since every $b+1$ by $b+1$ submatrix of M is non-singular. Let $N(j)$ be the $b+1$ by $b+2$ submatrix of M formed in the obvious way from the j th of these $a+b+1$ sets of rows. Its first row consists of the first of M 's rows which occurs in the set. Its second row is M 's second. And so on. Now for each $x \in V$ it is possible to form the $b+2$ by $b+2$ matrix $Y(j,x)$ from $N(j)$ by appending a last (i.e. $(b+2)$ nd) row

$$\begin{aligned} x &= (x(1), x(2), \dots, x(b+1), x(b+2)) \\ &= (Y(j,x)[b+2,1], Y(j,x)[b+2,2], \dots, \\ &\quad Y(j,x)[b+2,b+1], Y(j,x)[b+2,b+2]) \end{aligned}$$

The $b+1$ dimensional vector subspace of the vector space of rows with $b+2$ entries taken from F determined by $N(j)$ is the set

$$U(j) = \{x \mid \det(Y(j,x)) = 0\}$$

Evidently the first row f of M belongs to $U(j)$ for every j since $Y(j,f)$ has first and last row equal to f for every j .

So when $b+1$ of these $b+1$ dimensional vector subspaces of the $b+2$ dimensional vector space of rows of $b+2$ entries taken from F are intersected their intersection is the line through the origin which also contains the vector f which is the first row of M . The equation $\det(Y(j,x))=0$ is, of course, a linear equation of the form

$$c(j,1)x(1)+c(j,2)x(2) \dots +c(j,b+1)x(b+1)+c(j,b+2)x(b+2)=0$$

where $c(j,t)$ is a determinant of some $b+1$ by $b+1$ submatrix of M . These are non-zero, and pair-wise unequal by the way M was produced. And, because of the foregoing, they probably appear to be approximately randomly selected from F .

But now look at what happens when only b of these subspaces is intersected to form a two-dimensional vector subspace. This means choosing integers

$$1 \leq j(1) \leq j(2) < \dots < j(b) \leq a+b+1$$

and solving the simultaneous equations

$$\begin{aligned} \det(Y(j(1),x)) &= 0 \\ \det(Y(j(2),x)) &= 0 \\ &\vdots \\ \det(Y(j(b),x)) &= 0 \end{aligned}$$

for x , by using Gauss elimination, then choosing a basis of two vectors for this space of all such x . The two-dimensional vector space in question contains the first row of M . But the randomness of the choices of the members of M should mean the following:

- *Hypothesis 2*—Consider the collection of all vectors in this two dimensional subspace which have exactly one entry equal to 1 and which have pairwise distinct entries none of which is zero. Any two members of $F \setminus \{0,1\}$ will be represented approximately equally often in the count of multiplicities of occurrence of members of $F \setminus \{0,1\}$ as entries in the vectors of this collection.

If this is correct then isolation of this two-dimensional subspace sheds no light whatever on how to recover the

key. The recovery system, when you have $b+1$ subspaces $U(j)$ is to solve the system

$$\begin{aligned} \det(Y(j(1),x)) &= 0 \\ &\vdots \\ \det(Y(j(b+1),x)) &= 0 \end{aligned}$$

as above. The solution is a line through the origin. A basis for it is a single vector $g=(g(1),g(2), \dots, g(b+1),g(b+2))$ which is some non-zero multiple of f , the first row of M , which contains the key as one of its entries. You know g , not f . But for each entry $g(i)$ of g it is easy to find the $h(i) \in F$ such that $g(i)h(i) \equiv 1 \pmod{p}$. The $b+2$ vectors

$$\begin{aligned} h(1)g \\ h(2)g \\ \vdots \\ h(b+2)g \end{aligned}$$

are the only multiples of g which have 1 as an entry. Therefore f is among them, and the key k is among the entries of f . So one of the $(b+2) \uparrow 2$ entries on the list of vectors above is the key. And the key is not equal to 1, which occurs once among the entries of each vector. So there are

$$(b+1)(b+2) \leq z(z+1)$$

candidates to be tested. One of them will pass the test.

REFERENCES

1. Blakley, G. R., and I. Borosh, "Rivest-Shamir-Adleman public key cryptosystems do not always conceal messages," *Computers and Mathematics with Applications*, Vol. 5, 1979 (in press).
2. Gilbert, E. N., F. J. MacWilliams and N. J. A. Sloane, "Codes which detect deception," *The Bell System Technical Journal*, Vol. 53, 1974, pp. 405-424.
3. Morris, R., N. J. A. Sloane and A. D. Wyner, "Assessment of the National Bureau of Standards proposed federal Data Encryption Standard," *Cryptologia*, Vol. 1, 1974, pp. 281-306.
4. M. Rabin, "Probabilistic algorithms," in *Algorithms and Complexity* J. Traub (ed.), Academic Press, 1976.
5. Rivest, R. L., A. Shamir and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Communications of the ACM*, Vol. 21, 1978, pp. 120-126.

Applications for multilevel secure operating systems

by JOHN P. L. WOODWARD

The MITRE Corporation
Bedford, Massachusetts

INTRODUCTION

The need for secure computer systems has been identified in many areas of DoD operations, but in the past these systems have not been built in a secure manner because a secure operating system on which to run has not existed. Now that verifiably secure minicomputer operating systems are becoming a reality, applications for secure systems are becoming more clearly thought-out, designed and implemented. This paper surveys some proposed DoD and non-DoD secure computer applications.

We will discuss the applications presented in this paper in the context of their implementation on a secure operating system. This paper will survey four applications:

- a. The "guard" application;
- b. Secure database management systems;
- c. Secure message processing systems; and
- d. Secure front-ends.

We will discuss the first two of these applications in more detail, and the other applications are discussed in Reference 1.

Before the discussion of secure computer applications, the following section describes the past and present work that is leading to the completion of a mathematically verified Kernelized Secure Operating System (KSOS), and briefly discusses the organization of kernel-based secure systems. This organization provides a framework for our survey of major applications for secure computer systems in the next sections. The conclusions highlight some of the major points of the paper.

SECURE COMPUTER SYSTEMS

Background

The need to process multiple levels of data in a single computer system has been recognized within DoD for a long time. This need led the Air Force Electronic Systems division in 1972 to study how one could verify that a system meets DoD security requirements.² This need for verification led to the development of a mathematical model that

describes DoD security policy in terms of a "reference monitor"—an abstract mechanism that controls the flow of information within a computer system by mediating all accesses by subjects (processes, users) to objects (files, I/O devices).³

Given the concept of a reference monitor and a mathematical model that embodies a specific security policy, one can formulate an inductive proof in terms of secure system states, proving that security is preserved in moving from state to state. Several models that embody DoD security rules have been developed^{4,5} and each of these models has two basic rules: the simple security condition and the *-property (read "star-property").⁶

The simple security condition mandates that a subject cannot read an object unless the security level (in DoD terms, a classification and a set of categories) of the subject is greater than or equal to that of the object. This rule is a precise statement of the basic premise behind DoD security, and is analogous to controls in the people-paper world.

The *-property dictates that a subject cannot write an object unless the subject's security level is less than or equal to that of the object. The *-property is motivated by the need to prevent a program operating on behalf of a user from reading information that the subject (user) is cleared to read and writing this information into a container of a lower classification level, thus allowing access to the information by subjects at lower levels.

The hardware and software mechanism that implements a reference monitor and enforces the rules of the model is called a kernel (or security kernel). To adequately provide security, an implementation of a kernel must have three properties: 1) It must mediate every access of every subject to every object; 2) it must be isolated from other code in the system so it cannot be modified or tampered with; and 3) it must be mathematically verifiable, i.e., it must perform its functions correctly. The kernel satisfies the first property by creating and controlling an environment in which all non-kernel software must run. The second requirement is generally satisfied by relying on hardware domain mechanisms. The third property is satisfied by developing the kernel with a formal methodology that allows proof that the kernel enforces the desired policy correctly. This methodology⁷ uses a formal specification technique to describe the kernel interface, and allows the correctness proof to be carried out

in (at least) two steps: a proof that the top-level (kernel interface) specification obeys the rules of the model,⁸ and a proof that the kernel code correctly implements this interface.⁹

Some of the earliest kernel work was done under Air Force sponsorship by The MITRE Corporation. MITRE developed a simple kernel for the PDP-11/45 computer.¹⁰ This kernel, designed with no particular operating system in mind, was successfully implemented and verified. The top-level specification was verified with respect to the model and example lower-level proofs carried out.

A logical extension of the early MITRE PDP-11 kernel work led to a MITRE/Honeywell project to design a kernel for the MULTICS operating system,¹¹ also under Air Force sponsorship. MITRE specified a MULTICS kernel and verified it with respect to the model.¹²

Other major kernel developments have produced prototype kernels for the UNIX[®] operating system, running on PDP-11 computers. Prototype Secure UNIX systems have been developed by UCLA¹³ (under DARPA sponsorship) and MITRE¹⁴ (under Air Force and DARPA sponsorship). Experience with these prototypes has contributed to the current development of a production-quality Kernelized Secure Operating System (KSOS)^{15,16} by Ford Aerospace and Communications Corporation (FACC), under the sponsorship of DARPA and other government agencies.

The KSOS project has two phases, a design phase and an implementation phase. The design phase was carried out by two contractors, TRW¹⁷ and FACC.¹⁸ Both contractors produced designs for a somewhat machine independent design for a secure operating system that emulated the UNIX operating system. FACC was chosen to implement their design on a PDP 11/70 computer.

The organization of secure computer systems

The organization of KSOS is similar to previous secure system designs. Secure systems are most often divided into three portions: the kernel, the trusted (or privileged) processes, and the Emulator. We will discuss each in turn.

The security kernel

The kernel, the concrete implementation of the software portion of a reference monitor, encapsulates the security-related functions of a conventional operating system and provides an external interface which allows multiple processes to access data files and I/O devices. Kernels typically operate on the basis of the following assumptions:

- a. The kernel provides a process-structured environment.
- b. Each process (supported by the kernel) has available to it a set of objects that can be created, deleted and accessed only by using defined (kernel provided or mediated) operations.

- c. Each process is confined with respect to the defined protection policy.
- d. Each process can execute whatever programs it pleases; because it is confined it cannot, by construction, violate security.

It is important to note that for any security kernel of some sophistication, the security kernel not only controls access to objects, it must define the objects and construct operations to access these defined objects.

Most kernels also support the notion of a privileged process—a process that is allowed (privileged) to violate some or all of the rules of the security model being enforced. Since a privileged process can potentially violate security, it must be constructed and verified with the same rigor applied to the kernel itself. Privileged processes are often referred to as "trusted processes," because they are trusted (because of their verification) either not to violate security or to violate the security rules in a known and controlled manner.

Trusted processes

Trusted processes augment the capabilities of the kernel itself, and provide security-related functions more properly performed above the kernel interface. Examples of trusted processes are: a process to downgrade files; a process to handle logins at terminals (reading user names and passwords), etc. In the UCLA prototype secure UNIX system,¹³ trusted processes are used to set scheduling policy, protection policy, and create a primitive, flat, file system.

The emulator

The emulator portion of a secure operating system runs on top of the kernel and creates an operating system interface out of the interface provided by the kernel and the trusted processes. Because the emulator runs in a kernel-provided process, and typically does not need any privileges, it does not need to be trusted or verified. Most secure minicomputer operating system developments to date have emulated the UNIX operating system.

A SURVEY OF MAJOR TYPES OF SECURE APPLICATIONS

There are two major types of applications for secure operating systems, unilevel and multilevel. We will discuss each in turn.

Unilevel applications

Unilevel applications involve the use of a secure operating system as a whole to allow many users, each operating at a single level, to use the same computer at the same time. Such operation of a non-secure operating system would be

[®] UNIX is a trade/service mark of the Bell System.

in violation of DoD security rules because you cannot demonstrably prevent a low-level user from accessing higher-level data.

There are many instances within DoD where periods processing and multiple computers are used to perform unilevel operations at several levels. Periods processing refers to a mode of computer operation whereby the system is used to process data of different classifications at different periods of the day, because the multiple levels of data cannot be simultaneously processed without overclassification. Use of secure operating systems in environments such as those running unilevel applications can greatly increase machine utilization and decrease hardware costs due to security.

Unilevel applications rely on the secure computer system to provide complete isolation between users of different security levels. Very frequently, the real driving requirement is for controlled *sharing* among users of different classifications. The applications that permit this are called multilevel.

Multilevel applications

Very often a computer system user needs to process several levels of information at the same time. This need is addressed in some installations by operating in a mode known as "system high:" each user of the system is cleared to the level of most sensitive data being processed, and can operate on any data up to and including that level (within the constraints of the "need-to-know" policy enforced at the installation). The problem with this type of operation is that data tends to become overclassified: all data leaving the system must be considered highly classified until a manual review process allows the data to be properly classified. Although the "system high" mode of operation allows the processing of several levels of data, it is not properly termed multilevel, because it does not preserve the different classification levels of the data it processes.

Whereas the unilevel applications we have previously discussed, if run on a secure operating system, would typically run on the emulator portion, multilevel applications generally run on either the emulator or kernel portions. The basic premise behind kernel-based operating systems is that any unprivileged code running on the kernel, no matter how malicious it is, cannot violate any security rules. Thus, the emulator portion of a secure system is not trusted, and exists only to provide a richer operating system interface than a kernel alone would. Therefore, applications that run on secure operating systems can run on the emulator interface, or they can run directly on the kernel interface. The latter is often the case if the features provided by the emulator are not relevant to the desired application, or if the application requires very high performance.

Furthermore, some portion of a multilevel application might need to be privileged to perform its task. If this is the case, then the code must be trusted to perform its function properly, and the code must run directly on the kernel because the Emulator is not trusted.

Because of these different options for implementation of

multilevel applications, we will survey some multilevel applications that have been proposed to date, and indicate how each application would run on a secure system. Each application has either been worked on in the past, or is presently under development.

The guard application

In military operations, there is a great need to be able to interconnect computers of different classification levels: evolving defense systems depend heavily on the capability of passing information between computers operating at different levels. Unfortunately, such connections are very difficult to implement in a secure manner. This problem would be made much easier if all computers had secure operating systems available, but such is not the case.

There is also a recurring need to make a subset of classified data available for use at a lower classification level. "Sanitization" and "downgrading" must be performed to do this without compromise. To better illustrate this need, consider a highly-classified computer that maintains an intelligence data base. This data base typically contains pieces of data with attributes that make them very sensitive. However, to be operationally useful, the data is needed at lower levels where command decisions are made. Thus there is a great need to be able to "sanitize" the data. Sanitization is often accomplished by removing the sources (sensors, etc.) of the data or by reducing the precision of the data. If the computer with the intelligence data base is not secure, then we cannot trust a sanitization and release function performed on the computer not to allow the release of highly-classified data at a low level.

Currently, there are several methods available to provide a sanitize and release function. The simplest involves reading the information to be sanitized from one terminal, and entering the sanitized version on another terminal. The two terminals involved are not connected to the same system; there is no electrical connection between the two systems. A more sophisticated solution involves a single CRT terminal that can be connected to either system (by means of a switch), but only one system at a time. In this mode, data from the intelligence computer is read onto the screen of the CRT and sanitized by using the local editing capabilities of the CRT. Then the CRT is disconnected from the intelligence computer and connected to another, lower level system, and the data is read into the lower level system.

The problem with these solutions is that they are time-consuming and cumbersome. The best solution would be to use a secure operating system on intelligence computer, and directly connect it to a lower-level system, providing a facility on the higher-level computer to securely sanitize data before releasing it at the lower level. This solution may not be viable however, because existing intelligence data bases are on computers for which no secure operating system exists, and which are, in some cases, not securable (because of lack of necessary hardware features).

The Guard application is an intermediate solution to this problem. The Guard is a secure minicomputer system that

acts as an interface between two computers of different security levels and allows data to flow between them in a secure and controlled manner. The Guard can provide better throughput than the switched terminal described above, and is much more flexible in its capabilities. The Guard application uses trusted processes and unlevel processes to accomplish its functions. The Guard will be described in more detail in a later section.

Secure database management systems

Using a secure computer system allows one to process multiple levels of data without having to overclassify the data. Using the system high mode of operation for example, all data of classification SECRET and below can be processed on a unilevel SECRET computer, but all outputs of the computer must be considered SECRET until a manual review process allows them to be downgraded. If multiple levels of information are to be organized and accessed in a meaningful manner, it makes sense to use some form of Database Management System (DBMS). However, most current database management systems do not take classification levels of data into account, so using them to process multiple levels of data would result in all of the data being processed at one level, causing some of the data to be overclassified, an adverse side effect to be avoided if possible.

Since the advent of secure system development work, there have been several attempts to incorporate multilevel security into a DBMS. These DBMS designs have relied to varying degrees on the kernel of the underlying secure operating system for protection of data in the database. Work in the design of secure database management systems has shown that although they can be designed to function in a secure environment, the environment provided by many past and present kernel designs may not be ideally suited to building a Secure DBMS. Another concern in the design of secure database management systems is the nature of the user interface to the database.

A later section will survey recent work in this area and describe some problems that still need to be solved.

Secure message processing systems

Another military application for secure computer systems is message-processing systems. These systems automate the task of day-to-day military message communication. These systems are in reality transaction-based database management systems, but have a much stronger requirement for good user interfaces than many other database applications if they are to be widely used by military staff.

Recently, there has been an effort to evaluate the use of such systems in a secure environment, and to design secure message-processing systems to run on secure operating systems. DARPA, the Navy and CINCPAC are conducting a Military Message Experiment to evaluate computer-aided message handling systems in an operational military envi-

ronment. MITRE's security-related role in the experiment has been to investigate the security ramifications of such systems: to identify the kernel primitives needed to implement the systems, and to identify the impacts that security imposes on the user interface.

Several message systems were developed during the experiment, and the SIGMA system developed by the Information Science Institute of the University of Southern California¹⁹ was chosen for installation. SIGMA is an interactive message handling system providing computer-aided message handling services for the receipt, filing, retrieval, creation and coordination of military messages. Although SIGMA runs on the (insecure) TENEX operating system, it presents a user interface that reflects DoD security policy. A key concept in the design of SIGMA is that of a multilevel terminal: a terminal with multiple "windows" on the screen, each of which can potentially contain information at different classification levels.

If such a system were implemented on a secure operating system, it would have to be integrated closely with the kernel. Either the kernel would have to be able to deal securely with the concept of a multilevel terminal, or some kind of trusted process would have this responsibility. FACC, as part of their KSOS effort, is investigating the applicability of KSOS as a host for multilevel terminals and secure message processing systems.¹

Secure front ends

Early in our Multics effort we discovered that to make a Multics system fully secure and able to handle multiple levels of terminals attached to the system in a feasible manner, that we had to make the Multics front-end terminal controller secure also. The front-end terminal controller has to multiplex the data streams coming from many terminals into the Multics computer, and it was quickly recognized that if this multiplexing were not done properly, untold security violations could result. We could have used multiple controllers each dedicated to a single level, but this usage of resources is wasteful and inflexible. Unfortunately, the Datanet 355 controller being used did not have the hardware features necessary to support a security kernel. The Multics project undertook, with Honeywell, the development of a secure communications processor, called the SCOMP.^{20,21} Currently, the SCOMP hardware (a modified Honeywell Level 6 minicomputer) is complete and Honeywell is building a version of KSOS to run on the SCOMP.

Thus, front-end controllers are another major type of application for secure computer systems. Indeed, this application is not limited to terminal controllers, though. Secure front-end controllers have application in secure computer networks, also. Front-ends have many different functions and require varying amounts of security support from secure systems. Some front-ends would need much trusted code and therefore run on the kernel interface. Some more sophisticated front-end applications may also need the type of file systems supported by the emulator portion. Currently, Ford Aerospace and Communications Corporation is look-

ing into the applicability of KSOS as a base for a Network Front-End for the World Wide Military Command and Control System, WWMCCS.

A DEEPER LOOK AT THE GUARD APPLICATION

The Guard application described earlier is being developed by Logicon under DARPA sponsorship at the Advanced Command and Control Architectural Testbed (ACCAT) in San Diego. The Guard application is expected to run on KSOS when KSOS becomes available. ACCAT, a joint DARPA/Navy undertaking, provides an ideal environment for Guard development. The Guard application is often referred to as the ACCAT Guard.^{22,23}

The ACCAT Guard system is a minicomputer that provides an interface between two computers or networks at different classifications. Hereafter, the computer or network of lower classification will be referred to as the LOW computer or network, and the higher classification computer or network will be referred to as HIGH.

Guard allows the two computers/networks of different classifications to communicate by providing (1) an upgrading facility to pass data from the LOW computer/network to the HIGH computer/network, and (2) a sanitization and downgrading facility to pass properly sanitized data from the HIGH computer/network to the LOW computer/network.

Two general classes of data transfers are provided by ACCAT Guard. The first is ARPANET network mail transfers. Network mail can normally flow among computers on the LOW network or among computers on the HIGH network, but cannot flow between the two networks at different classifications. ACCAT Guard allows mail to pass between the LOW and HIGH networks in a secure, controlled manner.

Second, Guard allows users on the LOW computer/network to query Datacomputer²⁴ databases residing on HIGH computers, and allows a properly sanitized response to be sent to the requesting user. Additionally, Guard accepts queries either in English or in Datalanguage, the Datacomputer database query language. English queries are translated into Datalanguage by a Guard operator.

The Guard minicomputer is connected to the LOW and HIGH networks through Private Line Interfaces (PLIs)²⁵ over the ARPANET. PLIs are encryption devices that allow a computer with a specific key to securely communicate with other computers having the same key. Thus the Guard has two distinct ARPANET connections that are at different security classifications. Figure 1 shows the connection of the Guard computer via PLIs and the ARPANET to the other computers. The LOW and HIGH computers are prohibited from directly communicating because their keys are different. They can communicate only through the Guard computer, which has two PLIs and keys to communicate with both the LOW and HIGH computers. Thus the Guard computer must control the communication between the LOW and HIGH computers. Note that there could be other computers in the network with PLIs keyed the same as the

LOW or HIGH computers. All such computers that share the same key form a secure "subnet" of the ARPANET. Thus the Guard can be viewed as an interface between two networks of different classifications.

Guard operating personnel

There are two types of personnel designated to operate the Guard system, Guard operators and Security Watch Officers. The main responsibility of the Guard operators is sanitization. A Guard system can have many Guard operators, whereas it has only one Security Watch Officer, whose function is to review all data downgraded by the Guard and approve or deny the downgrade. Thus the Security Watch Officer has responsibility for the security of the system. The specific duties of both Guard operators and Security Watch Officers are identified below in the context of each type of data transfer.

Guard operation

As outlined briefly above, Guard provides two types of communication between LOW and HIGH computers: network mail and database queries. Moreover, each of these two types of communication can be initiated by a user on a LOW computer or a HIGH computer. The operation of the Guard in handling each of these four cases is described in general terms below.

HIGH network mail

HIGH network mail is mail sent from a LOW computer and intended for delivery to a HIGH computer. No security violation is involved when sending mail from LOW to HIGH. The HIGH network mail enters the Guard system as mail through the LOW ARPANET interface, passes through the Guard Software with no human intervention, and leaves the Guard via the HIGH ARPANET interface. Thus neither the Guard operator nor the Security Watch Officer is involved in Guard processing of HIGH network mail.

LOW network mail

LOW network mail is mail sent from a HIGH computer and intended for delivery to a LOW computer. Such a transfer has a potential for a security violation if the data in the mail is classified higher than the LOW level for which the mail is intended. Therefore, LOW network mail is processed by the Guard system as follows.

First, the mail enters the Guard system as mail through the HIGH ARPANET interface. From there it is manually reviewed by the Security Watch Officer who must determine if the mail is free from information classified higher than the LOW level. If the Security Watch Officer determines that the mail can be sent, the mail leaves the Guard via the LOW

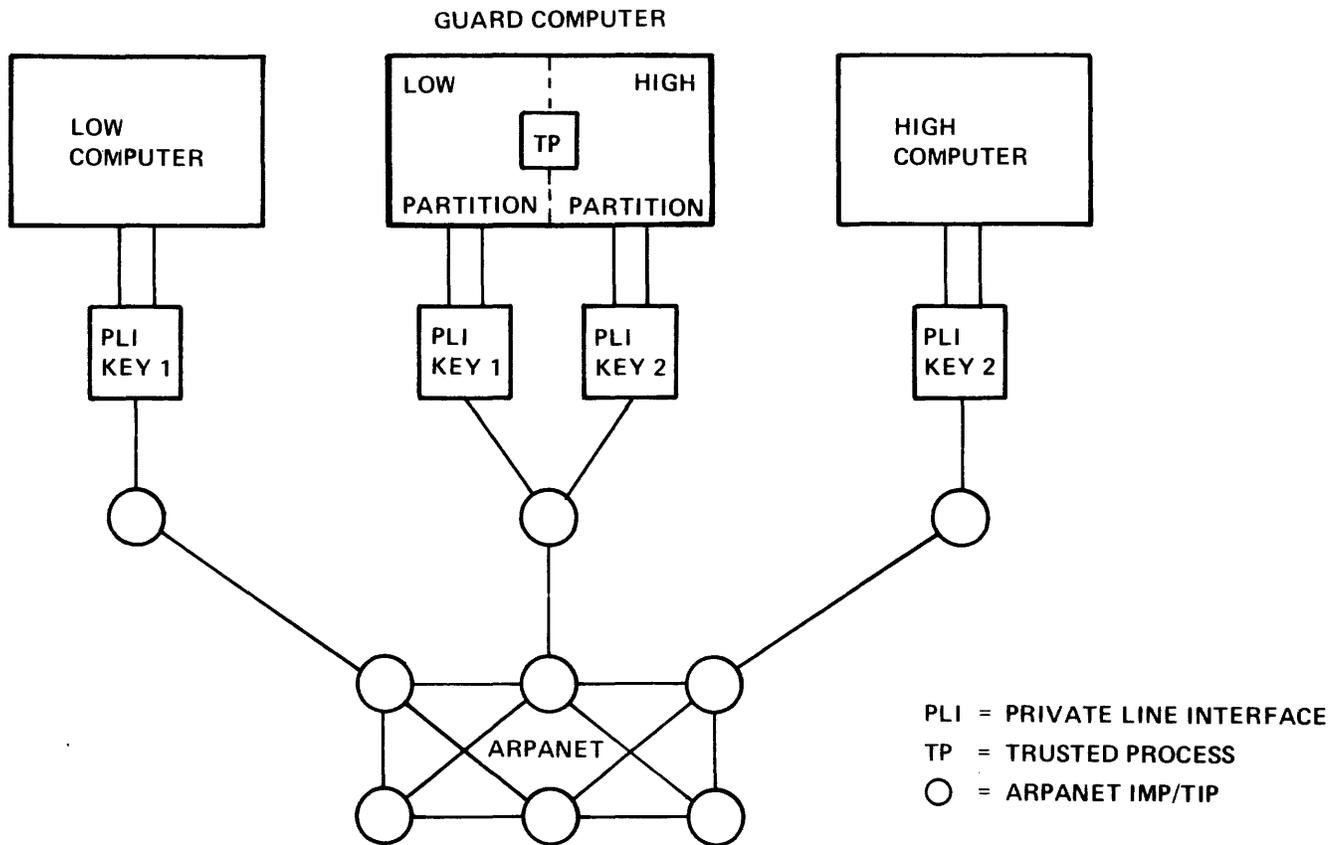


Figure 1—ACCAT guard ARPANET connections.

ARPANET interface. If the mail cannot be sent, the mail is sent back to the sender marked as rejected. Thus, only the Security Watch Officer is involved in Guard processing of LOW network mail.

HIGH database queries

A HIGH database query is a query originating on a LOW computer and intended for a HIGH computer database. Passing a query from LOW to HIGH does not involve a security violation. However, passing the reply from HIGH to LOW could potentially involve a security violation if the reply is not properly sanitized. HIGH database queries are processed by the Guard as follows.

First, the query enters the Guard system as mail through the LOW ARPANET interface. If the query is in Datalanguage, the query is passed through the Guard without human intervention, and passes through the HIGH ARPANET interface to the HIGH Datacomputer. If the query is in English, it is translated into Datalanguage by a Guard operator, and then passes to the HIGH Datacomputer through the HIGH ARPANET interface.

Replies coming from the HIGH Datacomputer (through the HIGH ARPANET interface) contain potentially classified information and are processed by the Guard system before passing to the requester. The replies are in Datalanguage, but when sanitized, may take on a different form. A

Guard operator reads the query and its response and attempts to sanitize the response by editing it to remove information pertaining to the nature of sources, etc. The Guard operator (sanitization officer) either sanitizes the response or replaces it with a message indicating that there was no response, and passes the query and the sanitized response to the Security Watch Officer, who must make the final determination of whether or not the response is passed on to the requester through the LOW ARPANET interface. If the Security Watch Officer decides against sending the response, the sanitization officer is so notified so he can do a better job of sanitization.

LOW database queries

A LOW database query is a query originating on a HIGH computer intended for the LOW Datacomputer. Passing such a query from HIGH to LOW involves a potential for a security violation if the query contains some data classified higher than LOW. However, passing the reply from LOW to HIGH does not involve a security violation. LOW database queries are processed by the Guard as follows.

The LOW database queries enter the Guard system as mail through the HIGH ARPANET interface. If the query is in Datalanguage, it is reviewed by the Security Watch Officer who allows or disallows the query. If the query is

disallowed, it passes back to the requester as mail (through the HIGH ARPANET interface) marked as rejected. If the query is allowed, it passes through the Guard to the LOW Datacomputer.

If the query is in English, it is translated into Datalanguage by a Guard operator, and then passes to the Security Watch Officer for review. If the Security Watch Officer allows the query, it passes to the LOW Datacomputer. If however the query is disallowed, the Guard operator is so notified so that he can take the appropriate action (i.e., to translate again or send the query back to the requester as rejected).

The reply from the LOW Datacomputer can pass back to the requester (from LOW to HIGH) without a security violation, and does so without human intervention. The reply is in Datalanguage.

Guard design

The programs that implement the Guard system are divided into three types of processes, LOW, HIGH, and TRUSTED. The integration of these processes with KSOS is shown in Figure 2. LOW processes have a security level

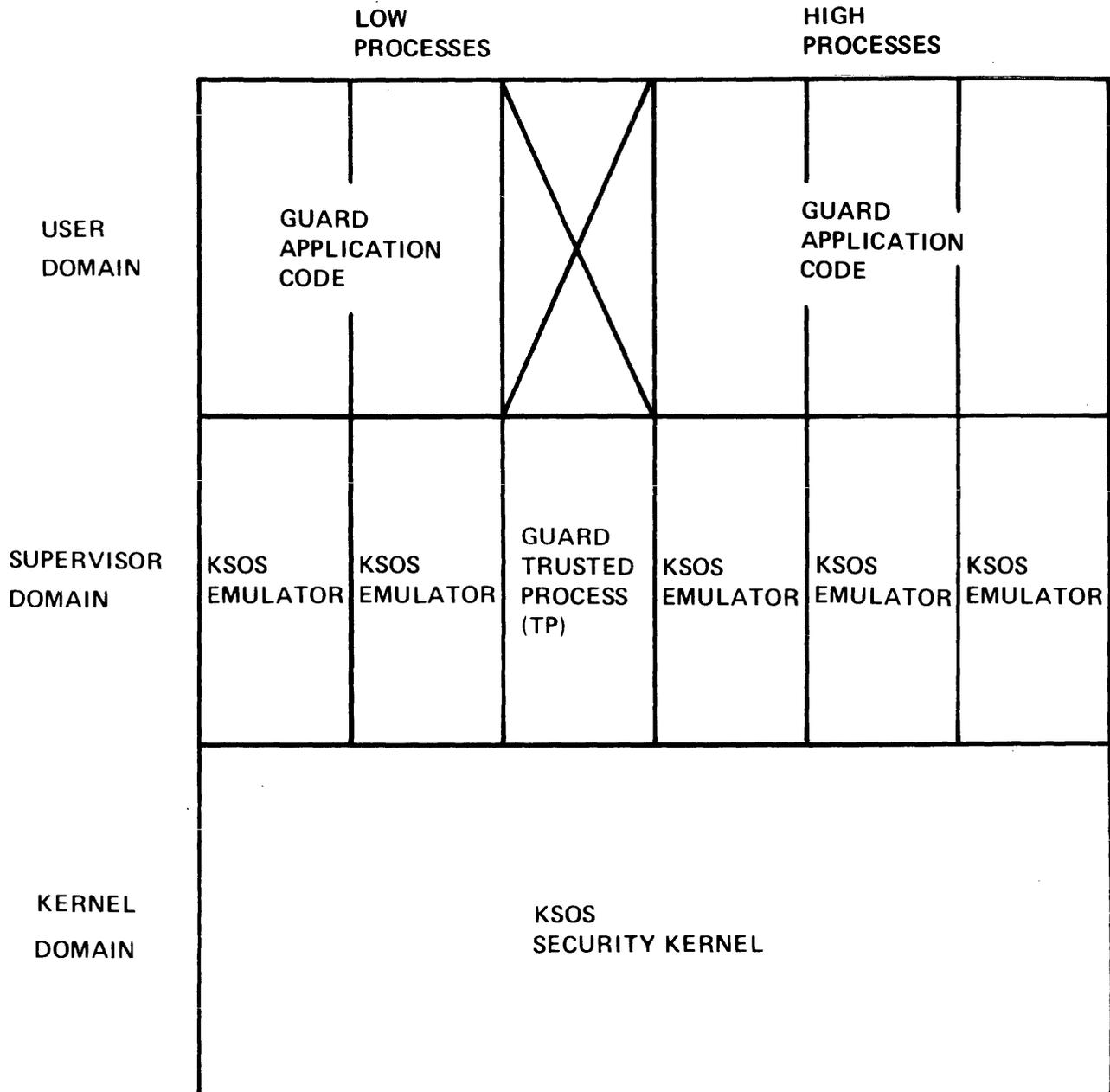


Figure 2—Integration of guard processes with KSOS.

equal to that of the LOW ARPANET connection, and have read/write access only to files at the LOW level. Similarly, HIGH processes have the same level as the HIGH ARPANET interface, and have read/write access only to files at the HIGH level. The TRUSTED processes are able to read and write both LOW and HIGH files, and therefore has the capability of passing data between LOW and HIGH processes.

The LOW and HIGH Guard processes deal with the LOW and HIGH ARPANET interfaces, respectively, and HIGH Guard processes provide the interface that Guard operators use to perform their functions of English to Datalanguage translation and HIGH response sanitization.

Just as the kernel is the encapsulation of the security-related portion of an operating system, the Guard TRUSTED processes are the security-related portion of the Guard code, and are therefore the only portion of the Guard that must be verified. The Guard TRUSTED processes run directly on the kernel, and have little or no knowledge of the environment created by the UNIX emulator. The other Guard processes run in a UNIX environment, and communicate with the TRUSTED processes via kernel-provided inter-process communication (IPC) messages.

The guard TRUSTED processes

Since the TRUSTED processes are the only security-related portion of the Guard application, they are the only Guard processes that we will discuss in more detail. The TRUSTED processes provide the interface to the Security Watch Officer and provide one major function, that of downgrading.

The downgrade function allows a HIGH process to send some data to a TRUSTED process for downgrading and release at the LOW level. This function is used to:

- Downgrade LOW Mail (mail sent from a HIGH user to a LOW user).
- Downgrade LOW Queries.
- Downgrade sanitized responses to HIGH Queries.

The downgrade function must be provided by a TRUSTED process because its operation violates the *-property. When the TRUSTED process receives some data for downgrading, it notifies the Security Watch Officer, who can read the data and decide whether or not the data can be released at the LOW level. The TRUSTED process releases the data only when so instructed by the Security Watch Officer. Thus the Security Watch Officer is solely responsible for what data is released at the LOW level.

Verification of the TRUSTED downgrade function involves demonstrating that the only data allowed to flow from HIGH to LOW by the TRUSTED process is data that has been seen by the Security Watch Officer and approved for downgrading. The Security Watch Officer's terminal is connected directly to the TRUSTED process through the kernel, so there is no unverified code between the terminal and

the TRUSTED process, and hence no chance for spoofing the Security Watch Officer.

A DEEPER LOOK AT SECURE DATABASE MANAGEMENT SYSTEMS

As mentioned earlier, the area of secure database management systems is one that has received some attention in the past, but that has many avenues yet to be explored. In this section we review some of the Air Force-sponsored work, and present some problems yet to be solved.

The Air Force Electronic Systems Division sponsored several research and development efforts in the design, specification, and validation of secure database management systems. Of primary importance are the contributions of System Development Corporation²⁶ and I. P. Sharp Associates Ltd.²⁷ Both of these studies helped to identify and clarify the key technical issues in secure database management technology.

The SDC secure data management system

The SDC effort concerned the design of a secure relational data base management system that interfaces with the multilevel environment provided by the secure Multics Operating system. SDC concluded that a relational data base could comfortably exist within the multilevel environment. The following technical evaluation of the SDC work is taken from Reference 26.

"The objective of this work was to develop a model and design of the security-related portions of a Data Management System (DMS). The model and design effort focused on those portions of a traditional DMS which are affected by the security constraints of the operating system. The result is a design framework which provides the basis for the development of a complete DMS which only has to draw on conventional DMS design technology.

"The mathematical model of a secure DMS encompasses DoD-based security policies which the DMS is to enforce with respect to some of the basic DMS operations. The modeling effort encompasses the modeling of the various levels of the DMS and its operating system interface.

"The modeling work suggested a design for a relational data management system that interfaces with the multilevel environment provided by the secure Multics operating system. The design utilizes the protection provided by the operating system in such a manner that the DMS contains no code which has an impact on the security of the data in the system and, thus, whose correctness must be verified. The DMS is designed according to the principle of least privilege; hence, the DMS operating as part of the user's process has no privileges that are not also afforded the user."

The DMS described in the SDC work is designed to store database information in the storage containers provided by the underlying Multics operating system: segments. In order to contain no security-relevant code, the DMS must use a number of unilevel segments, because segments are the smallest unit of protection in secure Multics.

The SDC work is important because it shows that a secure DBMS can exist on a secure system without any additional security-relevant code. However, the SDC effort does not address the additional flexibility that could be gained if smaller units of data could be efficiently protected. The SDC effort is a good example of a DBMS design around existing secure operating system primitives.

The I. P. Sharp protected DMS tool

The approach taken by I. P. Sharp is unique in that rather than working from existing kernel designs and primitives, they investigated the design of kernel primitives that would support the implementation of a family of secure data management systems. The primitives identified are referred to as the "DMS Tool." The study shows that the DMS Tool is general enough to apply to data management systems implemented as a dedicated DMS, as an application on a secure operating system, or in a computer network.

The view of the data in the I. P. Sharp work was relational.²⁸ The I. P. Sharp study recommended that each relation be assigned a single security level, i.e., the data in each relation must be considered at one level. Data from several levels cannot be gathered into a single relation and maintain their individual classifications. The resultant relations would have the classification of the most highly classified data that made up the relation.

The MITRE secure INGRES system

MITRE, working from these past studies, attempted to impose security constraints on the INGRES relational database management system, and to integrate the resultant secure INGRES system with the MITRE secure UNIX prototype.²⁹ This effort is important because it was one of the first actual implementations of a secure DMS on a prototype secure operating system. Like the SDC study, the MITRE effort worked with existing kernel primitives to investigate their sufficiency for supporting a secure DMS.

The approach taken by MITRE was also very similar to the SDC approach, in that no security-relevant code was added to the INGRES system. Use was made of the objects provided by the MITRE Secure UNIX kernel to accommodate the relations in an INGRES database.

The design of the MITRE Secure UNIX file system, like that of UNIX, is hierarchical, consisting of directories which may contain other directories or data files. In the MITRE design, data files in a directory assume the same access level as the directory. Consequently, the security level of an INGRES relation (a data file) must be at the same level as its database (a directory). Although this limitation does re-

duce the convenience of the secure INGRES system, the coordination of INGRES with the MITRE Secure UNIX necessitated the mapping of relations in this manner. Adhering to these restrictions, a user is still able to perform multilevel operations on relations in the INGRES database directory structure.

In order to use secure INGRES to process multiple levels of information, the user must create a database at each security level to be included in the database. Also, a user is able to "read" information from a file at a security level lower than his current level, established at login time. As a result, a new relation can be created by combining information obtained from relations in databases at access levels less than or equal to the level of the database being added to.

Areas for future work

There are two problems that have recurred in much of the past secure database design and implementation work:

1. The fact that protection of very small objects is difficult, and often results in relational database designs that force relations to be at the same level.
2. The fact that user interfaces to secure database management systems are seemingly difficult to design in such a way that the user is not greatly hindered by the security features.

The solution to the first problem seems to lie in the proper design of security primitives, and the implementation of protection features in the right places. For example, a DBMS can rely on the protection primitives provided by the underlying secure operating system, or it can, using trusted processes, provide its own protection, possibly at a much finer grain. Currently trusted processes are rather difficult to verify, so they are used sparingly. However, as verification technology develops, it will become much easier to design with trusted code. IBM has made a survey of many database management systems to determine the relationship between operating system and database system security.³⁰ Their conclusion is that it is most convenient to separate the design of kernel primitives and DBMS security features.

The solution to the user interface problem has been addressed in the context of message processing systems,³¹ which are really special cases of a DBMS. The techniques proposed in this paper need to be further studied and applied to more secure database designs.

CONCLUSIONS

Secure computer systems are moving rapidly out of the "research arena" and into the field. The KSOS effort will soon demonstrate that multilevel secure operating systems can be built feasibly and cost-effectively. Many important applications for KSOS and other secure operating systems

are being designed and developed now, and the potential for future applications is tremendous.

REFERENCES

1. Padlipski, M. A., et al., "KSOS: Computer Network Applications," 1979 National Computer Conference.
2. "ESD 1974 Computer Security Developments Summary," MCI-75-1, Air Force Electronic Systems Division (AFSC), L. G. Hanscom Field, Bedford, Massachusetts, December 1974.
3. Anderson, J. P., "Computer Security Technology Planning Study," ESD-TR-73-51, Volumes I and II, James P. Anderson & Co., Fort Washington, Pennsylvania, October 1972.
4. Bell, D. E., and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," M74-224, The MITRE Corporation, Bedford, Massachusetts, October 1974.
5. Walter, K. G., et al., "Primitive Models for Computer Security," ESD-TR-74-117, Case Western Reserve University, Cleveland, Ohio, January 1974.
6. Bell, D. E., and L. J. LaPadula, "Secure Computer Systems," ESD-TR-23-278, Volumes I-III, The MITRE Corporation, Bedford, Massachusetts, November 1973-June 1974.
7. Bell, D. E., and E. L. Burke, "A Software Validation Technique for Certification. Part 1: The Methodology," ESD-TR-75-54, Volume 1, The MITRE Corporation, Bedford, Massachusetts, April 1975 (AD 009849).
8. Millen, J. K., "Security Kernel Validation in Practice," *Communications of the ACM*, Vol. 19, No. 5, May 1976, pp. 243-250.
9. Robinson, L., P. G. Neumann, K. N. Levitt and A. R. Saxena, "On Attaining Reliable Software for a Secure Operating System," 1975 *International Conference on Reliable Software*, Los Angeles, California, April 1975, pp. 267-284.
10. Schiller, W. L., "The Design and Specification of a Security Kernel for the PDP-1145," ESD-TR-75-69, The MITRE Corporation, Bedford, Massachusetts, May 1975 (AD A011712).
11. Schiller, W. L., P. T. Withington and J. P. L. Woodward, "Design and Abstract Specification of a Multics Security Kernel," ESD-TR-77-259, Vols. I-III, The MITRE Corporation, Bedford, Massachusetts.
12. Ames, S. R., J. K. Millen, "Interface Verification for A Security Kernel," *INFOTECH State of the Art Report: System Reliability and Integrity*, Vol. 2, INFOTECH International, pp. 1-22.
13. Popek, G. J., et al., "UCLA Data Secure UNIX—A Securable Operating System: Software Architecture," SDPS-78-003, University of California, Los Angeles, California, August 1978.
14. Woodward, J. P. L., and G. H. Nibaldi, "A Kernel-Based Secure UNIX Design," MTR-3499, The MITRE Corporation, Bedford, Massachusetts, November 1977.
15. McCauley, E. J., et al., "KSOS: Design of a Secure Operating System," Ford Aerospace and Communications Corporation, Palo Alto, California.
16. Berson, T. A., et al., "KSOS: Development Methodology For a Secure Operating System," Ford Aerospace and Communications Corporation, Palo Alto, California.
17. "Kernelized Secure Operating System—System Specification," TRW, Redondo Beach, California, April 1978.
18. "Secure Minicomputer Operating System (KSOS) System Specification (Type A)," WDL-TR7808, Rev. 1, Ford Aerospace and Communications Corporation, Palo Alto, California, July 1978.
19. Ames, S. R., and D. R. Oestreicher, "Design of a Message Processing System for a Multilevel Secure Environment," 1978 National Computer Conference, Anaheim, California, June 1978, pp. 765-771.
20. Broadbridge, R., and J. Mekota, "Secure Communications Processor Specification," ESD-TR-76-351, Honeywell Information Systems, Incorporated, Federal Systems Division, McLean, Virginia, June 1976.
21. Rolfe, G., and J. Carnall, "Detail Specification for the Security Protection Module (SPM)," ESD-TR-76-366, Honeywell Information Systems, Incorporated, Aerospace Division, St. Petersburg, Florida, September 1976.
22. Woodward, J. P. L., "ACCAT Guard System Specification (Type A)," MTR-3634, The MITRE Corporation, Bedford, Massachusetts, August 1978.
23. "ACCAT Guard Computer Program Development Specification (Type B5)," ARPA-78C0323-01, LOGICON, San Diego, California, October 1978.
24. "Datacomputer Version 1 User Manual," Computer Corporation of America, Cambridge, Massachusetts, August 1975.
25. BBN Report 1822, Appendix H, Bolt, Beranek, and Newman, Cambridge, Massachusetts, 1977.
26. Hinke, T. H., and M. Schaefer, "Secure Data Management System," RADC-TR-75-266, System Development Corporation, Santa Monica, California, November 1975.
27. Kirkby, G., and M. Grohn, "On Specifying the Functional Design for a Protected DMS Tool," ESD-TR-77-140, I. P. Sharp Associates, Ltd., Ottawa, Canada, March 1977.
28. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, Vol. 13, No. 6, June 1970.
29. Wagner, B. N., "Implementation of a Secure Data Management System for the Secure UNIX Operating System," MTR-3524, The MITRE Corporation, Bedford, Massachusetts, September 1977.
30. Fernandez, E. B., and C. Wood, "The Relationship Between Operating System and Database System Security: A Survey," *Proceedings of the 1977 IEEE COMPSAC Conference*, pp. 453-462.
31. Ames, S. R., "User Interface Multilevel Security Issues In a Transaction-Oriented Data Base Management System," MTP-178, The MITRE Corporation, December 1976.

The foundations of a provably secure operating system (PSOS)

by RICHARD J. FEIERTAG and PETER G. NEUMANN

SRI International
Menlo Park, California

INTRODUCTION

PSOS has been designed according to a set of formal techniques embodying the SRI Hierarchical Development Methodology (HDM). HDM has been described elsewhere,¹⁻³ and thus is only summarized here. The influence of HDM on the security of PSOS is also discussed elsewhere.⁴ In addition, Linden⁵ gives a general discussion of the impact of structured design techniques on the security of operating systems (including capability systems).

HDM employs formally stated requirements, formal specifications defining the design of each module in a hierarchical collection of modules, and formal statements of the module interconnections. In the case of PSOS, there is a formal model describing the requirements of the basic protection mechanism, and additional formal models of the requirements of various applications (e.g., Reference 6). HDM provides the formalism and the structure that make the formal verification of the system design and implementation possible and conceptually straightforward. This formal verification consists of formal proofs that specifications satisfy the desired requirements,⁶ and subsequently that the actual programs for the system and its applications are consistent with those specifications.²

The design of PSOS has been formally specified using a SPECification and Assertion Language called SPECIAL.⁷ These specifications¹ define PSOS as a collection of about 20 hierarchically-organized modules. Each module typically is responsible for objects of a particular type defined by that module. From the user point of view, the most important modules are those for capabilities, for virtual memory segments, directories, user processes, and for creating user defined abstract objects. Some modules are to be implemented in software, some in firmware, and some in hardware—as dictated by the efficiency required.

Capabilities provide the protection mechanism for all such objects in PSOS, and are discussed in the next section. The subsequent sections of this paper summarize the development methodology used in PSOS, present the protection mechanism provided by PSOS capabilities, exhibit its properties, show its applicability in developing data and procedure abstractions, and contrast the PSOS approach with the kernel approach to achieving secure systems. There are

many important issues relating to the use of capabilities in PSOS (and other computer systems) that are not presented in this paper. Many of these issues are discussed in the references cited here.

PSOS CAPABILITIES

The concept of the capability has appeared in several other operating systems (e.g. References 8-13). Although capabilities are a fundamental part of the design of each of these systems, they all differ in the way they use and interpret capabilities. PSOS differs from its predecessors in its uniform use of capabilities throughout the system and in the simplicity and primitive nature of the basic capability mechanism.

Each object in PSOS can be accessed only upon presentation of an appropriate capability to a module responsible for that object. Capabilities can be neither forged nor altered. As a consequence, capabilities provide a controllable basis for implementing the operating system and its applications, as there is no other way of accessing an object other than by presenting an appropriate capability designating that object.

Each PSOS capability consists of two parts, a *unique identifier* (uid) and a set of *access rights* (represented as a boolean array). By definition neither part is modifiable, once a capability is created.

- *Unique Identifiers*—PSOS generates only one original capability for each uid. Any number of copies can be made of a given capability, but making a copy requires presenting an existing capability for which a copy is to be made. Therefore, a procedure or task that creates a new capability with some uid knows that the only capabilities that can have that uid must have been copied either directly or indirectly from the original. In other words, the creator of a capability with a given uid is able to retain control over the distribution of capabilities with that uid.
- *Access Rights*—The set of access rights in a capability for an object is interpreted by the module responsible for that object to define what operations may be per-

formed by using that capability. The interpretation of the access rights is constrained by a *monotonicity rule*, namely that the presence of a right is always more powerful than its absence. The interpretation of the access rights may differ for different objects, but the monotonicity rule must always apply.

The access rights for a segment capability (as interpreted by the segment manager) indicate whether that capability may be used to write information into the designated segment, to read that information, to call that segment as a procedure, and to delete that segment. In PSOS, a directory contains entries, each of which is a mapping from a symbolic object name to a capability. Each directory is accessed via a capability for that directory. For directories, the interpretation of access rights is done by the directory manager. The access rights for a directory capability indicate whether that capability may be used to add entries to the designated directory, to remove entries, and to use the capability contained in that entry.

A copy of a capability may be made, but the resulting capability cannot have any access rights that the original capability did not—as is seen from the following list of possible operations upon capabilities. (There are other access rights, meaningful to capabilities of all types, to be discussed under storage permissions.)

THE PSOS PROTECTION MECHANISM

Capabilities provide the basis for a flexible protection mechanism, as follows:

Tagging of capabilities

In PSOS, capabilities can be distinguished from other data because they are tagged throughout the system (i.e., in the processor, and in both primary and secondary memory) by means of a tag bit inaccessible to programs. Consequently, the hardware can enforce the nonforgeability and unalterability of capabilities.

Operations upon capabilities

There are only two basic operations that involve actions upon capabilities (as opposed to actions based on capabilities, which is the normal mode of accessing objects), as follows.

`c=create__capability` creates a new capability (i.e., with a previously unused uid) having all access rights.
`cl=restrict__access(c, mask)` creates a capability with the same uid as the given capability `c` and with access rights that are the intersection of those of the given capability `c` and the given maximum (`mask`): i.e., it creates a possibly restricted copy.

Store permissions

The second capability operation described above appears to permit unrestricted copying of capabilities. For certain types of security policies this unrestricted copying is too liberal. For example, one may wish to give the ability to access some object to a particular user but not permit that user to pass that ability on to other users. Because simplicity of the basic capability mechanism is extremely important to achieve the goals of PSOS, any means for restricting the propagation of capabilities should not add complexity to the capability mechanism.

A few access rights (only one is currently used by PSOS itself) are reserved as *store permissions*. This is the only burden placed on the capability mechanism. The interpretation of the store permissions is performed by the basic storage object manager of PSOS, namely the segment manager. Each segment in the system is designated as to whether or not it is capability store limited for each store permission. If a segment is capability store limited for a particular store permission, then it can contain only capabilities that have that store permission. This restriction can be enforced by a simple check on all segment-modifying operations.

By properly choosing the segments that are capability store limited, some very useful restrictions on the propagation of capabilities can be achieved. The restriction used in PSOS is not allowing a process to pass certain capabilities to other processes or to place these capabilities in storage locations (e.g., a directory or interprocess communication channel) accessible to other processes. (Other restrictions are also possible using store permissions, such as restricting a capability to a subsystem or a particular invocation of a subsystem. For example, see Reference 1, page II-25.) More general means for restricting propagation of capabilities and for revoking the privilege granted by a capability can be implemented as subsystems of PSOS. The store permission mechanism has been selected as primitive in the system because it achieves the desired result with negligible additional complexity or cost.

DATA AND PROCEDURE ABSTRACTIONS

PSOS consists of a collection of data and procedure abstractions constructed in a hierarchical fashion as shown in Table I. Each level in the hierarchy represents a collection of abstractions introduced at that level. Abstractions at higher (numbered) levels are implemented using abstract objects introduced at lower levels in the design. It is unimportant whether an abstraction is implemented in hardware, firmware, or software. It is reasonable that abstractions introduced at lower levels be implemented largely in hardware or firmware and that abstractions introduced at higher levels be implemented largely in software. However the demarcation between hardware and software is not established by the design, and it is quite possible that abstractions occurring throughout the system be implemented as hybrids, i.e., partially in hardware and partially in software.

TABLE I—PSOS Abstraction Hierarchy.

Level	Abstractions
16	user request interpretation
15	user environments and name spaces
14	user input-output
13	procedure records
12	user processes and visible input-output
11	creation and deletion of user objects
10	directories
9	abstract object manager
8	segments and windows
7	pages
6	system processes and system input-output
5	primitive input-output
4	arithmetic and other basic procedures
3	clocks
2	interrupts
1	registers and other storage
0	capabilities

It is convenient to group the levels of Table I into generic categories as shown in Table II. The generic categories collect abstractions satisfying similar goals. At the base of the hierarchy is the capability mechanism, from which all other abstractions in the system are constructed. Above the basic capability mechanisms are all the physical resources of the system, e.g., primary and secondary storage, processors and input/output devices. From the physical resources are constructed the virtual resources. These virtual resources present a more convenient interface to the programmer than the physical resources, permit multiplexing of the physical resources in a manner largely invisible to the user, and allow the system to allocate the physical resources so as to maximize their efficient use. Next in the PSOS hierarchy comes the abstract object manager, providing the mechanism by which higher-level abstractions may be created. As will be discussed in detail, it is possible to construct higher-level abstractions based solely on the capability mechanism; however, the abstract object manager provides services that make construction of such abstractions easier. The top two categories in the generic hierarchy include community abstractions and user-created abstractions. The community abstractions are intended to be used by a large group of users, e.g., by all the users at a particular site. Such abstractions may be simple utility routines such as a compiler, or may actually create and control access to new virtual resources such as directories. The user abstractions are those intended for use by a limited group of individuals.

Of the properties stated previously, there are two impor-

tant ones that make the PSOS capability particularly useful in the construction of abstract objects.

1. The capability serves as a *unique* name for an abstract object.
2. The capability is unforgeable.

This means that a capability can be used as a name (guaranteed to be unique) by which an abstract object can be referenced, and access to the object can be controlled by limiting the distribution of the capability.

In addition, there are several important pragmatic reasons why PSOS capabilities are useful as a naming and protection mechanism for supporting abstract objects.

1. The capability mechanism has a very simple implementation. This allows capabilities to be built into the system at the lowest level of abstraction, thus making capabilities available for the most primitive objects.
2. Capabilities are uniform in size, making them easy to manage.
3. The inclusion of access rights in capabilities permits efficient fine-grained control of access to objects.
4. Capabilities can be written into storage (including secondary storage) and retrieved from storage in the same manner as other data, and therefore have many of the properties of other data.

Capabilities serve as names or tokens for all objects of PSOS. It is because the basic capability mechanism is so simple in concept and in implementation that construction of the most primitive objects (e.g., input/output channels, processors, and primary memory) as well as the most complex system objects (e.g., directories and user processes) and user application objects (e.g., a data management system) is possible using capabilities. This promotes a high degree of uniformity throughout the system and eliminates the need for many special-purpose facilities.

Objects that have many properties and operations in common and are managed by a single program are said to have a common *type*; that program is called a *type manager*. The type manager implements operations on an abstract object in terms of operations on the more primitive objects used to represent the abstract object. The type manager must be able to determine which objects are part of the representation used to implement an abstract object denoted by a given capability. In other words, a type manager must be able to map the unique identifier of a given capability into capabilities for its representation objects. The capability mechanism of PSOS does not predispose a type manager to any particular implementation of this mapping. Different type managers will require diverse mapping algorithms, depending upon the number of abstract objects and representation objects they must manage, the desired efficiency of operations on the abstract object, the desired simplicity of the mapping algorithm, and numerous other factors. For example, the segment type manager uses a mapping algorithm that is in almost all cases extremely fast; however, the algorithm is

TABLE II—PSOS Generic Hierarchy.

Level	Abstractions	PSOS Levels
F	user abstractions	14-16
E	community abstractions	10-13
D	abstract object manager	9
C	virtual resources	6-8
B	physical resources	1-5
A	capabilities	0

quite complex, requiring implementation in both hardware and software. Extreme speed is essential to the operations of the segment type manager because the segment operations are used very frequently (at least once on every instruction). The directory type manager uses a less speedy algorithm because fast access is not essential.

Although the capability mechanism of PSOS does not prescribe a particular mapping algorithm, the system does provide some assistance in managing abstract objects. The *abstract object manager* provides a set of operations by which type managers can associate capabilities for abstract objects with the capabilities for their representation objects. The type manager can then retrieve the representation capabilities by presenting to the abstract object manager the abstract object capability. This is done in such a way that *only* the type manager program itself can obtain the representation capabilities, and then *only* upon presentation of the abstract object capability. The abstract object manager performs the mapping from abstract object capabilities to representation object capabilities, some of the bookkeeping functions necessary to implement abstract objects, and some storage allocation. Although the abstract object manager is intended to be useful and appropriate for a wide variety of type managers and does make the programming of a type manager much easier, it is only a service and is not essential to the construction of type managers.

The capability mechanism itself could have been constructed with many of the facilities of the abstract object manager included. This would have resulted in a capability mechanism that would be more elaborate and—for some applications—more efficient and easier to use. This is the approach taken by other capability systems cited above. On the other hand, such a capability mechanism would have required a more complex implementation. More significantly, the capability mechanism could then not have been placed at the lowest level of abstraction in the system design, and some of the physical and virtual resources of the system could not have been implemented using capabilities—requiring a different means for reference. Although having several different naming schemes is possible (and common in most systems), it destroys the uniformity, conceptual simplicity, elegance, ease of use, and possibly the efficiency of the system. It is for this reason that PSOS has a very simple, but fully general, capability mechanism, and that programs enhancing the use of the capability mechanism can be introduced as extensions at higher levels of the design.

As noted above, there is no clearly-delineated system boundary in PSOS. One would normally draw the system boundary at the interface to the community abstractions. However, all the programs that implement the community abstractions (such as directories or user processes) could be provided by users as user programs. The community programs have no special privilege other than claiming resources at initialization by taking possession of certain capabilities. For example, the user-process type manager takes possession of the capabilities for certain system processes which it then multiplexes to create many user processes. If the system's user-process type manager did not claim all the

available system processes, then it would be possible for a user to provide a different user-process type manager with the same or different facilities. Similarly, the abstract object manager has no special privilege at all. A user might program his own abstract object manager if he so desired.

The abstractions at or below any level in the design of Table I form a consistent and useful system. Clearly, the lower the level chosen as the "top level" of the system, the more primitive that system will be. If all of the physical resources (levels 1 through 5) are present, then the full PSOS could be reconstructed on the restricted system, but more likely, one would construct a somewhat different system. Thus, the PSOS design represents a family of systems. One can choose the level that provides the best set of resources to fulfill the needs of the desired system without having to include unnecessary facilities. Then one can augment this level with new type managers to create abstract objects appropriate to the desired applications. Writing such "system" type managers requires no additional skill or privilege other than that required to write ordinary user programs. The distinction between a "system" program and a "user" program is thus indeed blurred.

PSOS RELATIONSHIP TO KERNELS

Several recent operating systems have been constructed using a "kernel" architecture. Such systems include the Kernelized Secure Operating System (KSOS)^{14,15} and two precursor systems developed at MITRE^{16,17} and UCLA.¹⁸ The term *kernel* is used loosely in the literature, but for the purpose of this discussion a kernel is that part of the operating system that is both necessary and sufficient to satisfy certain requirements of the system. For example, if the essential requirement of a system is that it enforce a certain security policy, then that part of the system that enforces the security policy constitutes the kernel. By this definition, a kernel is meaningful only with respect to some requirement or some set of requirements. The kernel must contain all those parts of the system that pertain to meeting the requirements, i.e., there is no part of the system outside the kernel that can cause the system not to meet its requirements. Also, the kernel can contain only those parts of the system that are necessary to meet the requirements, i.e., the kernel should not contain anything that does not pertain to the meeting of the requirements. The reasoning behind kernel-based architectures is that since a kernel contains only that part of the system essential to meeting requirements, it can be small, compared to the system as a whole, and therefore has a better chance of being correct.

One of the main advantages of the kernel approach is the clear statement of purpose of the system. Since a kernel is meaningful only with respect to some explicit requirements, these requirements serve as the statement of purpose of the system. The other main advantage is the enhanced probability of correct operation. Since the programs that are critical to the correct operation of the system are isolated in the kernel, a great deal of attention can be paid to getting this code right, and less attention can be paid to other system

code that may be important but is not critical. The relatively small size of the kernel significantly improves the chances of applying formal verification techniques to the programs in the kernel in a cost-effective manner, where applying these techniques to the entire system would be unwieldy.

There are, as one might expect, some disadvantages to the use of kernels. Kernels cannot be casually modified because, by definition, all the code in the kernel is essential to meeting the requirements of the system, and any modification is likely to cause the system to deviate from that which is required. One must take extreme care to be sure that a change in the kernel will not compromise its correct operation.

In order to be able to construct a small kernel, the requirements must be fairly narrow and highly specific. Such requirements limit the applications for which the kernel is useful. For example, if the requirement of a kernel is to enforce a particular security policy, then only applications requiring that policy can be reasonably implemented using that kernel. It is not possible to implement another security policy that is inconsistent with the given security policy.

Yet another of the major problems with the kernel approach is the difficulty of designing a system in such a way that those programs essential to meeting the requirements are isolated from the nonessential programs. Finally, experience with the systems mentioned above indicates that kernels are still quite large. Clearly the size of a kernel depends upon the requirements it is supposed to meet, but reasonable requirements tend to require a large part of the system to be part of the kernel. Large kernels do not enhance one's confidence in the correct operation of the system.

Consider, for example, the kernels of KSOS and of the MITRE system. The requirement of these kernels is that they enforce a multilevel security policy. Upon close examination of these systems, it is seen that what is labeled the "kernel" is not really the kernel at all, but is only a part of the kernel. These systems have so-called *trusted processes*, namely programs that are internally able to violate the requirements, but whose external interface is consistent with the requirements. These trusted processes include programs for file system backup and retrieval, I/O spooling, and network interfaces. These programs are not labeled as part of the kernel because their function is in some sense peripheral to the main task of the system. However, their correct operation is as essential to meeting the requirements as any kernel program. If the system is to be proven correct, the programs that are used by the trusted processes must be formally verified. Inclusion of the code for the trusted processes into the kernel makes the resulting kernel much larger. This illustrates a difficulty in designing a small kernel.

It is a matter of judgment as to whether the advantages of the kernel approach outweigh the disadvantages. For the situation in which one has clearly defined, specific overriding requirements for which a small kernel can be constructed, then the kernel approach is ideal.

PSOS is well suited to situations in which one wants to support many applications with different or conflicting requirements. Because PSOS is highly extensible and easily supports different type managers with strong control over

access to objects and type managers, it makes possible the support of many different sets of requirements on one PSOS implementation. For example, several subsystems have been designed for PSOS that enforce different security constraints. A particular task could be constrained to have access to only one of these subsystems, but several tasks may be executing different subsystems simultaneously. In a sense, each of these subsystems can be viewed as a "kernel" for the tasks having access to them, but PSOS can support any number of such subsystems. Of course, one still has the problem of assuring the correctness of these subsystems and those parts of PSOS which the subsystems use. However, assuring the correctness of these subsystems on PSOS should be significantly easier than assuring the correctness of a stand-alone kernel, because each subsystem will be much smaller and simpler than it would be if it had to be implemented as a stand-alone system.

The UCLA system¹⁸ is an interesting case in that its requirements for security are very broad and general. The UCLA kernel attempts to be like PSOS in its ability to support a wide range of security policies simultaneously. However, the resulting requirement does not permit as wide a range of policies to be implemented, and the system design is not as uniform or elegant as the PSOS design.

SUMMARY

The capabilities of PSOS provide a flexible naming and protection mechanism that can be used to implement arbitrarily complex subsystems efficiently fulfilling a wide variety of requirements. The properties of PSOS that make this possible are summarized as follows.

1. The capability mechanism is extremely simple, with only two operations involving the creation of capabilities and none permitting the alteration of capabilities. There is no policy embedded in the mechanism.
2. The operations on capabilities can be completely controlled at the most primitive conceptual level of the system design and implemented in hardware. Capabilities are tagged and nonforgeable, and the protection they provide is not bypassable.
3. Capabilities and other PSOS facilities encourage strong modularity via the creation of data and procedure abstractions. Such abstractions are the basis of the design of the PSOS system itself and can be used equally well in application programming.
4. The capability mechanism can be used equally well for user programs, application subsystems, and system programs. There are no special protection mechanisms necessary to protect system programs.
5. The capability mechanism is fully general and can simultaneously support subsystems that implement arbitrary policies. Mechanisms for initialization, backup and recovery, and auditing for both PSOS and its subsystems can be constructed without subverting the protection mechanism.
6. The operations that can be performed on an object of

a particular type are precisely those defined by the type manager for that object. The operations permitted upon the particular object designated by a given capability are limited by the access rights of the given capability.

7. If a user is in possession of only one capability for an object, and he wishes to confer some or all of the access rights to another user (or to another program), he may create and pass a new capability whose access rights are a subset of those of the original capability. There is no way in which an additional access right can be introduced. (Note, however, that type managers must consistently enforce the monotonicity of access rights. That is, the presence of the right must be more powerful than the absence of that right. This is guaranteed for system-defined object types, and must be assured by the type managers for other types.)
8. Propagation of capabilities can be restricted by use of capability store permissions. The passage of a capability to other users can be prevented by not including process store permission in that capability's access rights.

Although no single commercially available computer has the facilities necessary to implement PSOS efficiently, each of the required facilities does exist on some computer. Therefore, the proper hardware support for PSOS can be implemented using established techniques. The formal techniques used to design PSOS make implementation straightforward¹⁹ and make formal verification of correct operation possible. All of the advantages summarized here can make PSOS and subsystems implemented on PSOS far more secure and reliable than contemporary operating systems.

ACKNOWLEDGMENTS

The design of PSOS was accomplished by the close co-operation of several people. Outstanding among these are Larry Robinson who is primarily responsible for the development of HDM and who played a major role in the early design of the system, Karl Levitt who designed security related subsystems, and Bob Boyer who is responsible for the formal mathematical theory.

REFERENCES

1. Neumann, P. G., R. S. Boyer, R. J. Feiertag, K. N. Levitt and L. Robinson, "A Provably Secure Operating System: the System, its Applications, and Proofs," SRI International, Menlo Park, California, February 1977.
2. Robinson, L., and K. N. Levitt, "Proof Techniques for Hierarchically Structured Programs," *Communications of the ACM*, Vol. 20, No. 4, April 1977.
3. Robinson, L., K. N. Levitt, P. G. Neumann and A. R. Saxena, "A Formal Methodology for the Design of Operating System Software," in *Current Trends in Programming Methodology*, R. T. Yeh ed., Vol. 1, Prentice-Hall, Englewood Cliffs, New Jersey, April 1977.
4. Neumann, P. G., "Computer System Security Evaluation," *AFIPS Conf. Proc.*, NCC 1978, Anaheim, California, January 1978, pp. 1087-1095.
5. Linden, T. A., "Operating System Structures to Support Security and Reliable Software," *Computing Surveys*, Vol. 8, No. 4, December 1976, pp. 409-445.
6. Feiertag, R. J., K. N. Levitt and L. Robinson, "Proving Multilevel Security of a System Design," *Proc. ACM Sixth Symposium on Operating Systems Principles*, November 1977, pp. 57-65.
7. Roubine, O., and L. Robinson, *SPECIAL Reference Manual*. SRI International, Menlo Park, California, January 1977.
8. Lampson, B. W., and H. E. Sturgis, "Reflections on an Operating System Design," *Communications of the ACM*, Vol. 19, No. 5, May 1976, pp. 251-266.
9. Lampson, B. W., "Dynamic Protection Structures," *Proc. 1969 AFIPS Fall Joint Computer Conference*, Vol. 35, AFIPS Press, Montvale, New Jersey, 1969, pp. 27-38.
10. Sevcik, K. C., "Project SUE as a Learning Experience," *Proc. AFIPS 1972 Fall Joint Computer Conference*, Vol. 40, AFIPS Press, Montvale, New Jersey, 1972, pp. 571-578.
11. Wulf, W. A., et. al., "HYDRA: the Kernel of a Multiprocessor Operating System," *Communications of the ACM*, Vol. 17, No. 6, June 1974, pp. 337-345.
12. Needham, R., "Protection Systems and Protection Implementations," *Proc. 1972 AFIPS Fall Joint Computer Conference*, Vol. 41, AFIPS Press, Montvale, New Jersey, 1972, pp. 571-578.
13. England, D. M., "Capability Concept Mechanism and Structure in System 250," *Proc. IRIA International Workshop on Protection in Operating Systems*, Institut de Recherche d'Informatique et de Automatique, France, 1974, pp. 63-82.
14. McCauley, E. J., and P. Drongowski, "KSOS: Design of a Secure Operating System," NCC '79, New York, New York, June 1979.
15. Berson, T., and J. Barksdale, "KSOS: Development Methodology for a Secure Operating System," NCC '79, New York, New York, June 1979.
16. Millen, J. K., "Security Kernel Validation in Practice," *Communications of the ACM*, Vol. 19, No. 5, May 1976, pp. 243-250.
17. Schiller, W. L., "The Design and Specification of a Security Kernel for the PDP-11/45," ESD-TR-75-69, The MITRE Corporation, Bedford, Massachusetts, March 1975.
18. Popek, G. J., and D. A. Farber, "A Model for Verification of Data Security in Operating Systems," *Communications of the ACM*, Vol. 21, No. 9, September 1978, pp. 737-749.
19. DeLashmutter, L. F., Jr., "Steps Toward a Provably Secure Operating System," *Spring '79 COMPCOM, Digest of Papers*, February-March 1979, pp. 40-43.

A security retrofit of VM/370

by B. D. GOLD, R. R. LINDE, R. J. PEELER, M. SCHAEFER, J. F. SCHEID and P. D. WARD

System Development Corporation
Santa Monica, California

INTRODUCTION

The VM/370 Security Retrofit Program is a continuing research and development initiative, funded by the Defense Advanced Research Projects Agency (DARPA), with additional funding provided by the Canadian Department of National Defense. The program's primary goal is the security retrofit of a popular commercial operating system, VM/370.¹ Two approaches were originally planned: (1) the design of a feasible, formally verified security kernel to VM/370 and (2) a "hardening" effort to repair known VM/370 penetration weaknesses. It was subsequently decided not to proceed with the VM/370 hardening task because of the uncertainty of the end result: correction of known security flaws does not guarantee the absence of exploitable, but not yet detected, security flaws in the hardened system.

In the first year of the research program, the feasibility of adding a security kernel to VM/370 was studied and a kernel design for the system was produced. The retrofitted system is called KVM/370 (for Kernelized VM/370). The security enforcement mechanism, the kernel, must implement a reference monitor² that enforces a security policy. A security kernel is a reference monitor that:

- a. Mediates all attempts to access security objects;
- b. Is protected from the tampering attempts of either the control software or the users;
- c. Is verifiably correct.

A security policy has been evolved that will permit as general a form of controlled sharing of machine resources and classified data as possible within the constraints of defining a kernel that will:

- a. Be verifiable with respect to enforcing that policy;
- b. Have an acceptable effect on overall VM/370 performance;
- c. Require minimal rewriting or replacement of existing code in the VM/370 Control Program (a retrofit);
- d. Preserve a maximum compatibility with VM/370 applications.

Although KVM/370 has been developed primarily for the defense and intelligence communities, its security policy can

be applied to other environments as well. For example, the system can operate in a private-sector environment where privacy² safeguards are necessary. At this time, several commercial organizations as well as other agencies in the Department of Defense that are outside of the intelligence community are considering the possibility that KVM/370 may satisfy their "secure" data processing requirements.

The KVM/370 effort has been inspired by the belief that encapsulation of multiple, individual copies of an operating system under a virtual machine monitor system can provide a practical, secure operating system. SDC's experience with IBM's VM/370 supports this belief. Even though the commercially available implementations of VM/370 continue to be insecure against planned intrusion,³ this system appears to have sufficient potential to warrant the present retrofit effort.

RETROFIT STRATEGY

A methodology was developed for partitioning the VM/370 control program (VM/370-CP) into security-relevant and nonsecurity-relevant modules. The decision process is based on the principles of least privilege and least common mechanism,² defining security-relevant code in CP as that code which executes privileged instructions or the code which accesses global system data (i.e., control blocks traversing security levels). In this way, security-relevant CP modules are directly identifiable.

It was found in the first year of the project that most system data need not be truly global, but global only over the Virtual Machines (VMs) at a given security level. The VMs at a given security level* could be supported by a combination of a formally verified kernel operating in real supervisor state⁸ and a Non-Kernel Control Program (NKCP) executing in real problem state and consisting of all non-security-relevant VM/370-CP code. The NKCP would execute as a virtual machine, having access only to global

* A security level {C, K} consists of a hierarchical classification C from the ordered set {unclassified, confidential, secret, top secret}, and a category K consisting of a subset (possibly empty) of the set of special access compartments (e.g. NATO, CRYPTO, etc.). The categories form a partial order under set inclusion.

system data for the virtual machines it is supporting at the given security level.

In principle, there are significant differences between a security retrofit to VM/370 and a new design of a secure VM/370. In both cases it is necessary to design and specify the security enforcement mechanisms for the security kernel, as well as to derive the set of formal security invariants the kernel must preserve over the system. It is found, however, that much of the code in an operating system that virtualizes a computer, such as that found in VM/370-CP, either has no security relevance or can be trivially modified so that it no longer has any security relevance. Hence, much of the existing code which provides functional capabilities to the virtual machines is essentially usable as it stands.

Secondly, it is observed that such code, in fact all of VM/370, could be virtualized. The strategy suggested by these observations involves designing and verifying a relatively small body of code which is just powerful enough to provide primitive virtualization and which controls all forms of I/O access with respect to the security policy. It is thus conceptually possible to run numerous copies of VM/370 atop this simple kernel, each running in virtual (as opposed to real) supervisor state. Since the kernel is the final arbiter and all access to real devices must eventually pass through it (i.e., these accesses all require invocation of privileged instructions in real supervisor state), no action of the virtualized VM/370-CP can compromise security. The users would run their programs atop the untrusted copies of virtualized VM/370. If a virtualized VM/370-CP were to attempt to perform actions contrary to the security policy, the kernel would prohibit such actions from taking place. These potential denials of service could be avoided by deleting the related code from the virtualized VM/370-CPs, but it is important to observe that these matters have no effect upon the enforcement of security since (1) the kernel was designed to enforce a specific security policy, (2) the kernel was formally verified to support the enforcement of that policy, and (3) the correctness proof of the kernel made no assumptions about any of the virtual machines running atop the kernel, particularly none with respect to an NKCP itself.

In the interest of enhancing the performance of such a kernelized system, it might be necessary to give certain system modules access to multilevel system data. These are the modules which control the sharing of real system resource among virtual machines at different security levels. In order to maintain system security, it is necessary to ascertain that such resource management modules properly utilize the privileges granted them by the added common mechanism. Such modules become trusted processes. Where possible and practical, the trusted processes are to be given the same formal verification the kernel processes receive.

Where this is not practical or possible,** the trusted pro-

cesses are subject to a thorough audit for the presence of errors or Trojan Horses² encapsulated into a limited address space with restricted reading and writing privileges, and restricted so that they operate in real problem state with virtual addresses. These latter processes are known as semi-trusted processes.

SECURITY POLICY

The KVM/370 kernel is designed to enforce a military security policy. This requires the preservation of two security properties, the "security condition," and the "confinement condition" (also known for historical reasons as the "*-property", pronounced "star property").⁴ These properties are described in terms of three types of entities: subjects, objects and security levels. Subjects are the active elements of the system for which data access must be controlled (e.g., users, processes). Objects are the data or data containers, access to which must be controlled by the kernel. There is a security level associated with each subject or object which describes the degree of clearance of the subject or sensitivity of the object. A partial order, called dominates, is defined on the security levels. Specific interpretations of these elements are to follow.

The security condition requires that no subject may access an object for the purpose of reading or updating unless the level of the subject dominates that of the object. The confinement condition demands that a subject may have write access to an object (permission to both read and write) only if the subject and object are associated with precisely the same security level.

In the main, subjects in KVM/370 are interpreted as the individual NKCPs. The kernel provides isolation among the NKCPs, but provides little or no additional isolation between VMs under the same NKCP beyond that already provided in VM/370. Since all VMs operating under the same NKCP act at the same security level, the kernel protects each VM from other VMs at different security levels, but not necessarily from VMs at the same level. Global processes, which must interface with several NKCPs at different levels, are also subjects.

The objects in KVM/370 are collections of data areas on direct access storage devices (DASD), or entire DASD volumes, tape volumes, unit record devices, real core pages and processes, and VM working environments (control blocks, scratch storage registers etc.).

During 1977, the evolution of United States National Security Policy was studied in an effort to make KVM/370 more responsive to the modifications that were being made to Executive Orders 11652 and 11905. The requirements of Executive Order 12065 clearly state that it is essential that computer systems not divulge information to unauthorized individuals on the one hand, while prohibiting the overclassification of data on the other hand. KVM/370 enforces the confinement condition to prevent unauthorized declassification of data, and produces detailed historical collateral classification information for every new volume created by the system in order to justify its classification as a function of the classifications of all data to which the virtual machine

** The semi-trusted processes serve as schedulers and allocators of global resources and have the potential to be used as an illicit signalling path in violation of the confinement condition by modulating the global state variable, TIME. There is no known method for formally demonstrating that an algorithmically correct, Trojan Horse free, scheduler cannot be manipulated by users in such a way that the users can cause clock time to become a signal to other users.

that created it had access. This historical classification information may then be reviewed by a security officer possessing original classification authority for the data.

OVERALL SYSTEM ARCHITECTURE

Figure 1 represents the architecture of Kernelized VM/370 (KVM/370), consisting of the following domains:

1. The kernel and verified trusted processes, executing in real supervisor state (about 6000 lines of JOVIAL);
2. The audited semi-trusted processes, having access to some global system data, executing in real problem state, but having access only to virtual addresses (about 10,000 lines of assembly language);
3. The NKCPs, one per security level, having access to system data for the supported security level only, executing in real problem state, having access only to virtual addresses (about 70,000 lines of assembly language);
4. The user VMs, each controlled by the appropriate NKCP for its security level, executing in real problem state.

It was intended that all kernel code and trusted process code would be written in a strongly-typed Pascal-based programming language such as the EUCLID⁵ language in order to facilitate formal verification.

However, as the time for system implementation drew near, it was found that there were no available production quality compilers for EUCLID or any other thoroughly-typed Pascal-based programming language that would permit efficient system programming on an IBM System/370 base machine. The requirement that the system programming language possess the capability of addressing and manipulating IBM System/370-specific data structures is essential since the kernel must analyze, prepare, and maintain numerous tables and control blocks whose structure is dictated by the hardware. In order to provide for the future verification and certification of KVM/370, it is desirable that a maximum of detail on the manipulation of these data structures be expressed in the higher-order language rather than in assembly language. Lastly, it was essential that the compiler reliably produce highly efficient executable code, lest the performance costs of the system be impractically high. It was also necessary that the compiled code not require a run-time support package for its execution, since the run-time package could be a possible source of security compromise (e.g., Trojan horses, trapdoors, etc.).

After serious consideration of numerous languages for which compilers either existed or were proposed, it was decided by ARPA that system efficiency was of sufficient importance to permit the use of a programming language that was not Pascal-based, providing it satisfied the other exigencies for the project.

The language selected for the implementation of KVM/370 was the J3 dialect of the JOVIAL Programming Language. JOVIAL is a system programming language providing direct description of machine-specific data structures, as

well as access to the full instruction set of the machine. JOVIAL is not a verification-oriented programming language. Consequently, while the formal specifications of KVM/370 will be formally verified against the security policy enforcement criteria, the first implementation will not be formally verified. Subsequently, when a production quality compiler exists for a verification-oriented systems programming language, KVM/370 can be recoded in that language and then verified.

DESIGN TRADEOFFS

The user's system-use expectations will have an impact on the system architecture, size of the kernel and trusted processes, overall system performance, and level of effort required for implementation and formal verification. Resource scheduling and management can be performed on either a system-global or an NKCP-local basis. If done on a system-global basis, the size of the kernel and trusted processes is increased, the interface between the NKCPs and the global processes becomes more intricate, verification becomes more difficult and costly, system modification becomes less facile, but system performance improves. If done on a local basis with most resource management decisions performed by the NKCPs and perfunctory reconciliations performed by the kernel, the opposite results hold; system design, implementation, verification and interfaces are simplified, while system performance may be adversely affected.

In terms of greatest all-around adaptability to applications, ease of implementation and verification, and best multilevel security, we concluded that:

- DASD page areas will be global
- Main page frame management will be based on global allocation with global page replacement
- Multilevel shared reentrant systems will be provided with all shared pages locked into core
- The CPU will be scheduled by the NKCPs.

KERNEL DESIGN

The kernel and trusted processes are the only portions of KVM/370 whose formal specifications will be formally verified. The system is being designed such that there are no "upward" functional dependencies (i.e., at level of abstraction i , no function depends for its correct operation on any function from level of abstraction j , if $i < j$).^{6,7} In this way, it can be demonstrated that no trusted code depends on the correctness and non-maliciousness of any untrusted, unverified code. Further, the formal proof of correctness of KVM/370 will require only (1) the kernel and trusted processes be shown to satisfy the requirements of enforcing the security policy, and (2) a demonstration of the absence of unauthorized signalling capabilities within the semi-trusted processes.

In the case of the Start I/O request to the kernel, the entire channel control program⁸ is copied into a portion of the kernel's domain where it is protected from modification

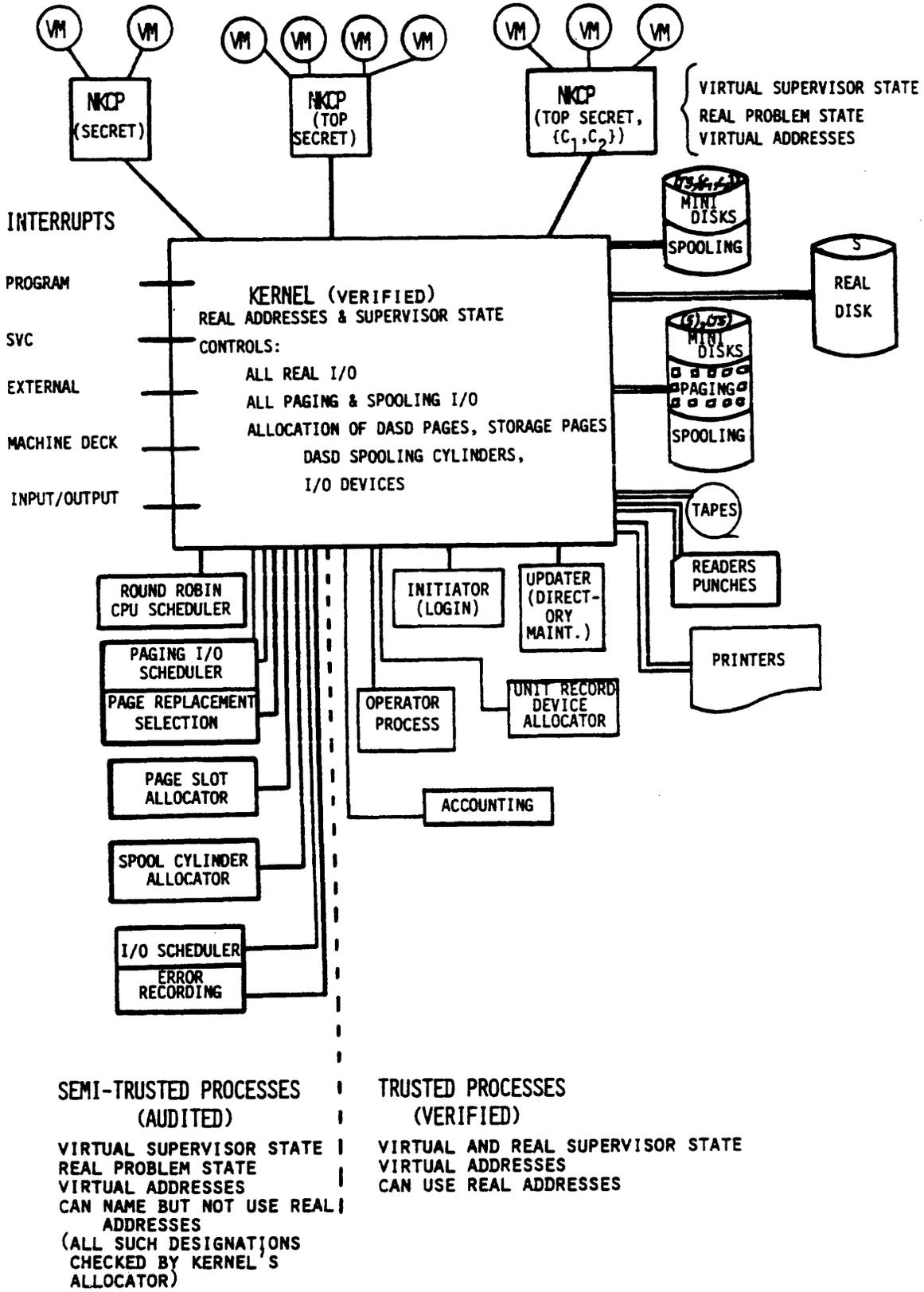


Figure 1—KVM.370 system architecture

by asynchronous attack.² Address translation is performed on the channel program. Then it is legality-checked by the kernel to guarantee that the program performs only valid accesses, that all referenced pages are locked into core, and that the channel program is not self-modifying and contains no puns or other security threats.⁶ The I/O scheduler is then invoked and control is eventually passed to the dispatcher module, simulating a Start I/O Fast Release.⁸ When the I/O interrupt finally takes place, the relevant pages are unlocked, and the condition code is passed as an interrupt to the appropriate NKCP.

Each security level has a unique address space for use by its NKCP. With the exception of the functions enumerated below, no NKCP can communicate outside its and its VMs' address spaces. Spool files, virtual channel-to-channel adapters (CTCA),⁸ and inter-VM messages are handled by the NKCP and consequently cannot violate the kernel's enforcement of the security policy.

The notable exceptions are:

- Append-up for writing machine error records and accounting data which are processed at the highest security level in the system
- Read-down to obtain access to a DASD whose classification is dominated by the clearance of the user's VM.

For purposes of design simplicity, each NKCP will appear to be uninterruptible just as VM/370-CP currently is, i.e., the NKCP's critical regions will be preserved. The NKCP may, in practice, be interrupted by the kernel, but only if no NKCP shared variable (e.g. its set of active page frames or real addresses) is modified while it is servicing a user request. This constraint on NKCP-shared variables is the result of considerable effort in the area of kernel-NKCP interaction. The consequences of this design decision, as well as the considerations that led to it, are detailed in the Appendix. An NKCP terminates a critical region (a locally uninterruptible code segment) when it either schedules a VM or when it issues an I/O request.

NKCP DESIGN

Code is security-relevant if it can influence unmediated I/O directly through the use of privileged instructions that manipulate devices or user domains, or indirectly through the use of data structures that either contain security enforcement data or can be viewed and modified by processes operating at different security levels. Data is security-relevant if it contains global information which traverses several security levels.

It appears that a considerable amount of CP code is security-relevant because it makes wide use of global system tables. Many of these tables could be distributed so that each copy contains only data relevant to a unique security level. The code manipulating these distributed tables could easily be made reentrant so that most of the code could lose

its security relevance. (Privileged instructions would still be security-relevant, however.)

An alternate approach is driven by identifying non-security-relevant code in CP and virtualizing it out of the privileged execution domain. The remainder, plus additional security enforcement code, becomes the kernel. The more code that is virtualized, the less there is to verify. Apparently, the efficiency of this system degrades as code is virtualized out of the kernel. This is because of the increased context switching between the NKCP and Kernel and the unavailability of global data required for optimal resource scheduling.

SYSTEM VERIFICATION

Formal verification of kernel and trusted processes at the specification level will be of two kinds. These functions will be shown to correctly implement the sharing policy between subjects and objects in terms of the basic security principle and the confinement condition. This form of proof involves demonstrating that all system state transitions preserve a set of security invariants. The proof of correctness will be achieved with the assistance of an automated verification tool which enhances the credibility of the formal logical demonstration. The second phase of the formal verification process is formal proof that the trusted state transition functions themselves obey the confinement condition with respect to the system objects they read and write. The analysis techniques required for this verification involve simulated symbolic execution of source code.^{9,12} The formal verification of KVM/370's specification has been postponed until after installation of the prototype.

POSSIBILITY OF HARDWARE ERRORS

One of the problems considered by the project was the possibility of violations of the security policy occurring because of failure of the hardware security controls. Some possibilities considered were:

- Failure of the privileged operation protection mechanism;
- An error in address translation⁸ or the Translation Lookaside Buffer (TLB);
- Failure of the storage protection mechanism;
- Misinterpretation of a Channel Command Word (CCW) by a channel;
- An I/O device responding to the wrong device address;
- Mishandling of a command by an I/O device.

Errors in the operation of the Central Processing Unit CPU and inboard channels are considered unlikely if the system receives proper maintenance. In addition, it appears to be difficult to guard against this type of failure. For similar reasons, we decided to ignore the possibility of an I/O device responding to the wrong address because address recognition logic on the S/370 channel is fairly simple. This left us

with the possibility of an I/O device mishandling a channel command. The most probable case of this type seemed to involve seeks on moving head devices: the possibility of a mechanical error moving the access arm to the wrong cylinder.

Questions had arisen as to the possibility that certain Direct Access Storage Devices (DASDs) were liable to incorrect seeks with a frequency that increased with their age. SDC conducted an investigation to establish hard data on the reality of these reported threats. Investigation of the logic of 3330-type devices showed a vanishingly small probability of a missed seek, inasmuch as the device controller counts the tracks electronically as they pass under the head. We also forced over 20,000 seeks of 350 cylinders and tested for an incorrect home address after each one. We found no missed seeks, nor were any seek retries reported in the error log for that day. We now believe that the redundancy checking built into the 3330 provides sufficient reliability for multilevel security applications.

Obviously, if a DASD accesses the wrong cylinder on a seek command, it is possible for a malicious user to read whatever is on the cylinder accessed, or to write incorrect information into those records. If the probability of a missed seek going undetected by the controller is as high as 0.1 percent, a user who causes a large number of seeks can reasonably expect to gain unauthorized access to data belonging to other users several times a week.

The results of the study of the 3330 have obviated the necessity of having to limit each DASD volume to a single level of security, as had been contemplated prior to the experiment. However, some installations may use older or other direct access storage devices which may not be as reliable as the IBM drives that we tested. As a result, we decided to provide a mechanism for protecting against misseeks, albeit at some cost. I/O requests are separated into paging/spooling requests and all others.

We note that all modern DASD devices can have an eight-byte key added to each paging block without affecting the number of pages which will fit on a cylinder. It was decided to write in each key the real cylinder number, track and record number, VMid and virtual page number that it contains, encrypted by a key that is determined at system start-up and unique for each security level. This would make the task of the would-be penetrator hard enough to discourage any attempt to use this mechanism.

For general I/O, it is possible to include a read-home-address CCW after each seek, and to have the kernel validate the home address on completion of the channel program. This would increase the average time required to execute a channel program by one-half revolution (about eight milliseconds). Since this represents nearly a 100 percent overhead in I/O operations, it was decided to partition devices into two classes: trusted devices and untrusted ones. A device is considered trusted if (1) it has been designated by the installation's security officer as trusted, (2) no misseeks on that device have gone undetected by the hardware (these usually result in unit-check with no-record-found), and (3) less than some threshold number of hardware-detected mis-seeks have occurred. If any of these conditions

is not satisfied, the device is regarded as untrusted. Home address verification is applied only when (1) the device is regarded as untrusted, (2) the I/O is not for paging or spooling, and (3) the I/O has been requested by an untrusted process (NKCP or scheduler/allocator).

ELIMINATION OF KNOWN SECURITY FLAWS

There are about 30 security flaws known in VM/370. A hardened version could be produced by fixing these errors, but there might be other errors which had not been detected and the fixes might themselves introduce new security flaws. KVM/370 goes further and eliminates both known and unknown security flaws. It is instructive, however, to see how the presence of a security kernel and the constraints it places on the system design eliminate some of these errors.

Almost every known security flaw in the VM/370 system involves the input/output functions.³ This is because there is no address space validation of input/output by the hardware other than that performed by the storage protection keys. Therefore, VM/370 must check the validity of all channel programs and relocate all virtual addresses. This includes both main storage addresses and DASD cylinder addresses in seek arguments and home addresses. The same I/O logic is repeated for several different requirements: virtual spooling support, virtual console support, virtual channel-to-channel adapter support, and a special VM/370 I/O interface. Each variation of this support means that errors may be present.

These errors occur in the translation of channel programs as a result of the complexity of the channel command language. For example, the same word in a channel program might be used as a command or as an operand address depending upon the execution sequence of the program.³ Since the System/370 architecture allows puns in the channel program (a word's interpretation depends on whether it is received as the leading or trailing portion of a long command), it is possible to surreptitiously bypass checking by these modules, and access DASD records without authorization.³

Under KVM/370, channel command words are not permitted to take on different meanings depending on the sequence of execution. Primarily, this means that an NKCP is not permitted to submit certain channel commands with transfers or with certain modifier bits set. This does not preclude users (VMs) from constructing such channel programs, it merely requires the NKCP to put them into a standard form before submitting them to the kernel. Further, these commands will be copied into the kernel's data space and translated and modified there, preventing their modification by an NKCP between the time of translation and time of execution. Also, self-modifying channel programs will not be permitted, such as those used by the OS/360 Indexed Sequential Access Method (ISAM).¹

Certain VM/370 penetrations³ dependent upon simultaneous input/output and CPU execution are being countered by removing from the address space of the requestor all pages which are buffering input. This applies to both NKCP

and user VMs. In the event either an NKCP or a VM needs access to such a page, its execution will be delayed until the I/O completes and the page is made available. For example, under VM/370, careful timing of asynchronous execution could be used to exploit a bizarre oversight in condition-code checking to gain a total system penetration (real supervisor state).³

Although the treatment of storage and timing channels^{10,13} is beyond the scope of a system such as VM/370, they must be controlled in a military system such as KVM/370. In this respect, we will thoroughly audit all semi-trusted processes for Trojan Horses since they control resources shared between different security levels. Hence, it will not be possible to transmit information over a covert communication channel at a high enough bandwidth to make such attempts worthwhile. The above discussion deals only with the security flaws that are already known. KVM/370, however, is designed around a formally specified security kernel. Once the specification is verified, there is good reason to believe that no possibility of violations of the security condition or overt violations of the confinement condition exist in the design. Further, by recoding the security kernel and trusted processes in a programming language designed for verification, a near-certainty of the absence of security flaws could be obtained by verifying the code against the already verified specification.

CURRENT STATUS

The feasibility of performing a VM/370 security retrofit was demonstrated during the first year of project activity. An informal system design was produced, identifying the major security kernel functions. The input and output parameters were defined and their effects on KVM state variables were described.

From this, the security kernel and trusted processes were formally specified in the SDC specification language, INA JO, a strongly-typed dialect of the first-order predicate calculus. After the system data bases were defined, the coding of the NKCP and semi-trusted processes was started, and the implementation of the kernel and trusted processes was begun in the J3 dialect of JOVIAL.

KVM/370 security policy was re-examined in light of the modifications to the National Security Policy as defined in Executive Order 12065, 28 June 1978. KVM/370 security policy now takes account of both discretionary and non-discretionary aspects of security.

Integration testing of KVM/370 is in progress as of this writing.

PLANS

It is expected that system testing and integration will conclude by late summer of 1979. At that time, KVM/370 will be installed in a testing environment within the Defense Communications Agency Engineering Center. Here, the prototype system will be evaluated on a set of selected

benchmark workloads and its performance will be tuned to the extent possible within the constraints of the security policy. In this way, the first steps can be undertaken toward determining the costs of multilevel security on an IBM System/370 mainframe. Initially, test cases will be run under varying conditions in a periods processing environment.² This will establish a basic scale against which the operation of KVM/370 can be judged. These will be followed by selected KVM/370 runs that approximate the periods processing approach: one NKCP (one color); two NKCPs (two colors), etc. Various test case workloads will be defined and specific measurements will be performed. Acceptance of KVM/370 will be made by relating dollar costs to run time, and by evaluation of the periods processing approach as it impacts user requirements.

CONCLUSION

In this paper, we have presented a design strategy for performing a retrofit to VM/370 which will provide a multilevel secure operating environment. The strategy is heavily based on the principles of least privilege and least common mechanism. The research and development activities described in this paper transpired in the period March 1976 through January 1979. The implementation of KVM/370 is currently in progress and it is anticipated that a prototype version of the system will be installed within the Defense Communications Agency in the late summer of 1979.

ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions and suggestions made by the following individuals: E. Book, M. Branstadt, E. Burke, W. Carlson, J. Fraley, T. H. Hinke, D. Hollingworth, A. K. Jones, H. C. Lauer, R. Lyons, C. A. Melkerson, M. Moriconi, M. Orceyre, G. J. Popek, G. Schroeder, W. M. Shasberger, D. H. Thompson, S. T. Walker, C. Weissman, J. H. Yott, and D. Zucker. We would also like to extend our sincere appreciation to our technical editor, B. L. Sargeant, and L. L. Parris, who provided typing support.

REFERENCES

1. *IBM Virtual Machine Facility/370: Introduction*, IBM Publication GC20-2800. May be obtained from IBM Corporation, Data Processing Division, 1133 Westchester Avenue, White Plains, New York, 10604.
2. Saltzer, J. H. and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, Vol. 63, Number 9, September 1975, pp. 1278-1308.
3. Attanasio, C. R., P. W. Markstein and R. J. Phillips, "Penetrating an Operating System: a Study of VM/370 Integrity," *IBM Systems Journal*, Vol. 15, No. 1, International Business Machines Corp., 1976, pp. 102-116.
4. Bell, D. E. and L. J. LaPadula, "Secure Computer Systems: A Refinement of the Mathematical Model," MTR-2547, Vol. III, MITRE Corp., Bedford, Massachusetts, 28 December 1973.
5. Lampson, B. W., J. J. Horning, R. L. London, J. G. Mitchell and G. J. Popek, "Report On The Programming Language Euclid," Xerox Re-

- search Center, University of Toronto, USC-ISI and UCLA respectively, December 1976.
6. Janson, P., "Using Type Extension to Organize Virtual Memory Mechanisms," MIT/LCS/TR-167, Massachusetts Institute of Technology, September, 1976.
 7. Reed, D. P., "Processor Multiplexing in a Layered Operating System," MIT/LCS/TR-164, Massachusetts Institute of Technology, June, 1976.
 8. *IBM System/370 Principles of Operation*, IBM Publication GA22-7000. May be obtained from IBM Corporation, Data Processing Division, 1133 Westchester Avenue, White Plains, New York, 10604.
 9. Denning, D. E., "Secure Information Flow in Computer Systems," PhD. Thesis, Purdue Univ., Computer Science Dept., West Lafayette, Indiana, May 1975.
 10. Lamson, B. W., "A Note on the Confinement Problem," *Communications of the ACM*, October, 1973, pp. 613-615.
 11. Bell, D. E. and L. J. LaPadula, "Computer Security Model: Unified Exposition and Multics Interpretation," ESD-TR-75-306, MITRE Corp., Bedford, Massachusetts, June 1975.
 12. Millen, J., "Security Verification in Practice," *Communications of the ACM*, Vol. 19, No. 5, May 1976, pp. 243-250.
 13. Schaefer, M., B. Gold, R. Linde and J. Scheid, "Program Confinement in KVM/370," *Proceedings 1977 Association for Computing Machinery Conference*, October 1977, pp. 404-410.

APPENDIX A

A solution to the "load real address" problem

Background—During the first year of the KVM/370 project, attention was paid to what is known as the "Load Real Address" problem. This problem is concerned with the fact that an NKCP needs to be able to "locate" certain pages of the VMs under its control. This is handled in VM/370-CP by the Load Real Address (LRA) instruction.⁸ The LRA may be used for channel program translation, or to locate the operand(s) of a privileged operation that CP is simulating.

The problem which arises in KVM/370 is that the decision to "steal" a page is made by a global process, not under control of the NKCP. Consequently, there is no guarantee that the page, once "located" by the NKCP, will stay at the same real address, or even remain in main storage, long enough to be used. In order to avoid frequent and embarrassing denials of service, it is necessary to guarantee that a virtual page stays in the same place from the time the NKCP has been given its real (or "real") address, until the NKCP is no longer relying on the address given for that page. The following discussion expands on the complexities of the problem and presents what is believed to be a solution to it.

Related Considerations—When the problem arose, several points of view were held concerning real addresses. It would probably simplify the kernel-NKCP interface if the NKCP were allowed to access the real addresses of the pages under its control. On the other hand, there are several high bandwidth data channels involving real page addresses.¹³ The current design calls for the NKCP to gain access to pages containing operands by having them placed in its own address space. (The kernel inserts their real addresses in the NKCP's page table.) This allows the NKCP to read and/or modify data for instructions that it simulates for its VMs,

without knowing the real addresses of the pages. For channel program translation, the NKCP will leave virtual addresses in the Indirect Address Word [IDAW] lists, which the Request-I/O handler will translate to real addresses.⁸

In order to simplify the kernel's handling of process scheduling, it was decided to treat NKCPs as logically non-interruptible. The kernel will refrain from presenting the NKCP with interrupts during its operation (just as VM/370-CP is designed to run with interrupts disabled). The kernel will also refrain from running any other NKCPs until the current one signals the end of its critical region.

The Solution—The operation of an NKCP is considered as a critical region from the time it is entered until it dispatches a VM or relinquishes the CPU. Any interrupts taken by the kernel during this period will be handled to the extent possible by the kernel and schedulers, but no other NKCPs will be dispatched, even if the current NKCP's time-slice ends. Interrupts requiring action by any NKCP, including the current one, will be stacked until the end of the critical region. Any pages swapped in at the NKCP's request or to which the NKCP gains access (by "attach page") will be placed under a temporary lock which prevents their page frames from being stolen during the critical region. The critical region (and temporary lock) will end when the NKCP makes either a Dispatch-VM or a Release-CPU kernel call. If an NKCP requires that a page be at a fixed address for a longer period of time, it must make an explicit Lock-Page call.

Note that such locks (either temporary or long-term) cannot be attached to a page which is not present or which is being used for a conflicting purpose. For example, a page that has been stolen and is being swapped out cannot be locked unless the requester can reclaim the page (this ability is not supported by the current KVM/370 design). A page which is being used for the buffering of input cannot be locked for CPU usage or output, nor can a page being used for the buffering of output be locked as an input buffer.

This approach has the following consequences:

1. No page will be stolen from an NKCP or its VMs except when the NKCP is running a VM or waiting for an interrupt. (The latter case is equivalent to the dispatcher's loading a wait-state PSW because there is no work to do). VM/370-CP always checks page status when a new request arrives from a VM by doing a LRA and/or calling the real page manager (DMKPTR). Similarly, NKCP does not rely on page locations remaining constant across such operations, so stealing a page cannot cause the NKCP to make an erroneous assumption.
2. If VM/370-CP requires that a page keep the same real address after a call to the dispatcher, it explicitly requests DMKPTR to lock the page. Thus, if the NKCP requires a page to stay in main storage across Dispatch-VM or Release-CPU calls, it must explicitly request a lock on that page. Otherwise the kernel or Select routine will be permitted to steal the page if it is the "best" page to steal.
3. Since the kernel will call other NKCPs (that is, by

invoking the CPU scheduler) only when the current process makes a Dispatch-VM or Release-CPU call, there can be no "surprise" loss of the CPU to another NKCP. This means that the kernel need not save and restore the registers in the KPROCBLOK for each process. Instead, a single level of register storage will suffice (for General registers and timers only, since neither the kernel nor the trusted/semi-trusted process will use the Floating registers). Each NKCP must be written so that it saves its own registers whenever it makes a call that invokes another process or ends its critical region. (This is not directly related to the LRA Problem, but simplifies table design.)

Locks—The kernel provides three types of locks to NKCPs.

1. A temporary lock is attached to each page which is swapped in at the request of an NKCP or to which the NKCP is given access as a result of an attach-page request. The lifetime of the temporary lock is the NKCP's critical region (i.e., until the NKCP releases the CPU or dispatches a VM). The temporary lock prevents the page from having its frame stolen; the NKCP may release the page, however, which cancels the temporary lock.
2. An I/O lock is attached to each page used in an I/O request. The page must be in main storage when the request-I/O call is made. The lifetime of the lock is concurrent with the I/O request (the lock is released when the requested I/O operation completes). The I/O lock can be cancelled only by cancelling the I/O operation.
3. A long-term lock is attached to a page at the request of any NKCP with access to that page. The long-term lock is permanent until cancelled by a specific request. Only the NKCP which requested the lock can cancel it. The kernel calls Lock-Page and Unlock-Page will be provided for this purpose. This type of lock can be used by an NKCP in response to an operator "LOCK" command or while gathering multiple pages (e.g. for an I/O operation) to insure that pages obtained earlier do not get swapped out while obtaining other pages.

All three types of locks protect the page from being stolen until the NKCP is finished with them. The second and third types of locks also prevent the NKCP from releasing the page until the lock has been cancelled.

If the SELECT routine (a semi-trusted process) attempts to select a page for which a lock exists, it will be re-entered to select another page. The kernel will refuse to steal a frame from a locked page. If the NKCP attempts to release a locked page (via release-page) or to swap out such a page, the result depends on the type of lock(s) attached to the page. If only a temporary lock is attached to the page, the temporary lock will be released and the request honored. If an I/O lock or a long-term lock is attached to the page, the request will be denied.

APPENDIX B

A general countermeasure for quota-type leakage paths

The allocation of objects from a global pool of finite size allows use of that limited size as a communication path for covert transmission of data. The sending process repeatedly requests resources from the pool until a request is denied. At that point the sender knows the pool is exhausted and can release the CPU and allow other processes to run. The receiver requests a few objects from the pool, then releases them. The sender releases a large number of objects into the pool to send a one, or exhausts the pool to send a zero. The receiver receives a one or zero depending on whether its requests are satisfied. Other processes may introduce noise by exhausting the pool with legitimate requests or releasing objects they no longer need. Such noise can be filtered out via normal redundancy techniques. The state of the pool is a variable shared between sender and receiver, creating a storage channel whose bandwidth is dependent on the frequency with which such requests can be made.

Analysis of the preliminary design of KVM/370 reveals a number of such resource pools:

- Disk Pages (Page Slots)
- Main Storage Pages (Page Frames)
- Spool Cylinders
- Temporary Disk Cylinders
- Kernel Table Entries
- Kernel Storage for Dynamic Creation of Tables

A number of countermeasures have been adopted to control the use of these pools as communication channels.[1] Prediction is used on page slots and entries in the KVMTABLE and PROCESSLIST. Whenever a user attempts to Log In (a relatively infrequent event), a check is made to determine whether the necessary page slots and table entries are available. The user is denied access if they are not. [2] Temporary disk cylinders are subpooled; each security level has a private pool from which it makes allocations. No global pooling of TDisk is provided. [3] Requests for main storage pages and spool cylinders are never refused. The satisfaction of a request for a page frame or spool cylinder is reported by an interrupt which may occur immediately or after an arbitrary period of time. This converts a potential storage channel into a timing channel and lowers the bandwidth. (A process which exhausts the pool is unable to free the entries it has requested until the necessary I/O has been performed).

However, some types of requests for kernel tables cannot be predicted, and their satisfaction is not dependent on I/O.

Further, subpooling kernel storage by security level would be extremely wasteful of main storage which is a precious resource. The communication channels involving these quotas are being tolerated but restricted in bandwidth. Whenever a process request is refused because of exhaustion of a kernel table or storage pool, a return code is provided

indicating to the process that some quota has been exhausted (without specifying which one). After that the process will not be permitted to make another such request for a period of time. Until that time period is over, any request by that process depending on such a quota will be denied without checking the resource pool and the return code will indicate "too soon." In this way, the communication channel is

limited to one bit per time period. By setting the time period to .1 second, these communication channels are restricted to ten bits per second.

This technique can be used on any system that has a real-time clock and can be used on any resource pool. It can be applied instead of or in addition to other countermeasures for control of quota-type data channels.

KSOS—The design of a secure operating system*

by E. J. McCAULEY and P. J. DRONGOWSKI

*Ford Aerospace and Communications Corporation
Palo Alto, California*

INTRODUCTION

This paper discusses the design of the Department of Defense (DoD) Kernelized Secure Operating System (KSOS, formerly called Secure UNIX).** KSOS is intended to provide a provably secure operating system for larger minicomputers. KSOS will provide a system call interface closely compatible with the UNIX operating system. The initial implementation of KSOS will be on a Digital Equipment Corporation PDP-11/70 computer system. A group from Honeywell is also proceeding with an implementation for a modified version of the Honeywell Level 6 computer system.

KSOS will be capable of handling information at various security levels (a security level is a combination of a hierarchically-ordered classification category, like SECRET or TOP SECRET, and a possibly null set of compartments, like "No Foreign Dissemination" or specialized need-to-know compartments). The goal of the system is to provide strong assurances that it is impossible for an unprivileged user to cause an information compromise.

At its outer interface, KSOS will appear to be closely similar to the UNIX operating system.¹³ The only changes are to tighten the security checking on some of the operating system calls, and to add several new calls which individual UNIX sites had previously added to their systems. Existing applications programs written for UNIX will run without modification or recompilation on KSOS, providing that they do not violate the security rules of the system. At last count there were several hundred application programs for UNIX, ranging from simple utilities through sophisticated compilers, data management systems, text processing systems, and powerful editors. (This paper was completely prepared on a UNIX system, as is all documentation for the KSOS project.) All of these programs should run on KSOS without modification.

This UNIX-like interface is provided by a software component called the UNIX Emulator. The UNIX Emulator

transforms the user's UNIX operating system calls into (sequences of) calls to the Security Kernel. The Security Kernel is the heart of the system. The Kernel implements the reference monitor concept.¹ Briefly, through a combination of hardware and software checking, the Kernel monitors every access attempt by each user process. The Kernel will be shown to make the correct decision on whether to permit or deny the access attempt.

One important distinguishing characteristic of KSOS over the prototypes which have preceded it^{5,8} is that it contains a full range of support software. Included in this "Non-Kernel System Software" (also called Non-Kernel Security-Related Software) are components which support the day-to-day operational functions of the system: secure spooling of line printer output, portions of the interface to a packet-switched computer network, etc. Also included are components for the continuing maintenance of the system such as consistency checks of the file system, and system generation support. Finally, there are components to support the administration of the system, such as adding and deleting users, changing the security levels that a given user may access, and other functions.

The schedule for KSOS calls for its delivery in the fall of 1979 after the conclusion of a full series of testing. The KSOS development contract specifies that the system shall have a full MIL SPEC documentation package. The primary documents defining KSOS are detailed "design to" specifications which are called "B5 Specifications."^{3,6,9} The Kernel B5 Specifications⁶ include formal, mathematical descriptions of the Kernel written in a language developed by SRI International called SPECIAL.¹⁵ SPECIAL is a formal, non-procedural language for describing the behavior of systems in the manner suggested by Parnas.¹⁰ In addition, technical reports have been delivered detailing our plans for verification of the system's security properties,¹⁶ for the tools and techniques to be used in implementation,⁴ and for the long term maintenance and support of the system.⁷

The remainder of this paper begins with a discussion of the influences on the design. As with any design project, it is impossible to identify all of the factors which cause a given course to be taken, so only the strongest influences are discussed. Next the design itself is presented. Here the emphasis is on the more novel aspects of the design. In addition to the usual things expected from an operating

* The work described in this paper was performed under ARPA Order 3319, Contract MDA903-77-C-0333 administered by the Defense Supply Service Washington. Various DoD Agencies are funding the work. The conclusions presented are those of the author and are not necessarily those of the Government or Ford Aerospace.

** UNIX and PWB/UNIX are trademarks of the Bell System.

system. KSOS provides a number of features that aid in the creation of encapsulated secure environments. The paper concludes with a few remarks on how KSOS may be used effectively.

INFLUENCES ON THE DESIGN

External design goals

The overall design goals for KSOS are:

1. The system must provide provable security, i.e. its design and mechanization must be oriented towards the proof of its security properties.
2. The emulation of the UNIX system call interface must be as faithful as possible given the constraints of the security model.
3. The performance of the system should be "good," specifically, the performance should be comparable to that of a UNIX system.
4. The Kernel should be usable by itself as a simple, secure operating system.
5. The design should be amenable to implementation on other hardware bases.

The need for provable security had the most profound impact on the design. First, it dictated the basic structure of the system. A Security Kernel would function as a reference monitor.¹ The Kernel would mediate all access attempts in the system. Because the Kernel would potentially be proven to operate correctly, its behavior would have to be formally specified. Further, the size of the Kernel would have to be kept to a minimum to make formal specification and eventual verification tractable. Although only representative code proofs were planned, the Kernel would have to be implemented in a language suitable for code proofs.

Because the UNIX call interface had to be emulated faithfully and efficiently, the Kernel interface became "UNIX-flavored." However, because non-UNIX applications of the Kernel were planned, there was strong pressure to keep UNIX-specific structures out of the Kernel. As will be seen below, the Kernel has no knowledge of the format, or semantics of UNIX-specific constructs such as directories or load modules (UNIX a.out files). This knowledge is encapsulated outside the Kernel.

It was recognized that a large class of KSOS applications would not require the flexibility and added power of the UNIX interface. Rather, many of them would be built directly on the Kernel. Thus, the Kernel had to provide all of the features commonly found in an operating system. This meant that the Kernel would include somewhat more functionality than the absolute minimum.

Hardware limitations

Although KSOS was intended to be a machine-independent design, it will be implemented on real machines with

various hardware limitations. The PDP-11/70 has two significant limitations. First, process switching is expensive because a large number of processor and memory management registers must be individually saved and restored. Thus, architectures which require extensive process switching are to be avoided.

The PDP-11/70 does not lend itself to the creation of virtual machine environments that include direct control of single user i/o devices. The problem stems from the granularity of the virtual address to real address mapping, and from the logical addressing of i/o registers. In KSOS on the PDP-11/70, all devices are managed by the Kernel; no attempt is made to provide devices in the user's "virtual machine."

In fairness to the PDP-11 design it should be remarked that none of these hardware limitations are especially burdensome; they merely influence the design to take advantage of the strengths, and to avoid the weaknesses of the hardware base.

The design methodology

The design of KSOS is strongly influenced by the design methodology used on the project. KSOS is being designed and implemented using a blend of the "classical" methods with the formalism of the Hierarchical Development Methodology (HDM)¹⁴ developed by SRI International. HDM emphasizes formalism throughout the project. The system's security requirements are formally stated as properties to be satisfied by an abstract description of the design. This design is described in a mathematical, non-procedural language, SPECIAL.¹⁵ The security properties of the design are established by proving theorems that are derived from the design and the mathematical model of the security requirements. The implementation language is selected to allow its correspondence with the specifications to be proven. All of these steps force the designer to be precise and exacting in the statement of the system design. They make "kludges" very obvious at an early date. The design methodology strongly encourages a hierarchical decomposition of the design.

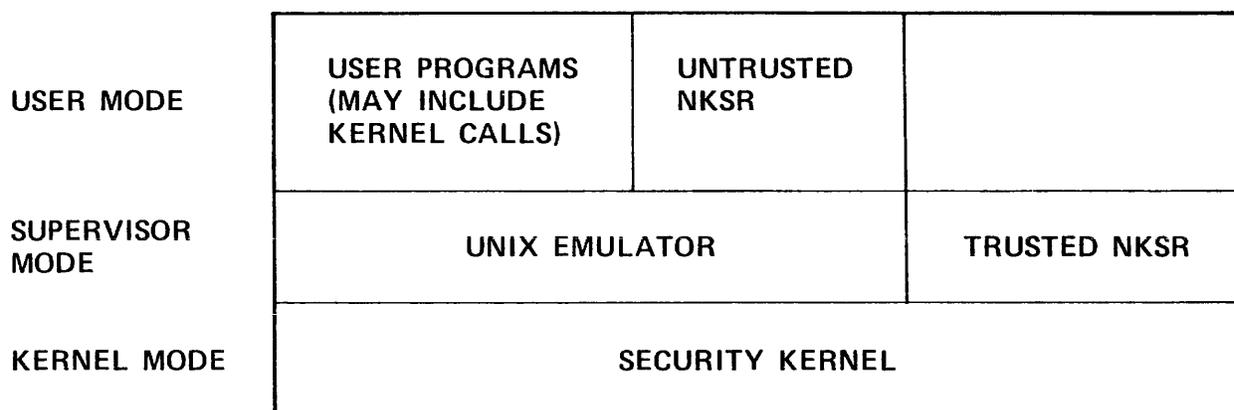
KSOS DESIGN

KSOS is composed of three components:

1. The Security Kernel
2. The UNIX Emulator
3. The Non-Kernel System Software

The relationship of these components is shown in Figure 1.

The Security Kernel's function is to provide a simple operating system which can be shown to be secure. The Kernel centralizes the control of all the resources in the system. It mediates each access attempt by a user process and only permits those accesses which comply with the access control policy. The Kernel resides in the most privileged address space of the machine (called "kernel mode"



(NKSR: NON-KERNEL SECURITY RELATED SOFTWARE)

Figure 1—KSOS system structure.

on the PDP-11/70) where it has access to all of the raw hardware and memory management facilities.

Logically, the UNIX Emulator is a part of each UNIX process which on the PDP-11/70 resides in the "supervisor mode" address space of the process. Its function is to map the user's UNIX system calls into the corresponding Kernel call(s).

The Non-Kernel System Software is a collection of autonomous processes performing support services for the system. Like UNIX, KSOS does not have services like login embedded in the operating system. Rather, these services are performed by "trusted processes" which reside outside of the Kernel. Except for the fact that these processes have the privilege to selectively violate the rules of the Kernel, they are just like any other process. Because the Emulator is "untrusted" and is not intended to be verified, it cannot be used by trusted software; rather, such software must use the Kernel directly.

The KSOS Security Kernel

Viewed as an abstract machine, the Kernel's function is to create the objects of its interface (processes, process segments, files, devices, and subtypes) from the basic hardware resources of the system, and to mediate all access attempts to these objects.

The Kernel enforces three distinct types of access checking. The first is the enforcement of DoD security policy. This checking is the verification of that fact that the user has the proper clearance and need-to-know for reading the information (the "simple security property"), and that information cannot be downgraded by writing it to a file at a lower security level (the "security *-property").

The second type is the enforcement of an integrity policy described in Reference 2. Integrity is a mechanism for protecting system data bases, programs, etc. against modification while allowing them to be read by any process. It is formally defined to be the mathematical dual of the security

model. We have found this integrity model to be overly restrictive, as its originator suspected. However, it does provide an additional, essential dimension of protection. Development of a more effective integrity model would seem to be a meaningful research topic.

The third type of access checking performed by the Kernel is discretionary access checking. Unlike the first two types of checking, the discretionary access checking is completely under the control of the user. The user may, at his discretion, permit or deny access by other users to the objects he owns. KSOS enforces a discretionary access policy similar to that of UNIX. For each object there are (logically) nine bits that specify read, write, and execute/search access by the owner, others in the same group as the object, and all others. We recognize that this discretionary access policy has limitations when compared to more sophisticated schemes, such as the access control lists used in Multics. However, it is simple, and requires a small fraction of the support mechanisms needed for access control lists.

The Kernel supports five different types of objects:

1. Processes
2. Process segments
3. Files
4. Devices
5. File subtypes

All Kernel objects have the same type of name called a SEID (Secure Entity Identifier). Further, every object, regardless of its type, has a block of information associated with it that includes all the information needed by the Kernel to mediate access attempts to the object. This block is called the "type independent" information. Because objects, regardless of the object type, have homogeneous type independent information, access checking by the Kernel is greatly simplified. All that must be checked is that information may flow from the source to the destination. For example, if a process wishes to read a file, the source is the file and the destination is the process. In the KSOS Kernel,

two functions perform all the access checking (one for security and integrity checking and one for discretionary access checking).

Processes

Processes are the only active agents in the KSOS design. To adequately emulate UNIX, KSOS processes must be cheap and plentiful. For example, each UNIX command is run as a separate process. Processes in KSOS will require only modest amounts of Kernel resources. Most of the Kernel data for a process will be swapped in and out with the process, reducing the amount of locked down Kernel memory space for the process tables.

Processes may possess privileges ("trusted processes") that enable them to perform functions that require reduced checking by the Kernel (e.g. changing the classification of a file) or which may require that additional checking be performed in the process (e.g. logically mounting part of the file system). The privileges that may be given to a process have been designed following the concept of "least privilege." That is, the granularity of the privileges is quite fine, and quite specific. Many service processes possess only a single privilege, and many privileges are possessed by only one process. Thus, the KSOS Kernel is designed to create encapsulated environments for critical functions. Privileges are obtained from the process image file (load module) from which the process was initialized. Two Kernel calls, `K_invoke` and `K_spawn`, are used for the controlled invocation of privileged software. `K_invoke` functions by replacing the entire process with a user-specified intermediary process. For the invocation of trusted software, this intermediary is a trusted "bootstrap" that, in turn, replaces itself with the requested process image file, and sets the privileges of the process from the values in the image file. `K_spawn` performs the same function in a new process created as part of the `K_spawn` function. This mechanism allows knowledge of the format and semantics of process image files to be kept out of the Kernel. Thus, the bootstrap encapsulates the function of initiating trusted software with minimal Kernel support.

In addition to the `K_spawn` mechanism, new processes may be created by the `K_fork` call, which is similar to the UNIX fork call. `K_fork` creates a "clone" of the caller, a new process that is an exact copy of the caller. The only difference between the two processes (parent and child) is the return value from the `K_fork` call. Such a mechanism is required for the accurate emulation of the UNIX fork call.

Processes normally run at a single security level. The only exception to this is the part of the Non-Kernel System Software that changes the user's working security level. For inherently multi-level applications, the preferred design would be to create a trusted multiplex/demultiplex ("mux/demux") process which directs commands and i/o to processes running at each level needed. This would be preferable to having these per-level functions performed within one process which changes its level because such a process

would be larger and more complicated than the mux/demux process. Verification of the correctness of a process becomes significantly more difficult as the process size and complexity increase. One example of this preferred architecture is the KSOS network interface. A small trusted process separates the multi-level data stream from the network into several streams. Each stream has data of only one security level in it. The mono-level streams from the processes are similarly combined by the trusted process into a single, multi-level stream.

Standard UNIX is acknowledged to be deficient in the area of Inter-Process Communication (IPC). KSOS provides significant improvements in this area. The Kernel supports both an event IPC mechanism and shared segments. The event mechanism allows one process to send a message to another process, and (optionally) to cause the receiving process to be interrupted analogously to receiving a hardware interrupt. The full set to security checks is performed for each IPC attempt. That is, information must be able to flow from the sender to the recipient, and the recipient must have permitted such information flow. Finally, a process may enable and disable the pseudo interrupt mechanism, so that it will not be interrupted during some critical operation. (Shared segment IPC is discussed below.)

Process segments

A process segment is a portion of the virtual address space of a process. The process segment is not tied to the native memory management hardware of a particular machine. The KSOS process segment may be of any size from a hardware-limited lower bound up to the entire virtual address space of a process. A process may have only some of its segments actually mapped into its address space. At its creation the segment may be declared to be sharable, in which case other processes can "rendezvous" with it and map it into their address spaces. This allows for very high bandwidth communication between the processes. Naturally, they must establish a protocol that guarantees that the segment will not be corrupted through unsequenced use. The process may elect to have only some of its segments actually mapped into its address space. In particular, several segments for the same part of the address space could exist. This mechanism is used by the trusted mux/demux processes discussed above. The data segments are shared between the trusted mux/demux and the processes servicing each logical stream. The mux/demux maps in a particular segment to a well known location and puts/extracts the data for that stream into/out of the segment.

One other use for shared segments is shared text (program) segments. It is possible to have a pure text segment shared between multiple processes, thus reducing the overall memory requirements for the system. KSOS allows a segment to be locked in memory, or to be retained in the swap area for faster accessing. The designer of the KSOS-based system is offered considerable latitude in trading space for time.

Files and devices

The Kernel file structure is flat and uniform. That is, there are no Kernel assumptions about the internal structure or contents of files. Directories and other higher-level constructs are mechanized outside the Kernel. The UNIX Emulator creates UNIX-like directories by interpreting the contents of Kernel files. This allows a designer working directly with the Kernel to create a different type of directory structure if desired. Kernel files are accessed by blocks. There is no Kernel buffering of file *i/o*. Rather, the *i/o* is done directly into the requesting user's address space. Kernel *i/o* is asynchronous, that is, the call returns to the user as soon as the *i/o* has been internally queued. An IPC message is sent to the user upon *i/o* completion. (The inclusion of asynchronous *i/o* is a relatively late addition to the KSOS design.)

Kernel devices are like a special type of file, as in UNIX. Terminals have only the lowest level echoing support in the Kernel. Higher level functions like erase/kill processing are done outside the Kernel.

KSOS supports removable file volumes. The mechanism is similar to the UNIX mount mechanism with some significant additions for protection. Because of the possibility for removing a volume, files are limited in size to one volume. Presently the design allows for support of at least 300 Mbyte disks, with extensibility to 600 and 1200 Mbyte disks possible. These large disks may be partitioned into one or more extents, referred to as "mini-disks" which may be independently utilized as virtual disks.

Subtypes

The KSOS subtype mechanism is one of its more novel features. The subtype mechanism is designed to allow the selective encapsulation of a class of files. Each file is a member of a subtype class. For example, "normal" files are in the null subtype class. Files which are UNIX directories are the "UNIX directory" subtype class. The accesses to files in a given subtype class may be restricted. The subtype restriction on UNIX directories is that anyone may read a directory, but only a process whose effective user ID is the Directory Manager may write them. These subtype restrictions are in addition to the other types of access checking (security, integrity and discretionary). The access restrictions for a given subtype apply to all files of that subtype.

There are many other possibilities for using subtypes. For example, they could allow "peaceful coexistence" of two separate directory structures as might occur if there were two different Emulators, say one for UNIX and one for another operating system. Subtypes could also be used to control what could be done to files that mechanized the internal structure of a data base management system. Only processes that were known to correctly manipulate the structure would be allowed to change it. The subtype mechanism provides the KSOS Kernel with a significant type extension feature in that it lets the Kernel support encapsulation and control of objects without having the Kernel be cognizant of the syntax and semantics of the object.

sulation and control of objects without having the Kernel be cognizant of the syntax and semantics of the object.

Secure terminal interface

In the secure system it is necessary to have an "unspoofable" path to trusted services. ("Spoofing" occurs when an unprivileged user process pretends to be a privileged process. For example, a nefarious user starts a process that imitates the login sequence, and waits for an unsuspecting victim to type in his password.) In KSOS each terminal is (logically) two devices, the normal terminal device and the secure device. Only privileged Non-Kernel System Software is able to use the secure device. When the user types a reserved attention character (currently BREAK), the normal path is blocked, and the character stream is switched to the secure path. Listening on the secure path is a service process which will cause the desired secure service to be performed. Because the normal path is blocked, rather than killing off any process using it, it is possible for the user to start doing something, temporarily abandon it while requesting some secure service, and resume the activity after the secure service is completed. This is the mechanism by which the user is able to change his working security level. The Secure Terminal Interface is illustrated in Figure 2.

Auditing

DoD security policy requires that certain security-related events be captured for auditing purposes. In KSOS this occurs in two ways, as shown in Figure 3. In the first case, the Kernel captures the events it knows about and generates an IPC message to the Audit Capture process. The second mechanism is that the Non-Kernel System Software captures the event. This second case is necessary because the Kernel cannot tell that certain significant events, like a user login, have occurred. The Audit Capture process does only a minimal amount of processing and then simply places the event record into an audit log. Although it is not within the scope of the current KSOS contract, this audit log could be processed to look for suspicious (sequences of) events.

The UNIX Emulator

The UNIX Emulator is almost completely defined by its two interfaces. It must transform the system calls of the UNIX interface into sequences of Kernel calls. In the design of KSOS a serious attempt was made to get a good "impedance match" between the Emulator and the Kernel, while not having the Kernel be strongly UNIX-dependent.

Our view of the Emulator has evolved significantly. Initially, the Emulator was viewed as not much more than a set of subroutines that resided in a different address space. The functions performed by the Emulator were isolated to one process, except for the obvious cases of interaction with

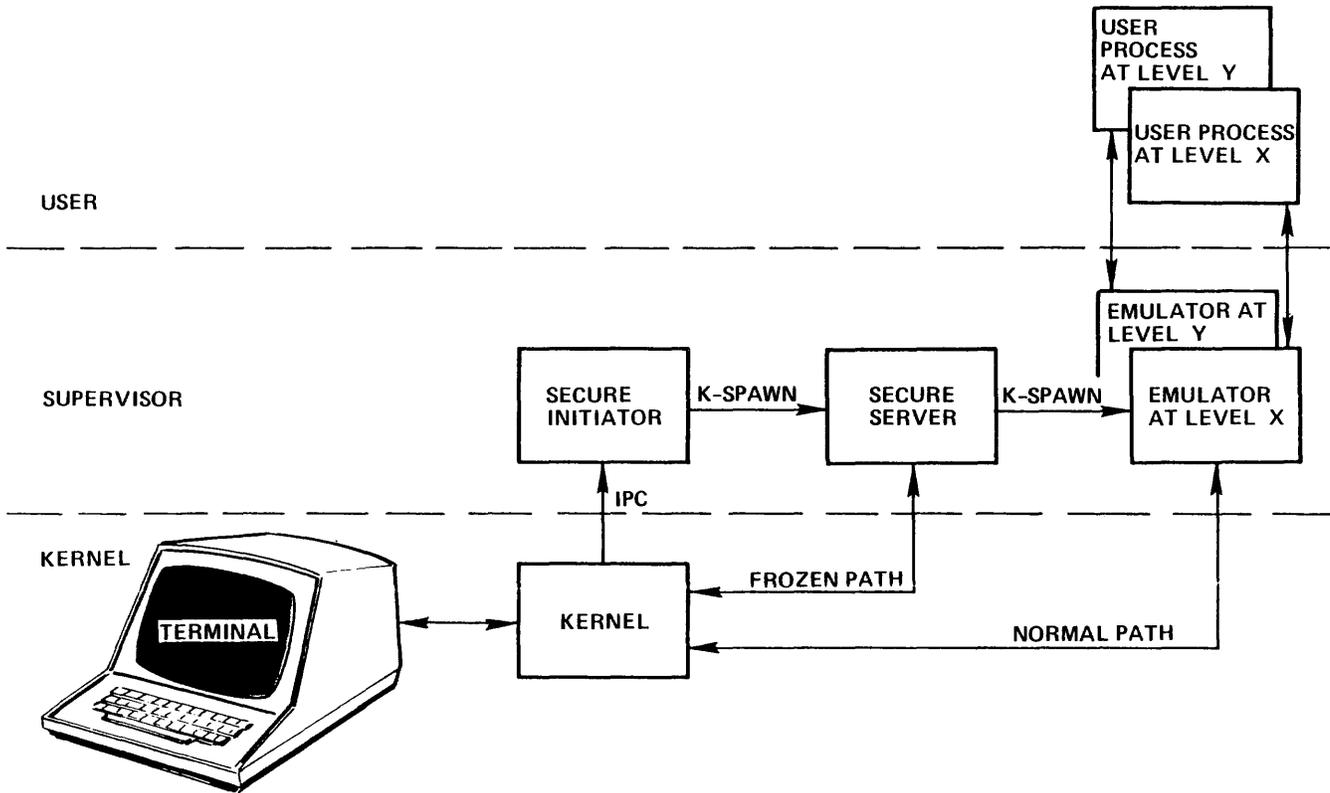


Figure 2—Secure terminal interface.

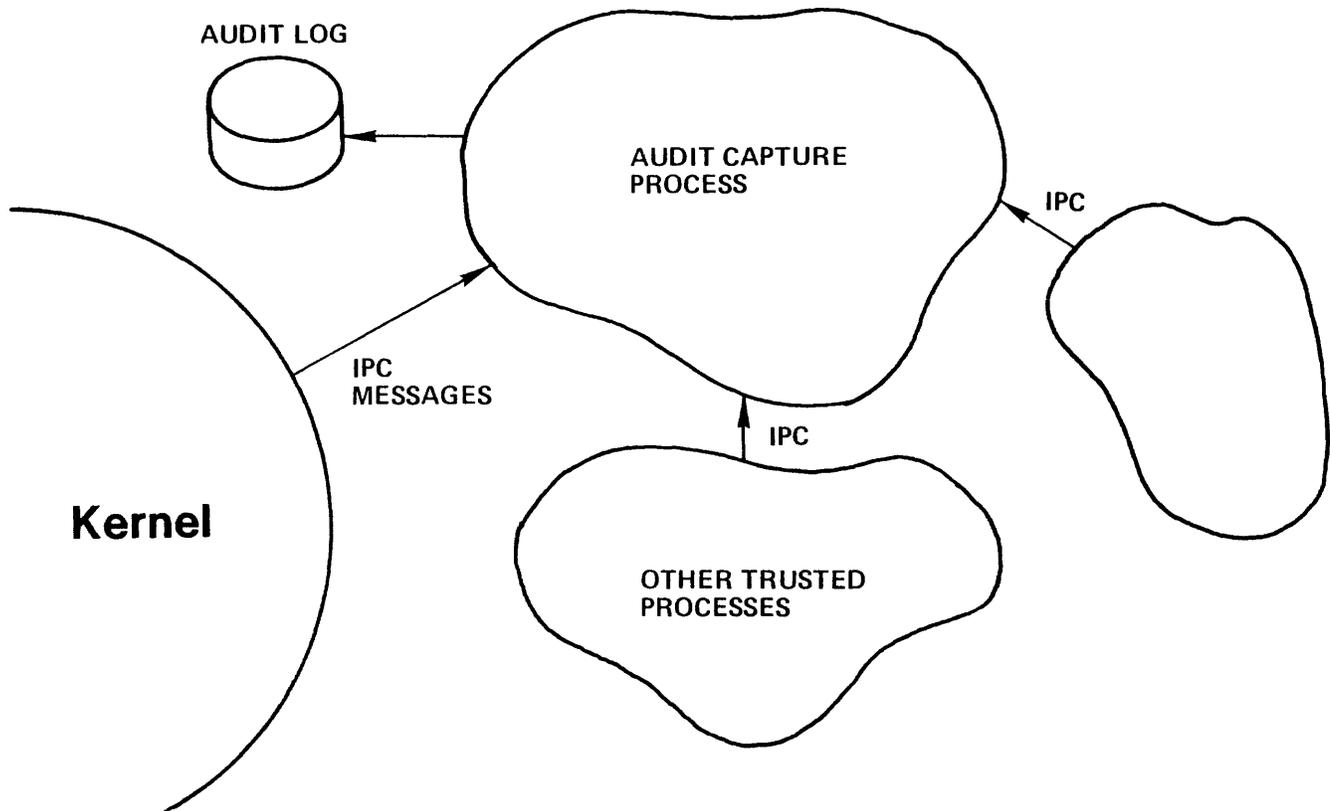


Figure 3—Audit information flow.

other processes via the UNIX pipe mechanism and the `ptrace` system call.

While this view simplifies the Emulator, it is incorrect. At the UNIX interface, a process is indirectly aware of the presence of other members of its "process family" (a process family consists of all the processes that are descendents of the process started at login for a given user). In particular, things like the seek pointers to open files are shared among the members of a process family. Our view of the Emulator now is that it provides an operating system for the process family. The Emulator not only creates the UNIX-level objects from the Kernel-level objects but also provides for controlled sharing of these UNIX-level objects.

It should be remarked that the UNIX interface is perhaps not as "clean" as one would like. There are several subtle ways in which a great deal of the internal mechanization of the system is manifest at the interface. It is debatable whether these things are "bugs" or are "features!"

UNIX directory management

One of the major functions of the Emulator is the creation of the UNIX file system from the more primitive file system provided by the Kernel. The Emulator caches the block i/o supported by the Kernel to provide the byte stream i/o supported by the UNIX interface. The Emulator also is where UNIX directories are managed. The final design of the UNIX directory management function is the result of a long series of (occasionally heated) debates on where directories would be mechanized. Initially they were to be completely managed by the Emulator. However, this was prior to the birth of the subtype notion, and there was no way to guarantee the integrity of the directory structure. In particular, trusted software could not depend upon the directory structure. Then it was proposed to move part or all of the directory management function into the Kernel. This seemed to solve the integrity problem, but opened a new and more serious problem of making the Kernel cognizant of the structure and semantics of directory files, and thereby making the Kernel very UNIX-specific. Finally, the subtype idea was proposed. The Kernel would know that directories were "special," and would aid in the preservation of their integrity. However, the Kernel would not be aware of the internal structure or semantics of directories.

The current design has the Emulator performing all the directory interpretation functions (i.e. recursively searching for names in directories), but writing directories is only done by the Directory Manager. The Directory Manager is a program that is `K_spawned` into execution whenever an Emulator needs to modify a directory. It starts its life running as the user `dir_mgr` who owns the directory subtype. After getting permission for write access to directory subtyped objects, the Directory Manager reverts its identity to that of the requesting user. From there on, the Kernel will enforce security, integrity and discretionary access checking. Thus, the user cannot trick the Directory Manager into modifying a directory that the user cannot access. This architecture may be criticized as being too slow, since creating a new

process via `K_spawn` is moderately time-consuming. However, measurements on one of our UNIX systems in a software development environment suggest that modification of directories is a fairly infrequent occurrence.

Computer network support

The Emulator contains the bulk of the support for the computer network interface. Initially, KSOS will "speak" Version 4 of the Transmission Control Protocol (TCP)¹² including the Internet Datagram Layer.¹¹ This protocol appears to be on its way to becoming a future standard within DoD.

Although no networks presently exist that can handle multiple security levels, this architecture envisages their development and is designed to support them. To support a multi-level network, the Network Daemon would be trusted, so it could handle the multi-level stream to/from the network. The remainder of the TCP functions performed by the Emulator would be untrusted, since they are at only one level.

The basic structure of the KSOS network interface was discussed above, and is illustrated in Figure 4. There is a Network Daemon which handles the Internet Datagram protocol, and enough of the TCP to separate the i/o stream from the network into separate streams for each connection. In each Emulator is the majority of the TCP functionality. All of the functions relating to sequence number maintenance, window maintenance, acknowledgment, and retransmission are in the Emulator. This is possible because these are per connection functions, and need not be globally managed. These two portions of the TCP function communicate using the Kernel-supported IPC mechanisms. The shared segment mechanism is used for bulk data passing, and the event mechanism is used for synchronization, and for "commands" to the TCP Daemon and responses from it.

The non-Kernel system software

The purpose of this component of the KSOS system is to provide the software tools to support a KSOS system. The Non-Kernel System Software is divided into four groups:

1. *Secure User Services*—Software that manipulates the security levels of users and files. Also included in this class are all functions that require a secure ("unspoofable") path to the service.
2. *System Operation Services*—Software that performs continuing services for the system, such as the Network Daemon, line printer spooling and interuser mail.
3. *System Maintenance Services*—Software that performs occasional services primarily in the area of checking and repairing the consistency of the file system. Also included are the system generation functions. Individual KSOS sites can generate their system to suit the hardware configuration available.
4. *System Administrative Services*—Software that aids

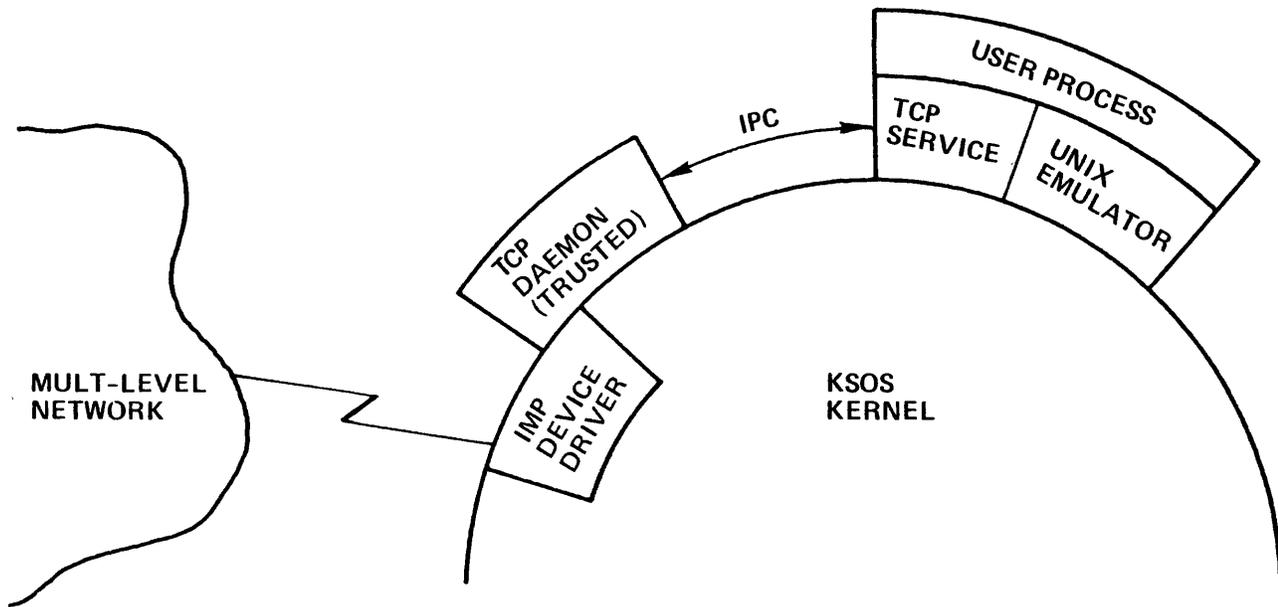


Figure 4—Network interface structure.

the System Administrator in controlling the system. Our goal has been that the System Administrator need not be a computer expert to perform his functions.

The Non-Kernel System Software described is a minimally complete set. Clearly there are large numbers of additional utilities that would be desirable. It is expected that this class will be supplemented extensively as KSOS matures.

KSOS APPLICATION CONSIDERATIONS

There are two broad classes of KSOS applications, each with different considerations. The first is applications that utilize the full KSOS system, i.e., applications based upon UNIX. KSOS should appear to these applications to be only slightly different than a standard UNIX operating system. Because KSOS provides a UNIX-like interface, meaningful secure applications can be built using the existing software. UNIX is one of the best systems in existence for the creation of new products by novel combinations of existing packages, and KSOS will preserve this flexibility. Such applications can, however, be made easier in some cases via the direct use of KSOS Kernel calls.

The second class of applications is those which use the Kernel directly without an Emulator. The Kernel provides many features that make it an attractive operating system in its own right. It offers excellent i/o performance, a range of IPC options, and many features that ease the design of multi-level applications. Because the Kernel is "UNIX-flavored" without being heavily UNIX-dependent, it is possible to create application environments that are an amalgamation of the features provided by different operating systems.

KSOS facilitates the creation of encapsulated environments that can be used for a variety of purposes. This

encapsulation allows objects to be manipulated only by software known to perform correctly. In many cases only a small part of a multi-level application actually deals with data at different security levels. By encapsulation of these functions in a small trusted process, it is possible to build multi-level applications that minimize the amount of trusted (and therefore expensive) code.

ACKNOWLEDGMENTS

KSOS is being created by an exceptionally talented and dedicated team. It is a pleasure to acknowledge their contributions. The Government team on KSOS also deserves acknowledgment for their efforts to make KSOS a reality. Finally, credit must be given to Ken Thompson and Dennis Ritchie of Bell Laboratories for the creation of UNIX. We still marvel at the sophistication and elegance of their product.

REFERENCES

1. Bell, D. E. and L. J. LaPadula, "Secure Computer Systems," ESD-TR-73-278, Vols. I-III, MITRE Corporation, Bedford, MA, November 1973-June 1974.
2. Biba, K. J., "Integrity Considerations for Secure Computer Systems," MTR-3153, MITRE Corporation, Bedford, MA, June 1975.
3. "KSOS UNIX Emulator Computer Program Development Specification (Type B5)," WLD-TR7933, Ford Aerospace and Communications Corporation, Palo Alto, CA, September 1978.
4. "KSOS Implementation Plan," WDL-TR7799, Ford Aerospace and Communications Corporation, Palo Alto, CA, March 1978.
5. Kampe, M., C. Kline, G. Popek and E. Walton, "The UCLA Data Secure UNIX Operating System," Technical Report, University of California at Los Angeles, Los Angeles, CA, July 1977.
6. "KSOS Security Kernel Computer Program Development Specification

-
- (Type B5)," WDL-TR7932, Ford Aerospace and Communications Corporation, Palo Alto, CA, September 1978.
7. "KSOS Maintenance and Support Plan," WDL-TR7810, Ford Aerospace and Communications Corporation, Palo Alto, CA, March 1978.
 8. "Draft B5 Specifications for the MITRE Secure UNIX Prototype," Private Communication, 1977.
 9. "KSOS Non-Kernel Security-Related Software Computer Program Development Specification (Type B5)," WDL-TR7934, Ford Aerospace and Communications Corporation, Palo Alto, CA, September 1978.
 10. Parnas, D. L., "A Technique for Software Module Specification with Examples," *CACM*, Vol. 15, No. 5, May 1972, pp. 330-336.
 11. Postel, J. B., "Internetwork Protocol Specification," Version 4, Information Sciences Institute, University of Southern California, Marina del Ray, CA, September 1978.
 12. Postel, J. B., "Specification of Internetwork Transmission Control Protocol—TCP Version 4," Information Sciences Institute, University of Southern California, Marina del Rey, CA, September 1978.
 13. Ritchie, D. M. and K. Thompson, "The UNIX Timesharing System," *CACM*, Vol. 17, No. 5, May 1974, pp. 365-375.
 14. Robinson, L., K. N. Levitt, P. G. Neumann and A. R. Saxena, "A Formal Methodology for the Design of Operating System Software," in *Current Trends in Programming Methodology*, R. T. Yeh (ed.), Vol. 1, Prentice-Hall, April 1977.
 15. Roubine, O. and L. Robinson, *SPECIAL Reference Manual*, 3rd ed., Technical Report CSG-45, SRI International, Menlo Park, CA, January 1977.
 16. "KSOS Verification Plan," WDL-TR7809, Ford Aerospace and Communications Corporation, Palo Alto, CA, March 1978.

UCLA Secure Unix*

by GERALD J. POPEK, MARK KAMPE, CHARLES S. KLINE, ALLEN STOUGHTON, MICHAEL URBAN
and EVELYN J. WALTON

University of California at Los Angeles
Los Angeles, California

INTRODUCTION

There has been considerable interest for some time in developing an operating system which could be conclusively shown secure, in the sense that the information stored on behalf of a heterogeneous user population was safely protected from unauthorized access or modification, even in the face of skilled attempts to do so. Early attempts to attain this goal consisted largely of auditing an existing system through attempts at circumventing the controls, and then revising the implementation code to block any successful paths that were found. Unfortunately, this approach failed to produce a secure system, largely because third generation operating systems contain so many errors that "penetration audits" followed by patches inevitably led to a system whose controls were still easily penetrated.

However, there was an even more fundamental limitation to the early approaches, frequently mentioned; testing proves the presence but not the absence of bugs. A more strictly constructive method was required, by which it would be possible conclusively to demonstrate the correctness of the security controls. It was hoped that this goal would result in a much superior system in other respects as well. The experience to be reported here strongly bears out that expectation.

The UCLA Data Secure Unix operating system is intended as a demonstration that verifiable data security with general purpose functionality is attainable today in medium scale computing systems. More specifically, the UCLA system has the characteristic that data security, the assurance that data can not be directly read or modified without specific permission, is enforced via a limited amount of kernel software. High levels of care are being applied to demonstrate that the security properties of that software are correctly implemented. In addition, the system is designed so that confinement can be demonstrated by audit of some additional, isolated code.

To achieve these goals, a number of design and implementation principles have been integrated into a single system. These include a tightly constrained base kernel, a second-level policy kernel, a well known and accepted

operating system interface, implementation in the high-level language Pascal, and application of formal, semi-automated program verification methods to the source code of both kernels.

The system interface is essentially identical to Unix as released by Bell Laboratories,⁹ and the software presently runs on DEC PDP-11/45s and PDP-11/70s. The kernel structures and verification procedures, together with the choice of language, provide a powerful means by which the system's security and integrity can be demonstrated and assessed. Support of the Unix interface illustrates the robustness and functionality of the resulting system.

However, the kernel and verification goals imposed significant constraints on the size, complexity and general architecture of the system. The result therefore is quite different from what would have been expected otherwise. System integrity improvement results from the significant reduction in common mechanism operating on behalf of all users, a characteristic that was necessary to make verification and certification of the system practical. Nevertheless, in retrospect, we are unaware of any decision forced by these goals which has not also had the effect of simplifying the system's structure and improving overall reliability and integrity. Significant performance penalties are not expected either. The primary cost in obtaining a secure operating system appears to be found in the care required during design and development.

In the next sections we outline the UCLA Unix architecture, together with explanations for the design choices. Verification and the programming language are also discussed, and illustrative examples of the effects of Unix functionality on the system's operation are given.

OVERALL ARCHITECTURE OF UCLA UNIX

The UCLA Unix architecture contains a number of major modules, whose relation to one another is suggested by Figure 1. The kernel should be thought of as an operating system nucleus which provides about a dozen primitive operations callable from user processes. That is, the kernel implements a number of abstract types and the valid operations on each type. It is the only module in the system empowered to execute hardware privileged instructions.

* This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-77-0211.

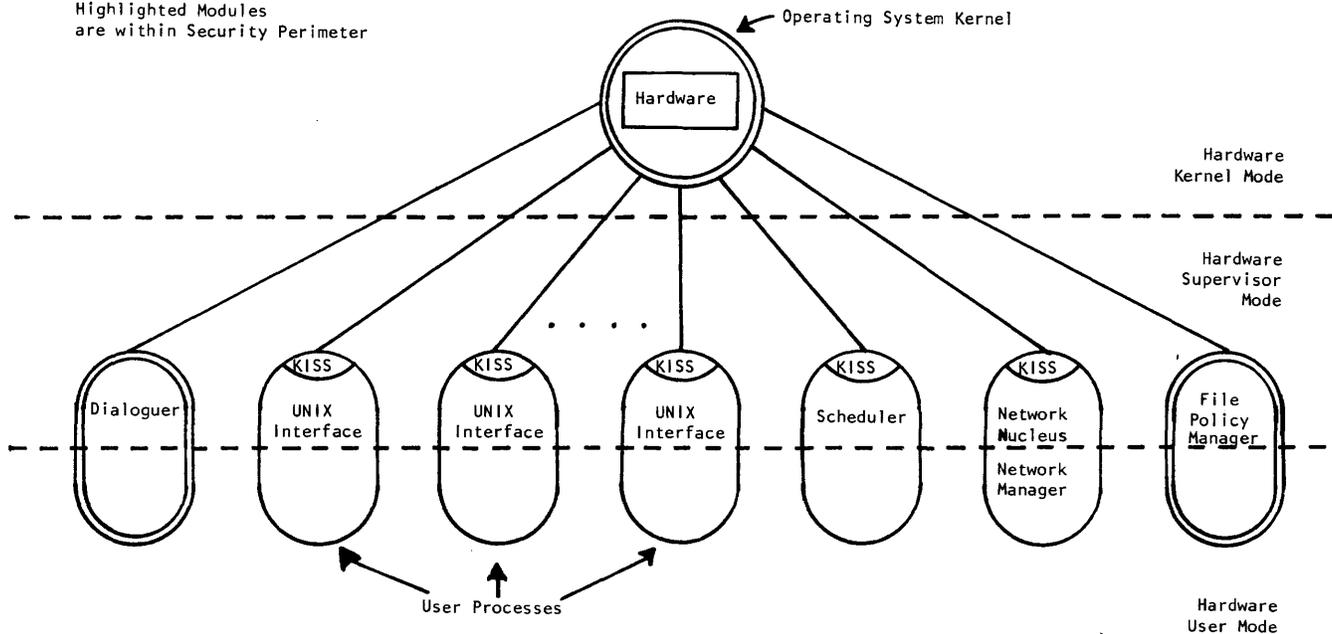


Figure 1—UCLA Secure Unix architecture.

One of the abstract types implemented by the kernel is process. A process contains two address spaces (supervisor and user mode on the large PDP-11s). An operating system interface package resides in one address space. In the other, application code is run. When an application program makes an operating system call, control passes to the O/S package which interprets the call. If necessary, the package issues kernel calls or uses kernel facilities to send messages to other processes to accomplish the needed action. All such calls or messages are controlled by the kernel. Each process is a separate protection domain. The access rights of the domain are represented by capabilities: a C-list for each process is maintained by the kernel.

There are several processes that are special, in that they perform system-related functions. Overall system security depends on the correct operation of two of them.** One, called the policy manager, is the process capable of altering the data upon which protection decisions are made, and is thus the site where various security policies may be implemented. Type extensions to kernel objects, including file systems, typically would also be supported here. In the UCLA system, security policy plus suitable primitives for the Unix file system to support protection of individual files are built in the policy manager process. The second, "dialoguer," process initially owns all terminals (i.e. has capabilities for all of them) and is responsible for user authentication. It tells the policy manager what user is to be associated with a given process.

There is one further process which differs from the typical

** One might say they are within the "security perimeter." Their size is not large compared to the kernel described here. It should be emphasized that the design is such that only software within this security perimeter must be correct to block system penetration.

processes employed for applications programming. However, this one, a scheduler, is not relevant to data security. It contains short-term resource management policy for CPU and main memory: process scheduling, page replacement strategies and the like. UCLA Unix is a demand paged system: when a process page faults, the scheduler is informed by the kernel so that an appropriate swap call may be issued at some later time by the scheduler. All of its security relevant actions are accomplished through kernel instructions, however.

Thus, in normal operation a user first logs into the dialoguer. That process then sends a message to the policy manager, who initializes a process for the user and moves the user terminal to the new process by issuing appropriate capabilities. Process initialization as well as normal computation take place within the domain of the given process. Additional resource requirements or file activity is accomplished through messages to the policy manager. Process switching occurs whenever a given process invokes the scheduler process, or when an appropriate clock interrupt forces such an invoke. The scheduler can then run whatever process it wishes. Page faults also force an invoke of the scheduler, so that it can initiate appropriate page swapping.

THE UCLA KERNEL AND ABSTRACT TYPES

The kernel can alternately be viewed as a basic, stripped down operating system or as an implementor of a number of abstract types, together with the operations on those types. One of its more notable features is the fact that a significant number of facilities, normally found in large systems, are included in it despite its very small size and straightforward structure. The basic kernel consists of ap-

proximately 760 lines of Pascal code, not including I/O support. The PDP-11 does not have any channels, so that the functions of channel programs must be written as CPU code. I/O support in the UCLA kernel is composed of two portions: a device-independent internal interface of approximately 300 lines, and as many device-dependent drivers as are required by devices present on a given machine configuration. These are quite small, and for the UCLA installation, supporting many peripherals, approximately 300 lines of code are required altogether. These numbers are relevant because it is intended the entire kernel be subjected to program verification procedures. Given current verification capabilities, and well structured code, this goal is attainable.

The UCLA kernel implements a fixed number of types, the four listed below. Type extensibility as illustrated by CAL-TSS or Hydra is not provided, although simple extensions appear feasible. The implemented types, together with the permitted operations, are discussed below.

Processes

The process object is defined to consist only of the usual state variables plus one small page. It does not include the process virtual memory. As a result, kernel calls such as *Invoke* can be quite simple, merely moving data from tables to CPU registers and vice versa. All process relevant kernel calls are controlled by capabilities. It is not possible to send or receive interprocess communication, for example, unless in each case a capability is present in the process' C-list.

The process abstraction has been carefully developed to permit a large number of processes to be alive—500 on a PDP-11/70 would not be unreasonable. To do so, it is necessary that very little locked down memory be required per process, despite the fact that there are asynchronous events taking place (such as I/O completions and messages from other processes) which can occur when all the memory of a process is swapped out. The process must be informed of these events. However, the obvious solution, kernel queues, are undesirable since they increase verification difficulties, lead to overflow problems when queue space is exhausted and introduce confinement problems. The UCLA kernel avoids this problem by a number of methods, including a generalized page faulting structure and efforts to keep as much per process information as possible in swappable pages allocated to the given process. As a result, very little main storage must be reserved for an active process. Further, the complexities in two level process mappings, as suggested for Multics are avoided.

The operations available for objects of type process are as follows:

- a. *Invoke*
- b. *Initialize*
- c. *Zero-relocation-register*
- d. *Return*
- e. *Send-interrupt****
- f. *Set-interrupt****

*** These operations are available for all objects.

Invoke moves the state variables of a process into the CPU registers, after first saving those of the currently running process, mostly into one of that process' pages. *Initialize* clears the state variables of a process and grants those few capabilities needed for the process to bootstrap itself. The *Zero-relocation-register* call is used by the process to adjust its virtual memory. The process first sets a data structure in the page shared between it and the kernel to indicate the desired virtual memory page, and then zeroes the associated register, so that the subsequent page fault will cause the kernel eventually to reload the register with the desired value. *Return* is used by a process to change its current site of execution, either giving up control to the scheduler process, checking to see if any interrupts have arrived, or changing from supervisor to user mode. *Set-interrupt* and *Send-interrupt* are used in the system's inter-process communication facility. Both calls give as a parameter the name of an object with which the signal is to be labelled. *Set-interrupt* is used by a process to enable its ability to receive signals labelled with the named object, and *send-interrupt* is the associated inter-process signalling mechanism. *Send-interrupt* also passes a small amount of data, and is the means by which the system supports very low delay inter-process communication (*ipc*). High bandwidth *ipc* is done through shared pages together with this interrupt mechanism.

Pages

Pages are the abstract storage unit supported by the kernel. All pages have a fixed home location on secondary storage, which is not deallocated when the page is swapped into main memory. There are two page sizes in the current implementation, with memory frame sizes set at sysgen time to minimize kernel complexity. In order to access a page, a process must first obtain a capability for the page. Then the process indicates where in its virtual address space the page specified by the capability is to appear. At that point the process can attempt to refer to the page. If it is in core, the hardware register will be loaded and the reference will succeed. If not, the process will page fault, as will be described. Since each page is a separate object, controlled sharing of individual pages is easily done.

The only operations on pages are:

- a. *Swap-in*
- b. *Reflect*
- c. *Free*

“*Swap-in*” copies the secondary storage version of a page into main memory, changing the name of the object associated with that destination page frame to the new page. The secondary storage copy is preserved. “*Reflect*” updates the secondary storage version to match main memory. Neither of these operations gives the caller access to the contents of the page, so that the operation can be issued by untrusted code. “*Free*” deletes a page from main store (only permitted if the secondary storage version is current.†)

† This (trivial) operation is present only so that pages can be moved from place to place in main store. Without “*Free*,” this movement would be more difficult, since the swap call nops when issued if the specified page is already in main store.

To simplify management of physical memory, main memory is divided into two statically allocated regions, one for each page size. That is, one contiguous region in main memory is devoted to small page frames, the other to large page frames. The portion of main memory allocated for each size can be varied at system generation time by changing a compile-time constant.

A similar situation exists for disks, except that the number of regions is variable and changeable at kernel system generation time. A given physical disk unit is broken into an arbitrary number of smaller logical mini-disks. A mini-disk is constrained to contain a fixed number of pages of a given size and object type. Mini-disks, however, are not required to be of uniform size. Hence one mini-disk might have X small pages, while another mini-disk has Y small pages. While large and small pages are not allowed to co-exist on the same logical mini-disk, they can easily exist on the same physical disk. This structure simplifies address calculation within the kernel for pages on disk, yet still allows the system manager freedom to rearrange the configuration of large and small segments on the disk to minimize disk seek and transfer times.

Pages have a home address on disk, and the name of a page can be used to determine this home address in a simple way. Main store is simply a disk cache; in-core pages are therefore merely copies of disk pages.

Devices

I/O operations to all devices, including terminals, are controlled by the same capability mechanism as all other operations. However, devices such as terminals are treated as two devices—an input part and an output part. Two capabilities are therefore required to read and write a terminal, but as a result kernel internals are simplified.

Completion interrupts are handled just like any other process notification. All those processes with capabilities to receive interrupts from the device, with interrupts enabled, and with appropriate access, will receive a notification when the device generates it.

The device operations are as follows:

- a. Start-i/o
- b. Completion-interrupt
- c. Status

“Start-i/o” initiates all I/Os except swaps and reflects. The “Completion-interrupt” is the hardware generated call which typically signals completion of a previously started I/O. As an entry point into the kernel, it is little different from any other call. “Status” I/O causes no input/output operation, but returns status of the previously started I/O.

Capabilities

The capability is the basic kernel representation of protection information—which objects a process is entitled to

access. Each process has associated with it a C-list containing those capabilities, stored in pages that can be swapped, but which are directly accessible only to the kernel. §

Each capability consists of four fields. First is the name of the object to which this capability refers. Second are the access rights provided. Next is a “guess” value which the kernel uses to attempt to quickly find the entry in a kernel table which maps the object indicated by the capability to a physical location. In the case of pages, the guess is the index into the kernel page table to the slot where that page entry last appeared. It in fact may have been moved by subsequent Swaps and Reflects, so if the entry does not match, a search of the table is required. That event is relatively rare however. The last field in the capability is of no relevance to the kernel, but can be set via the Grant call. The Policy Manager uses it to record the file descriptor with which the page or device is associated.

The only operation on capabilities is:

- a. Grant/revoke

It adds a specified capability to a specified slot in a specified process’ C-list. What processes can issue this call is also controlled by capabilities.

The operations on capabilities are thus quite limited. Revocation is accomplished by granting the null capability into the C-list slot that contains the capability to be revoked. There is no means by which processes can directly pass capabilities. While this fact limits what can be done with capabilities, it also greatly simplifies many issues and avoids a number of the criticisms of certain capability systems, especially the danger of not knowing how access to an object has propagated. As a result, the kernel can more accurately be viewed as containing no security policy. All such decisions regarding rights transfer, including initial granting of rights, are made only by the software running in the process which has the ability to issue Grants. The Policy Manager is the only such process in UCLA Unix.

The C-list composes a local name space for the process. This name space has two effects. First, through message exchanges with the policy manager, the user has complete control over which C-list slot contains a given capability, thereby permitting local management over the name space. Fabry¹ points out the significant advantages of this facility. Second, kernel names are not visible to user code. Instead, the capability contains that name. Therefore user code, being unaware of the actual object names, cannot use them as a means to breach confinement.

Types and operating systems

Other authors¹¹ have noted that the usual views of abstract types to be found in programming languages are not quite suitable for operating systems because of finite resources

§ The policy manager is given read access to capability pages so that it need not keep separate track of which capabilities for pages in a file are outstanding. See the discussion of the policy manager for further information.

and circular dependencies. In Multics, for example, the process manager depends on the page abstraction, since the manager is contained in pages, while the page manager is a process and hence depends on the process manager. In a revised design for Multics, abstract types are used in a sophisticated, multiple layered manner to solve these problems.¹¹

However, as noted by Gaines,² the method required need not involve a sophisticated solution at all, and is largely composed of static allocations. This is the approach embodied in the UCLA kernel. Processes, pages and devices are neither created nor destroyed. There are as many pages as there is space on secondary storage for them. The number of processes is fixed by the size of the kernel process table. Devices are added at system generation time. This static view is not really a limitation, since the Policy Manager reuses process "bodies" and pages by reinitializing them via kernel calls. Many systems include these size limitations anyway, although perhaps not so explicitly. As a result, the kernel type structure is exceedingly simple, and yet robust enough for fairly general operating system activity, as illustrated in the sixth section on Unix functionality. Further, the entire kernel is small enough to be locked down in main memory, in space removed from page management, blocking circular dependencies.

Kernel names

The names for kernel-supported objects were designed to maintain several important properties with the minimum of mechanisms: a) Unique names for all objects, b) clear knowledge of object types at all times, and c) avoidance as much as possible of complex name to location mappings, which must be maintained by kernel code if object protection is to be at all meaningful. Since these names are not visible to normal user processes, who see only C-list indexes, considerable design freedom was present. Therefore, names were chosen to represent the home location of the object: a page name consists of the disk device and block number. Hence no disk map need be maintained or interrogated by the kernel.

Paging, segmentation and scheduling

UCLA Unix, unlike standard Unix, is a demand paging system. All user disk I/O, including swapping of the process virtual memory space and file activity, occurs via the paging mechanism.*

Page faulting is invisible to all processes except the scheduler, which is invoked by the kernel when a fault occurs, so that it can start a swap. There are actually two "faults" involved in accessing pages. The most significant, just described, occurs when a page is not core-resident. The other, called a register fault, occurs when the page is resident but

the relevant page register is null. This case is handled in a highly efficient way—a user map table is checked by the kernel to see which capability (and therefore which page) is desired. The appropriate value is then placed in the register and user execution continues.

All kernel calls which require swappable pages to be in core in order for successful execution first check to see if the necessary pages are indeed available. If not, the call completely unwinds itself (a trivial act, since no kernel table updates are made until all checks complete successfully), the process state is reset as described above, and the scheduler is notified as in any other page fault. Even invoking a process, which requires the page that contains the user's registers, operates in this manner. Thus page faults involving kernel-primitive instructions appear to user processes just as page faults involving hardware implemented instructions (that is, they are completely transparent).

The preceding outline suggests how the UCLA system provides a complete virtual memory and file system with only a simple set of paging primitives in the kernel. This simplicity was achieved by two major decisions. First, the virtual memory facilities were decomposed into that which had to operate correctly in order to maintain the security and integrity of the system (Swap, Reflect, and Completion-interrupt) and the rest of the virtual memory mechanism (page replacement algorithm, interaction with CPU scheduling, etc.). This decision had a significant effect on the system's resulting simplicity. Second, file activity and process memory swapping were combined into one mechanism. In standard Unix, main memory is broken into two areas—one to hold user process images, and the other for I/O buffers. Each area is managed separately. The I/O buffers are replaced in LRU order, while scheduling of process images is handled differently. All disk I/O buffers are the same size, while process images vary. The code used to handle I/O buffers is in large part different from that used to handle the movement of process images, and significant parts of both collections of code are important to the system's security and integrity.

In UCLA Unix, only one mechanism, paging, exists, and much of its support has been moved out into a scheduler which can not affect the integrity of the system. As explained earlier in the section on capabilities, the user domain also carries some of the responsibility for virtual memory management. By placing some of the responsibilities in the domain for which the action is being taken, error propagation is further limited. Application code is of course unaware of that responsibility, since the O/S interface is performing the task.

Firmware implementation

The UCLA kernel has been developed to be a candidate for firmware implementation. To be practical, it is helpful if each call behaves as much as possible as a separate instruction, with no need to be interrupted in execution, nor to issue I/O calls for which the results affect the instruction's behavior, since I/O is typically slow relative to micropro-

* A physical disk can alternately be treated as a device, and Start-I/Os issued to it. However, a disk treated in this manner cannot also hold pages.

gram cycle speeds. These criteria are met by the UCLA kernel. Therefore, it differs significantly from architectures such as Multics or related work.^{6,7} In both of those systems all of the operating system, including inner rings in Multics and kernel software in the case of MITRE, must be considered as part of the user process. Any process can be suspended in the middle of execution in the inner ring or kernel mode, respectively. Neither of those systems lend themselves to firmware considerations, the MITRE work because of the architecture, and Multics because of its size and architecture.

Verification impacts

Verification of a full scale operating system is a multistep process, and the methods employed at UCLA are outlined by Popek,⁸ with more detail available from Kemmerer.⁹ The effect that the verification and certification goals had on the system architecture was exceedingly positive. Often a design choice presented itself, without any clear basis for resolution except maximizing verification ease. In retrospect this criterion was quite effective in making decisions and avoiding design pitfalls. Further, when it became clear subsequent to implementation of certain parts of the system that verification would be difficult, those portions were redeveloped. A good example of this case will be outlined in the section on I/O Interfaces.

Sequential code

The current state of verification tools does not permit proof of parallel programs. Since semi-automated aids are, in our view, essential, this constraint implied a kernel design and implementation in which each call ran from start to completion without interruption, including the interrupt handlers. The UCLA kernel is built in this way, and so most of it can be proven by standard verification methods.

The cost of this design choice results from delayed servicing of interrupts which arrive while a kernel call is in progress. To minimize this problem, each call is designed to run very quickly—approximately one millisecond or less. To do so, no kernel call may do I/O of its own while in the midst of execution, since virtually all devices respond rather slowly relative to this criterion. While millisecond delays in interrupt servicing may not be suitable for heavy real time activity, it appears quite acceptable for interactive systems, which is the nature of Unix.

I/O Interface

The PDP-11 does not have any significant channels: instead the device registers are wired into physical address locations and "channel" functions are executed by CPU code. Since all devices address main memory (and secondary storage) in terms of absolute addresses, I/O management is therefore necessarily a kernel responsibility. This is un-

fortunate, for several reasons. First, device semantics are quite complex and difficult to interface with the semantics of the programming language in which kernel code is written. Next, devices are probably the single largest source of changes to the kernel, since as new types of devices are added, additional verified kernel code is required to manage the device's actions. To minimize the impact of these problems, kernel I/O code was redesigned to provide a device independent level of I/O abstraction within the kernel. Code above that level is not concerned with any of the device details. Code below it implements device dependent issues, including any device dependent protection controls. The I/O abstraction level appears similar to a channel interface, with well defined opcodes and operands.

This I/O abstraction level is quite important, likely more so than the process abstractions mentioned by other authors, since at least half of the operating system kernel is concerned with I/O.^{11,6} As a result of its use, device semantics have been isolated to the low-level drivers. See Walker¹² for more information.

THE POLICY MANAGER

The Policy Manager is the major security relevant process in UCLA Unix. It is responsible for implementing a shared file system, for maintaining whatever security policy is to be supported by the system, and for part of the action of process initialization, which occurs every time a Unix fork operation takes place. Each of these issues is discussed below. Long term resource allocation can also be implemented in this process, but currently is not.

The file system and protection policy

User code must see a file structure which is identical to the Unix tree of directories. However, one should not immediately conclude that the entire directory structure and other file support should be implemented in trusted code. In fact, one can make the following argument, largely independent of the security policy to be enforced.

Most code to be run in the user domain strictly should not be trusted to be correct, at least not to the same standards as the verified secure kernel and policy manager. However, all names, including file names, are either issued, interpreted or transmitted through that code. Therefore, it makes little sense to verify the directory-naming scheme of a file system when significant amounts of unverified code issue the names or are in the path leading to the file system. The best one can do, it appears, is to provide the user with a reliable means to specify a process profile which characterizes the categories of files to which the process is to be allowed access. Profile specification and alterations, together with the association of labels with the file on which categories are based, must therefore be done in a guaranteed reliable way if the verified protection and integrity of the entire operating system is to have any meaning. That necessary secure terminal facility will be discussed in the seventh section.

The file protection labels provided in UCLA Unix consist of a very large variety of "colors." Each file can be labelled with some number of them. Each user (principal in Saltzer's terminology¹⁰) has a fixed color list associated with him. It is understood that a user potentially can access a file only if his color list covers that of the file. The actual profile for a running process can be set to any subset of the user's color list. There is a separate profile for read and write.

Since there are a large number of colors, many of the usual protection policies can be implemented using them. Public files are labelled with the color `public` and all users have that color in their list. Denning has noted that military security policy is essentially a lattice, and that the relations of sets and subsets provides just the lattice required. Individual file names are had by assigning a given color to a single file. This color system is still evolving as experience is gained with the user protection interface, especially in the area of control over changes to color lists. Additional detail is provided by Urban.¹⁴

Given the preceding view of file system protection, one can profitably decompose its implementation into two parts, one a common mechanism relevant to security and integrity, the other executable in the domain of the requesting user process. The common mechanism can support a simple, flat file system. Files are the only significant data type, and a color list is one of the attributes of a file. The simple file system mechanism must include complete space management—disk-free lists and maps specifying which pages belong to which files, together with software to manage these data structures.

Many of the facilities normally thought of as part of the file system can be provided by software in the individual process domains as part of the O/S interface—directory structure, maintenance and searching; end of file indicators and other file status information such as usage locks. Directories are then contained in files, and access to directories is controlled in the same way as access to any other files. Assuming that the common mechanism in the policy manager is verified correct, users can affect one another only through the use of files to which they share access. Once again one expects system integrity to be further enhanced, as errors in higher level file system code are confined to the domain in which they occur.

Process initialization and forking

The policy manager must also be involved when new processes are created, since a kernel process body must be initialized and appropriate capabilities need to be granted to the new process. As much as possible however, one wishes process bootstrapping to take place within the domain of the new process. In UCLA Unix, the normal procedure for process forking is as follows. The requesting process sends a message to the Policy Manager requesting the new process as a member of the same user family. The Policy Manager records the user to be associated with the new process and issues a kernel `Initialize` call, which zeroes a process body, grants two capabilities to that process, and sets the program

counter and status to standard values. The capabilities point to a standard boot code page and the `arg-block` page respectively.** A third capability is granted by the policy manager upon process request to give the process the ability to communicate with its forking parent. From here on, initialization takes place wholly in the domain of the new process. The process begins by attempting to execute its boot code, which may cause a page fault. These are handled normally. Eventually the boot code will load the O/S interface and presumably a Unix Shell into its address spaces.

Other policy manager responsibilities

In UCLA Unix, the Policy Manager is also responsible for control over access to the other kernel-supported objects besides pages—processes and devices. Devices appear as special files and inter-process communication takes place through pages (which appear as part of a file). Therefore, colors are uniformly employed for access control in these cases too.

An ARPANET connection is provided in UCLA Unix; access to it must be controlled and support for initial network connection activities is required. Capability based encryption is used to protect each connection individually. See the section on Secure Computer Networks.

THE KERNEL INTERFACE SUBSYSTEM

Since the kernel is an operating system nucleus of minimum size and complexity, one can properly expect that it is not a convenient base to build on. Traditional systems provide a good deal of "extension" for convenience. While at first glance the O/S interface has this responsibility, it should be noted that a considerable amount of code is written to run directly on top of the kernel—the O/S interface, the network manager, process initialization, and the scheduler, for example. Each of these need basically the same extensions—capability management, inter-process communication support, virtual memory code, and some file system interfaces. Therefore we have developed an intermediate interface between the O/S interface and the kernel. The software which implements it provides a much more convenient interface to the kernel and is called the Kernel Interface SubSystem (KISS). As an extension mechanism, the KISS manages the entire environment of the process. In general, no other code in the process makes kernel calls, sends messages to the scheduler or policy manager, etc. Thus this software package has primary responsibility for maintaining a convenient "virtual machine" for the user process.

The KISS of course runs as part of the user process domain, and is architecturally contained in the same address

** The boot code is actually the Kernel Interface SubSystem discussed in the fifth section. The `arg-block` page is read/write shared between the process and the kernel, and serves as the means for passing arguments and return values for kernel calls.

space of the process as the O/S interface. The KISS can be viewed as an inner ring in the sense of Multics, and if appropriate hardware were available, that would be an effective means of implementation.

THE UNIX INTERFACE

The operating system interface has the responsibility of providing a user program interface which is as much as possible identical to standard Unix.*** It handles user system calls either by performing them itself if possible, or making the appropriate kernel calls for service requests to the policy manager to get the desired action accomplished. Parts of the Unix O/S interface are actually composed of code from the standard Unix operating system. Most of the changes consist of wholesale deletions of functions, resulting from the fact that many of those functions are redundant given the available kernel facilities and the fact that the O/S interface is essentially a single user system. All scheduling support could be removed, since scheduling is done in a separate process. A more drastic change concerns I/O buffering. In standard Unix, buffers contain significant structure to aid in multiuser and LRU operation. In UCLA Unix, most of that function disappears since it is done by the paging mechanism supported by the kernel and scheduler. I/O support is replaced in the O/S interface by code that requests file opens and relevant page capabilities from the Policy Manager, and maps those pages to the interface's virtual memory. Then the interface merely tries to reference data on the page to move it to the user, and the usual page faulting and swapping action takes place.

New code in the interface largely consists of the KISS, changes to the interface/KISS boundary, ipc support, and maintenance of the process hierarchy. This last issue will now be discussed.

The file system

The Unix interface has a significant portion of the responsibility for making the user view of the file system equivalent to standard Unix. This task consists of all directory support, including searching, working directory control and the like. Once the desired logical file name is found in a directory, a file open request of the policy manager can be made using that name.† Directory searches are done by first opening the containing file, like any other. It is the responsibility of the Unix interface to manage its open files in such a way as to keep the working directory open most of the time to minimize search costs.

*** There are certain actions possible in standard Unix which will be blocked by the security policy of the secure system.

† The logical file name is essentially an inode number. (Pointer to file descriptor in the Unix file system.)

Forking and process hierarchies

In standard Unix, a given user can have a process family active for him. The family is hierarchical in the sense that parents have certain rights over children. However, intra-family protection is not really effective, since any member of a family can convince any other member to destroy itself, and to take other undesirable actions, via standard Unix functions.

Therefore, process hierarchies should not be supported by kernel code, and so in UCLA Unix, members of a process family cooperate among themselves to effect family behavior. Of course, the support for process families is provided in the O/S interface, so that user software need not be concerned. This design choice simplified the kernel, and in light of the observations just made, had little or no effect on the actual protection functionality provided.

In the implementation, each process of a family has a capability for a shared page, set up by family members. In that page, data structures are maintained by the O/S interface so that intra-family relationships are properly supported. In doing so, the kernel notification facility is used to great advantage. Unix typically performs a great deal of "one to n " notification—one process issuing a signal intended for the rest of the family. The kernel Send-interrupt call is designed to support this behavior efficiently, as well as to be adaptable for other uses.

SECURE USER INTERFACE

In order for any user to have assurance that the protection controls of a system are operating in the manner desired, it is crucial that he be sure of the values to which protection policy data have been set. Further, when login takes place, there is an issue of mutual authentication: the user wishes to be sure that he is interacting with the secure system interface, not some clever user simulation of it which collects passwords. For both of these reasons, UCLA Unix contains a small dialoguer process to which the user terminal can be reliably connected. The user causes his terminal to be switched to the dialoguer by typing a predefined sequence of break characters.§ The kernel supports the terminal switch through maintenance of a terminal state. A terminal can be thawed or frozen. Capabilities are granted by the Policy Manager giving access to terminals only when thawed, or only when frozen. When the break sequence is detected, or when a line drop occurs, the line is marked frozen. The Policy Manager grants frozen access only to the dialoguer, thawed access in all other cases. In this way, the user can move his terminal to the dialoguer, accomplish whatever change is desired, such as changing process profiles, and then move the terminal back, all without disturbing the state of computation of the process at all so that it can be continued.

§ Kernel recognition of the break sequence is not expensive since PDP-11 hardware requires character by character terminal input handling anyway.

THE SCHEDULER

Whenever it is time for a process invocation decision to be made, the Scheduler is invoked, either directly by a user process (i.e. when it wishes to sleep) or by a clock interrupt. The kernel makes available a considerable amount of system data through a pseudo device, so that the scheduler can make sophisticated resource allocation decisions, about both memory and the CPU. Centralizing both classes of resource control permits effective coordination of allocation decisions and therefore potentially higher performance. A large class of scheduling policies can be implemented in this process. Some of them have confinement implications but provide better performance potential than those which do not. This architecture permits the system operator to make the confinement/performance tradeoff, since there is no kernel effect from scheduling policy changes.

The one potential drawback of a separate scheduler process is that it doubles the actual number of process invocations over what is really needed. This overhead is of little consequence if context switches are relatively cheap, not really the case for UCLA Unix.*

SECURE COMPUTER NETWORKS

When security is of concern in a computer network, encryption of the lines is generally a necessity, because those lines are not considered safe from tapping or spoofing. However, the usual approach is to encrypt and decrypt the data external to the central machine and its operating system.

It should be recognized that the software resident within the operating system responsible for managing the network is both complex and relevant to security and integrity. In standard Unix with an ARPANET Network Control Program (NCP), the NCP, operating as a common mechanism, is of comparable size and complexity to the whole operating system.** Typically, one wishes to protect each network connection separately from each other connection, but the NCP manages them all, including moving data from user buffers through the NCP and out to the network interface device.

Given the availability of a secure operating system, one can entertain the idea of extending the "ends" of the encryption path deep into the operating system. For example, the user process, as it hands data over to the NCP, could be forced to cause the data to be encrypted, so the network software is treated merely as part of the insecure transmission channel. That data would not be decrypted until the receiving NCP handed it over to the destination user. If each

* Context switches on the PDP-11 are in general fairly slow. Therefore, the scheduler is to be changed so that it is not invoked at every process switch, but instead periodically gives the kernel advice about which processes are to be run. In this way, most of the scheduling algorithms remain out of the kernel, but the additional overhead of having two context switches per (desired) context switch is eliminated. That work was not done when this paper was authored.

** The NCP being considered was developed at the University of Illinois.

connection were encrypted with a separate key, then NCP errors and misdelivery within the host operating system would not affect security. If suitable error correction is incorporated with the encryption, then integrity problems can also be detected.

The main problem in this approach is the initial connection establishment protocol—how to permit users to supply the NCP with parameters telling which site and what type of connection should be established, without large confinement channels in the system. For a discussion of these and related issues, see Kline.⁴ The method of solution outlined there has been implemented in UCLA Unix. The additional kernel code to support secure network operation was quite small. Further, most of the original NCP was kept unmodified, although its lower level was altered to match the kernel interface.***

PROGRAMMING LANGUAGE ISSUES

The programming language employed in software development is usually recognized to have a significant effect on that effort; however when the goal of development includes verification, the effect is heightened. The specific language issues break down here into two groups—those concerned with systems programming, and those concerned with the scale of the verification steps.

Systems programming issues arise in the same way that they occur in most high-level systems programming languages. It is necessary to be able to express details of the hardware in the high-level language, such as interrupt vectors, hardware device registers, or special instructions. These facilities must be available in the programming language, but in a way that minimizes the effect on the semantics of the rest of the language.

Virtually all the security and integrity relevant code in UCLA Unix is written in a slightly altered Pascal. Obvious verification problems were removed from the language, such as pointers, variant records and various sources of aliasing.⁵ I/O facilities were also deleted, since we were building I/O mechanisms, among other functions. The run-time package needed to support Pascal I/O would have been useless baggage, and since it typically would be written in assembly code there would be little chance of ever verifying properties of its operation.

It was also necessary however to add features to Pascal to permit systems programming, as remarked above. Very few additions were actually necessary, and were limited to the following:

1. The ability to declare a variable to be stored at a fixed physical location (to initialize interrupt vectors, access device control registers, etc).
2. Assembly language procedures (so that special hardware instructions could be expressed as a procedure call).

*** The Illinois NCP "kernel" was rewritten.

3. The ability to have procedures which take array parameters whose length is determined at call time (to remedy the most significant limitation of Pascal).

We also developed an extensive library system to support independent compilation of program modules, and yet force type integrity across module boundaries. The compiler and library system force recompilation of modules when needed for compatibility with another module which has been altered. This facility is needed since the verification work depends on type enforcement. The language, compiler, and library system are discussed by Walton.¹³

There are many issues concerned with the scale of the verification effort. It is believed that over half of the original verification effort could be avoided if the language contained more reasonable controls over aspects of program behavior. One of the more obvious examples concerns the integrity of global variables. An important portion of the assertions to be verified state that most of the kernel variables have not been altered by the routine being considered. (After all, much of the statement of security concerns what is not to happen.) These assertions, in the form of a large invariant, could be simply handled by scope controls in the language, such as the Import/Export lists of Euclid.⁵ Then compile time enforcement could be employed and the verification task correspondingly simplified. UCLA Pascal has been modified to provide Import Lists.

Another example where the verification task can be eased concerns array bounds checking. In Pascal, many subscripts can easily be out of range, and therefore potentially reference data other than the given array, violating type rules. There are four reasonable ways to deal with this problem—subscript checking could be done by hardware, by runtime software generated by the compiler, by runtime software explicitly inserted by the programmer, or it could be verified in many cases that subscripts do not get out of range. The PDP-11 hardware base does not provide any reasonable way to itself check subscript references.[†] The UCLA Pascal compiler does not implement array checking code. Therefore, a combination of the remaining choices were taken. The resulting assertions which need to be proven compose a significant fraction of the total verification to be done. Clearly here is a fertile area for language support or enhanced verification tools.

ARCHITECTURAL OBSERVATIONS

UCLA Unix comprises the first verifiably secure, full functionality operating system with a fine grain of protection. The experience gained in its design and development

led us to several conclusions. Most obvious, secure operating systems are feasible to develop, although the development cost is likely to be considerably greater than if highly reliable security and integrity were not such a serious goal. However, the result is a system which appears to exhibit considerably enhanced reliability and integrity, and because of the strict modularity, is easier to modify. Performance does not appear to be seriously affected by the architectural constraints imposed by the various goals. That is, the net result of the security goal seems to be a better system in general.

It should be noted, however, that one of the central ideas to the success of the work, kernel-structured architectures, requires considerable rethinking of the usual operating system architecture views if it is to be effectively employed. Much of the standard operating system wisdoms must be reexamined, or the result will be a "kernel" that is in fact overly complex and not suitable for a rigorous demonstration of correct security and integrity enforcement.

In conclusion, it appears that the goal of obtaining secure operating systems, at least for centralized, medium scale machines, has been largely reduced to (high quality) engineering, with the most significant progress required in program verification.

BIBLIOGRAPHY

1. Fabry, R., "Capability Based Addressing," *Communications of the ACM*, Vol. 17, No. 7, July 1974, pp. 403-412.
2. Gaines, R. S., Private communication, 1977.
3. Kemmerer, R., "Verification of the UCLA Security Kernel: Abstract Model, Mapping, Theorem Generation and Proof," PhD Thesis, UCLA Computer Science Department, 1978.
4. Kline, C. S., "Protection Mechanisms for Operating Systems and Networks," PhD Thesis, UCLA Computer Science Department, 1979 (forthcoming).
5. Lampson, B. et. al., "Report on the Programming Language Euclid," *SIGPLAN Notices*, Vol. 12, No. 2, February 1977.
6. Millen, J., "Security Kernel Validation in Practice," *Communications of the ACM*, Vol. 19, No. 5, May 1976, pp. 243-250.
7. Organick, E., "The Multics System, an Examination of its Structure," MIT Press, 1971.
8. Popek, G., and D. Farber, "A Model for Verification of Data Security in Operating Systems," *Communications of the ACM*, September 1978.
9. Richie, D., and K. Thompson, "The Unix Timesharing System," *Communications of the ACM*, Vol. 17, No. 7, July 1974, pp. 365-375.
10. Saltzer, J., and M. Schroeder, "The Protection of Information in Computer Systems," *proceedings of the IEEE*, 1975.
11. Schroeder, M., D. Clark, J. Saltzer, "The Multics Kernel Design," *Proceedings of the Sixth Symposium on Operating Systems Principles*, W. Lafayette, Indiana, November 1977.
12. Walker, B., "Verification of the UCLA Security Kernel: Data Defined Specifications," Masters Thesis, UCLA Computer Science Dept., November 1977.
13. Walton, E., "The UCLA Pascal Translation System," UCLA Computer Science Dept. Technical Report, January 1976.
14. Urban, M., "A Policy Manager for UCLA Secure Unix," Masters Thesis, UCLA Computer Science Dept., 1979 (forthcoming).

[†] The new, upward compatible DEC VAX/780 does.

KSOS—Development methodology for a secure operating system*

by T. A. BERSON and G. L. BARKSDALE, JR.

Ford Aerospace & Communications Corporation
Palo Alto, California

INTRODUCTION

The goal of the Department of Defense Kernelized Secure Operating System (KSOS) project is to design, implement and prove a secure operating system. Specifically, it is desired that KSOS be designed and proven to enforce a security model, derived from the security practices of the Department of Defense, referred to as "multilevel security."

The proof required for KSOS is rigorous proof in the mathematical sense. The necessity of preparing for proof of a program so large and complex as an operating system has led to the adaptation and, where necessary, development of design and implementation methodologies for KSOS which are departures from the usual methods of systems programming. Specifically, KSOS has required formal specification of operation system design, automatic theorem generation, automatic theorem proof, selection and use of a verifiable programming language, and verification of operating system programs. KSOS represents the first industrial application of many of these techniques, and is breaking new ground in the construction and proof of large scale computer systems. This paper describes what methods were chosen for KSOS and how they are being applied.

BACKGROUND—THE MODEL AND THE SYSTEM ARCHITECTURE

The multilevel security model attaches a tag known as an *access level* to every object managed by the system and places constraints upon the valid relationships between the access levels of interacting objects.

The design of KSOS has been described in detail elsewhere.¹⁻⁵ A brief description of its architecture provides sufficient background for a discussion of the methodology. A KSOS system is comprised of

1. A kernel,² which performs operating system functions and which has the responsibility of enforcing the se-

curity policy. The object types supported by the KSOS Kernel are processes, process segments, files, devices and subtypes. The Kernel is motivated to perform actions upon these objects by sequences of calls to the routines which the Kernel provides at its interface with the rest of the system. An important mechanism for enforcing the security policy is provided by comparisons, made at kernel call time, between the access levels of the caller and of the objects the caller seeks to manipulate.

2. An emulator,⁴ which uses the facilities of the Kernel to fabricate an (arbitrary) environment for user programs. The emulator being prepared initially for KSOS emulates the UNIX** operating system.⁶ The use of an emulator is convenient for applications which seek to exploit existing software, but is not strictly necessary. The KSOS design envisages that certain applications will make direct use of Kernel facilities.
3. Support software⁵ to aid in the day-to-day operation of the system, (e.g. secure spoolers for line printer output, dump/restore programs, portions of the interface to a packet-switched communications network, etc). These are collectively referred to as "Non-Kernel System Software" (NKSS). Because of its varied responsibilities, portions of the NKSS must from time-to-time be allowed to violate the security model in order to operate correctly. These portions are referred to as *privileged NKSS*. To a large extent, they represent a mismatch between the idealizations of the multilevel security model and the practical needs of a real user environment. The design of KSOS allows for these violations but seeks to minimize them by providing for the economical definition of finely grained privileges and mechanism for Kernel security support of user defined extended types.⁷

The KSOS components for which proof is required are those which are responsible for enforcing the multilevel security model, i.e. the Kernel itself and those portions of the privileged NKSS.

The remainder of this paper is organized around a schema

* The work described in this paper was performed under ARPA Order 3319, Contract MDA903-77-C-0333 administered by the Defense Supply Service, Washington. The opinions expressed are those of the authors and not necessarily those of the Government or Ford Aerospace.

** UNIX and PWB/UNIX are trademarks of the Bell System.

for the construction of provable systems. Initially only a brief overview of the schema is presented. This is followed by an elaboration of each step of the schema, first in general terms and then in terms of its impact upon KSOS methodology.

CONSTRUCTION OF PROVABLE SYSTEMS

Figure 1 illustrates a general schema which, in principle, can be used to construct proofs that systems conform to arbitrary policies. It will be seen that the overall proof is constructed from two sub-proofs, namely

- P1—Proof that the system design conforms to the desired policy (property).
 P2—Proof that the implementation conforms to the proven design.

From these there follows

- P3—Proof that the implementation conforms to the desired policy.

Successful application of this schema requires that a great many careful preparations be made before the proofs are attempted. The various methodologies adopted for KSOS were chosen to ease the burden of these preparations. These methodologies have so far been useful in this role. In addition, the discipline of following rigorous design methodologies has yielded software engineering benefits which were unanticipated at the time the methodologies were adopted. These will be discussed in more detail below.

FORMAL STATEMENT OF DESIRED PROPERTY

Consider first those preparations involving the desired property of the design. Generally, there exists some informal statement of this property. Informal statements, written in natural language, are designed and may be adequate for human interpretation. They are however unsuitable as a touchstone for mathematical proof as they lack sufficient precision and are not in an easily manipulatable form. The informal statement of the property must therefore be con-

verted into a suitable formal statement. Great care is required at this stage to ensure that the formal statement accurately and adequately represents the intention of the informal statement.

In the case of KSOS, informal statements of the desired security property are to be found in regulatory documents.^{8,9} These informal statements embody an intuitive notion of military security policy. They are widely applied and well understood. A mathematical model approximating this policy was developed by Bell and LaPadula.¹⁰ This model has been utilized as a formal policy statement in the design and verification of a security kernel¹¹ during a project which was a predecessor to the present KSOS project. Another similar model was described by Walter.¹² The formal policy statement being used for KSOS was prepared by generalizing from these two models and formulating the generalization in terms amenable to proof. A informal description of the model may be found in a companion paper.⁵ Full details of the KSOS formal statement are shown in Reference 13.

FORMAL SPECIFICATION OF SYSTEM DESIGN

The next step in construction of a provable system is expression of the system design in a fashion which is suitable for the construction of a proof. Such an expression is referred to as the system's *formal specification*. A formal specification may be viewed as a set of equations which describe the possible "states" of the system.

Viewed this way, the preparation of formal specifications is not attractive to system designers. There are two main problems. The first is that the sort of state abstraction required to formulate the equation set is not the same sort of functional or data abstraction in which the designer is trained and which (following tradition) he would otherwise use to express his design. The second problem is that adequately detailed specification of useful systems requires a large and unwieldy set of equations throughout which it is difficult to maintain consistency.

Hierarchical development methodology

Fortunately both problems may be alleviated by use of appropriate computer-based techniques. The techniques chosen for use in KSOS are embodied in SRI International's Hierarchical Development Methodology (HDM).¹⁴⁻¹⁶ HDM addresses formal specification problems by providing mechanization for each of a series of steps required for system design and production. These steps are the decomposition of a design into a partially ordered hierarchy of modules, the specification of each module, the specification of the interfaces and mappings between modules and, eventually, the proof. All of these functions are performed by an integrated collection of supporting programs.

In HDM as used for KSOS, each module is considered to represent an incremental abstract machine. This is implemented upon the abstract machines coming below it in the hierarchy. Each module is specified in terms of the data

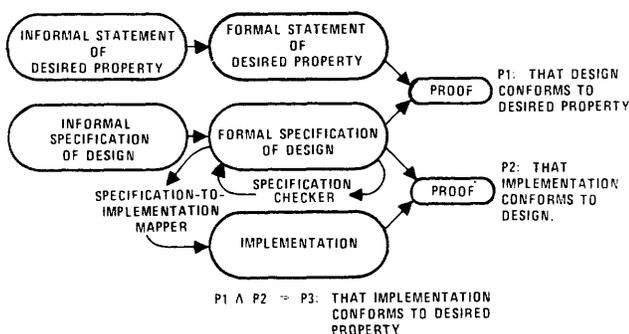


Figure 1—Schema for the construction of provable systems.

types it manipulates, and in terms of functions. There are several kinds of functions. Primitive V-functions represent the state of the abstract machine. They have a value. Derived V-functions also have a value which is computed from the value(s) of primitive V-functions. O-functions represent operations upon the state of the abstract machine by specifying how the machine's state *after* the operation is related to the machine's state *before* the operation. OV-functions combine operation and state. This model of functional decomposition follows roughly from the early ideas of Parnas.¹⁷ A large reduction in the problems of abstraction is due to this model.

The abstract machines are specified in a nonprocedural language called SPECIAL (for SPECification and Assertion Language).¹⁸ SPECIAL has the advantageous property that the effects of a computation may be specified independently from that computation's implementation. This is a vital precondition to design proof (P1 in Figure 1). In addition, this property allows the designer to concentrate upon the structure and functionality of his system, and to defer decisions about data representation and even algorithm choice until implementation. SPECIAL is supported in HDM by a language processor, called the Specification Checker. This provides a number of largely syntactic tests which aid the designer in maintaining consistency of definition within individual modules and throughout a hierarchy of modules.

Formal specification of KSOS

The externally visible design of the KSOS kernel was decomposed and specified using HDM. Twenty modules were used to establish and support the functionality of the kernel interface. The hierarchy of these modules is shown in Figure 2. Each of these modules has been specified in sufficient detail to allow all the externally visible effects of each kernel call, or of any sequence of kernel calls, to be determined by inspection of the specifications. The size of

each module is dependent upon the nature of its abstraction and the ease of specifying how that abstraction might be produced in terms of functions available from lower level abstractions.

There are 34 KSOS Kernel calls. The specification of the kernel's visible actions contains the definition of about 240 functions, comprised of roughly 3000 lines of SPECIAL. The complete formal specifications of the KSOS Kernel have been published in Reference 2, and those of the privileged portions of the NKSS in Reference 3.

A small example (the single function SEGrendezvous) taken from the Kernel formal specifications is included as Figure 3 to give the reader exposure to the style and content of the work. SEGrendezvous is part of the mechanization of shared segments. The function is a derived V-function. It returns a value computed from the values of other V-functions. It takes three parameters, pSeid, rdvSeid, and da. The names "seid," "daType," and "RdvType" are user-defined type names. The semantics of SPECIAL call for the EXCEPTIONS to be evaluated in turn. If any of them are found to be TRUE, its associated label is "returned" as an error value and no further EXCEPTIONS are checked. If all of the EXCEPTIONS are FALSE the return value, segSeid, is calculated according to the specification in the DERIVATION section of the function.

Stated in slightly different terms, the precondition of a SPECIAL function is the conditional conjunction of the negations of its EXCEPTIONS. The postcondition of a SPECIAL function depends upon the type of function being considered. In the case of derived V-functions (such as this example) the postcondition is the DERIVATION. In the case of O-functions, the postcondition is the conjunction of the function's EFFECTS.

Drawbacks and benefits of formal specifications

It is unfortunately true that the formal specifications of KSOS are difficult to read. This point is amply illustrated by Figure 3. Their poor reliability is due in part to the syntax of the SPECIAL language, and in part to the specification style adopted (style continues to evolve: see Reference 19). The specifications are nevertheless popular with KSOS designers and implementors. This is because they provide a medium in which design decisions can be expressed, discussed and recorded with precision and with assured continuation of design consistency. The designers and implementors communicate effectively in terms of the formal specifications. This is certainly a major benefit and it was unanticipated when the methodology was adopted.

Additional unanticipated benefits derive from the constraint of working with a hierarchical decomposition. We have found it extremely difficult to make a clean decomposition and formal specification of a kludge. On several occasions during work on KSOS, difficulty in formulating a specification for a design has encouraged prompt reexamination and subsequent simplification of that design. In other words, formal specifications help designers to understand and evaluate their product.

LEVEL	NAME	ABSTRACTIONS
19	KER	kernel call interface
18	SPF	special functions
17	PRO	process operators
16	IPC	interprocess communication
15	FCA	file capabilities, open files
14	SUB	file subtypes, type extension
13	MFS	mountable file systems
12	PST	process state
11	PVM	process virtual memory
10	SEG	segments
9	FIL	file contents
8	FST	file state
7	SMX	security model
6	PRV	privilege control
5	DIF	device independent functions
4	TII	object type independent information
3	SYL	system level
2	SEN	secure entity names, system namespace
1	DIF	device dependent functions
0	MAC	machine

Figure 2—KSOS Kernel abstraction hierarchy.

```

VFUN SEGrendezvous (seid pSeid,rdvSeid; daType da)->seid segSeid;

    EXCEPTIONS
KSegBadName: ~(EXISTS rdvTe x INSET SEGrdvTable() :
                x.nameSeid=rdvSeid);
KSegBadLevel:~(EXISTS rdvTe x INSET SEGrdvTable() :
                x.nameSeid=rdvSeid
                AND SEGshareCheck(pSeid,x.segSeid,da));

    DERIVATION
LET rdvType x= SOME rdvType y | y INSET SEGrdvTable()
    AND y.nameSeid=rdvSeid
    AND SEGshareCheck(pSeid,y.segSeid,da)
    IN x.segSeid;

```

Figure 3—Formal specification of the function SEGrendezvous.

DESIGN PROOF

Consider now the next step in the provable system schema—the proof that the design conforms to the desired property. Slightly more rigorously, we wish to prove that the formal specification of the design implies the formal statement of the desired property. In practice, this inference is not proven directly. Instead, the formal specifications are processed to generate formulas relating the states of the specified design to the desired property. An attempt is then made to prove these formulas. For KSOS, the generated formulas are such that, for each specified function, the access levels of the objects manipulated are related to the access level of manipulator in accordance with the formal statement of the security policy. The formulas themselves take the form of inequalities upon access levels.

The formula generator has information about the computational model of HDM and about the "semantics" of SPECIAL. It also has implicit information about the formal model of security. An anticipated generalization of the formula generator would accommodate arbitrary formal models, perhaps expressed in SPECIAL.

KSOS exhibits considerable novelty in its use of an automatic design proof environment. During a predecessor project,¹¹ the generation of formulas and their proof as design theorems was done manually. This manual work was labor intensive and mind-numbing; it required great vigilance against error. In KSOS the cost of proof and the risk of error are reduced by utilizing an automatic formula generator coupled to an automatic general-purpose theorem prover.²⁰ To our knowledge, KSOS is the largest program for which automatic design proof along the lines sketched above has been accomplished.

Proving the published KSOS design entails the generation

and proof of about 500 formulas. This is routinely accomplished in an entirely automatic fashion. The complete process requires about 10 CPU-minutes on a DEC KL-10. This dramatic reduction in proof cost has made it feasible to include a feedback path not shown in Figure 1 whereby formal specifications giving rise to unprovable theorems are modified and the proof then retried. By this mechanism even details of the design can be coerced into conformance with the desired property.

IMPLEMENTATION

Proof P1, that a formally specified design implies a desired policy, is a major milestone in the schema. The next logical step is to produce an implementation of that design. For this, an implementation language must be chosen. The translation from formal specification to implementation is, in part, automatable. However, completion of the task requires application of traditional inspection, review and testing methods. As the KSOS implementation effort is only just beginning at this writing, we are forced, in this section and the next on Program Proof, to discuss our plans, not our results.

Choice of implementation language

The peculiar nature of operating system programming places some well known demands upon programming languages. In addition to these, KSOS places the additional requirement that the system implementation be verifiable, i.e., P2: proof that the implementation conforms to the specification.

The following are the requirements for the KSOS pro-

programming language:

- a. The language must be well defined and be supported by a stable, efficient compiler, which produces efficient code.
- b. The language must provide “modern” control structures, data structures, abstract types, type safety and machine-dependent scopes.
- c. The language must be compatible with HDM and be amenable to axiomatization.

A short list of likely system implementation languages was prepared. These were Euclid, Modula, ILPL, Gypsy, Pascal, C and Ada. Of these, the most suitable on technical grounds appears to be Euclid. However, difficulties encountered by the implementors of a compiler for Euclid have led to our choice of Modula as the KSOS implementation language.

Mapping a formal specification into code

Several levels of documentation and specification have been developed for KSOS. First, a system-level specification was produced.¹ Next, a design specification was developed that included both prose and formal specifications for the major components of the system (e.g., Kernel, Emulator, NKSS).²⁻⁴ And finally, a product specification was developed for each of the major components of KSOS. Each of these specifications is more detailed than its predecessor and defines the implementation more exactly. In concept, each successive specification provides a refinement of the ideas presented in earlier, higher-level specifications.

Care must be taken to avoid the constant danger of inconsistency, both within a given specification level and between levels. To guard against this, we have established the primacy of formal specifications in all questions. Thus, each of the managers, designers and programmers on the KSOS project has at least a reading knowledge of SPECIAL.

There are some aspects in which any formal specifications bind their admissible implementations. Specifications written in the style used for KSOS bind the structure, functionality and local assertions of the implementation. In order to exploit this binding we plan to create and use a software tool which will map from the formal specification domain into the implementation domain, producing implementation language skeletons for use and refinement by the implementors.

There is, in principle, no requirement that a given formal specification be implementable. Neither is there a requirement that the structure of an implementation follow that of its specification. It seems to us, however, that both of these requirements increase the practical utility of incorporating formal specifications in a methodology for program development, and we therefore strive to meet them. We have shown how the KSOS formal specifications are used both manually and automatically to provide implementation guidance. The goal of decomposing the system specification into an easily implementable hierarchy was identified early in the

KSOS project. The effectiveness of this choice will not be known until the implementation is complete.

Implementation environment

KSOS is being implemented in the environment provided by the Programmer's Workbench (PWB/UNIX).^{21,22} This program-development, management, maintenance and testing tool provides the facilities needed to carry out the complex development and maintenance activities required by the KSOS project.

The development plan for KSOS requires that several small programming teams concurrently write and test portions of the system. Configuration control will be maintained through use of the PWB/UNIX's Source Code Control System (SCCS).²³ SCCS will be used to control all forms of machine-readable text (e.g., design notes, formal specifications, implementations, test plans) created in conjunction with KSOS implementation. Use of SCCS will provide a complete audit trail of systems development, the ability to reconstruct any version of the evolving system and the basis for subsequent maintenance of KSOS.

Inspection and Review

The accuracy with which verified formal specifications can be implemented has been the subject of much concern to the KSOS project. How can one ensure that an implementation will perform exactly the specified function and no other? Complete code proofs (i.e. P2) of the implementation would perhaps answer this question, but they are not anticipated for KSOS.

The function of ensuring an accurate match between the formal specifications and their implementation is therefore assigned to a formal inspection process using the techniques described by Fagan.²⁴ Two levels of formal inspection are planned. The first, called Design Completion Review, authorizes release of module designs for Critical Design Review by the KSOS customer. This review takes place when the detail design has reached the level that each design statement corresponds roughly to ten or fewer statements in the implementation language. The second inspection, called Code Completion Review, is scheduled after the first diagnostic-free compilation of the complete module.

The focus of each inspection is to ensure conformity of the detailed design and implementation to the proven formal design specification. Every module must pass these two inspections.

Testing for specification compliance

The quality assurance efforts for KSOS seek to provide a series of convincing demonstrations of the security and completeness of the system. Inspection and review are two contributors to quality assurance; testing is another.

The formal specifications are a useful guide to test case

selection. In particular, test cases are selected such that there is at least one test case for:

- a. The TRUE and FALSE cases of every specified EXCEPTION condition.
- b. Both $x=TRUE$ and $x=FALSE$ conditions of every specified IF (x) THEN . . . ELSE.
- c. Every specified $(x)=>Q$.
- d. Every possible type of x in every specified TYPE-CASE (x) OF . . .

Testing of a function is based on the proven formal design specification of that function. Test cases are automatically generated from the formal specifications and are then subject to an inspection process similar to those used for inspection of design and of code.

IMPLEMENTATION PROOF

The final step required to complete the schema for production of proven systems is to prove that the implementation conforms to the proven specification. Hoare has shown^{25,26} how such proofs may be constructed. In practice, these proof techniques have been successfully applied to isolated algorithms (e.g. Reference 27) and have led to spectacular insights about program construction.²⁸ However, there have not been any implementation proofs of operating systems.

All the necessary methodological preparations for a complete implementation proof of KSOS are being made. There exist, of course, formal specifications. The implementation language was chosen to allow the formulation of proof rules. A theorem prover (the same one which is used for the design proof) is at hand. The KSOS contract calls for axiomatization of the KSOS implementation language and for creation of the necessary verification condition generator. However, it requires only "illustrative" code proofs, i.e. only portions of the implementation will be proved.

These proofs will not be sufficient to complete P2, proof that the implementation conforms to the design. Nonetheless, they will serve a very important function. They will illuminate the state-of-the-art in automatic program proof, providing not only experience in the necessary techniques but also quantitative data as to the tractability and economics of proving large programs. There is no doubt that an estimate of the effort required to perform a complete implementation proof of KSOS will be made, based upon data derived from the illustrative proofs.

SUMMARY

The KSOS project is extremely significant in the field of program development methodology. It makes initial industrial use of a number of techniques which were previously used only in academic and research environments. In particular, it is novel in its large-scale use of formal specification, automatic theorem generation, language axiomatization

and automatic theorem proof. These new techniques have been successfully integrated with more traditional ones such as programming teams, inspection and review. Careful utilization of these combined methodologies allows construction of a rigorous proof that the KSOS system meets its stringent security requirements. More generally, the methodology mix and the experience gained in applying them to KSOS open the way for routine construction of computer systems whose vital properties can be convincingly proven concurrently with the development of the system.

REFERENCES

1. "KSOS System Specification (Type A)," WDL-TR7808 Revision 1, Ford Aerospace and Communications Corp., Palo Alto, CA, July 1978.
2. "KSOS Security Kernel Development Specification (Type B5)," WDL-TR7932, Ford Aerospace and Communications Corp., Palo Alto, CA, September 1978.
3. "KSOS Non-Kernel Security Related Software Development Specification (Type B5)," WDL-TR7934, Ford Aerospace and Communications Corp., Palo Alto, CA, September 1978.
4. "KSOS UNIX Emulator Development Specification (Type B5)," WDL-TR7933, Ford Aerospace and Communications Corp., Palo Alto, CA, September 1978.
5. McCauley, E. J., "KSOS: Design of a Secure Operating System," NCC, New York, NY, June 1979.
6. Ritchie, D. M., and K. Thompson, "The UNIX Timesharing System," *Communications of the ACM*, Vol. 17, No. 5, May 1974, pp. 365-375.
7. Berson, T. A., "Type Extension in KSOS," WDL-TR7967, Ford Aerospace and Communications Corp., Palo Alto, CA, January 1979.
8. Executive Order 11652, Office of the President, Washington, DC.
9. DOD Directive 5200.1-R, Department of Defense, Washington, DC.
10. Bell, D. E., and L. J. LaPadula, "Secure Computer Systems," ESD-TR-73-278, Vols. I-III, MITRE Corp., Bedford, MA, November 1973-June 1974.
11. Millen, J. K., "Security Kernel Validation in Practice," *Communications of the ACM*, Vol. 19, No. 5, May 1976, pp. 243-250.
12. Walter, K. G., W. F. Ogden, J. M. Gilligan, D. D. Schaeffer, S. I. Schaeen and D. G. Shumway, "Initial Structured Specifications for an Uncompromisable Computer System," Case Western Reserve University, Cleveland, OH.
13. "KSOS Verification Plan," WDL-TR7809, Ford Aerospace and Communications Corp., Palo Alto, CA, March 1978.
14. Neumann, P. G., R. S. Boyer, R. J. Feiertag, K. N. Levitt and L. Robinson, "A Provably Secure Operating System: The System, Its Applications, and Proofs," SRI International, Menlo Park, CA, February 1977.
15. Robinson, L., and K. N. Levitt, "Proof Techniques for Hierarchically Structured Programs," *Communications of the ACM*, Vol. 20, No. 4, April 1977.
16. Robinson, L., K. N. Levitt, P. G. Neumann and A. R. Saxena, "A Formal Methodology for the Design of Operating System Software," in *Current Trends in Programming Methodology*, R. T. Yeh (ed.), Vol. 1, Prentice-Hall, Englewood Cliffs, NJ, April 1977.
17. Parnas, D. L., "A Technique for Software Module Specification with Examples," *Communications of the ACM*, Vol. 15, No. 5, May 1972, pp. 330-336.
18. Roubine, O., and L. Robinson, *Special Reference Manual*, 3rd. ed., CSG-45, SRI International, Menlo Park, CA, January 1977.
19. Berson, T. A., "Elements of Formal Specification Style," WDL-TR7968, Ford Aerospace and Communications Corp., Palo Alto, CA, May 1969.
20. Boyer, R. S., and J. Moore, *A Computational Logic*, to appear in the *ACM Monograph Series*. Academic Press, 1979.
21. Ivie, E. L., "The Programmer's Workbench—A Machine for Software Development," *Communications of the ACM*, Vol. 17, No. 5, October 1977, pp. 746-753.
22. Dolotta, T. A., R. C. Haight and J. R. Mashey, "The Programmer's

-
- Workbench," *Bell System Technical Journal*, Vol. 57, No. 6, Part 2, July-August 1978, pp. 2177-2200.
23. Rochkind, M. J., "The Source Code Control System," *IEEE Transactions on Software Engineering*, Vol. 1, No. 4, December 1975, pp. 364-370.
 24. Fagan, M. E., "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 182-211.
 25. Hoare, C. A. R., "An Axiomatic Basis for Computer Programming," *Communications of the ACM*, Vol. 12, No. 10, October 1969, pp. 576-583.
 26. Hoare, C. A. R., "Proof of Correctness of Data Representations," *Acta Informatica*, Vol. 1, 1972, pp. 271-281.
 27. Gries, D., "The Schorr-Waite Graph Marking Algorithm," Dept. of Computer Science, Cornell University, Ithaca, NY, 1977. To appear in *Acta Informatica*.
 28. Dijkstra, E. W., *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

KSOS—Computer network applications

by M. A. PADLIPSKY, K. J. BIBA and R. B. NEELY

*Ford Aerospace and Communications Corporation
Palo Alto, California*

INTRODUCTION

Background

The need for multilevel security in computer systems has become well known. In the military, lacking such systems makes costs higher than they should be because of the need either to replicate facilities or perform "color changes" (shutting down and purging systems between uses at varying levels) in order to deny less-cleared users access to highly-classified information, and desirable functions which would require the controlled intermixing of data at different security levels are simply not yet done. The Government's concern with such matters is amply reflected in Reference 9. Outside the military, it is clear that most if not all funds transfer systems, for example, would benefit from the other side of the security coin—that is, although the major military threat is compromise of data, the major financial threat is alteration of data. In both broad areas, a free-standing multilevel secure operating system would be a distinct asset. The Kernelized Secure Operating System is meant to be just such a system, and in these terms alone is of considerable interest. This paper, though, will address potential applications of KSOS in areas other than as a free-standing system.

Aside from use as a free-standing system,¹² it is essentially the case that all other currently-envisioned applications of KSOS will involve intercomputer network environments. Under current consideration at varying levels of intensity are the use of KSOS as a containing operating system for communications subnetwork processors ("packet switches") themselves, Network Front-Ends, Network "Front Doors" (in which case the associated Host plays a "Back-End" role), mini-Hosts (to support users at terminals), and nodes for the processing of military messages. Although only the last of these has at present been analyzed in considerable detail, all are interesting and all will be touched upon in some detail.

KSOS's use in such applications is, as the title suggests, the main theme of this paper. However, it should be obvious that the mere act of inserting a secure component into a network architecture does not mystically make the network itself secure. Therefore, another important consideration which must be addressed is that the broad issue of just how to make a network secure is a complex one, and not nec-

essarily "known" in any but the abstract sense. Not only is there little in the open computer security literature which deals with networks per se (as opposed to free-standing, single systems), but the two examples with which we are most familiar^{7,14} are by two of the present authors—and each has reservations about the other's (fortunately, the reservations concern form far more than content). Thus, as we consider the use of KSOS as containing operating system for various network components in forthcoming sections, it will be an additional concern to keep track of how these components interrelate with the other components of the assumed or actual nets in question to effect the security of the network as a whole, and in order to do so sensibly we feel constrained to begin with what might appear to be a digression from the main theme of the paper—a discussion of security issues in networking in the abstract.

Security issues in networking

The problem

As will be discussed in more detail below, the use of data communications networks in the implementation of information systems has materially increased the vulnerability of data to compromise and unauthorized modification. Substantial efforts have been made to determine the cause of these vulnerabilities and remove them. This paper will present an overview of the techniques used to ensure the security of data transiting communications networks. In a sense, the point at issue is whether a network can be said to be "secure" if (and only if?) each of its components can be said to be "secure." While the prerequisite of adequate procedural, personnel, and physical security measures is acknowledged, these issues are not addressed here. Our concern is for technical measures that can be taken, within a network, to ensure data protection in accordance with an established security policy.

Security definition

Whether for free-standing systems or for networks, information system security addresses the problem of ensuring

that protected data is accessed in a legitimate manner by authorized users. The overall security problem is generally decomposed into three related but separable concerns:

- Data security—Ensuring that protected data is not disclosed to unauthorized users.
- Data integrity—Ensuring that protected data is not modified by unauthorized users or in a proscribed manner.
- Denial of service—Ensuring that access to systems is not maliciously denied.

Current security technology has developed effective measures only for the first issue. The problems posed by data integrity and denial of service are quite difficult, with no currently accepted general solutions available. However, the means used to ensure the data security of a computer system often greatly increases the system's data integrity and service availability likelihoods. The following discussion will, though, be limited to explicit discussion of data security protection techniques.

Security threats and vulnerabilities

The design of secure systems explicitly assumes not only the presence of external users that may attempt to penetrate the system, but also the existence of malicious software embedded within the system that may be used to improperly obtain information. Information systems (again, whether free-standing or networked) are susceptible to penetration threats presented from two sources:

- External threats—Presented from "outside of" the system (e.g., by users at terminals or, network-specifically, by "wire-tappers" on the transmission medium).
- Internal threats—Presented from software resident within the system.

A second classification dimension considers the means by which information is compromised:

- Overt—Transmitted data is directly compromised (e.g., packet delivered to an unauthorized user).
- Covert—System control information is used to illicitly "leak" information (e.g., network-specifically, use of the packet transmission rate as a carrier to be modulated with "leaked" information).

Ample evidence is available as to the ease in which free-standing systems designed without security in mind can be penetrated by external means.⁴ Further evidence is available as to the ease with which trusted individuals (particularly software personnel), if subverted, can embed "Trojan Horses" within software systems permitting exploitation of

internal threats. Many current systems permit mechanisms that can be exploited by Trojan Horse software to "leak" compromised information at terminal speed (100-1000 b/s) and several examples can be shown to permit even higher bandwidth.

Although the state of security kernel technology lends credence to the belief that multilevel secure free-standing operating systems are near at hand, the computer security literature does contain one interesting open problem which bears heavily on computer *network* security. This problem is referred to as the "Confinement Problem." Identified by Lampson,¹⁰ the Confinement Problem deals with the possibility that unverified code (say, a compiler) invoked by a highly-cleared user might, unknown to the user, have been maliciously programmed to release to less cleared individuals classified information which it acquires while executing with the first user's access privileges. The means of releasing the information are called "channels," which fall into three classes. The first class, "storage channels," consists of files, registers, memory locations and the like. The second class, "legitimate channels," is taken to cover billing information and the like. The final, most pernicious class, "covert channels," consists of modulating shared resources such as processor usage or paging rate in a manner observable by some other, less cleared user. As Lipner observes,¹¹ proper application of the fairly well known security kernel theoretic " *-property " can deal with the first two classes, but covert channels remain a difficult problem.

Of course, it is possible to argue that for complete security, *all* code must be verified, but this is precisely the exercise which the kernel approach is intended to eliminate. We assume it will not be the case that entire operating systems will be verified for the foreseeable future (until and unless such verification can be highly automated), and inquire instead into other means of dealing with covert channels. Here we become involved with policy considerations, for the examples cited in the literature tend to deal with rather slow channels and in some cases conscious decisions have been made not to deal with explicit blocking of covert channels in order to avoid the attendant inefficiency. Networks, however, can entail such high speeds that it is at least plausible to assume that steps must be taken to prevent them from being used as covert channels. (After all, the Host's interface to a local terminal is currently usually no faster than 1200 bits per second and it "knows" where the terminal is physically, whereas the interface even to an ARPANET packet switch is in the 100-300,000 bps range and the bits are going into another computer which may itself contain malicious software.)

As discussed in more detail elsewhere,¹⁵ several covert channels may be identified that could permit hostile programs in network Hosts to communicate at military teletype speeds with hostile programs in CSNPs even if the contents of actual messages are protected by encryption and the CSNPs are kernelized. Fundamentally, the channels modulate the observable parameters of transmissions (address, length, and timing) to communicate information despite the protection of the data portions of the transmissions.

Security countermeasures

To effectively prevent exploitation of these threats a threefold approach is required:

- Determine the security policy to be provided and enforced by the network.
- Provide the mechanisms to enforce this policy.
- Ensure that the mechanisms operate reliably and cannot themselves be subverted.

Secure systems technology

The technology for the development of secure systems is based upon three components (the first two apply to both single systems and networks, the last is network specific):

- An effective software development methodology, thoroughly based on the constructive approach to software reliability.⁶
- A system architecture that completely identifies the software and hardware responsible for security, protects it from tampering, and ensures that it provides a compromise-proof execution environment for all other software (e.g., Reference 12).
- Prudent use of encryption for the protection of data during transmission through an insecure medium.

Network security

A secure network can be considered to be composed of two elements:

- Hosts or subscribers.
- A data communications subnet.

The most common architecture for the data communications subnet is currently a network of communications switches, routing packets of data from sending to receiving Hosts via point-to-point communications lines. However, the security requirements for the communications subnet are not limited to packet switched architectures. Other subnet architectures having similar requirements include those using wideband satellite channels as a communications backbone and local networks based on communications busses or radio.

We assert without further argument that a network (Host and subnet) can be considered secure if:

- Multilevel network Hosts properly protect data while the data is resident within the Host and properly label data with its classification when submitted to the communications subnet for transmission to another Host.
- The communications subnet restricts unilevel Hosts to receive and transmit only data labelled with the classification each is permitted to process.
- The communications subnet maintains the integrity of transmitted data, particularly its classification label.

- The Confinement Problem is suitably dealt with, either by policy decision or by means detailed below.
- Communications lines are protected from compromise or modification (usually through encryption).

ALTERNATIVE SECURE NETWORK ARCHITECTURES

One more apparent digression from the main theme seems relevant: In light of the cost of "verification" of the separate components of a network, it is tempting to consider whether a network can safely be partially verified, in the sense of verifying only the Hosts and/or their Network Front Ends (NFEs) or only the communications subnetwork processors (CSNPs), and not the other logical level. That is, can Hosts/NFEs be prevented from allowing compromise to occur, or can CSNPs be enlisted to block attempts to transfer illicitly-acquired information from compromisable Hosts, in each case without the active cooperation of the other?

Host-Only verification

By "Host-Only" we actually mean a secure network architecture wherein the CSNP level is not verified. In order to be assured that the CSNP cannot "spy" on data or deliberately misroute it, there must, of course, be an appeal to some sort of rather sophisticated encryption hardware. We note this fact here and pass on, as a discussion on encryption is well beyond the scope of this paper.

Because networks can in general consist of heterogeneous Hosts, we cannot simply assume that all Hosts will be multilevel secure. So, if only the Hosts' logical level of a network is to be verified, given that some to-be-netted Hosts cannot in principle be verified, it must be assumed that such Hosts would be front-ended by verified NFEs and that such Hosts have no access to the "outside world" except through their NFEs. (Note that from "the network's point of view" a front-ended Host is indistinguishable from a conventional Host.) The key point is that we want to be able to posit the trusted nature of the Network Control Program involved (by which we mean the network-related software which interprets the Host-Host and Host-Switch Protocols in either the Host itself or the NFE) so that we can consider whether the Confinement Problem can be dealt with by means of having the (trusted) NCP block the channels through which compromise could occur.

Taking what appears to be the strongest channel first, the use of message lengths as codes could basically be blocked by a trusted NCP's padding up to packet boundaries with null characters, which would then be encrypted and undetectable by the presumed confederate code in the CSNP. The problem with padding, of course, is that it inflicts a throughput penalty on the network in question. Also, it is not foolproof, as in networks that allow multi-packet transmissions "short" (single packet) and "long" (maximum packets) transmissions could be used as "bits." Indeed, the number of bits necessary to express that maximum number

of packets are essentially usable by a hostile transmitting program if throughput is being conserved by only rounding up to packet boundaries instead of absorbing the full penalty of rounding up to maximum length.

Blocking the timing channel is also simple in principle, but may cost even more dearly when it comes to efficiency. That is, the trusted NCP could maintain an effectively-constant rate of observable traffic by furnishing dummy traffic when it has nothing to transmit and/or by transmitting only at fixed time intervals. The impact of such measures would have to be carefully assessed for given proposed networks.

Assuming dummy traffic, the address field channel can at least be masked by the expedient of "sending" the dummy traffic to random addresses (where it would, of course, have to be detected and discarded). This tactic, too, would have an unfavorable impact on throughput. Further, close analysis by specialists would be required to determine for given circumstances just what rate of illicit communication could still be maintained through the "noisy" channel the address field would still represent.

We see, then, that the effects of attempting to block the confinement channels when the CSNP is not verified are at best conjectural. In circumstances where either the threats to the software are deemed to be acceptably low (i.e., where the Confinement Problem is dismissed as a matter of policy) or the difficulties in verifying the CSNP are unacceptably high, however, the Host-only partial verification approach would seem to be better than nothing.

CSNP-only verification

In the Host-only verification case, we did not have to look closely at the nature of the verification because the key issue was the effects of attempting to have the NCP block the confinement channels which otherwise could be used to compromise classified information. In the CSNP-only case, however, the nature of the verification is extremely important, for it has been proposed that a particular kind of CSNP verification can lead to a secure network despite the potential presence of confinement channels (cf. Reference 14).

The reasoning can be expressed as follows: Suppose a given Host contains a hostile program attempting to convey information to a confederate outside the Host. Suppose further that its CSNP contains a hostile program which will attempt to aid the Host-side program by observing the cited confinement channels and, by virtue of being within the CSNP, passing the derived information on to a human agent at some other Host (presumably an unclassified Host) on the net—a Host, that is, which would otherwise be at an inappropriate level to communicate with the first Host. If it could be guaranteed that the second act of communication can and will be prevented (i.e., that the presumed hostile program in the CSNP cannot "talk to" the agent), then it is a matter of indifference whether or not the first communication has taken place (i.e., the Host's and CSNP's hostile programs' "talking to" one another). The claim, then, is that a properly kernelized switch can indeed prevent just

such forwarding of information as is necessary to make the confinement channels work.

For Hosts without multilevel security, the CSNP must "know" the current level of operation. For Hosts with multilevel security, the Host is trusted to label each transmission to the CSNP. In both cases, the code that manages the Host-CSNP interface resides within the CSNP's kernel. Thus, any message from the Host to the CSNP will have a label representing the security level of its contents indelibly associated with it. Then, by virtue of the *-property, any untrusted code elsewhere in the CSNP which deals with a given message (the routing algorithm, say) should be debarred from writing to any destination which has an inappropriate level (the code that manages the CSNP-to-network/communication line interface also being within the kernel). So only if there were some means of beating the *-property within the CSNP could effective compromise occur. Although a purist might argue that finding such a means is not logically impossible, the overall case for basing the CSNP on a security kernel seems to be a strong one.

Interim conclusions

We see from the previous and somewhat sketchy arguments that it would be more desirable, from the viewpoint of secure intercomputer network architecture, to have verified comm subnet processors than verified Hosts/Front-Ends if only the one or the other could be obtained. Unfortunately, when we return to the main thrust of this paper and begin to consider applications for KSOS in networking, the a priori case for attempting to use it as the containing operating system for a CSNP turns out to be the weakest of the cases for networking applications of KSOS currently under consideration. However, it should be realized that it is only the somewhat esoteric Confinement Problem which militates against the desirability in the abstract of the more feasible "secure NFE" architecture.

SECURE CSNP APPLICATION

The first potential networking application we will address in some detail has already been touched upon—What of using KSOS as the containing operating system for the communications subnetwork processor (usually, but not necessarily, the "packet switch") itself? The motivation for such an application is abundantly clear, for the in-principle case for overall network security is strongest when a verified CSNP is available (with, of course, suitable encryption or physical protection of relevant transmission media).

In practice, however, the following a priori argument would have to be disproved or otherwise avoided before a KSOS-based CSNP would make sense: The most important aspect of CSNP operating systems is that they must be in some sense "minimal"; that is, as opposed to general-purpose time-sharing operating systems, CSNP operating systems must be able to display the performance characteristics of so-called "real-time operating systems" in order to

achieve the level of throughput necessary for their role in the network. KSOS, on the other hand, is specifically intended to support a general-purpose time-sharing system in its primary design role. Thus, for example, several file-system-related primitives are included in the KSOS kernel which would simply "get in the way" of real-time considerations. So there must be considerable caution exercised on the fundamental performance issue, simply because of the natures of the two sorts of beasts involved.

A nearly obvious counter-argument suggests itself, of course: Why not simply excise those portions of KSOS which are superfluous to real-time use? The answer appears to be a further question: Would doing so necessitate re-verification of the resultant system (assuming, that is, that there are no primitives missing for real-time use)? If so, then it is by no means clear that whatever "leg up" we get from the fact that a similar system has already been verified is worthwhile in comparison to the verifying of a different system which was specifically designed for real-time use. At this writing, we know of no concerted efforts to investigate these issues. However, as will be seen directly, similar questions arise in the on-going investigation of another potential KSOS application, so the situation in regard to CSNPs is not purely speculative. Further, it is an interesting theoretical problem to consider whether the verification process of original KSOS might not be arguably invariant under strict subtraction of primitives, in which case appropriate surgery could be performed "for free."

SECURE NETWORK FRONT-END APPLICATION

By "network front-end," we mean a separate system, closely coupled to a Host, which is interposed between the Host and a network (or networks) for the primary purpose of offloading as much network-related software as possible (including the generic Network Control Program at least, and process level or applications protocols as feasible) from the Host, for reasons of efficiency and/or implementation ease. We note that in the present context the attachment strategy may be either "rigid" (emulating a device or devices "known" to the Host—typically either a common device controller or specific terminal(s)) or "flexible" (interpreting a compact Host-Front End Protocol in both Host and NFE), despite the fact that in the general networking arena the flexible approach is theoretically far superior.

The motivation for a secure NFE (SNFE), then, would be the sum of the motivations for a conventional NFE (CNFE) plus whatever assurances of overall network security the SNFE might entail. Clearly, when one wishes to attach older operating systems to networks, not only is it desirable to avoid consuming limited space and time resources by implementing an "inboard" NCP, but it is almost inconceivable that such Hosts could be made secure in their own right. Thus, a secure NFE seems quite attractive in principle.

Architecturally, the crucial issue is what to do about the Confinement Problem. As we have already noted, without highly specialized crafting of the CSNPs it is possible that

malicious software in the Host could establish covert communications channels to malicious software in the CSNP. Presumably, were the CSNP so crafted as to defeat the confinement channels, there would be little point in interposing SNFEs anyway, as they would almost surely be less throughput-efficient than CNFEs. The implicit security architecture in which an SNFE makes sense, then, must be one involving Host-only partial verification, wherein all Hosts not multilevel secure themselves must be mediated by SNFEs. Of course, it also makes sense to require a SNFE if it is desired to outboard the NCP of a multi-level secure Host for either efficiency or flexibility reasons even in a security architecture wherein the CSNPs are verified. Also, as noted earlier, the cost of blocking confinement channels is at present conjectural, assuming that the channels are not ruled out of consideration as a policy matter.

If such provisos are acceptable, the question reduces to the appropriateness of KSOS as the containing operating system for an SNFE. As in the case of KSOS for a CSNP, the major concern is that the demands of the role are essentially real-time. It is the case, however, that limited experience with CNFEs suggests that these demands are less onerous in the NFE area than in the CSNP area.

SECURE NETWORK FRONT DOOR APPLICATION

By "network front door" we mean an NFE-like system with a potentially significant difference—rather than offload network software for the purpose of doing general networking, the Front Door is interposed between Host and net for the sole purpose of allowing a given application on the Host to be made available to remote users. The primary currently-envisioned application of such a "Back-End" Host would be to run a pre-existing data base management system.

The motivation for a Secure Network Front Door (SNFD) is quite apparent: It is axiomatic that "security" cannot be retrofitted to existing systems; yet forthcoming networks will make it possible for users at various security levels to access existing systems which run important applications sub-systems, particularly data base management systems (DBMSs); so if an SNFD can enable such users to access such systems with some degree of confidence that data will not be compromised in so doing, it might well be superior to doing frequent "color changes" on the relevant Hosts. It must be noted, though, that the degree of confidence is debatable. The problem is that a sufficiently penetrated Back-End Host might "lie" to its SNFD about what output is destined for which user, thus achieving direct compromise rather than the sort of indirect, Confinement Problem compromise we've been focusing on. Allowing only one user in from the net to the Host at a time might mitigate matters, but the potential difficulty remains, and, as with the broad Confinement Problem issue, must be dealt with as a policy matter.

Architecturally, the accuracy of the levels of the users must be guaranteed, either by verified CSNPs or by a model which ensures that all active access to the net is via a Host or a mini-Host with multilevel security; the Back-End Hosts

would be allowed only passive access—in the sense of being debarred from initiating logical connections “to” the net. Between the passive role of the Back-End Host (which effectively eliminates address modulation), the interactive nature of the DBMS application (which suggests transmissions will—or can—be kept short so as to minimize the likelihood of length modulation), and the consideration that no user-written code would need to be invoked in the Back-End, it can be argued that the SNFD, while not in principle leak-proof, at least does not present a particularly high Confinement Problem risk.

As to the applicability of KSOS to the SNFD, a priori concern with performance issues is certainly far less than in the CSNP and NFE cases, for we envision a user load “through” the Front Door comparable to that for free-standing KSOS use. Further, the sort of load presented—which we take to be fairly modest interpretation of the query language traversing the logical connections in order to confirm that the user is not directing the Back-End to deal with a data base access to which is not appropriate given the level of the connection—appears to be no worse than comparable to the computational load one might anticipate in the program development environment. Indeed, unlike the CSNP and NFE applications, it is at least plausible to imagine that the standard KSOS operating system emulator might also be usable for the SNFD application, thus easing implementation even more.

SECURE NETWORK MINI-HOST APPLICATION

By “network mini-Host” (NMH), we mean a system essentially dedicated to the supporting of terminal users whose real computational work is to be performed on the full-scale Hosts attached elsewhere on the net. The NMH is a familiar concept from existing networks, particularly the ARPANET, where the terminal support (but not the packet switch) aspects of the “TIP” (Terminal IMP) are almost exactly what’s implied by the NMH.

Motivation for a conventional NMH is largely economic. It was, after all, much less expensive to sprinkle TIPs around the countryside than H6180s, PDP10s, or whatever the full-scale Host of one’s choice is. Similar considerations would of course also apply to secure networks; but here there would also be a compelling security-architectural principle available as well. For, as has already been implied, in situations where the level of the user must be guaranteed to be accurate, it is possible to fulfill the goal by forcing all active access to the net to come through a multilevel-secure point, whether full-scale Host or NMH. (This model would be sufficient, but is not, as it happens, necessary, in that given a secure comm subnet it is possible—though awkward—to reflect “color changes” of unilevel secure Hosts and to simply multiply NMHs to the extent necessary to allow them to be unilevel too. On the other hand, for environments in which it is not feasible to have a secure subnet, it would seem that control of all entry points is a very desirable attribute.)

As with the SNFD, the NMH application would appear

to place only modest demands upon KSOS, *qua* operating system. In fact, the NMH use for KSOS seems to be the most likely of them all to come about, largely because it essentially comes “for free.” Although some fine-tuning might be necessary in order to support the rather large numbers of terminals typically associated with this sort of application, at this stage it seems likely that simply not making compilers available on NMH versions of KSOS would liberate enough capacity for doing terminal support comparable to that done on similar contemporary systems. (Given the full range of utility software, a “midi-Host” could also be envisioned.) In terms of additional programming effort implied, if the Host-Host Protocol which “comes with” KSOS is applicable, only relatively straightforward process-level protocols need be done—and at least a primitive virtual terminal protocol will be available at some level.

MILITARY MESSAGE PROCESSING SYSTEM APPLICATION

The foregoing applications were, for various reasons of space, time available, and point in time of writing, necessarily discussed in rather general terms. We conclude with an application the investigation of which has proceeded far enough that we can offer a noticeably more specific treatment.

Introduction

By “Military Message Processing System” (MMP), we mean an essentially free-standing system which serves as a node on a network of, at least, many such systems (and potentially on a general-purpose network), in order to process prescribed types of text messages in prescribed fashions. Thus, the network security architectural points already raised come into play here as well. For example, provided that they can be constrained to communicate only with one another, secure MMPs would be usable on unverified subnets (subject to Confinement Problem considerations), and, given verified subnets would be still more economical by virtue of being multilevel secure—in that they would obviate the necessity of unilevel replication. (We take the motivation for MMPs to be self-evident.)

Under Navy sponsorship, considerable research has gone into Military Message Processing, under the collective heading of “the Military Message Experiment.”^{2,8,16,17,1} Military Message Processing systems are fully automated, or “third-stage” (cf. Reference 3, Table 2), message-handling systems. This means that a MMP system must be capable of complex message composition, distribution, selection, and analysis. To perform such tasks, any MMP implementation must rely on fully sophisticated and complete computer support. We believe that the KSOS Kernel provides such support in a way well suited to MMP-style communications. Without question, a MMP system could be implemented using KSOS itself as the underlying operating system (con-

sider the Mitre implementation under Tenex¹). Indeed, all the required functionality, without the extra (not desired) functions and structures particular to the UNIX* time-sharing system, are available from the Kernel alone.

MMP requirements and their satisfaction

Using several documents generated by the MMP community, we have compiled a number of either mandatory or desirable properties of MMP systems. Such properties are now presented within the classifications *security*, *functionality*, and *performance*.

Security

Several documents (Reference 13, p. 6, Reference 1, p. 1, Reference 17, pp. 2, 5) establish standard DoD security policy as the basis for security controls within MMP. Such policy includes the classifications UNCLASSIFIED, CONFIDENTIAL, SECRET and TOP SECRET; and the concept of information compartments such as NATO or NOFORN. Each entity within the system is considered to be at a particular security level, which encompasses one classification and a set of compartments. The security policy prohibits flow of information from a higher classification to a lower classification, or from one compartment into another. In addition to security controls, MMP systems must provide discretionary controls to implement privacy measures (Reference 13, p. 6, Reference 17, p. 2, Reference 16, p. 17). These are used to control access to information by individuals and groups.

The security area is clearly satisfied by KSOS. Not only is the DoD security policy deeply embedded in the KSOS Kernel, but also the Kernel will be verified mathematically, using proof tools developed in part by Stanford Research Institute (cf. Reference 6), to adhere to the policy. In addition, the Kernel provides UNIX-style discretionary access controls on all types of objects.

Functionality

Functionality required for MMP systems is discussed in this section; where appropriate, the Kernel's satisfaction of the requirements will be indicated. First, the more primitive support functions of a system are discussed. Then the user level functions are discussed, along with their implications for system functionality.

- **Direct—Support functions**

Support of processes. The need for large numbers of processes (on the order of tens of processes per user), and for

flexible handling of them, is emphasized in several references (Reference 1, p. 5, Reference 17, p. 19, Reference 2, p. 20). Some references indicate that one or two processes are necessary for each security level classification convoluted with exact set of compartments, (e.g., Reference 17, p. 19). Specifically, the Kernel must be able to:

- a. Support many processes
- b. Allow fast process creation and deletion
- c. Switch between processes with little overhead (most processes do small amounts of work at a time)

While it is difficult to predict the level of performance of the Kernel in these aspects of process handling, there is good reason to expect it will do well. The ability to provide a satisfactory implementation of UNIX would imply at least the first two, and possibly the third. Since it is the Kernel, and not KSOS, that is under discussion, one can conceive of small special-purpose limited-context processes with characteristics suited to a MMP system, but not available under UNIX.

Corresponding to the ways messages are traditionally handled (Reference 8, p. 12, Reference 2, p. 21), inter-process communication must be available in both "message" mode and "interrupt" mode. Message mode communication places a message in a queue where it stays until explicitly read by the receiver. Interrupt mode communication preempts the receiver's execution so that the message is handled immediately. Both styles of inter-process communication exist in the Kernel as currently conceived.

Support of files and devices. Ames (p. 5) indicates that size of information objects should depend on the application itself. Ideally, then, the underlying system ought to support efficiently objects of greatly varying size. In particular, very small objects on secondary storage must be reasonable to use. The flexibility of the KSOS kernel provides for such objects.

The actual structure of a file system in a MMP system is likely to be quite different from what conventional operating systems provide. Thus, another advantage of the Kernel is its provision of a fast, simple underlying flat file system out of which an appropriate structured file system can be built.

The major security issue in device handling is providing a secure user interface, or secure terminal handling. A significant choice to be made in this regard (and regarding all devices) is whether a terminal is at a single security level at any time, or whether it may be used as a multilevel device. Also, the security level may be established externally, via established lines to limited-access terminals; or internally, via user identification. The Kernel allows single-level terminals with level established via user identification. Other important device uses are hardcopy output (line printers, etc.), and archiving and disk save and restore (magtape). Complete (level-multiplexed) handling is provided for such uses by the Kernel, as well as for secure terminal handling.

* UNIX is a trademark of the Bell System.

- Indirect—User-level functions.

The remarks of this system indicate briefly the nature of the user interface of a MMP system. It is not possible to show point by point that the Kernel can support the given requirement. However, after thorough examination of the Kernel design while considering these requirements, we are convinced that none of them needs any capability not found in the Kernel.

Message attributes. Messages must be divided into sections (header, address, subject, text, comments), each of which possesses an independent security level (Reference 1, p. 3, Reference 17, p. 7, Reference 8, p. 11). A message also must possess a priority, or degree of urgency, that affects its handling.

Sending. Sophisticated message composition facilities are provided. External documents (files, other messages) may be referenced and included in a message without losing context. Special handling of composition is provided for responding to received messages, including direct reply and forwarding. During the composition of a message (any time until released by the composer), the classification of the message may be changed (including being downgraded). Cf. Reference 13, p. 2, Reference 17, p. 13, and Reference 8, p. 36.

Receiving. Received messages may be displayed and printed in various ways. Messages may be moved into user files. Messages to be examined may be selected in various ways, including order of receipt and by content. Cf. Reference 13, pp. 3, 4; Reference 17, p. 12; and Reference 8, pp. 12, 15, 20.

Administration. Terminals and users are identified to assure proper clearance. A specified minimum of on-line storage for messages must be supplied. An archive facility for messages exceeding on-line storage is provided. Unusual or critical events are logged. Cf. Reference 17, p. 15, Reference 13, p. 6.

Performance

We use "performance" to mean how well the user perceives that the system does its job. In other words, given the system possesses the required functionality at all levels, does the user get comfortably fast responses, is the system's interaction with him generally comprehensible. In the user environment in which MMP systems must exist, high performance in this sense is absolutely necessary. The users do not come from computer science backgrounds, and are not likely to be tolerant of long waits, low reliability or a large amount of training in order to use the system. Of course, until KSOS is actually subjected to the test of real usage by real users, we can only conjecture about its performance in this sense, but we know of no reason to suppose it will be other than acceptable.

Cost/Benefit summary

A major cause of inefficiency in MMP systems up to now has been the mismatch between the external requirements

and the underlying support facilities. We have shown informally that the KSOS Kernel is well matched to the needs of MMP, providing in an efficient, flexible manner the essential security, functionality, and performance. The Kernel is obviously not a MMP system in itself, but the effort to develop such a system using the Kernel is likely to be less than that experienced in other instances, also because of the suitability of the Kernel for this application. Little or no modification of the Kernel (almost surely none requiring reverification) will be required to enable its use for this application.

CONCLUSIONS AND FUTURE DIRECTIONS

In conclusion, we observe that, based on the foregoing, there are several networking applications in which KSOS may be expected to play a role, either in whole or in part. (That is, either all of KSOS, Kernel, trusted processes, and Emulator, may be employed, or just the Kernel and, perhaps, the trusted network software). Investigations are actively under way in the Front End and Front Door areas, and the Military Message area has already been investigated to a sufficient extent to engender reasonable confidence in KSOS's utility. However, it is fair to say that none of the currently-considered application areas is at present a "sure thing." Therefore, it would seem that the most important future directions for KSOS networking applications to take lie in the following realms: the issue of removing superfluous primitives must be resolved; other efficiency areas must be investigated; closer scrutiny must be given to the overall network security implications of interposing verified components at various points in specific network architectures; and, although it was not touched upon in the body of the paper, we would be remiss not to mention the overriding open issue of secure networking—just what are the right sorts of "secure" higher-level protocols to implement in the secure components which are becoming available?

REFERENCES

1. Ames, "User Interface Multilevel Security Issues in a Transaction-oriented Data Base Management System," Mitre Corp. MTP-178, January 1977.
2. Ames and Oestreicher, "Design of a Message Processing System for a Multilevel Secure Environment," (exact citation not available; distributed internally).
3. "Automated Message Handling" (exact citation not available; distributed internally).
4. Anderson, J. P., "Computer Security Technology Planning Study," ESD-TR-7357, James P. Anderson and Company, Fort Washington, Pa., October 1972.
5. Bell, D. E., and L. J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," ESD-TR-75-306, The MITRE Corp., Bedford, MA., July 1975.
6. Berson, T. A., et al., "KSOS: Development Methodology for a Secure Operating System," to be published at the 1979 National Computer Conference.
7. Biba, K. J., "Multilevel Secure Network Architecture," 1978 National Telecommunications Conference, December 1978.
8. Goodwin, Mitchell and Tasker, "Concept of Operations for Message Handling at CINCPAC," Mitre Corp. MTR-3323, October 1976.
9. Herrmann, R. J., "DoD Requirements for the 80's," *Proceedings of the NOVA AFCEA Computer and Communications Security Conference*, April 1978.

-
10. Lampson, B. W., "A Note on the Confinement Problem," *Comm. ACM*, Vol. 16, No. 10, October 1973, pp. 613-615.
 11. Lipner, S. B., "A Comment on the Confinement Problem," MTP-167, The MITRE Corp., Bedford, Mass., November 1975.
 12. McCauley, E. J., et al., "KSOS: A Secure Minicomputer Operating System," WDL-TR7811, Ford Aerospace and Communications Corporation, Palo Alto, CA., August 1978, to be presented at the 1979 National Computer Conference.
 13. "Military Message Experiment Selection Criteria" (exact citation not available; distributed internally).
 14. Padlipsky, M. A., "An Architecture for Secure Packet-Switched Networks," *Proceedings of the Third Berkeley Workshop of Distributed Data Management and Computer Networks*, August 1978.
 15. Padlipsky, M. A., D. W. Snow and P. A. Karger, "Limitations of End-to-End Encryption in Secure Computer Networks," MTR-3592, The MITRE Corp., Bedford, MA, May 1978.
 16. Stotz, R., et al., "Information Automation," *USC/ISI 1977 Annual Technical Report*.
 17. Tangney, Ames and Burke, "Security Evaluation Criteria for MMP Message Service Selection," Mitre Corp. MTR-3433, June 1977.

Considerations in the employment of blind computer professionals

by JAMES A. KUTSCH, JR. and KIMBERLY B. KUTSCH

West Virginia University
Morgantown, West Virginia

INTRODUCTION

According to sections 503 and 504 of the Rehabilitation Act of 1973, employers are required to hire qualified handicapped persons and, where necessary, to provide reasonable accommodations relating to accessibility to their actual job station and to work environs—including rest rooms, recreation areas, eating facilities, etc. Further, companies holding federal contracts are required to have an Affirmative Action Program to seek out qualified handicapped individuals as potential employees.

When a person is handicapped, there is a tendency for this potentially productive person to be excluded from the work force. Yet, given opportunity and ambition, these people rise just as far as they would under more 'normal' circumstances. Why, then, are they discriminated against? Perhaps one reason is lack of understanding of the considerations necessary when employing the handicapped.

In this paper, we will deal with the problems encountered by the blind in the field of computer science.

THE BLIND PROGRAMMER

It is assumed that we will be dealing with a singularly disabled, i.e. blind, reasonably intelligent and trained computer professional. The most immediate problem he or she will have is in handling the output from the computer. Obviously, the input to the computer is no problem, since key punches and terminals are keyboard devices. There is only a slight inconvenience involved in learning the location of the special symbols, as each device differs somewhat. This, however, is also an inconvenience to the sighted programmer.

Historically, there are many solutions to the problem of output.¹⁻³ The simplest is to have someone read the printed material. The drawback, of course, is that this ties up two people to do one job. This is, however, an adequate solution if two programmers are working side-by-side on the same problem, or if the blind programmer is acting in a consultant's capacity for a limited time.

Braille, in which any blind computer professional should

be highly skilled, is another common solution. The biggest advantage is that it allows the programmer independence. Further, several vendors produce a Braille print train, which, when installed in a standard printer, will imprint a series of raised dots on the paper. Terminal devices with Braille output are also available.⁴

Braille can also be produced on a conventional printer, using a software conversion program which converts print characters to a series of periods and spaces. These are printed on a printer which has a soft cushion of some sort (usually an elastic band) placed behind the paper. Thus, the periods, when printed, make dents in the paper, which when read from the reverse side, appear as raised Braille characters. The Braille characters are a matrix of dots, two wide and three high. Counting horizontal and vertical spacing between characters, 40 Braille characters would be 120 printer positions wide and four lines high when printed using this method. The disadvantages with Braille output are bulk of listings and expense of hardware print mechanisms (if that approach is taken). Further, stock printer paper is not as heavy as normal Braille paper and the Braille dots have a tendency to be flattened back into the paper, especially in very thick and heavy listings.

Morse code and musical chords have also been examined for their usefulness with varying degrees of success. If the programmer already knows Morse code and has been using it for output, it can be very useful. It requires no additional equipment as software drives the computer's alarm bell. Learning so complex a skill in order to handle output, however, is an unfair request to put on an employee. Musical chords representing letters have even less promise as a solution and in effect have never really gotten off the drawing board.

A major contribution to the problem of reading printed material was made by Telesensory Systems Inc. (TSI) when they began producing the 'OPTICON' (Optical to Tactile Converter).^{5,6} With this system, a small camera is moved across a line of print. A tactile image of each character so scanned is presented to the reader's finger through a matrix of vibrating reeds. The user of the OPTICON can, without assistance, read a printed page. However, it takes considerable training and a high level of dedication to learn and become proficient with the device.

READING MACHINES

The author's own research has led to the development of a talking computer terminal. It consists of a keyboard for input, a speech synthesizer for output presentation, and a microprocessor to drive the synthesizer.⁷ The microprocessor communicates with the host machine via an RS-232 interface. The terminal can be used on any system in place of the conventional hard copy device or CRT. Input from the keyboard is sent through the RS-232 interface to the host machine. Output from the host machine, which would normally appear on the screen or paper, is processed by the software in the microprocessor and sent to the synthesizer producing spoken English output. The device can be used as a terminal on any computer system which supports asynchronous RS-232 communications.

Through one or all of the above means, the blind computer programmer can conquer the problem of computer output. It is necessary, for the employer, to discuss what methods each particular programmer utilizes and set a clear understanding of what, if any, equipment this individual will need.

OTHER CONSIDERATIONS

A friendly and accessible secretarial staff is a large asset to a blind employee. This staff can provide valuable services by reading mail and memos, proofreading correspondence, etc. The few minutes of time necessary to glance at a memo or read the mail can increase the productivity of the programmer immensely. In an environment devoid of secretaries, a receptionist or keypunch operator could be asked to fill this need.

Another important consideration is the type of environment the blind individual will be working in. Team programming and group efforts are ideal atmospheres for very productive work. There is always someone around with whom the project can be discussed.

Finally, there is the problem of orientation. It will be necessary for someone to spend some time helping the new employee learn the location of equipment and rooms. This should take no more than a few days. Once oriented, a blind person can move about safely, freely and independently. There may be special problems pertaining to each individual. For instance, a Seeing Eye dog may accompany his master. In this case, it is necessary to locate an out-of-the-way area for the dog to be walked and to acquaint the rest of the staff with the animal. These dogs are extremely well trained and should not present any problems to fellow workers.

If the office is located in a building or complex with other businesses, it is helpful to be acquainted with the general location of these other businesses. This should require only a few minutes of additional time.

CONCLUSION

Usually, blind computer professionals will have educational backgrounds similar to that of sighted computer professionals. However, some may have attended special schools for the blind which offer training in data processing. While a blind individual may come from a slightly different background and may require additional tools for his trade, he should be given the same opportunities and trial period as a sighted employee. Thus, he can prove his merit (or lack of it) based on his abilities and performance regardless of blindness. He can be as valuable and productive an employee as any of his fellow staff members; therefore the criterion for hiring, promoting and firing should be the same for all.

It is presumptuous to try to offer a "guideline" for hiring the blind. Each individual and each job require a different set of considerations. As an employer, do not hesitate to discuss how the new employee would be expected to function, i.e., how he might handle a batch environment compared with an interactive system.

Housing conditions in the area also bear mention. The blind employee will want to live within walking distance, on a bus or train route, or where fellow workers car pool to work without sacrificing desirability of neighborhood or school system which are important to his family. It is not necessary to be hesitant to discuss cause of blindness. Frankness and openness during the initial interview can eliminate misunderstanding or uneasiness later.

The biggest problem facing a handicapped person who tries to enter the professional work force is not his education, training, or ability but rather the education and awareness of the rest of the world in accepting the handicapped individual and realizing his potential.

REFERENCES

1. Rahima, M. A., and J. B. Eulenberg, "A Computing Environment for the Blind," *Proc. AFIPS Natl. Comp. Conf.*, Vol. 43, 1974, pp. 121-124.
2. Rahima, M. A., and J. B. Eulenberg, "Modes of Information for the Blind Programmer," *Proc. Assoc. for Comp. Mach. Annual Conf.*, 1974.
3. Cooper, F. S., J. H. Gaitenby, I. G. Mattingly and N. Umeda, "Reading Aids for the Blind: A Special Case of Machine to Man Communications," *IEEE Trans. Audio and Electroacoustics*, Vol. AU-17, No. 4, December 1969, pp. 266-269.
4. Loeber, N. C., "Proposed Braille Computer Terminal Offers Expanded World to the Blind," *Proc. AFIPS Fall Joint Comp. Conf.*, Vol. 39, 1971, pp. 79-87.
5. Bliss, J. C., M. H. Catcher, C. H. Rogers and R. P. Shepard, "Optical-To-Tactile Image Conversion for the Blind," *IEEE Trans. Man-Mach. Syst.*, Vol. MMS-11, March 1970, pp. 58-65.
6. Bliss, J. C., "A Relatively High-Resolution Reading Aid for the Blind," *IEEE Trans. Man-Mach. Syst.*, Vol. MMS-10, March 1969, pp. 1-9.
7. Kutsch, J. A., Jr., "A Talking Computer Terminal," *Proc. AFIPS Natl. Comp. Conf.*, Vol. 46, 1977, pp. 357-362.

Hiring a deaf computer professional*

by KAREN K. ANDERSON

Rochester Institute of Technology
Rochester, New York

and

PHILIP W. BRAVIN

IBM Corporation
New York, New York

INTRODUCTION

The computer industry has a critical need for skilled professionals. Employers are already hard-pressed to find qualified workers and look with interest to the growing population of technically trained deaf individuals as a new source of competent employees. Interest in hiring deaf computer professionals has also been accelerated by employers' growing sense of social responsibility and their need to comply with government regulations concerning affirmative action in hiring and promoting handicapped individuals. This willingness to employ deaf professionals must be accompanied, however, by specific guidelines for the actual hiring and accommodation of such workers. This paper addresses the practical concerns of managers who wish to hire deaf professionals into their organization.

CHARACTERISTICS OF THE DEAF POPULATION

There are 13 million people in America who have some type of hearing impairment.¹ About 1.8 million of these are considered "deaf," meaning that their hearing is not functional for the normal purposes of life. Since deafness is an invisible handicap, however, this group and its characteristics are relatively unknown to the general population.

Misleading terms like "deaf and dumb" and "deaf-mute" have reinforced common misunderstandings of the deaf population's intelligence and communication skills. In fact, IQ scores of the deaf follow the same normal distribution as the hearing population. Deafness does not, therefore, imply a lack of native intelligence. Neither does deafness imply the inability to vocalize. Most deaf individuals find that their speech sounds promote successful communication, especially among people who know them; but some individuals

prefer not to use their voices because their speech sounds have not proved helpful in communicating with others.

Most hearing people overestimate the ease with which a deaf person can lipread. Actually, lipreading is a difficult skill to master. Only 26 percent of all speech is visible on the lips. Even the best lipreader cannot lipread everything that is said. Familiarity of the speaker is again a significant factor in successful lipreading. The challenge of individual speech styles and facial characteristics such as flowing mustaches can be met with practice in repeated meetings.

One of the most significant results of deafness is unknown to the general hearing population. Specifically, early onset of deafness can seriously hinder language development. Although there are some deaf individuals who can read and write very fluently, others have limited reading and writing skills that misrepresent their intelligence and understanding. The facile assumption that a deaf individual's hearing and speech impairments can be easily overcome by reading and writing ignores the relationship between hearing and language development and leads many people to underestimate an individual's knowledge and potential.

"Deafness" is a generic term, since the word describes a handicapping condition with many variables such as degree of hearing loss, listening skills and lipreading and speech abilities. Other factors such as personality, education and intelligence have nothing to do with deafness *per se*. Although there is value in knowing the potential impact of deafness, it is vital to recognize the uniqueness of each deaf individual.

RECRUITING SOURCES

There are specialized institutes within the United States that are dedicated specifically to the education of the deaf. These institutes exist because the education of deaf individuals is a challenging task. On the average the educational level of the deaf population is well below that of the general population. Only 12 percent of the population seek post-secondary education, and only six percent receive baccalau-

* The techniques suggested in this paper reflect the collective experience of the authors and the community of deaf professionals but in no way represent policy of either author's employer.

reate degrees. Since most of the college-educated deaf now graduate from specialized schools, these schools are significant recruiting sources.

Currently there are several post-secondary schools with significant histories of educating deaf individuals. The oldest of these schools is Gallaudet College in Washington, D.C. Founded in 1864, Gallaudet is a liberal arts college for deaf students only. There is no computer science major *per se*, but degrees are granted in computational mathematics and in business with an emphasis on data processing.²

In 1969 the National Technical Institute for the Deaf (NTID) was established on the campus of the Rochester Institute of Technology (RIT) in Rochester, New York. As a college of RIT, NTID offers certificates, diplomas and associates degrees in data processing. In addition, NTID supports deaf students pursuing bachelor's degrees in RIT's School of Computer Science.^{3,4}

Utah State University, New York University and California State University at Northridge have growing programs for the deaf in computer science.⁵ In addition, special educational services for the deaf are becoming available throughout the country in response to Sections 503 and 504 of the Rehabilitation Act of 1973.⁶ As access to programs opens, the number of qualified deaf graduates grows.

Employers can recruit these graduates through the placement offices of the degree-granting institutions. Deaf baccalaureate degree students at RIT, for example, are encouraged to follow the standard procedures of RIT's Central Placement Office. Colleges offering special programs for deaf individuals, however, usually augment placement services with additional services tailored specifically to their needs. These services can include special assistance during the application/interviewing process, orientation activities for the managers and co-workers of a new hire and troubleshooting consultation during the first weeks of employment. NTID and Gallaudet in particular provide a wealth of materials and personnel to assist employers in hiring a deaf professional.

Justifying the commitment required to hire a hearing-impaired person on a full-time, permanent basis can sometimes seem an overwhelming obstacle to employers. This impediment is most often an artifact of the imagined needs and problems of a deaf individual rather than accurate information about such a person. Fortunately, the desire of the employer to minimize risk with a new hire is complemented by the desire of educators of the deaf to maximize the experiential learning component of formal education. Thus, most special programs for the deaf either require or encourage work experiences as part of their programs. A summer or one or more cooperative work periods provide an excellent opportunity for employers to gauge the impact of hiring a deaf individual. The spectrum of support services supplied for permanent placement is generally available for co-op placement, also, and can help make even such short term commitments truly enriching experiences.

INTERVIEWING AND FOLLOW-UP

The initial contact a deaf person makes with an interested employer can occur in a variety of forms such as replying

to a newspaper advertisement, mailing in a resume, requesting an application form, or appearing in person at an on-campus recruiting interview. None of these forms of contact guarantee that the employer will be forewarned that the applicant is deaf and there is no legal requirement to that effect. Indeed, the applicant's deafness may only be revealed in his communication attempts, whether in writing or in person. An employer learning of the applicant's handicap in this way may become surprised, confused and hesitant to carry out the interviewing and recruiting process. It is imperative, however, that the employer actively proceed with the usual routine of interviewing.

Communicating with a deaf person at the initial in-person interview is, at best, an opportunity for the employer to assess the deaf person as a whole and to observe the nature of his listening and communication abilities. A good strategy for the interviewer is to start speaking slowly and let the deaf person monitor or control the pace and mode of the interview. The deaf person may indicate his inability to lipread and request that the interviewer write everything on paper. On the other hand, the deaf person may not need to resort to pencil and paper, and the interview can take place in the usual manner (i.e. communicating orally). In fact, during the course of a day when a deaf person is interviewed by several people, the deaf person may use pencil and paper with one interviewer and communicate orally with another. The interviewers should be aware of the various communication modes that may be employed. Also, it is essential for the interviewer not to show any hesitation or displeasure when asked by the deaf person to speak slower or use pencil and paper.

Another strategy to consider is the use of a sign language interpreter at the interview. It should be emphasized that the need for such a service should be specified by the deaf person, since some deaf persons do not find it necessary to have an interpreter while others may need one. If the need for an interpreter is stated, the organization should make arrangements to obtain one through a local referral or vocational rehabilitation agency that serves deaf people. One of the reasons why interpreting services are mentioned is that there are legal considerations, such as Sections 503 and 504 of the Rehabilitation Act, which create a mandate for employers to make their facilities accessible to the handicapped in general. In the case of deaf people, communication accessibility is a consideration.

The variety in the degree of hearing and speech and other communication skills must be borne in mind; but, once the communication considerations have been dealt with, the scope and content of the interview should focus on the usual issues such as the qualifications of the prospective employee, his ability to contribute to the overall objectives of the company, his ability to get along with his peers and managers and other necessary attributes required to make him a successful employee.

Finally, if the organization wants to follow up the interview by offering the prospective employee a job, indicating to the applicant that the organization could not find a suitable position, or asking for more information, it is important that the organization make such efforts directly, in writing if possible. Direct contact or writing is preferred because

the applicant may not have direct access to a telephone. Telephone communication through a third party runs the risk of misunderstanding because of distorted or abbreviated messages.

ORIENTATION

In any new job setting the environmental barriers, whether they be social, physical, or technical, can be overcome by the orientation process. This process is simply an educational one in which the new environment is described to the new hire to make him comfortable in his new setting. In the case of a deaf new hire the orientation process is not radically different from that for a new hire who is hearing. Depending on the organization, however, the process can differ for the deaf employee's peers and his management, since they generally need some orientation in this case also. This type of orientation, in both a formal classroom and informal social setting, will be described.

The best approach to making a deaf new hire's peers and management feel comfortable and removing the mysterious aura surrounding deafness is to make them aware of deafness in general. The more they understand deafness, the more receptive they will be to the new hire and the new hire to them. This receptivity is essential because the population-at-large tends to have a stereotyping attitude toward a specific population. The process of orientation will assist in the removal of the stereotyping factor and help make the view of the person towards a new hire more individualized. In the process the new hire will blend into the work setting rather than stand out on account of his deafness.

There is a wide range of materials that organizations can use in their formal training programs for the new hire's management and coworkers. These materials can include movies, books, pamphlets and manual alphabet reference cards, to list a few. Such materials can be obtained from the colleges, institutes and organizations that serve deaf people and from appropriate personnel representatives in corporations. Some corporations may have materials they use to conduct formal training programs; others may have materials they use on an informal basis. Appendix A lists the names and addresses of several sources of orientation information and Appendix B lists the names of several corporations with experience in hiring deaf individuals.

Informal orientation is a process that is best left to the new hire, who usually takes the initiative in educating his peers about aspects of deafness that are not covered by the formal training materials. Informal training may take the form of stories and anecdotes that have a deaf theme or informal instruction in sign language or the lifestyle of a deaf person (e.g. how he "hears" the doorbell, his "telephone"). This approach usually is entertaining and goes a long way in removing the so-called environmental barriers.

The orientation process for the deaf new hire may differ only in the way information is presented, depending on the deaf person's communication skills. It should be emphasized that the deaf person should receive the full benefit of orientation programs and not be given a half-hearted treatment because of his communication limitations. The new hire's

communication skills can be inferred from the initial interview process.

As essential as it is for any organization, the orientation process may be even more important for situations involving a deaf new hire. Its value should not be minimized, for such a program implemented correctly and positively goes a long way in creating a congenial working atmosphere for the new hire, his peers and his management.

TRAINING

In any organization heavy emphasis is placed on training and education for employees, especially in the computer field with its rapidly changing technology. This training gives the employee the opportunity to keep pace with technology. At the same time the organization maintains a group of qualified and educated people to meet its changing needs.

When a student makes the transition to the business world, a great deal of practical material must be digested before the student makes himself a useful employee. To maintain his usefulness, he must avail himself of professional development activities throughout his career.

The medium through which training is presented to the employee has greater impact on a deaf employee than on a hearing employee. To give a simple example, an audiotape is as useless to a deaf person as a videotape without sound is to a blind person. In a classroom setting there are several points to consider for a deaf person such as relevancy of the subject matter, the size of the class, and the communication skills of the deaf individual. In any case the deaf individual should make his needs known. They can take the form of preferential seating, notetakers, or even a sign language interpreter. Some deaf people may find that none of these aids are necessary, and others may find some or all of them necessary.

There are an increasing number of courses offered in both audiotape and videotape forms. For a deaf person the audiotape does create significant problems. The deaf person can receive the full benefit of the tapes by having a sign language interpreter interpret the tape, by having a secretary transcribe the contents, or by getting a copy of the script that was used to prepare the tape. The latter method is most practical, and such scripts are usually available. Videotape has similar ramifications for a deaf employee because it is harder to lipread a TV screen than to lipread in person. Thus, although the employee may benefit as his peers from the visual aids that are incorporated in videotape training, he will not necessarily do much better with the "talking face" type presentation than with an audiotape. Finally, voice-over presentations are fundamentally as difficult for the deaf person to follow as audiotapes.

Other media through which courses are offered include computer-assisted instruction and programmed instruction (self-paced instruction) as well as training in the form of reading manuals. These media are ideal for deaf persons.

If an organization acknowledges the value of training programs, it should be able to guarantee that the deaf employee receives the full benefit of these programs.

MEETINGS

Meetings usually provide a forum for people of a given organization to share relevant information. Meetings take many forms—one-on-one, small groups of three to nine people, groups of 10 to 20 people, and an assembly type group of more than 20 people. The deaf person may require specific consideration in each of those settings. It should be emphasized, however, that there are no universal solutions for a deaf person in any setting. Useful strategies vary from one deaf person to another, depending on communication and listening skills as well as the group of people involved. The deaf person's problems are less acute if the group contains people who are familiar with the deaf employee.

In a one-on-one situation the communication process usually requires a simple adaptation. In some cases the person meeting with the deaf individual may know sufficient sign language, may talk at a speed appropriate for him to lipread, or may write down everything for him to read. In response the deaf person may write, speak, or even use sign language, if the person he is communicating with can read sign language.

In a small group the communication problems become somewhat more complicated since more people are involved. At this point written communication may be more difficult but can still be used. The deaf person may need to depend on one person in the group to follow the conversation, or he may be able to follow everything by lipreading alone. If the deaf person is the presenter, the environment differs and will depend greatly on his communication skills. The deaf person may have sufficient speech skills to conduct the meeting. On the other hand he may prepare a text or some notes for his co-workers to present for him. Communication in this kind of a setting is usually not a critical problem for a deaf person since the group is small enough for the deaf person to monitor.

The problem becomes more acute when the group is bigger. There are more personal relationships and communication links to take care of. If the deaf person has good lipreading skills, he may still run into problems at this type of meeting. For example, when one person stops talking, the deaf individual must find the next person who is talking. By the time he finds the speaker, that person may already be half-way through his statement. Similarly, if more than one person tries to speak at once or if someone interjects a parenthetical remark, the deaf person will not be able to follow them. If the meeting contains critical information for the deaf person, it may be necessary for the meeting to be controlled carefully. For example, it may be necessary for each person to raise his hand and be acknowledged before speaking. If the deaf person doesn't have adequate lipreading skills, he may need to depend on a notetaker or, if one is available, a sign language interpreter. In many cases a notetaker would suffice and could be a co-worker or secretary.

In an assembly-type setting the deaf person with adequate lipreading skills should be seated preferentially or, if he prefers, have a notetaker or sign language interpreter seated next to him. In any case the importance of the meeting will

determine whether a sign language interpreter is justifiable. The sign language interpreter is usually able to interpret the speaker's comments word-for-word, while a notetaker can at best give a good abbreviated summary of what has happened. Furthermore, the sign language interpreter can also reverse-interpret, i.e., repeat the deaf person's sign language statements or questions orally. In a setting of 500 people or more lipreading can be impractical, since it ceases to be effective when the speaker is more than eight to 10 feet away. In such situations a notetaker or an interpreter would be necessary.

WORK ASSIGNMENTS

It is not appropriate to attempt to identify specific work assignments that deaf employees can or cannot handle. The skills and interests of deaf employees are as varied as those of their hearing counterparts. An appropriate match between employee and task must be made by considering the training and qualifications of the individual. Managers are urged to be aware of their own stereotyping tendencies when considering work assignments. Many of their tendencies are vestiges of outmoded or changing work environments. For example, the increasing orientation toward terminals for interacting with the computer reduces the number of communication barriers between the programmer and the computer. In addition, the terminal becomes a handy device for aiding communication among deaf and hearing co-workers. The recent push for documentation produces real-time rather than after-the-fact documentation, and the introduction of efficient, graphical documentation tools such as HIPO and data flow graphs also ease communication among deaf and hearing. Deaf individuals have worked in essentially every capacity within a data processing environment. There are deaf data entry personnel, deaf computer operators, deaf programmers, deaf systems analysts, deaf project leaders and deaf managers. Sensitivity to the uniqueness of each deaf person and willingness to modify the work environment slightly to accommodate him can permit that person to function and grow in accord with his own interest and abilities.

CONCLUSION

The growing population of college-educated deaf individuals provides a new source of skilled workers to satisfy the critical need for computer professionals. There are specific, cost-justifiable techniques that can be used to overcome the communication problems that a deaf employee will face on the job. Schools, institutes and organizations serving the deaf offer materials and personnel to assist the employer in accommodating the deaf worker. Employers are urged to draw on this pool of competent professionals. They will find an increasing number of graduates who can be significant contributors to their organizations.

REFERENCES

1. Schein, Jerome D., and Marcus T. Delk, *The Deaf Population of the United States*, NAD, Washington, D.C., 1974.
2. Schuchman, John S., "Educational Opportunities in Computing—Gallaudet College," *Proceedings of the National Conference on Computing Careers for Deaf People*, ACM SIGCAPH, Arlington, VA., 1975.
3. Hurwitz, Tracy A., "Educational Opportunities in Computing—National Technical Institute for the Deaf," *Proceedings of the National Conference on Computing Careers for Deaf People*, ACM SIGCAPH, Arlington, VA., 1975.
4. Beil, D. H., and J. W. Panko, "Educational Opportunities for the Deaf in Data Processing at Rochester Institute of Technology," *ACM SIGCSE Bulletin*, Vol. 9, No. 4, December, 1977, pp. 79-84.
5. Jamison, Steven L., "Employing a Deaf Programmer," *Proceedings of ACM 78*, Washington, D.C., 1978.
6. "Nondiscrimination on Basis of Handicap," *Federal Register*, Vol. 42, No. 86, Part IV, Wednesday, May 4, 1977.

APPENDIX A

Sources of orientation information:

Alexander Graham Bell Association for the Deaf
(AGBAD)
1537 35th Street
Washington, D.C. 20007

Gallaudet College
7th and Florida Avenues, NE
Washington, D.C. 20002

National Association of the Deaf (NAD)
814 Thayer
Silver Spring, MD 20910

National Technical Institute for the Deaf (NTID)
Rochester Institute of Technology
1 Lomb Memorial Drive
Rochester, NY 14623

Registry of Interpreters for the Deaf (RID)
P.O. Box 1339
Washington, D.C. 20013

APPENDIX B

A partial list of employers of deaf computer professionals:

American Can
Anhauser-Busch
Boeing Computer Services
Bunker-Ramo
Grumman Data Systems
IBM
Kodak
Lockheed
McDonnell Douglas
Mobil Oil
Travelers Insurance
Union Oil
U.S. Dept. of Housing and Urban Development
Xerox

MIS effects on managers' task scope and satisfaction*

by DANIEL ROBEY

Florida International University
Miami, FL

The impact of computers on organizations has long been a topic of special interest to management and organization theorists. Opinions vary widely on the nature and importance of information technology's influence on the structure and process of organizations. For every ounce of technical optimism created by computer scientists, a pound of social pessimism is generated by behavioral scientists suspicious of technological impacts. Over the past 20 years much dialogue has raged in emotional and speculative tones without substantial progress made in our understanding of the issue. The armchair theorists who project less human organizational life because of the computer have generally operated without the benefit of research findings. The purpose of this paper is to shed some empirical light on one hotly debated issue: the impact of management information systems on managers' tasks and their evaluation of computer-induced changes in tasks.

Speculative arguments differ in their predictions of task impacts. Leavitt and Whisler⁷ forecasted the removal of meaningful content from middle managers' work and the resultant alienation of managers from their jobs. They suggested that need fulfillment would only be found off the job, and drew an analogy between middle management and blue collar workers, whose jobs have been affected by mechanization to the point of removing craft elements.

A counter argument by Anshen¹ forecasted enhancement of the manager's job because of the computer. He contended that the machine would relieve the manager of tedious routine and make more time available for creativity and unstructured problem-solving. The result would be greater job satisfaction, not alienation.

In either of these contrasting arguments, the motivational assumptions are not hard to trace. Both positions depend implicitly on the same model of Man as first offered by humanistic psychologist Abraham Maslow.⁹ This model posits that man is motivated by a hierarchy of needs. Basic physiological, safety, and social needs motivate behavior initially, and when they are satisfied the higher-order or ego needs motivate behavior. Needs for achievement, recognition, growth, and self-actualization are said to be important

motivators for those not driven by hunger or basic survival. Through the work of Douglas McGregor¹⁰ Maslow's theory of motivation has had enormous influence over management thought since the late 1950s. Other theories of work behavior with close ideological links to Maslow's work are Herzberg's⁶ Motivation-Hygiene theory, Likert's⁸ System 4, and Argyris's² personality-maturity theory. While details of these approaches differ, they all project man as motivated by the intrinsic satisfactions of meaningful work and as possessor of needs for achievement and professional growth.

The importance of task scope to management motivation is clear under these models of work behavior. "Enriched" tasks, which provide challenging problems and opportunities for achievement, are more motivating than those devoid of challenge or achievement possibilities. Recent work has identified tasks with greater variety, autonomy, identity, and feedback as ones with greater "motivating potential."⁵ While research support is inconsistent, it is clear that such a viewpoint underlies the predictions of computer impact on tasks and satisfaction. Briefly, we are led to expect that *if an information system reduces task scope, managers will be less motivated and satisfied. On the other hand, if a system increases task scope, managers should react positively with greater motivation and satisfaction.*

It is important that these notions receive empirical testing if we are to move away from pure speculation. With this in mind a research project was formulated to address this question and others. The study was conceived as an exploratory project because few standard hypotheses or research methods have been generated toward answering questions about the computer's impact on organizations. The next section of this paper briefly describes the overall project and the specific approach to assessing changes in tasks and managers' reactions to them.

THE CISM PROJECT

The project from which this research is drawn is titled Computer Information Systems and Management (CISM). Over a five-year period research teams in Denmark, Austria, England, West Germany and the United States have collected data in eight organizations which have recently installed computer-based management information systems.

* This paper is based in part upon research supported by the National Science Foundation under Grant No. MCS77-22486. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the author and do not necessarily reflect the views of the National Science Foundation.

Each of 130 system users spent approximately four hours in interviews and questionnaire completion. In addition, unstructured interviews with other managers, systems designers and subordinates were conducted in an effort to describe the impact of the information systems on managers and their organizations. A host of variables were measured, including managers' tasks, interpersonal relations and formal structure. The major output of the project is currently being assembled and will consist of an examination of different themes, each focusing on a particular area of computer impact. In addition, each national team is producing detailed case reports of the companies they studied.

Nine questions about the impact of the system on tasks were asked of each user. For the sake of analysis these task characteristics may be grouped into three categories:

- *Enriching factors*—which reflect intrinsically satisfying aspects of work.
 1. Degree of complexity in the task
 2. Number of problems recognized within the task
 3. Possibility of developing new ideas or methods
 4. Feedback on decisions
- *Structure factors*—which reflect rigidity and routine in the task.
 1. Degree of routine of the task
 2. Standardization of codes or terminology in the task
- *Load factors*—which reflect work pace and its variations.
 1. Work pace in the task
 2. Variations in work pace in the task
 3. Work load within the task

Each respondent was asked to indicate whether each item had increased, decreased, or remained the same as a result of computer introduction. In addition, where changes were reported, respondents were asked to evaluate the change as an improvement or deterioration in their jobs, or state that the change made no difference.

Most of the questions directed to members of the CISM sample sought changes in two managerial tasks. Because some managers performed only one primary task, data on both tasks were not uniformly available. Furthermore, some respondents chose not to complete the section of the questionnaire described above. Nonetheless, data for at least one of the two tasks were obtained from 85 respondents, and data analysis is based on this group. The sample is heterogeneous with regard to type of system used, employer, and nationality. The common experience among respondents is their familiarity with a pre-existing task as well as the new computer system for performing their jobs. Our sampling thus controls for possible extraneous sources of variance and enables a direct assessment of the computer's impact on tasks and users' evaluation of those impacts.

RESULTS AND DISCUSSION

Changes in the enriching factors are shown in Table I, where changes are cross-tabulated with evaluations on each

TABLE I—Changes in *Enriching Factors* and Managers' Evaluations of Them

Degree of Complexity in Task:	Evaluation of Change makes no			totals
	deterioration	difference	improvement	
increased complexity	2	4	37	43
no change*		22		22
decreased complexity	1	0	19	20
totals	3	26	56	85

Number of Problems Recognized:	Evaluation of Change makes no			totals
	deterioration	difference	improvement	
increased number	8	6	29	43
no change*		23		23
decreased number	1	0	11	12
totals	9	29	40	78

Possibility of New Ideas or Methods:	Evaluation of Change makes no			totals
	deterioration	difference	improvement	
increased possibility	0	3	52	55
no change*		25		25
decreased possibility	3	0	0	3
totals	3	28	52	83

Feedback on Decisions:	Evaluation of Change makes no			totals
	deterioration	difference	improvement	
increased feedback	0	1	54	55
no change*		25		25
decreased feedback	1	0	0	1
totals	1	26	54	81

* Where "no change" in the task characteristic was reported, the evaluation response was automatically coded as "makes no difference."

item. The total number responding to each question in this and following displays fluctuates between 78 and 85 because of missing data, but this does not seriously affect the results. The frequencies are also reported here without supporting chi square statistics because expected frequencies are too small in too many cells to make the test statistic useful. However, the results come through clearly without statistical testing. A large percentage of respondents perceive increases in enriching factors and evaluate these changes as improvements. On two dimensions, complexity and number of problems, a substantial minority of respondents perceive decreases and evaluate them as improvements. On the whole, however, increases in these challenging aspects are noted and considered to be improvements.

These findings seem to suggest that predictions from the Maslow theory are correct—that system users react favorably to work that increases challenge and opportunity for achievement. However, we must also examine changes in

task structure. Here the motivation theories predict decreased satisfaction if task scope is effectively reduced through computer changes.

As Table II shows, respondents overwhelmingly see greater routine and standardization in their tasks as a result of the computer systems. However, in contrast with our expectations, users react positively to these changes. On task routine, 50 percent evaluate the change as an improvement and almost 50 percent say the change made no difference. Responses are not quite as favorable for the standardization aspect, although only ten report that their jobs have deteriorated because of increased standardization.

In explaining results which run contrary to theoretical predictions, one often turns to alternative theory. One explanation of these results lies in expectancy theories of work motivation.¹¹ A version of expectancy theory for information system user is shown in Figure 1.¹² It shows the vital role that performance and both extrinsic and intrinsic rewards play in the motivation process. If an information system increases users' ability to perform, perhaps through more standard routines, and if rewards follow performance, users will be more satisfied and exert more effort. While our data do not test this proposition directly, many of our research personnel report informal comments from users which indicate the benefits of increased rationality in tasks. Standardization and routine remove chaos from the job and permit better performance. Therefore, these aspects are evaluated as job improvements.

Our final set of task dimensions involve load factors. Here

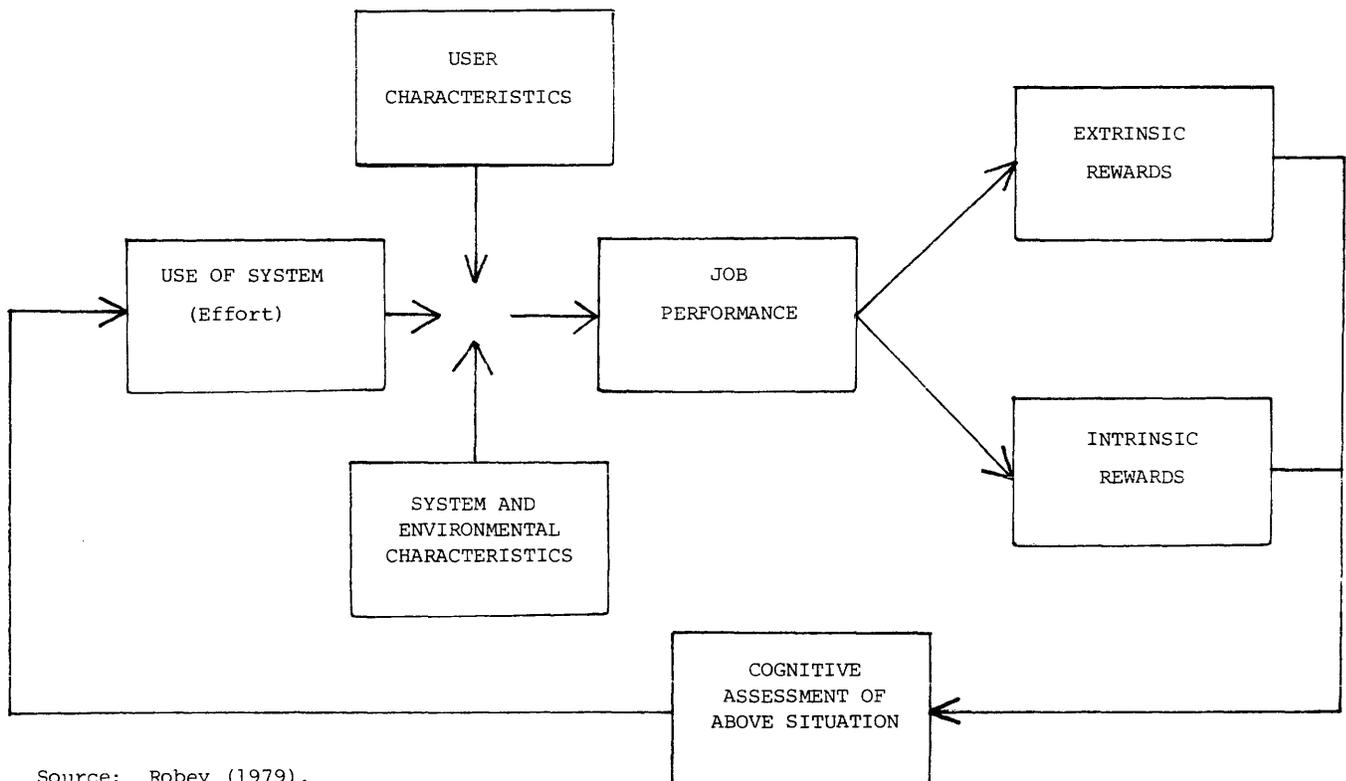
TABLE II—Changes in *Structure Factors* and Managers' Evaluations of Them

Degree of Routine of Task:	Evaluation of Change makes no			totals
	deterioration	difference	improvement	
increased routine	0	5	35	40
no change*		31		31
decreased routine	2	2	5	9
totals	2	38	40	80

Standardization of Task:	Evaluation of Change makes no			totals
	deterioration	difference	improvement	
increased standardization	10	15	22	47
no change*		37		37
decreased standardization	0	1	0	1
totals	10	53	22	85

* Where "no change" in the task characteristic was reported, the evaluation response was automatically coded as "makes no difference."

it is expected that users will evaluate increased load and load variations in a negative light. The data in Table III show that increases in work load and pace generally outweigh decreases, and that variations in work pace increase substantially. The evaluation of these three changes is not



Source: Robey (1979).

Figure 1—Model of user behavior.

TABLE III—Changes in *Load Factors* and Managers' Evaluations of Them

Work Pace in the Task:	Evaluation of Change makes no			totals
	deterioration	difference	improvement	
increased pace	9	7	24	40
no change*		32		32
decreased pace	0	1	9	10
totals	9	40	33	82

Variations in Work Pace:	Evaluation of Change makes no			totals
	deterioration	difference	improvement	
increased variations	19	10	19	48
no change*		26		26
decreased variations	3	1	3	7
totals	22	37	22	81

Work Load in the Task:	Evaluation of Change makes no			totals
	deterioration	difference	improvement	
increased load	9	3	17	29
no change*		35		35
decreased load	1	0	18	19
totals	10	38	35	83

* Where "no change" in the task characteristic was reported, the evaluation response was automatically coded as "makes no difference."

terribly negative, although users are most negative about variations in work pace. However, a substantial number of respondents evaluate increases in work pace and work load as task improvements, and this is in contrast with our expectations.

In explaining the findings on load factors we are led again to consider the expectancy model in Figure 1. Respondents may be responding favorably to increased work load because the computer systems permit them to achieve this additional work. While performance pressures have increased, users now have the tools to reach these higher output standards. Both intrinsic and extrinsic rewards may follow and increase the satisfaction and subsequent motivation of users. Part of the price for this greater productivity may be the increased variability of the work pace. In batch systems in particular users are forced to accept discontinuities in work pace to satisfy data processing schedules and machine availability. This aspect may reduce some of the improvements perceived to be related to increased output.

Some further insights into these findings might be gained by disaggregating the data. For example, on-line and batch users could be compared on the variability issue. Elsewhere, we have looked at task uncertainty as an intervening variable.⁴ These further analyses are beyond the scope of this paper, which has attempted to use as large a sample as possible to examine basic task effects. Disaggregation results in comparisons between much smaller subsamples, and we do not have the capacity to do too much sample-splitting

because our total sample is not large. Future CISM publications will show more detailed data analysis to the extent that it is warranted.

CONCLUSIONS

Our overall findings may be restated quite directly: information systems increase the presence of enriching factors, task structure and task load. All of these changes are evaluated favorably by users. An explanation of these results requires that we go beyond a Maslow-based motivation theory, which predicts only the first result. Satisfaction with increases in task structure and task load are better understood by adopting an expectancy theory model of user motivation.

In moving toward the broader implications of these findings, it is interesting to note that apparent counteracting changes have occurred in the tasks we studied. The computer does make organizations more bureaucratic in the sense that routine and standardization have increased. But it also generates nonbureaucratic changes: it increases task complexity, and enables users to apply new ideas and methods to a wider array of task problems. It appears possible that both changes can occur simultaneously, which suggests that many earlier arguments linking computers to task changes were grossly oversimplified. Our results show a complex combination of impacts, which serve to enrich as well as standardize the job. There is no logical conflict in this statement if we assume that jobs are composed of several independent dimensions. Unfortunately, task research has not isolated standard measurable dimensions which could be used in research like this.³ It does not appear, however, that the computer's impact on tasks can be fully understood by employing a simple, one-dimensional concept of task.

Of further interest is the pattern of evaluative responses by users. Even where tasks do become more bureaucratic (standard and routine), the change is not perceived as deterioration. The white collar worker is not alienated by the introduction of computer systems which increase task routine. Quite to the contrary, the middle manager has enthusiastically embraced the new technology largely, we feel, because it helps him improve his job performance. The computer removes guesswork, ambiguity, and structures the task so that it can be done more effectively. The user is not a loser in this transition, at least not in his own eyes. Very few respondents, even in companies with older tenured employees, regard the computer negatively. Nor are they particularly fascinated by it or treat it as a magnificent toy. It is simply a tool which permits work to be accomplished more effectively than before. To the extent that managers share the benefits brought about by better performance, they feel that their jobs have improved.

ACKNOWLEDGMENTS

I am indebted to the following who contributed to the research design and data collection: H. Lippold and E.

Reindl (West Germany), N. Bjørn-Andersen and P. Pedersen (Denmark), G. Weiser (Austria), and K. D. Eason, L. Damodaran, and T. F. M. Stewart (U.K.).

REFERENCES

1. Anshen, M., "The Manager and the Black Box," *Harvard Business Review*, Vol. 38, 1960.
2. Argyris, C., "Personality and Organization Theory Revisited," *Administrative Science Quarterly*, Vol. 18, 1973.
3. Dunham, R. B., "The Measurement and Dimensionality of Job Characteristics," *Journal of Applied Psychology*, Vol. 61, 1976.
4. Eason, K. D., "Computer Information Systems and Managerial Tasks," paper presented to the Conference on Computer Impact, Copenhagen, October 1978.
5. Hackman, J. R. and G. R. Oldham, "Development of the Job Diagnostic Survey," *Journal of Applied Psychology*, Vol. 60, 1975.
6. Herzberg, F., "One More Time: How Do You Motivate Employees?" *Harvard Business Review*, Vol. 46, 1968.
7. Leavitt, H. J. and T. L. Whisler, "Management in the 1980's," *Harvard Business Review*, Vol. 36, 1958.
8. Likert, R., *The Human Organization*, New York: McGraw-Hill, 1967.
9. Maslow, A. H., "A Theory of Human Motivation," *Psychological Bulletin*, Vol. 50, 1943.
10. McGregor, D., *The Human Side of Enterprise*, New York: McGraw-Hill, 1960.
11. Porter, L. W. and E. E. Lawler, III, *Managerial Attitudes and Performance*, Homewood, Ill.: Richard D. Irwin, 1968.
12. Robey, D., "User Attitudes and Management Information System Use," *Academy of Management Journal*, Vol. 22, 1979.

Some neglected outcomes of organizational use of computing technology—And their implications for systems designers

by M. LYNNE MARKUS

Case Western Reserve University
Cleveland, Ohio

INTRODUCTION

Twenty-five years after the first commercial application of computers, it now seems possible to assess their consequences on the basis of user experience. Until recently, most assessment was done on the basis of early predictions inspired by the technological capabilities of computers. But these predictions have missed the mark considerably. Not only did the technology of computing fail to stand still, but it has been demonstrated time and again that just because a capability exists does not mean that it will be used as intended.

The predictions were dramatic: increased centralization, reduction in the number of middle managers, decreased personal autonomy on the job.¹⁶ But the research findings were not dramatic: few structural changes occurred in organizations using computers, usually only those reflecting the creation of EDP departments.^{23,11} If anything, the findings have shown that the effects of computing are "subtle, limited and complex in their patterns."¹⁴

This subtlety and complexity makes computer impact research difficult to do. Consequently, not much is done, especially inside organizations. One manager of management systems in a major industrial corporation recently told me: "We go through all sorts of gyrations to justify our systems development efforts—cost benefit analysis, ROI calculations and so forth. But we have yet to go back to see whether we achieved the benefits we expected. We know that the benefits are too difficult to measure and too difficult to trace back to their causes." Academic research has begun to fill the gaps in knowledge about computer impact on organizations, but the research has sadly neglected some important types of outcomes.

RESEARCH FROM THE IMPLEMENTATION SCHOOL

Two general philosophies pervade the academic literature on computing use in organizations. One stream of research focuses on the factors contributing to success or failure of

computerized information systems. The second focuses on computer-induced changes for the individual employee or for the organization as a whole.

A number of researchers have tried to identify what factors influence or determine the success of information systems and management science projects. To call this rather diverse collection of operations researchers, management scientists and management information systems specialists the "implementation school" may be stretching a point. But writers such as Bean, et al.,³ Schultz and Slevin,²² Lucas,¹⁷ Gibson,⁸ Ginzberg^{9,10} and Alter^{1,2} share a common focus, first, on individual information systems and, second, on a specific type of outcome.

Both focal points of the implementation school distinguish it from the organizational school, to be discussed shortly. The first focal point, individual information systems, is a significant asset for the implementation school, because research has shown that different types of systems applications have different outcomes,^{23,11} and that there is something in the nature of application types which affects organizational outcomes.¹⁰ The organizational school, in contrast, tends to lump all types of applications together into indices of computerization for an entire organization, ignoring the previously-cited research and the fact that some parts of an organization may be unaffected by computing.

Limitations of the implementation school

The second focal point of the implementation school, however, puts significant limitations on the research of this school. This is the school's focus on one particular type of outcome, namely, what happens when the systems designer turns the system over to the user ready to be converted or used. Different researchers in the school have used different measures of this outcome: Some have used the success or failure of the system to be implemented, others have used rates of use, still others have used the satisfaction/acceptance or dissatisfaction/resistance of users. But almost all call their dependent or outcome variable "success of the information system," and almost all measure it at a point in time before the users have had a chance to use the system.

This focal point of the implementation school has led to some serious limitations on its research. The first limitation relates to time. The startup of a new or modified information system is roughly analogous to the startup of a new plant. One does not simply flip a switch and begin to use the new technology. There are false starts, glitches and occasional trips back to the drawing board. Many systems designers and users I have interviewed express the opinion that working the bugs out of a new system takes anywhere from six months to two years. One major implication of this is that the "permanent" effects of an information system in an organization may not be felt by users or observable by researchers until some six to 24 months *after* the system is installed, that is, after users have some experience working with it. And, because of its focus on success defined at the point in time when the system is installed or turned over to the user, the implementation school has not been able to examine outcomes of information system *use*.

Neglected organizational outcomes

There is another limitation resulting from the implementation school's focus on "success of the information system." This is the failure to recognize the organizational consequences of information systems. In the effort to construct measures of success or failure of computerized systems, defining and redefining "usage," "utilization" and "user satisfaction," implementation researchers have left the search for organizational outcomes to researchers who did not distinguish in form or content between payroll systems and investment analysis programs. Almost totally ignored by implementation researchers have been the outcomes which relate to social and political aspects of organization functioning.

One manifestation of this organizational "ignorance" is the implementation school's definition of the "user." Systems designers interested in researching the success of their systems have typically, and not unreasonably, been most concerned about the satisfaction or acceptance of the person or group who commissioned the new system and who may have also paid for it. Therefore, the client of the systems designer is often designated the user of the system. The problem comes in when there are other, hidden users of the system, maybe those who only supply data to it. Often these other users have different perceptions of the success of the system, different feelings of satisfaction with it, from the client group. Including these might radically alter the researchers' measures of system success.

A financial information system which I have recently studied illustrates this point. This system consolidates financial data from the divisions of a large, decentralized corporation. In response to my question, "Who are the users of the system?" I was told that the main users were members of a corporate staff accounting group. Further questions revealed that divisional accountants provided the data to the system and maintained it.

"Aren't the divisions also users of the system?" I asked.

"Well, actually, the divisions don't use the system as

much as they could. We're disappointed that they haven't used the variable report writing facility to generate new types of financial analyses."

Interviews with corporate accountants identified a long list of benefits with which the new system provided them: automatically consolidated financial statements, decreased workload, faster month-end closings, standardized input from divisions, more and better management information, greater flexibility around external reporting, decreased disruption of financial activities caused by internal organizational changes. In contrast, divisional accountants found the system to be an obligation with few or no benefits to them: "Creating and maintaining the database for this system were huge jobs. Our division already had a smooth-running system for providing this information to Corporate, and it took two years to iron all the bugs out of this one. Also, it doesn't do a thing for me. The data in it is not at a level of detail where I can use it, even if I did have a staff of programmers to help me use the report writer. So I need to maintain a dual system for my own internal reporting needs."

This example illustrates that not all users of an information system see the system in the same way and if the people in this organization had taken a more inclusive view of users than just the corporate accounting group, *they* would have seen a distribution of outcomes, of costs and benefits, of satisfactions and dissatisfactions, associated with the system. This distribution is influenced by the social and political factors in the organization, factors previously neglected by most implementation researchers (with the notable exception of Gibson⁸), who most often focus on the system itself and the psychological attributes of client-users.

RESEARCH FROM THE ORGANIZATIONAL SCHOOL

The second research stream, which I'll call the organizational school, includes the studies of organizational theorists and sociologists like Whisler,²⁴ Hoos,¹² Kraut,¹⁵ Blau,⁴ Stewart,²³ Reif,²⁰ Hofer,¹¹ Robey²¹ and Pfeffer.¹⁸ The organizational school examined the effects of computing on a full range of variables reflecting psychological and psycho-technical outcomes for individuals. The effects of computing on job task, autonomy, stress, satisfaction with work and employment patterns are some examples. But early on, the organizational school began to concentrate efforts on variables reflecting changes in the organization as a whole. And the area of impact which attracted the most attention was the effect of computing on power, authority and influence in the firm.

The first published predictions about computer impact on organizations had raised this issue. Leavitt and Whisler,¹⁶ in 1958, saw the potential of computing technology to store, process and analyze large quantities of data in a single place. They reasoned that access to this information would give managers the power centrally to control dispersed business activities. Managers had formerly had to delegate some of their authority, because of their limited ability to gather and digest information. The increased centralization of control

brought about by computer use would, they believed, reduce autonomy of middle managers and lower job satisfaction. Also, many middle management positions would be eliminated, replaced by computer applications.

The wave of research which followed these predictions did not significantly support them, however. Some industries, like insurance and banking, did appear to become more centralized, but researchers are still debating the issue. What they are debating, though, is how to study it, not whether the issue is important. The major contribution of the organizational school has been to identify changes in power, influence and control as potential effects of computing use and to legitimate the area of study. It made this contribution in spite of its tendency to focus on outcomes for the organization as a whole rather than on the more clearly delimited set of organizational outcomes which result from individual computerized information systems.

Outcomes for organizational power and influence

Recently, research in the area first investigated by the organizational researchers has begun to pay off. A number of studies, which may represent the beginnings of a third major school of research, have appeared in the last several years using the focus on individual information systems of the implementation school and the focus on outcomes for organizational power and influence of the organizational school.

Kling¹³ found "that computer-based systems reinforce the existing distribution of power in American municipalities. They provide differential support to mayors and city managers in smaller cities and to departments in the larger cities."

Bjorn-Andersen and Pedersen⁵ found that computing use in the business organizations they studied contributed to shifts in power among affected groups. By changing the basis of power of various organizational members, that is, by changing aspects like one's position of centrality in the flow of work through the organization, one's expert knowledge and one's access to up-to-date information, use of computing technology has created "winners and losers in the fight for influence."

Incidentally, Bjorn-Andersen and Pedersen noted that the early organizational research may have failed to identify these subtle changes, because the concept "centralization" is "too crude and only covering parts of the very complex interpersonal relationships potentially to be altered by the introduction of computer systems."⁵ Pfeffer¹⁹ lends weight to their observation. According to Pfeffer, real changes occurring in organizational processes of power and influence may not be immediately reflected in measures of organizational structure, like degree of centralization.

CASE ILLUSTRATION—ORGANIZATION AND SYSTEM OUTCOMES

A recently published case study by Conrath and du Roure⁶ illustrates two points I have been making; first, that benefits

from, and incentives for, using computing technology vary across user groups, and second, that the success or failure of the information itself is related to and perhaps secondary to organizational outcomes, such as changes in power and influence among user groups.

Conrath and du Roure's case describes the implementation of a comprehensive logistics system in a branch of the U.S. military. The new system provided on-line access to up-to-date status information for all materiel for which the branch had responsibility. Unfortunately, the case does not tell us who initiated the system, who supported its development, and who bailed it out with an expensive redesign when it was close to failure. We are, however, told the outcomes and some details of the organizational arrangements into which the new system was introduced.

Prior to the development of the new system, all data were collected into a monthly report for senior officers. On the average, the report was a month out of date. Consequently, it was not well used in the logistics decisions of the branch. Requests for materiel were routed to junior officers, who forwarded them up the chain of command to an officer of high enough rank to handle the request. Information about the requests was obtained by telephoning those believed to possess the information.

The new system collected all relevant information automatically and made it available to junior officers through on-line terminals. No telephone calls were required to obtain information. Furthermore, the system determined the optimal way to transport materiel from one point to another, minimizing distance.

When the new system was first used, "the greatest impact was a change in the effective structure of the organization. It went from one which had been very hierarchical, very vertical, to one which was primarily horizontal. . . . The chain of command almost seemed to be superfluous." The new system enhanced the power base of junior officers by giving them access to information. This started to erode the power bases of senior officers.

But then the reaction occurred. "The commanding officer (three-star rank) demanded that the old system be continued in parallel with the new. The roles of the more senior officers were to be maintained. The argument given was that the computer-based system was not yet sufficiently reliable. The underlying reason, however, was the effect that the information-communication system had on the perceived value of the authority structure. In fact, once the old system was reinstated, the stress on supplying all the required data to the new system was relaxed, and the new system did become less reliable. The cause-and-effect relationship, however, was the reverse of the way it was presented."

The senior officers had discovered that the new system eroded their power base, a trend which, if continued, would have undermined their authority. The parallel system allowed them to maintain their traditional position. But if the dual system had continued very long, the technical advantages of the new system would have been lost. Someone decided that this should not happen and authorized the redesign of the system to include "a monitoring system designed to provide the perception of participation and control to the

more senior officers, but a system which was computer-based and integrated into the original information-communication system."

CONCLUSION

The organizational school discovered that changes in power and influence are one major class of computing use outcomes. Research in this tradition, however, has generally failed to consider differential effects of different computer applications. The implementation school has had almost the reverse set of strengths and weaknesses: a focus on individual applications but a failure to consider power and influence.

The focus of implementation researchers regards as the user the person or group who requested system development. This, in turn, has encouraged implementation researchers to search for black-and-white outcomes, success or failure, within a single user group. This paradigm cannot distinguish between, for example, a system which failed because users did not have the appropriate cognitive style to use it and a system which failed because the initiating user imposed it upon other users who had sufficient power to sabotage it or to get around it in some way. Surely, these two causes of system failure would lead to different prescriptions for improving the activities of systems designers.

In the second instance, the organizational outcomes of the new system, namely, an undesirable imposed change, had a secondary effect on the information system itself, namely sabotage. The failure to recognize the interaction between consequences for the organization and consequences for the information system has been a limitation of implementation research that may be hindering the improvement of system design practice.

Sabotage is one kind of interaction between organizational outcomes and system outcomes. Maintenance of parallel systems is a second. Making changes to the information system itself after installation may be a third. Requests for changes to already implemented systems sometimes reflect user reactions to changes or disruptions in their organizational practices brought about by using a new computerized information system. This is clearly what happened in the Conrath and du Roure case just cited.

If the best aspects of both schools are combined, the application focus of the implementation school and the power and influence outcomes variables of the organizational school, the neglected outcomes are brought into focus. It then becomes possible to perform what can be called computer/organizational impact analysis. In this type of analysis, one traces organizational outcomes of computing use back to their causes. Changes in power and influence among various user groups can be traced back to the interaction of new system designs with existing organizational arrangements, as in the case by Conrath and duRoure. These same organizational outcomes can then be projected forward into probable consequences for the success or failure of the information system itself.

If systems designers performed computer-organizational

impact analysis before they designed and installed new systems, they might anticipate information system failures and learn to avoid them by manipulating aspects of system design and implementation with a view toward eliminating undesirable organizational outcomes.

Computer/organizational impact analysis might also help managers in organizations create desired organizational changes. Galbraith⁷ and Pfeffer¹⁹ have shown that computerized information systems can be used to change the structures and processes of organizations. However, the research has yet to tell us how to design computerized information systems to make these desirable organizational changes possible. Computer/organizational impact analysis may fill this gap, by shedding light on some of the neglected outcomes of computer use.

REFERENCES

1. Alter, Steven L., "Computer-aided decision making in organizations: a decision support system typology," Cambridge, Mass., Massachusetts Institute of Technology, Center for Information Systems Research, Report CISR-11, 1976.
2. Alter, Steven L., "How Effective Managers Use Information Systems," *Harvard Business Review*, Nov.-Dec., 1976, pp. 97-104.
3. Bean, Alden S., Rodney D. Neal, Michael Radnor and David A. Tansik, "Structural and Behavioral Correlates of Implementation in U.S. Business Organizations." In Randall L. Schultz and Dennis P. Slevin (eds.), *Implementing Operations Research/Management Science*, New York, American Elsevier, 1973, pp. 77-132.
4. Blau, Peter M., Cecilia McHugh Falbe, William McKinley and Phelps K. Tracy, "Technology and Organization in Manufacturing," *Administrative Science Quarterly*, Vol. 21, 1976, pp. 20-40.
5. Bjorn-Andersen, Niels, and Poul H. Pedersen, "Computer systems as a vehicle for changes in the management structure," Copenhagen, Denmark, Information Systems Research Group, Report 77-3, 1977.
6. Conrath, David W., and Gabriel du Roure, "Organizational implications of comprehensive communication-information (I-CS) systems, some conjectures," Aix-en-Provence, France, Institute d'Administration des Entreprises Centre d'Etude et de Recherche sur les Organisations et la Gestion, 1978.
7. Galbraith, Jay R. *Organizational Design*. Reading, Mass, Addison-Wesley Publishing Co., 1977.
8. Gibson, Cyrus F., "A Methodology for Implementation Research," In Randall L. Schultz and Dennis P. Slevin (eds.), *Implementing Operations Research/Management Science*, New York, American Elsevier, 1973, pp. 53-73.
9. Ginzberg, Michael J., "A Detailed Look at Implementation Research," Cambridge, Mass., Massachusetts Institute of Technology, Center for Information Systems Research, Report CISR-4, 1974.
10. Ginzberg, Michael J., "Implementation as a Process of Change: A Framework and Empirical Study," Cambridge, Mass., Massachusetts Institute of Technology, Center for Information Systems Research, Report CISR-13, 1975.
11. Hofer, Charles W., "Emerging EDP Pattern," *Harvard Business Review*, March-April 1970, pp. 16-171 (non-inclusive).
12. Hoos, Ida Russakoff, "When the Computer Takes Over the Office," *Harvard Business Review*, Vol. 38, 1960, pp. 103-112.
13. Kling, Rob, "Automated Information Systems as Social Resources in Policy Making," *Proceedings 1978 Annual Conference Association for Computing Machinery*, pp. 666-674.
14. Kraemer, Kenneth L., and William H. Dutton, "The Interests Served by Technological Reform: The Case of Computing," Irvine, Calif., The Urbis Research Group, Wp-78-58, 1978.
15. Kraut, Allen I., "How EDP is Affecting Workers and Organizations," *Personnel*, July-Aug. 1962, pp. 38-50.
16. Leavitt, Harold J., and Thomas L. Whisler, "Management in the 1980 s," *Harvard Business Review*, Nov.-Dec., 1958, pp. 41-48.

17. Lucas, Henry C., Jr., *Why Information Systems Fail*, New York, Columbia University Press, 1975.
18. Pfeffer, Jeffrey, and Huseyin Leblebici, "Information Technology and Organizational Structure," *Pacific Sociological Review*, Vol. 20, 1977, pp. 241-261.
19. Pfeffer, Jeffrey, *Organization Design*, Arlington Heights, Ill., AHM Publishing Corp., 1978.
20. Reif, William E., *Computer Technology and Management Organization*, Iowa City, Iowa, Bureau of Business and Economic Research, College of Business Administration, The University of Iowa, 1968.
21. Robey, Daniel, "Computers and Management Structure: Some Empirical Findings Re-examined," *Human Relations*, Vol. 30, 1977, pp. 963-976.
22. Schultz, Randall L., and Dennis P. Slevin, "Implementation and Management Innovation," In Randall L. Schultz, and Dennis P. Slevin (eds.), *Implementing Operations Research/Management Science*, New York, American Elsevier, 1973, pp. 3-20.
23. Stewart, Rosemary, *How Computers Affect Management*, Cambridge, Mass., The MIT Press, 1971.
24. Whisler, Thomas L., *The Impact of Computers on Organizations*, New York, Praeger Publishers, 1970.

An academic meets industry— Rethinking computer-based education and personalized systems of instruction

by KENNETH L. MODESITT

Texas Instruments, Inc.
Dallas, Texas

PROLOGUE

Education is a versatile animal. After belonging exclusively in the domain of the home for millenia, public education in the United States became institutionalized about 1800. Today, the school system in many cultures is often equated to "the" educational system. However, industrial institutions are also becoming acutely aware that education is a vital component of their structure, and in some cases, of their market.

I believe that education is vital in all three of the above institutions: home, school, and industry. If a role of education is to help us learn new ways, as well as to understand current and past concepts and events, then we are in for a great deal of education in the future. It hardly seems possible that Toffler's *Future Shock* is already eight years old, as we observe and share in rapidly changing lifestyles and institutions constantly today.

THE ACADEMIC

It is my deep conviction that a large part of education can occur in a framework other than in a formal classroom setting. In such a formal lecture/testing situation, students are most often evaluated on a basis of comparison with other people who happen to be in the class. The resulting attitudes are ones we've all seen or experienced: the confident, bright, fast-learning top 10 percent, the slower-learning bottom 10-20 percent who consider themselves "dumb" or "stupid," and the large majority in between—not really understanding much, but content to receive a "C" and get out.

It was attitudes such as these which prompted me several years ago to investigate alternative modes of education. The alternatives involved evaluating students on the basis of mastery of definable objectives, and not on how well or poorly or rapidly their classmates performed. I first investigated computer-based education (CBE) over ten years ago, became disillusioned with the quality, and left. When PLATO became viable in the early 1970s, my interest and activity were rekindled. Shortly thereafter, a friend intro-

duced me to Keller's Personalized System of Instruction (PSI). Combining CBE and PSI in a university framework proved very useful in eliminating many of the attitudes mentioned earlier.¹³⁻¹⁶ Then, a few years ago, I introduced explicit cooperation into these courses, lest students become too isolated.¹² The primary cooperative efforts were: study partners for PSI units, design of a computerized personal data base by a small group, mutual design and use of interactive programs, use of cooperative exercises and parties.

Approximately the same time as the last papers were published, the personal computer revolution began in earnest. The seeds were thus sown for a major professional transition: a long-abiding interest in the home, a career in higher education, professional training in computer science, and the personal computer. After balancing these for a year or two while continuing to teach in the university environment (and encouraging all my students to investigate machines for home use¹¹), I made the decision to return to the computer industry after a 13-year hiatus.

THE ATTRACTION OF INDUSTRY

Texas Instruments (TI) is a very natural professional home at this time. It has a long-standing interest in the individual consumer; it can deliver high-technology personal products at affordable prices; mini- and microcomputers are current offerings of the company; it has made a commitment to using an extremely well structured programming language (Pascal); a large educational effort is underway to deliver this tool; and most importantly, it actively encourages creative solutions to challenging problems. Consequently, my active interests in personal computing and CBE/PSI educational systems could be naturally joined to help deliver a useful, fun and challenging tool to homes everywhere. "Computing power to the people" became an area where contributions were definitely possible. In fact, TI received top billing in an excellent recent article on CBE, where the author stated: "... it is not wise to underestimate the ingenuity of the private sector in harnessing mass production to perceived consumer needs, whether in the home or in the

school."¹⁹ He then goes on to describe the popular Speak & Spell (TM) product of TI.

Texas Instruments has made a substantial effort to improve the scope and calibre of on-the-job education for its employees. And because of the diversity of employees in such a company, it is an ideal site for applying CBE/PSI principles developed in academia. Lessons learned in this environment can be utilized as TI also actively pursues a market beyond its own confines. This market is the general populace which can make fantastic creative contributions if given a versatile, well-designed home computer capable of educating *and* of becoming "educated" itself. An excellent article in a recent issue of *Computer* is a well balanced overview of some of these contributions and how they might come about in the 1980s, according to a group of experts.⁸

The ability of machines to become "educable" is my requirement for personal computers. They must be able to perform many useful and fun tasks. Pick up any copy of *Creative Computing*, *Personal Computing*, *Byte*, *People's Computers*, etc. for a vast array of marvelous application areas. But the same computers must also be able to be told how to do new things—no amount of ROM is adequate here. Historically, we have told the machine how to do new things by creating a new program for it. I doubt very much if the general public will program in the same ways you and I have for 20 years, but we must make it possible for them to "teach new tricks" to their new helper and game-player. Our natural language researchers in artificial intelligence should be a big help here. Smalltalk at Xerox PARC is an excellent start.⁹

POTENTIAL CONTRIBUTIONS

Both CBE and PSI, after a relatively sheltered life in academic institutions, hold great promise in other institutions, notably the home and industry. Control Data Education Company and others are deeply involved with CBE in industry.⁵ And it is rumored CDC is even looking at the home market.⁶ PSI, on the other hand, is almost exclusively used in schools. Therefore, one of the potential contributions is to formulate a strategy to maximize the effectiveness of both tools in an industrial environment. They have proven themselves to be viable in their original settings. Now we are in a position to respond to numerous calls for cooperation between the university and industry.^{4,17}

Upper-level management at TI has decided that the widely-recognized superiority of Pascal for many programming tasks will be acknowledged within a majority of future TI products. Consequently, they have funded the effort to produce the first industrial version of Pascal, called TI Pascal. The language, in both its sequential and concurrent forms, has been formally released.

Now formulating the (few) extensions to Pascal and implementing the associated compilers, interpreters, and run-time support systems is no mean feat. However, the educational effort required for widespread use of this tool is also considerable. The universities have been a great help here as they continue to replace their introductory courses

oriented to Fortran and assembly language with ones emphasizing Pascal and associated reliable design techniques. A recent issue of *Byte* contains many readable articles on Pascal.⁷ Textbooks are also rapidly appearing.^{2,3,18}

Within TI then, we are experiencing many of the problems associated with university classes: "too many" students for only a few instructors, not enough individual attention, time conflicts, student travel time, multiple entry levels, etc. In addition, there are some unique concerns. In particular, student time is worth money! This is a revolutionary concept in school, but a vital consideration in industry. Moreover, students do not really receive a letter grade. Rather, it is *assumed* that, after they have attended a Pascal course for n hours, they know Pascal sufficiently to be able to read and write Pascal programs.

A POSSIBLE SOLUTION

One of the possibilities under consideration by TI for delivering some courses involves both CBE and PSI. The following is one option.

1. Develop CBE materials using off-the-shelf hardware, and pieces of courseware, if available.
2. Use the product for a few classes until the majority of the major errors are removed.
3. Concurrently, develop TI software and hardware designed for a well-engineered CBE interface, relying on experience gained in the current system by users and authors.
4. Transport currently developed courseware to the new distributed TI system permitting interterminal communication.

At this point, students will take a lesson (or course) near their site at a TI system. User consultants will be available on-line in prime time, with a notes feature used at other times.

The course will usually consist of several units as in PSI, where each unit contains an introduction, objectives, suggested procedures (involving reading, program writing, taking CBE lessons, etc.), sample exercises and a multiple version unit test. Unit mastery will be demonstrated by passing the unit test at 80-90 percent competency, where each question will relate to a unit objective and be categorized by Bloom's educational cognitive taxonomy.¹ For readers unfamiliar with PSI, the excellent paperback by Keller and Sherman is recommended,¹⁰ in addition to earlier referenced papers of the author. Briefly, PSI is characterized by five attributes:

1. Mastery-based
2. Self-pacing
3. Non-lecture
4. Written materials
5. Proctors

The argument in favor of PSI runs something like this.

Students can gain in self-respect, and justifiably so, when they demonstrate they can master the material (a). But since students grasp material at widely varying rates, such mastery will occur only if the students are primarily responsible for pacing themselves (b). However, if they pace themselves, there is no way any lecturer could possibly speak to all students at their individual points of progress (c). But if the information is not transmitted verbally, how can it be done? Written material provides a partial answer (d). With the widely varying rates of progress through a course, there is no way one instructor could manage the evaluation process for 30 students working on 20 units of material, each with its own multi-version unit test. Hence, proctors become invaluable (e). They each, including the instructor, take responsibility for about ten students. That is, they go over the unit tests, answer questions, suggest resources, etc.

The first unit test will be graded on-line at the student's convenience by an instructor. If competency is not demonstrated, suggestions for further study will be given and another version of the test taken later. This will continue until mastery is achieved, and then the student will start the second lesson. For latter units, either the instructor or another student who had passed the unit earlier (internal proctor) will grade the test and make suggestions. This proctor can be at the same site as the student taking the test or at another TI site. In either case, the proctor can see the student's quiz and carry on a full dialogue.

Course mastery will be demonstrated by passing all units at mastery level. Statistics regarding completion times, number of unit test retries, proctors, comments on lessons taken, ill-stated questions or objectives, etc., are easily gathered in the proposed system and can be released to authorized personnel. For example, a cost center manager might wish to see how much money should be budgeted for students to take the TI Pascal course.

A FUTURE

I would expect that, should TI decide to try the preceding alternative to current course development and delivery, the company will also market a similar product eventually. A home computer can be utilized for educational lessons with dial-up access to instructors or proctors who might become widely-scattered friends over time. Components of the lessons can use the many exciting peripherals forthcoming: touch panels, audio output and input, video disks, synthesizers, etc. as well as our best and most versatile resource: people. If a student has difficulty understanding something, cooperation is encouraged! I think we are here to help one another, not to compete continuously.

SUMMARY

I have suggested that the institutions of home, school, and industry have much to gain by cooperation with one another.

My interest in providing computing power at an individual level has led me on an odyssey from industry to university life, and now back to a computer industry vitally involved with personal computing. Lessons learned in the university about how students learn better are suggested as viable options in the industrial environment. In particular, CBE (computer-based education) and PSI (Personalized System of Instruction) are conjoined to offer flexible courses internally. And from these efforts, modified ones can be made available to the public, permitting learning to occur wherever people happen to be, perhaps just finishing up with a rousing game of interterminal "Star Trek" or "Oregon Trail!"

BIBLIOGRAPHY

1. Bloom, Benjamin S. (ed.), *Taxonomy of Educational Objectives, the Taxonomy of Educational Goals, Handbook I: The Cognitive Domain*, David McKay Company, New York, 1956.
2. Bowles, Kenneth L., *Problem Solving Using Pascal*, Springer-Verlag, New York, 1977.
3. Brinch Hansen, Per, *The Architecture of Concurrent Programs*, Prentice-Hall, Englewood Cliffs, New Jersey, 1977.
4. Casmei, Howard B., "Computer-Based Education: An Approach toward Adaptive Learning Procedures," *ADCIS Summer Conference*, 1976, pp. 1-22.
5. Control Data Education Company, *Courseware Catalog*, Fall 1978.
6. Crudele, John, "Control Data May 'Teach' Home Computer Operations," *Electronic News*, September 11, 1978, p. 32.
7. Helmers, Carl (ed.), *Byte*, Vol. 3, No. 8, August, 1978.
8. Isaacson, Portia, and Jim Warren, et al., "Personal Computing," *Computer*, Vol. 11, No. 8, September, 1978, pp. 86-97.
9. Kay, Alan, and Adele Goldberg, "Personal Dynamic Media," *Computer*, Vol. 10, No. 3, March, 1977, pp. 31-41.
10. Keller, Fred S., and J. Gilmour Sherman, *The Keller Plan Handbook*, W. A. Benjamin, Menlo Park, Calif., 1974.
11. Modesitt, Kenneth L., "Beyond Independence: A View of Games and Personal Computing," *AEDS National Conference*, 1978, pp. 188-193. Revised version presented at *ADCIS National Conference*, 1979, pp. 911-921.
12. Modesitt, Kenneth L., "A Community of Individuals: Cooperation and Individualization in Computer Science Education," *National Computer Conference*, 1976, pp. 561-567. Also appeared in *International Journal of Computers and Education*, Vol. 2, No. 3, 1978, pp. 227-234.
13. Modesitt, Kenneth L., "The Tangled Triangle: Cooperation, Computer-Based Education and Personalized Systems of Instruction," *ADCIS Summer Conference*, pp. 320-324.
14. Modesitt, Kenneth L., "Personalized Systems of Instruction in Computer Science: 'Adult' Education," *National Conference on Personalized Instruction in Higher Education*, 1975.
15. Modesitt, Kenneth L., "An Excellent Mixture for PSI: Computer Science, PLATO, and Knowledge Levels," *ACM National Conference*, 1974, pp. 89-94.
16. Modesitt, Kenneth L., "PSI: A Valuable Addition to the Alphabet Soup for Computer Science Education," *ACM Special Interest Group on Computer Science Education Bulletin*, Vol. 6, June, 1974, pp. 37-44.
17. Pooch, Udo, and Richard Austing, et al., "Computer Science and Computer Engineering Education in the 80's," *Computer*, Vol. 11, No. 8, September, 1978, pp. 69-83.
18. Schneider, G. Michael, Steven W. Weingart and David M. Perlman, *An Introduction to Programming and Problem Solving with Pascal*, Wiley, New York, 1978.
19. Sugarman, Robert, "A Second Chance for Computer-Aided Instruction," *IEEE Spectrum*, August, 1978, pp. 29-37.

Recent developments in computers and society research and education

by RICHARD H. AUSTING

University of Maryland
College Park, Maryland

and

GERALD L. ENGEL

Old Dominion University
Norfolk, Virginia

INTRODUCTION

Since the advent of computers, there has been concern about the impact of computers on various aspects of society. Thousands of articles and numerous books have been written in the general area of computers and society. Computer professionals have been instrumental, though perhaps not always significantly enough, in the development of policies and legislation affecting the use of computers in the public domain. Studies have been conducted; courses have been introduced, primarily at the college level; seminars directed to computer professionals have been held; and public information programs have been sponsored, mostly on a local level.

While much of this activity is continuing, a number of developments during the past few years suggests that new (or renewed) emphasis is being placed in certain areas, particularly in research and education.

RESEARCH

One indication that research activity in any area is increasing is the establishment of a journal or section of a journal for publication of results. Research papers in computers and society have been appearing regularly in the *Communications of the ACM* since January, 1976 when "Social Impacts of Computing" was made a separate Technical Department with its own editor (R. Kling, University of California at Irvine). Also, the conference ACM 78 which was held in Washington, D. C. in December, 1978 included two sessions on research in computers and society. Six papers were featured in these sessions.¹

Books have appeared which give more stress to issues in the area of computer impact on society and analytical perspectives of the subject matter rather than expository treatments of applications as found in many works. Examples of the former are books by Arbib, Gerberick et al., Gotlieb and

Borodin, Hoffman, Mowshowitz, and Weizenbaum.²⁻⁸ These kinds of books are useful for computer professionals as well as for upper division or graduate students in a computer oriented degree program.

EDUCATION

Recent curriculum recommendations from ACM's Curriculum Committee on Computer Science (C³S) include an advanced-level course in Computers and Society. The report "Curriculum 78: Recommendations for the Undergraduate Program in Computer Science"⁹ specifies that the course at least be strongly recommended and should be required of all computer majors if sufficient material is not included in other required courses in the program. This recommendation constitutes one of the major differences between the report "Curriculum 78" and the same committee's report ten years earlier¹⁰ which did not include a course on computers and society.

Another recent development within ACM is also worth noting. The Elementary and Secondary Education Curriculum Committee, in December, 1978, began work on recommendations for material on computer impact to be taught in elementary and secondary schools. It is too soon to comment on this development, but the effort itself is significant.

CERTIFICATION

The Institute for Certification of Computer Professionals (ICCP), established in 1973, has exercised concern over the societal responsibilities of computer professionals in a variety of ways. In 1977, ICCP offered the Certificate in Computer Programming (CCP) examination which attempts to test minimum knowledge requirements for persons holding senior programmer positions. The content outline for the examination contains a section on computers and society.

ICCP also offers the Certificate in Data Processing (CDP) examination which is oriented toward management positions. Anyone who passes these examinations is expected to subscribe to codes of ethics and good practice (which were developed by ICCP) in order to obtain the certificate. Although relatively few (less than 20,000) persons hold the CDP and only several hundred hold the CCP, there is active discussion on the use of these examinations (and examinations for other positions under consideration by ICCP) as part of the minimum requirements for appointment or promotion to certain positions. This issue is a complex one with ramifications that are not subject matter for this paper. However, the institution of such requirements as criteria for obtaining a job title or, more importantly, for carrying out the duties of a position, will increase the awareness of computer professionals of their societal responsibilities.

For example, consider the development of an automated diagnosis program. A number of them have been implemented and more sophisticated ones are likely. Some of these programs have resulted from the interaction of medical personnel who knew relatively little about computers and programming and one or more programmers who knew relatively little about medical applications and who may not have known as much as one would like about programming, data structures, and relevant techniques. Who is to blame if a patient suffers because of a wrong diagnosis? Does the programmer have any responsibilities in this regard? Should a programmer be required to pass an examination such as the CCP before being placed in charge of the programming project? Certainly, there is no guarantee that persons will be more aware of societal responsibilities because they obtain a new title, but the intent would be to make them more accountable, hence, we hope, more concerned.

CODES

Within the last few years, codes of ethics, conduct, or good practice have been established by a number of computer societies (e.g., ACM, ICCP) which suggests the memberships' growing awareness and concern over the impacts of computers on society. Members are expected to subscribe to the codes, but computer societies have found it difficult if not impossible to enforce their codes. In an attempt to remedy this problem, the ACM Council at its meeting in June, 1978, adopted enforcement procedures for its Code of Professional Conduct after extensive discussion both at this meeting and at previous ones. The text of the procedures was published in the August, 1978 issue of the *Communications of the ACM*.¹¹

NSF GRANT TO ACM

In 1974, work began on a project of ACM's Education Board funded by NSF and entitled "A Study of Computer Impact on Society and Computer Literacy Courses and Ma-

terials." The project had three basic objectives:

1. To review and catalog materials related to computer and society courses and programs and to provide methods for dissemination of such information.
2. To identify minimum-knowledge-level requirements for computer literacy.
3. To develop behavioral objectives for various types of computer and society courses as well as develop decision mechanisms for materials for such courses.

Bibliography

The project committee's efforts to achieve the first objective resulted in 1976 in an annotated bibliography of over 2000 selected entries (dated, for the most part, after 1968) which was intended to provide resource material for teachers of both computer impact on society and computer literacy courses. A hierarchical information storage and retrieval system was developed and implemented under Charles H. Davidson at the Engineering Computing Laboratory, University of Wisconsin—Madison. Entries were coded by area, function, approach, level and type. Retrieval of all entries satisfying combinations of these categories was possible.

A review of the kinds of material written in the area of computers and society, and included in the bibliography, can be found in Austing, Cotterman, and Engel.¹² Briefly, the review indicates that over 90 percent of the material was expository in nature, ranging from the wonderment (e.g., computers are superior to humans) and fear (e.g., computers will cause vast unemployment) found in earlier literature to the more detailed discussions of applications found in later material. A very small percentage of the literature contained technical perspectives. We hope this percentage is increasing as suggested by the examples cited in previous sections.

While the bibliographic work was underway, the project committee participated in panel sessions and held open hearings at conferences, in addition to discussing ideas and experiences of interested and knowledgeable professionals from education, industry, government and selected groups from the public-at-large. These efforts provided substantial input relative to course content, educational level, minimum requirements and objectives of courses in computer impact on society and computer literacy.

The second phase of the project, also funded by NSF, began in July, 1977 and will continue into 1979. The following three activities were specified:

1. Further develop and refine the bibliography and the information storage and retrieval system supporting it.
2. Organize and conduct a workshop on computer impact involving individuals other than computer professionals.
3. Develop and disseminate a collection of position statements in the area of computer impact by various concerned professionals.

The workshop

The workshop played a major role in the approach to the other two activities. It was held on July 17-19, 1978 in Williamsburg, Virginia. The 33 invited participants represented as diverse a group as possible, the only common bond being an expressed interest or experience in one or more aspects of the societal impact of computers. Austing and Engel report on the organization, content and results of the workshop.¹³ Some of the results and recommendations lend emphasis to the fact that there are renewed efforts lately in the area of education regarding computers and society.

Specifically, a list of topics was developed from which course material could be used. This list did not constitute a taxonomy, nor was one expected at this stage. However, whenever a topic is selected for presentation, it is necessary to determine the level (impart knowledge, instill an attitude, develop a skill), audience and approach (lecture, discussion, examples only, technical aspects, project, etc.). Further, when developing material possible emphases must be considered, such as

1. Skills citizens need for coping with the computer impact.
2. Computer impact on work patterns and relationships within organizations.
3. Degree of high technology comprehensibility of professionals.
4. Computer impact on the political/economic/legal process.
5. Inherently socially problematic nature of computer technology.

The growing interest and public support for the computer literacy concept was cited as a reason to consider including suitable material at the pre-college level. More strongly, the workshop participants recommended that *all* high school graduates be computer literate. The rationale focused on the necessity to learn about computers before formal schooling is completed because computers do (and will continue to) permeate our society. To achieve the desired computer literacy, graduates should have knowledge of the following:

1. Historical perspective of computing.
2. Computer anatomy (includes parts of a computer, how computers work, algorithms, problem solving, system capabilities).
3. Uses of the computer (both types of uses such as information storage, simulation, DP, communications and areas such as business, science and technology, education, health care).
4. Social implications (such as careers, organizational changes, privacy).
5. Futuristics (for example, trends in artificial intelligence and robotics, innovation and new technology, communications).

6. Introductory level skill in algorithm design and programming (only if adequate access to computers is available).

At the college level, two kinds of courses were specified, one for the general education requirement (designed as a survey type course) and one for the computer science major (designed to help students learn to carefully analyze the social settings in which computing is used, to understand what social and historical forces give rise to different systems uses and designs, and to realize the impacts and value conflicts upon users and non-users of computers). The workshop participants recommended that a Computers and Society course, or equivalent knowledge, be a part of the education of every college student. Rationale cited the imperativeness for college-educated persons to have more than a superficial knowledge of computers and their impact because of the technical/information explosion and the corresponding expansion in computer usage, the need for the corresponding expansion in computer usage, the need for awareness about computers in consumer life and the need for a more informed citizenry in decision making where computers are involved.

Imparting knowledge to computer professionals and reaching them are two problems addressed by workshop participants. Delivery mechanisms (e.g., courses, professional development seminars, tutorials) must be intensive, be appropriate to the job environment and be geared to very specific groups. Computer professionals have a difficult enough time trying to keep up-to-date in their specific area. They will not always find the time to attend courses or seminars for the purpose of keeping abreast of the impact of computers in such wide-ranging areas such as transborder data flow, electronic funds transfer, communications and health care. They need a set of principles rather than broad knowledge to apply to societal impact issues relevant to applications in which they are involved. These principles can be developed through a highly-concentrated approach to a specific issue, possibly by means of a case study approach or by simulations and role playing.

The general public needs to be educated about computers but effective means are difficult to identify. Every computer professional can play a role in educating the public and, by so doing, put into action various aspects of social consciousness that would otherwise just be words. The public is inundated by computer applications (e.g., charge accounts, graphics on TV, advertisements, electronic games). Computer professionals could not only offer continuing education courses to transmit correct information to portions of the public, but also exert whatever influence they have (possibly through computer societies and associations) on business and government to promote policies encouraging proper use of computers where the public is involved. Workshop participants identified a number of specific information media and suggested uses of media groups for disseminating material about computers to the public. However, no readily available solutions were found.

One of the outcomes of the workshop was to broaden the base of material in the bibliography developed in the first phase of the grant by incorporating references from fields other than computer-oriented ones (e.g., philosophy, health care). The bibliography was also updated by adding more current references and, in some cases, deleting some which have been superseded or which were not as good a source as another reference. The bibliography now contains over 3000 entries and is a much more valuable resource to anyone intending to offer a course in the computer impact on society or computer literacy. After the termination of the grant in the spring of 1979, ACM will begin maintaining the bibliography. Information concerning its content and use can be obtained through ACM.

CONCLUSION

The activities previously described indicate the kinds of efforts underway in the area of the computer impact on society. They are at the level of affecting professional organizations of computer people and having an impact on schools, especially elementary and secondary ones. Research results are being disseminated more widely than before which, in turn, should encourage more creativity and development by people concerned with societal issues. All of these are hopeful signs of renewed emphasis on important issues in computers and society.

REFERENCES

1. *Proceedings of ACM 78*, Washington, D.C., December 1978, pp. 433-459, 659-683.
2. Arbib, M. A., *Computers and the Cybernetic Society*, Academic Press, Inc., New York, 1977.
3. The Ombudsman Committee on Privacy, D. A. Gerberick (Chm.), *Privacy, Security, and the Information Processing Industry*, ACM Los Angeles Chapter, 1976.
4. Gottlieb, C. C., and A. Borodin, *Social Issues in Counting*, Academic Press, Inc., New York, 1973.
5. Hoffman, L. J., *Modern Methods for Computer Security and Privacy*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1977.
6. Mowshowitz, A., *The Conquest of Will*, Addison-Wesley Publ. Co., Reading, Mass., 1976.
7. Mowshowitz, A., *Inside Information*, Addison-Wesley Publ. Co., Reading, Mass., 1977.
8. Weizenbaum, J., *Computer Power and Human Reason*, W. H. Freeman & Co., San Francisco, 1976.
9. "Curriculum '78: Recommendations for the Undergraduate Program in Computer Science." A Report of the ACM Curriculum Committee on Computer Science. (Eds. R. Austing, B. Barnes, D. Bonnette, G. Engel, G. Stokes), *CACM*, in press.
10. Curriculum Committee on Computer Science (C³S), "Curriculum '68, Recommendations for Academic Programs in Computer Science." *CACM* Vol. 11, No. 3, March 1968, pp. 151-197.
11. "Procedures for the enforcement of the ACM Code of Professional Conduct," *CACM* Vol. 21, No. 8, August 1978, pp. 708-709.
12. Austing, R. H., W. W. Cotterman and G. L. Engel, "Literature Resources and Computer Impact on Society and Computer Literacy," *Proceedings of the Computer Science and Engineering Curricula Workshop*, Williamsburg, Va., June 1977, 55-59.
13. Austing, R. H. and G. L. Engel, "Computers and Society: Report of a Workshop," *Proceedings of COMPSAC 78*, Chicago, Ill., November 1978.

Interactive monitoring of computer-based group communication*

by KATHLEEN SPANGLER, HUBERT LIPINSKI and ROBERT PLUMMER

Institute for the Future
Menlo Park, California

INTRODUCTION

Biofeedback is a procedure for monitoring unconscious or involuntary bodily processes and making them perceptible to the senses. The objective is to increase consciousness and therefore control of the bodily processes as a means of improving health. This paper is not about health or biofeedback, but it does describe a procedure that is perhaps analogous to biofeedback—the interactive monitoring of group communication through computers.

Over the last decade, a number of computer programs have been developed to support small-group communication; these include PLANET, EIES, PARTYLINE & DISCUSSION, CMI, CONFER, MINT and RIMS.** Since these programs act as a kind of gatekeeper for the communication process—directing participants to appropriate “activities” and ordering their “messages”—they can easily be extended to record important features of the group’s communication patterns. For example, during its development phase, the PLANET system included monitor software that collected and analyzed information about the time users entered and left a PLANET activity, the number of public and private messages sent, the number of words in public and private messages, the use of commands, the typing time and the number of computer resource units used, among other statistics. This information was used to evaluate the impact of the medium on group communication and it revealed several different styles of computer conferencing among users of the system.^{2,3,4}

The information from the PLANET monitor software was not available to those who were participants in PLANET conferences during their discussions. However, there is no technical reason why such information could not be made available to users of computer conferencing. It could then serve as a kind of biofeedback about the group communi-

cation process, increasing the group’s consciousness of its communication patterns and thereby giving them more control over those patterns.

An interactive monitor would allow the group to spot possible communication barriers—nonparticipants, isolated subgroups and poor access to important resources, for example. The group leader or the group as a whole could then determine whether some intervention would reduce these barriers. An interactive monitor could also provide insights into the efficiency of the group communication process; it could display to the group its volume of communication, the timeliness of information exchange, and the cost of communication. Over a long period of time, such a monitor could chart the history of the group, displaying informal changes in the group’s organization that may suggest the need for formal changes. In short, an interactive monitor could be used to evaluate and alter the group’s communication process.

IMPLEMENTING THE MONITOR—THE HUB SYSTEM

At the Institute for the Future, we have begun to implement this concept for groups with a specific communication purpose—the construction of large-scale policy models. We have designed a communication system known as HUB. The HUB system, which resides on the Bolt, Beranek and Newman PDP-10 computer in Boston, is a four-part communication system. The four parts include

- A computer conferencing facility that uses the PLANET program.
- A graphic communication facility (the “shared visual space”) that allows users to create graphic images from picture primitives and to comment on these images.
- A “program workspace” that allows users to run a variety of computer programs and to discuss them while they are being seen or to comment on them later.
- A “document workspace” that allows a group of users to develop a document jointly by making changes on a single version and annotating those changes in comments.

A HUB “switcher” makes it easy for users to access and

* This paper results from work supported by the National Science Foundation, Division of Mathematical and Computer Sciences under Grant No. MCS77-01424.

** PLANET was developed by the Institute for the Future; EIES by Murray Turoff at the New Jersey Institute of Technology; PARTYLINE & DISCUSSION by Rod Renner and Murray Turoff; CMI by Bell Canada; CONFER by University of Michigan; MINT by the Nonmedical Use of Drugs Directorate in Canada; and RIMS by Murray Turoff for the Federal Office of Preparedness. For a description of these systems, see Reference 1.

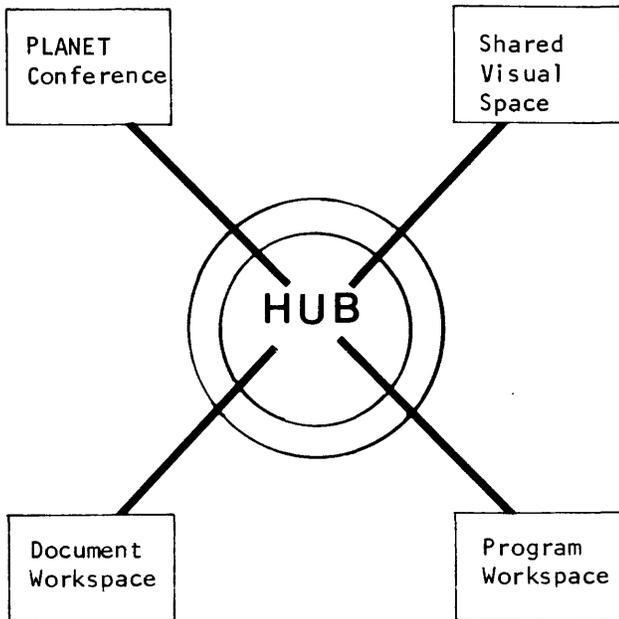


Figure 1—The HUB system.

move among each of these four systems, as illustrated in Figure 1. It is in this switcher that the HUB monitor is located and it is here that users interact with it.

IMPLEMENTING THE MONITOR—WHAT TO MONITOR

In designing the monitor, of course, one of the central questions is, What should be monitored? To answer this question, we began by considering the complex set of variables that combine to shape group communication. One useful way of grouping these variables is according to three categories: (1) group process, (2) individual communication styles and (3) task performance.** These categories suggest an outline for identifying the kinds of information that may be useful to a team of modelers in the course of a HUB conference.

Consider first the *group process* variables. One of the most important issues for groups of all kinds is participation. The level of participation of various group members reflects leadership patterns, productivity of the group, and the availability of resources to the group. It can also reveal subgroups who may have certain unique communication needs, but also unique problems in communicating with the larger group. In a HUB conference, we imagine that the following information about participation would be useful:***

- Volume of communication per day, week, or month for

** For a review of taxonomies of group communication through electronic media, see Reference 5.

*** Obviously, this list is not exhaustive; it does, however, include the major patterns that can be collected or calculated automatically.

the entire group (in both comments and words), for the entire system and for each part of the system (i.e., PLANET, program workspace, etc.).

- Length of comments in each part.
- Distribution of participation over time.
- Distribution of participation within each part and across parts of the HUB system.
- Dominant participants for each part; for the system as a whole.
- Percent synchronous participation for each part.
- Cost of communication per unit time.

Each *individual* in the group will have his or her own characteristic style of communication. The monitor should be able to provide a profile of this style. Many of the variables in such a profile are similar to those for the group as a whole. For example, the profile should probably include

- The volume of communication generated by the individual compared to the volume of the group as a whole.
- The individual's frequency of communication.
- The changing position of the individual in the distribution of participation over time and across parts.
- His/her use of programs compared to others in the group (number of runs initiated in the program workspace).
- Percent time spent in synchronous interaction.
- His/her private communication network (in PLANET).
- The "accessibility" of an individual to others in the group.†
- His/her cost to use the system per unit time, compared to the total group.

The *task performance* variables are perhaps most specifically related to the unique communication problems of modeling groups. These problems include a lack of communication between builders and users of models due to organizational barriers, perceptual barriers and different levels of skill in working with computers and mathematical concepts; a tendency for individual members of the modeling team to "do their own thing" rather than to work in a collaborative style; problems of documentation; and problems of confidence in the model due to difficulties in interpreting results, in understanding the model structure and in recognizing factors that are not considered in the model, all of which lead to questions about the validity of the model.§

Much of the information that would be useful in diagnosing and correcting these problems cannot be easily collected by an automatic monitor.* However, since the various parts of the HUB system are really designed to facilitate different

† "Accessibility" in a HUB conference can be measured by the time lag between the sending of a message and its receipt by any particular individuals. The longer the average time lag, the lower the accessibility.

§ For a full discussion of communication problems in the modeling process, see Reference 6.

* For example, an indicator of difficulties arising due to perceptual or disciplinary barriers is the use of specialized jargon by subgroups. A content analysis of the transcript would reveal such a pattern; however, it is not done easily by an automatic monitor.

tasks, the usage patterns for each of these parts provide a task-related profile of the group's communication. For example, the volume of activity in each part is likely to change as the focus of the project shifts from one task to another. These shifts can be monitored over time; individual participation patterns can be overlaid on these volume patterns. Such information would demonstrate who is involved in what types of activity at any point in time.

It will also be useful to collect specific data for each of the subparts to clarify issues of task performance. For example, it will be useful to know

For the shared visual space:

- The number of versions of each graphic image.
- The ratio of comments to graphic commands.
- The length of time to produce a completed graphic image.
- The frequency of use of graphic commands.
- The cost to produce a completed graphic image.

For the program workspace:

- The number of runs for each program.
- The number of comments per run.
- The number of synchronous runs.
- The number of times each run is reviewed.
- The number of unique input files.
- The ratio of program lines to comment lines.
- The cost of each run.

For the document workspace:

- The number of text changes over time.
- The ratio of text changes to comments.
- The number of times a document is printed in full.
- The average line length of text changes.
- The number of synchronous text changes.
- The number of reviews of text changes.
- The cost per text change (including and excluding comments).

In addition to this information, the group can evaluate its overall task performance by considering some of the data about the "time effectiveness" of information exchange in various parts of HUB. Thus, the HUB monitor should collect statistics on the length of time between any program run and its review by any other participant; between any graphic change and its review by any other participant; between any document change and its review by any other participant; and between comments in any of the four parts and their review by any other participant. Such measures will provide a very dynamic view of the communication process.

Many of the communication variables in these four categories are related; in fact, all of them can be calculated from a relatively small number of statistics, which the HUB monitor is designed to collect. In addition to these statistics that are collected unobtrusively, the monitor files include space for two other types of information: (1) hand-coded information about the message and (2) responses to structured

questions. Thus, it would be possible, for example, for someone to content-analyze a series of messages (e.g., for the use of jargon) and hand-code this information into the monitor to be displayed with network patterns for the messages. The second feature—responses to structured questions—would allow the conference organizer to poll the users' feelings about the group process, which would then also be available to supplement the statistics on communication patterns.

IMPLEMENTING THE MONITOR—HOW TO DISPLAY

The procedures for collecting monitor statistics are quite simple. The procedures for displaying them to the group are more complex. First, there are choices about how to represent information about all of the communication variables noted above. Some of these can be shown as simple ratios, but most of them call for some form of graphic display. Some of these, such as network graphs that must be constructed to show the strength of links, can be quite complex. Second are choices about how the users interact with the monitor—Do they automatically get a display of 20 or 30 pre-set graphs and figures or do they specify what they want to see? And if they specify what they want to see, how do they know what they can display? Closely related is a third question about how much control users have over what can be displayed. On one hand, there may be only one fixed representation for each major variable. On the other, the users might be able to manipulate the representation to emphasize certain aspects of it, or they might even be able to combine two or three variables to suggest correlations. Finally, there may be certain variables that are pre-selected as significant; in this case the monitor could automatically notify the group when this variable reached some critical level.

Because we are still in the process of developing the HUB monitor, we have not made all of these choices. However, it seems clear that the way in which the variables are displayed could make a significant difference in the impact of the monitor data on the group. Furthermore, it seems clear that there are a number of creative ways to display the information that might speak directly to problems that plague modelers. For example, labeling people by some label other than their names could provide some interesting insights. If a graph showing distribution of participation were labeled by disciplines rather than names, it might become clear, for example, that subgroups are developing along disciplinary lines. The group could then assess whether this pattern is appropriate for the phase of the activity or whether some important insights are being lost due to lack of effective communication between the subgroups. Thus, it seems that a primary criterion for decisions about how to display the monitor data should be flexibility to design displays that do, in fact, address particular problems that the group may have.

AFTER IMPLEMENTATION—THE IMPACT ON COMMUNICATION

We have suggested that an interactive monitor could have an effect on the group communication process that is comparable to that of biofeedback—namely, that it can increase consciousness and therefore control of the group's communication patterns. But we can now speculate about some more specific impacts—both negative and positive—of the type of monitor that we are implementing for the HUB system.

One possible effect is that there will develop a new role in the communication process for a group facilitator. We have already noted the tendency for such a role to develop in PLANET conferencing.³ In a HUB activity, this role might include responsibility for checking the monitor regularly and interpreting the results for the group. The facilitator might use the private message mode to counsel individual group members about their participation patterns or might use the displays in group sessions to consider the implications of the patterns for complaints or difficulties that the group may be experiencing.

A second likely effect of the monitor would be to encourage more experimentation with group structures. As the group becomes more conscious of its communication patterns, it may wish to intervene in these patterns by tinkering with roles and responsibilities and then following the monitor to observe the effects of the experimentation.

It seems very likely that such a monitor could and would be used to evaluate the group's performances and the performances of individual members. Certainly, it would make an individual's contribution to the group more apparent; it would also provide a reading on the more elusive measures

of performances, such as ability to get along with one's colleagues. Such an evaluation tool may indeed seem attractive to someone who is faced with objectively evaluating a group of people; however, there are also real dangers here. First, there is the possibility that the very existence of the monitor will inhibit some communication, and it may actually discourage the participation of some people altogether. Also, the monitor may encourage some people to "perform" for the monitor, to alter their behavior just to show up well in the monitor data. Finally, the use of the monitor could encourage evaluation of the *wrong* aspects of performance, particularly if the data that are easily collected are not the best data for evaluation.

The way that the HUB monitor will be used by groups and the effects on modeling communication remain to be assessed. Our plans call for field tests of HUB over the next two years. The role of the monitor will be an important focus of this evaluation.

REFERENCES

1. Johansen, Robert, Jacques Vallee and Kathleen Spangler, *The Camelia Report*, Institute for the Future, Report R-37, 1977.
2. Vallee, Jacques, et al., *Group Communication Through Computers*, Vol. 3, Institute for the Future, 1975.
3. Vallee, Jacques, et al., *Group Communication Through Computers*, Vol. 4, Institute for the Future, 1977.
4. Johansen, Robert, et al., *Group Communication Through Computers*, Vol. 5, Institute for the Future, 1977.
5. Johansen, Robert, Richard Miller and Jacques Vallee, "Group Communication through Electronic Media," *Educational Technology*, August 1974, pp. 7-20.
6. Lipinski, Hubert, Roy Amara and Kathleen Spangler, "Communication Needs in Computer Modeling," *Proceedings of the Winter Simulation Conference*, Miami, Florida, December 1978.

The status of women in health science computing

by LYNN L. PETERSON

The University of Texas Health Science Center at Dallas
Dallas, Texas

The computing profession is now just over 25 years old. During this 25-year period, women increased in the labor force from 34 percent of all women ages 16 and over in 1950 to 46 percent in 1975.¹ Since computing developed during the period when females became a larger proportion of the work force, the question arises as to whether women are equitably represented in this profession. The present decade has brought a consciousness-raising with respect to woman's role in the professions in general, with an interest in the role of women in the computing profession following naturally. The health care industries have become an ever more important and visible component of the U. S. Gross National Product, resulting in increasing scrutiny of the conduct within the health care professions. With this background, the question of the status of women in Health Science Computing appears to be a logical one to consider.

The ACM-SIGBIO Symposium on Health Computing Careers was held in June 1977 to provide a broad survey of present and future health computing careers. A basic understanding was that Health Science Computing was not just computer science practiced and applied in a Health Science Center setting, and it was not just medicine utilizing computers as one of the many specialized tools at its disposal. Health Science Computing was seen as combining capabilities of computer science and medicine and emerging as a discipline of its own to address areas of patient care, research, education and service.² However, if our concern is women's role in Health Science Computing, we can gain both knowledge and perspective by examining the two major components of its root—computer science and medicine.

COMPUTER SCIENCE AND WHERE WOMEN FIT

To look at those who are in the process of preparing for a career in computer science, we refer to a study done in the fall of 1975 by Mamrak and Montanelli.^{3,4} Their findings show clearly that the number of students enrolled in computer science programs has increased dramatically since the early 1970s. They also show a small but statistically significant increase in the enrollment of women in computer science at the bachelor's level during the period from 1971-1975. But a look at the numbers of students in bachelors, masters and doctoral degree programs shows a moderate decrease in the percentage of women enrolled and graduating as the degree level increased. It is possible that one of

the reasons for this is the lack of role models to encourage aspiration to higher-level career attainments. For the 1971-75 time period, the sex distribution of computer science faculty indicates a clear lack of availability of women role models in the higher academic ranks.⁴

Next, let us look at those who are actually engaged in computing as a career. In 1975, women made up 39 percent of those in the labor force.¹ At the same time, they comprised approximately 31 percent of those employed in computing at all levels.⁵ In the data entry positions, 99 percent were women. In programming and analysis, the percentage of women holding these job titles was no higher than 20 percent. An explanation of this situation could be the one offered by Weber and Gilchrist. Approximately 34 percent of the baccalaureate degrees awarded by American colleges and universities in 1971 went to women. If letters, nursing, fine arts, applied arts and home economics are excluded, the percentage of women drops to 26 percent. It is possible then that not enough women are receiving baccalaureate degrees in fields appropriate for entry into the computing profession to raise the percentage significantly through the mechanism of providing qualified applicants.

Another view of the role of women in the computing profession involved a survey of 425 women (77 percent responses) in data processing conducted in 1975 by Asprey and Laffan to determine how women felt about their status and their potentialities.⁶ The survey showed that 68 percent felt they had equal status with their colleagues in pay, promotions and overall. Some 70 percent of the respondents felt they had opportunities to advance to a senior level and more than half (actually 56 percent) felt opportunities to hold highly responsible management positions were available to them as women. The survey showed that the availability of part-time employment or of work on a contract basis made the computing field attractive to women with family responsibilities. Betty Maskewitz, now Director of BCTIC at the Oak Ridge National Laboratories, is quoted as saying "computing is a wonderful field for women—an exciting field for anyone regardless of sex or any other stupid qualifier."

While the computing field received a rather positive subjective evaluation by women employed in it, one might wonder if this field of endeavor is then hospitable to all. A recent *Datamation* article⁷ shows that the computing profession is clearly not for everyone, that those individuals who are

attracted to computing as a career do not represent a cross-section of working professionals. A survey reported in the article shows that data processing people show a strong need for personal accomplishment, for learning and developing beyond where they are, for being stimulated and challenged; in short, a high "growth need." In fact, of all the professions surveyed, computing professionals showed the highest level of "growth need." At the same time, data processing people show a negligible need to interact with others, the lowest "social need" of all professions surveyed. In fact, it is felt that, if asked, most programmers would probably say that they preferred to work alone in a place where they couldn't be disturbed by other people. Computing professionals then do not merely represent a cross-section of the work force, but have some distinctive characteristics. In the absence of arguments to the contrary, we can assume that these personality traits are represented in both sexes.

MEDICINE AND WHERE WOMEN FIT

A look at medicine, the other major component of the root of Health Science Computing, shows that women in academic medicine, the area of medicine most likely to be in contact with Health Science Computing professionals, do not fare as well. Judith Braslow of the AAMC staff stated⁸ that of 48,500 faculty members in the nation's medical schools, 15 percent were women. Furthermore, of the 38,973 full-time faculty, 10 percent of the M.D.s and 15 percent of the Ph.D.s were women. Subjectively, their situation was felt to be improving. These women felt freer to move as career advancement dictated than was previously the case, and many were willing to make the sacrifices involved in moving to administrative positions. Many were now being asked to serve on committees where the female point of view was desired but as a result were being greatly over-committed. However, they felt that in spite of the progress made there was still a dearth of women faculty role models.

MEDICAL COMPUTER SCIENCE

In order to qualify for many of the positions in the computing field, one is required to have the capacity for logical thinking instead of a specific preparation. Furthermore, for computing people such as data entry personnel, computer operators and junior programmers without a defined specialty, training in a specific applications area of computing is not required. It is, however, when we address the preparation of professionals in the specialty area of Health Science Computing that we are led to more academic considerations.

The academic training grounds of many Health Science Computing professionals are programs called by such names as Medical Computer Science, Medical Information Science, Medical Informatics or any of a variety of terms. In 1977, a survey of existing training programs in Medical Computer Science was conducted to update an earlier published study.⁹

An interim report on this survey,¹⁰ presented at the SIG-BIO Symposium on Health Computing Careers, is available from, and will be kept current by, The University of Texas Health Science Center at Dallas. This survey shows that, according to Medical Computer Science practitioners and educators, job opportunities in health computing abound for adequately trained individuals. This situation shows no signs of slackening in the near future. The survey showed there were 92 percent as many students enrolled in Medical Computer Science programs in 1977 as had graduated since 1970. The 358 people enrolled were to be thrust into the job market in the succeeding years, each having up-to-date training in medical computer science. However, they account for less than one-sixth of the projected requirements for biomedical computer specialists.¹¹ Computer applications in medicine seem to be on the increase without a commensurate increase in the development of programs in Medical Computer Science to supply the personnel. If the projected need is accurate, this situation will have to change or else the bulk of the personpower recruited for medical computer science positions will have to be trained on the job. Thus, whatever route is used to approach jobs in medical computer science, job opportunities seem bright whether resulting from the growing requirement for accurate information posed by various programs of utilization of health care and assessment of the quality of health care, or resulting from the increasing clinical applications of the computer.¹² Thus, for the foreseeable future, people with specialized training in Medical Information Science or Health Science Computing will have every expectation of finding employment which makes good use of their background.

STATUS OF WOMEN IN MEDICAL COMPUTER SCIENCE TRAINING PROGRAMS

In an attempt to investigate the status of women in the Medical Computer Science training environment, a pilot survey was conducted. A complete survey of the Medical Computer Science programs with respect to women's role is planned as a follow-up to the 1977 survey mentioned above. Those Medical Computer Science programs identified in the 1977 survey will be contacted with the intention of collecting data on the number of men and women currently enrolled in M.S. programs, in Ph.D. programs, in Post-doctoral programs, already graduated from M.S. and/or Ph.D. programs and currently serving on the faculty. The survey is planned for the spring of 1979 with results to be presented at NCC 79.

Since the number involved in the pilot study was small, impressions only from the pilot survey can be given, with more objective data to come from the survey itself. It can be seen from even this limited sample that, within Medical Computer Science departments, the role of women in these programs covers a wide spectrum. One program has no women faculty; one, representing the other end of the wide spectrum, shows a two-to-one male-to-female-faculty ratio. Several programs have no female graduate students: one, representing the other end of a wide spectrum, estimates

that its student population has a one-to-one male-to-female ratio. In one program, all graduates have been male; in another, all have been female.

From these limited glimpses, it appears that a program's openness to women depends heavily on the individual graduate program and that generalizations over all Medical Computer Science graduate programs will not apply. A comment from one spokesperson suggests that a program's openness to women depends also on whether the individual women want to give what it takes to meet the challenge of the graduate program. In speaking with representatives of various graduate programs, it was interesting to find frequent mention of a man or men who gave encouragement to a number of women during various stages of their academic development. Such individuals should obviously be recognized and encouraged. While female role models may be lacking in many areas, this type of person can serve a role model for both sexes.

ENCOURAGING GROWTH OF WOMEN'S ROLE IN HEALTH SCIENCE COMPUTING

These glimpses of women in computer science and in Medical Computer Science Training Programs shed some light on what might be done to encourage the growth of women's role in Medical Computing. Some positive suggestions:

- Encourage women who are inclined toward an interest in fields preparatory to computing to pursue that interest and get good high school and undergraduate training. Sex stereotypes which suggest that girls should not be good in math, for example, should not be perpetuated.
- If you are a woman in the field of Health Science Computing, consciously serve as role model whenever possible. Be visible. Talk about the problems and possibilities of women in this field. If you like your work, say so.

- Finally, be the very best you can in what you do. Don't shy away from a challenge—enjoy it!

By these means, and ones which others will, we hope, continue to suggest, we may ensure that qualified, interested women are not discouraged and/or prevented from engaging in careers in Health Science Computing.

REFERENCES

1. "Women Workers Today," U. S. Department of Labor, Employment Standards Administration Women's Bureau, 1976.
2. Peterson, L. L.. "Session Summary, Administration of Health Computing," *ACM-SIGBIO Symposium on Health Computing Careers*, Dallas, TX, June 1977.
3. Montanelli, R. G., Jr., and S. A. Mamrak, "The Status of Women and Minorities in Academic Computer Science," *Comm. ACM*, Vol. 19, No. 10, 1976, pp. 578-581.
4. Mamrak, S. A., and R. G. Montanelli, "Computer Science Faculties: The Current Status of Minorities and Women," Personal Communication, 1977.
5. Weber, R. E., and B. Gilchrist, "Discrimination in the Employment of Women in the Computer Industry," *Comm. ACM*, Vol. 18, No. 7, 1975, pp. 416-418.
6. Asprey, W., and A. W. Laffan, "Women Speak Out on DP Careers," *Datamation*, August 1975, pp. 42-43.
7. Couger, J. D., and R. A. Zawacki, "What Motivates DP Professionals?" *Datamation*, September 1978, pp. 116-123.
8. Braslow, J., *Association of American Medical Colleges Symposium on Women in Medicine*, New Orleans, LA, October 1978.
9. Mize, S. G., and D. J. Mischelevich, "Current Status of Medical Computer Science Programs in Biomedical Engineering Programs in U.S. Medical Schools," *J. Clin. Comput.*, January 1973.
10. Bayard, J. J., D. J. Mischelevich, M. A. Ball and J. S. Reisch, Interim Report—June 12, 1977, "Survey of U. S. Medical Computer Science Programs," *ACM-SIGBIO Symposium of Health Computing Careers*, Dallas, TX, June 1977.
11. Sias, F. R., Jr., "An Analysis of the Job Market for Biomedical Computer Scientists," *14th Annual Southwestern ACM Conference*, April 22-24, 1976.
12. Wasserman, A. I., "Job Opportunities in Medical Information," *ACM-SIGBIO Symposium on Health Computing Careers*, Dallas, TX, June 1977.

Women and minorities in the computer professions

by HELEN M. WOOD

National Bureau of Standards
Washington, D.C.

INTRODUCTION

Generally, while full equity may not have been achieved by women and minorities in computer occupations, the belief has been that their employment status was better and conditions were more favorable for full utilization of skills due to the relative youth of the computer field. The federal government, in particular, has been involved in computers since their beginnings, and in addition has taken a stance in support of both equal employment opportunity (EEO) and affirmative action.¹⁹ This position has been backed by laws (e.g., Civil Rights Act of 1964) and Executive Orders (e.g., E.O. 10590 and E.O. 11375) prohibiting discrimination on the basis of race, color, religion, sex, national origin, or age in federal employment. Accordingly, it is reasonable to assume that women and minorities would have more opportunities within the federal government, especially within the federal computer-related professions.

This report examines available statistics in order to assess the status of women and minorities employed in computer-related professions, both in the U.S. labor force, in general, and the federal workforce, in particular. The intent is to provide "gross" indicators of the current situation and perhaps stimulate additional analyses. The focus is on labor market experience (e.g., employment and unemployment rates, relative salaries), rather than the utilization of women and minorities from a human resources perspective. This paper will primarily address labor market-related issues. For a report on utilization of women and minorities in science and engineering, including an examination of such factors as science abilities and the relationship between career plans and career outcomes, see Reference 14. Overt or covert discrimination and societal, cultural or other causal factors which have been addressed elsewhere (e.g., References 7 and 8) are outside the scope of this report, as is any attempt to identify determinants for improvement.

Due to varying data collection and tabulation techniques, tabulated figures contained in this report may not add to totals. Likewise, totals may not necessarily agree across reports cited.

U.S. WORKFORCE

The minority portion of the total 1974 U.S. workforce was approximately 11 percent. Women comprised nearly 40 per-

cent of the workforce.¹⁴ The next section will examine the composition of the U.S. scientific and engineering workforce.

Scientists and engineers

The NSF studies show that in 1974 the scientific and engineering population of the United States was almost two million in size, or just under 1 percent of the total U.S. population.¹³ Of this total, which included individuals not in the active labor force (e.g., retired scientists), 1,100,000 were engineers and 900,000 scientists. Of the total population of scientists and engineers (S/E), about 1.7 million people were in the labor force (i.e., employed or seeking employment). Table I describes U.S. S/E employment by sex and race.¹³

In 1974, there were approximately 185,000 women trained in the S/E fields. Almost one-half of these women (89,000), however, were not participants in the labor force. That is, they were not employed and were not seeking employment. Those women who were in the labor force (about 96,000), represented six percent of the total number of employed scientists and engineers in the United States. The large number of women in S/E who were not in the labor force contrasts sharply with the figures for all male scientists and engineers. Only 12 percent (222,000 out of nearly 1.8 million) of all male scientists and engineers were not in the 1974 labor force.^{13,14}

Of the women not working, 20 percent had retired. The others were unemployed due to family responsibilities, ill health and "other reasons." Some of those "other reasons," including perceived career roadblocks and discrimination, were among the issues discussed at a 1977 American Association for the Advancement of Science symposium on covert discrimination and women in the sciences.¹⁵ Additional insights into factors contributing to low female labor force participation may be found in Kreps' report on American women in the work force.⁷

In 1974, minorities comprised less than five percent of all scientists and engineers (about 87,000). Minority scientists and engineers had a higher rate of participation in the labor force, however, than their non-minority counterparts. While about 15 percent of the non-minority scientists and engineers were not in the 1974 labor force, the figure for minorities was 8.5 percent.¹⁴ The higher labor force participation rate

TABLE I.—Employment of Scientists and Engineers, by Sex and Race: 1974

Field	Total	Sex		White	Total Minorities
		Male	Female		
Total	1,662,000	1,566,000	96,000	1,583,000	79,000
Physical scientists	156,000	141,000	14,000	147,000	8,000
Mathematical scientists	45,000	38,000	7,000	42,000	3,000
Computer specialists	122,000	101,000	21,000	116,000	5,000
Environmental scientists	44,000	42,000	1,800	43,000	1,000
Engineers	999,000	993,000	5,000	963,000	29,000
Life scientists	136,000	118,000	18,000	129,000	7,000
Psychologist	61,000	46,000	15,000	57,000	4,000
Social scientists	100,000	87,000	13,000	86,000	14,000

Note: Detail may not add to totals because of rounding. Source: National Science Foundation, Manpower Characteristics System.

for minority group scientists and engineers is somewhat surprising since a greater portion of minority scientists and engineers were women (one in five) than was the case for nonminorities (one in 10). These figures led the NSF to suggest that "the high labor force participation rate for minorities may reflect more favorable career opportunities for minority group scientists and engineers vis-a-vis minorities in the general population because of equal employment and affirmative action legislation. Also, given historical patterns of discrimination, it may reflect a greater commitment to gainful employment in science and engineering among minority group scientists and engineers."^{11,14} However, other factors such as the age, experience and reasons for non-participation in the labor force of both the minority and non-minority groups must be considered before any real conclusions can be drawn from these figures.

Now that general employment figures for scientists and engineers have been examined, it is meaningful to discuss the status of employment in the computer sciences.

Computer specialists

Recent attention has been accorded the status of women and minority computer science faculty and students.^{9,11} In a report on women in the computer industry,²⁰ Weber and Gilchrist examined employment figures for computer manufacturers and computer-user industries. The data they presented suggested "... that women are not receiving equal pay for equal work and may not be sharing equally in the opportunities for advancement." Some improvement was noted. For example, figures indicated that the percentage of women employed by manufacturers of electronic computer equipment was showing significant improvement, although still below the national average.

The NSF reports on S/E employment found that the second largest group of scientists in the U.S. 1974 population were computer specialists, numbering 125,000. Chemists ranked first at 138,000.¹³ In the NSF reports the term "computer specialist" was used to include

1. College faculty in the computer sciences
2. Computer programmers

3. Computer systems analysts
4. Computer scientists
5. Other computer specialists

In 1974, of the six percent of all employed scientists and engineers who were women, the largest proportion (22 percent) were categorized as computer specialists.^{13,14} Furthermore, 88 percent of the female computer specialists in the S/E population were employed, compared to, for example, 53 percent of the life scientists and 23 percent of the total scientists. This relatively high ratio between the number of female computer specialists in the population and their employment as computer specialists led the NSF to suggest that this could indicate more favorable employment opportunities for all computer specialists. This appears likely, as the employment figures for men and minorities were both approximately 100 percent in this field. The NSF also observed that the computer field is relatively new, having grown so rapidly that demand frequently exceeds supply. It was also speculated that "since traditional barriers to employment tend to fall in the face of a skill shortage, women may have found employment as computer specialists because they had the educational background and were available for employment."^{13,14}

The demand for computer professionals is further illustrated by data on the transition of scientists and engineers from school to work,¹⁴ which indicates that most of the women who planned careers and had degrees in the natural sciences and who did not work in those fields worked instead in the computer field.

A recent nationwide NSF survey shows that the numbers of men and women, respectively, working at or above the systems analyst level in the computer science area were 138,700 and 33,600 for 1975 and 134,900 and 31,300 for 1973.¹⁶ The apparent under-representation of women in the computer field in general could be attributable in part to a shortage of women receiving college degrees in the fields required by the computer industry.²⁰ The same might be true for the low representation in the more highly skilled computer positions.

Current figures from the National Center for Educational Statistics show the total and proportion of Bachelor's, Mas-

ter's and doctoral degrees awarded from 1970-1 through 1975-6.¹⁷ As shown in Table II, the number of degrees awarded to women in computer and information sciences continues to be fairly light. (Data is available on women and minority Ph.D.s and on science and engineering doctorate supply and utilization.^{6,12}

As indicated by Table III computer specialists were among the scientific fields reporting median ages under 30 years.¹⁴ The median age for women computer specialists was five years less than for men in the field. In fact, women scientists and engineers were in general younger than males with a median age of 27 and 38, respectively.

In Weber and Gilchrist's review of the status of women in the computer industry,²⁰ they concluded that "the available data suggest that women are not receiving equal pay for equal work and may not be sharing equally in the opportunities for advancement." While the annual salary differential between men and women is actually less for computer specialists—a difference of \$2,300—than for any other S/E field examined in the NSF study,¹⁴ this figure alone is not in itself a sufficient indicator that the situation is improving. Likewise, the fact that medical scientists had the largest salary differential (\$5,800) does not necessarily imply that greater inequities exist in that particular field. Salary differentials should instead be viewed as percentages of gross salaries, for clearly a small absolute differential could

TABLE II.—Total and Proportion of Bachelor's, Master's and Doctoral Degrees Awarded to Women 1970-71 Through 1975-76 By Broad Discipline Division

	Degrees Awarded 1970-71 to 1975-76					
	Bachelor's		Master's		Ph.D.'s	
	Total	% W	Total	% W	Total	% W
Agri. & Nat. Resources	94,425	10.6	17,311	9.7	5,965	3.5
Arch. & Env. Design	44,257	15.2	14,860	17.9	364	11.5
Area Studies	17,754	53.8	6,245	40.6	984	24.8
Biological Sciences	272,348	31.7	37,899	31.5	21,157	19.4
Business & Management	782,654	13.4	200,221	7.0	5,594	4.3
Communications	95,086	38.5	15,026	38.0	939	19.6
Computer & Info. Sciences	25,555	16.8	12,856	12.6	1,146	6.6
Education	1,077,546	73.6	653,509	60.0	43,258	27.0
Engineering	298,148	1.6	97,128	2.1	20,042	1.4
Fine & Applied Arts	223,890	60.6	46,653	46.7	3,663	26.5
Foreign Languages	112,532	75.9	25,056	65.7	5,257	42.4
Health Professions	233,993	77.5	54,764	60.5	3,327	23.8
Home Economics	86,905	96.3	10,747	91.4	862	69.1
Law	2,983	10.5	6,803	7.8	221	2.7
Letters	393,802	58.9	73,182	57.6	15,342	30.1
Library Science	6,238	93.0	46,504	79.0	392	40.8
Mathematics	128,233	40.0	28,473	31.0	6,257	9.4
Military Science	2,932	0.1	—	—	—	—
Physical Science	126,987	16.4	36,333	14.1	23,202	7.1
Psychology	283,726	49.7	37,093	41.3	13,114	28.7
Public Affairs & Services	126,511	45.4	74,309	45.9	1,442	24.5
Social Sciences	890,906	36.9	101,422	29.1	24,463	17.5
Theology	25,760	27.3	17,661	27.7	4,092	3.6
Interdisciplinary Studies	137,785	38.0	17,255	44.4	1,177	24.0
All Disciplines	5,490,974	44.4	1,631,312	42.9	202,260	18.6

Source: *Earned Degrees Conferred*, Series 1970-71 - 1975-76, National Center for Education Statistics.

TABLE III.—Median Age of Scientists and Engineers by Sex: 1974

Fields	Total	Men	Women
Total, all fields	36	38	27
Chemists	38	39	29
Physicists/astronomers	36	36	28
Other physical scientists	40	41	32
Mathematicians	32	32	29
Statisticians	32	34	28
Computer specialists	29	31	26
Earth scientists	37	38	27
Oceanographers	37	37	(*)
Atmospheric scientists	45	45	(*)
Engineers	40	40	28
Biological scientists	28	28	26
Agricultural scientists	39	40	35
Medical scientists	41	41	40
Psychologists	29	30	29
Economists	34	35	32
Sociologist/anthropologists	27	29	25
Other social scientists	27	28	25

Source: National Science Foundation.
* Too few cases to compute a median.

represent a large percentage of total salary, depending upon the average salary for the field.

Salary differentials reflect many variables, including type of employer (e.g., business, government), age (note the difference in median age reported earlier), work activity (e.g., R&D, management) and job experience. For example, in the U.S. workforce, a greater proportion of men than women were engaged in management and administration—the work area which typically has the highest salary. This was certainly the case for S/E fields surveyed in 1970, as is shown in Table IV.

Salary surveys made by the College Placement Council and cited in Reference 14 reveal an apparent narrowing difference in starting salaries for women and minority scientists and engineers. For individuals in computer science in particular, the national average monthly salary offers made to bachelor's degree candidates during the periods 1973-4 and 1975-6 were \$920 and \$1,035, respectively, for men and \$895; \$1,045, respectively, for women.

TABLE IV.—Median Annual Salaries of the 1970 Science/Engineering Labor Force, by Field and Work Activity: 1974

	Work Activity		
	R&D	Management or administration	Teaching
Total	\$18,400	\$22,600	\$19,200
Physical scientists	19,000	24,500	19,000
Mathematical scientists	21,400	24,200	18,000
Computer specialists	19,100	20,700	18,900
Environmental scientists	20,100	22,900	18,900
Engineers	18,300	22,600	20,400
Life scientists	17,900	19,000	18,500
Psychologists	19,500	22,200	18,700
Social scientists	20,200	23,100	19,800

Source: National Science Foundation, Manpower Characteristics System.

This (at best) elimination or (at worst) reduction of the gap in starting salaries, coupled with such factors as the relatively small number of women scientists and engineers and their younger median age, are likely to result in a reduction in average salary differential for men and women.¹⁴

Age and detailed salary information relating to minority scientists and engineers was not included in the NSF reports.

FEDERAL WORKFORCE

Recent Civil Service Commission statistics show that the federal government is well ahead in overall employment of minorities in professional, administrative, technical and clerical jobs. The private sector, however, has higher percentages of women and/or specific minority groups in some categories. These figures also show some increase in the proportion of minorities and women in both middle and upper level jobs, although Civil Service Commission Chairman Alan K. Campbell noted that the rate of progress has been slow.⁴

As of November 1977, minorities represented 21.6 percent of all full-time federal civilian employees. Women held 30.7 percent of those positions. Since 1975, minority full-time employment has steadily increased by an average of 1.2 percent per year, in spite of an overall decline of 0.2 percent for the full-time federal work force for this period.⁵

Scientists and engineers

As illustrated in Figure 1, in 1974 the federal government ranked behind business and industry (37 percent) and educational institutions (27 percent), by employing only 12 percent of all scientists. Most women engineers were employed by business and industry. The federal government, along with nonprofit organizations and State and local governments, each accounted for about 10 percent of the women scientists.

Figures compiled by the U.S. Civil Service Commission show that as of 1974 there were few minority and women employees in some of the "pure science" occupations. For example, the percentages of women and minorities respectively were Physics—2.4 and 3.7, Nuclear Engineering—0.6 and 3.5 and Chemistry—14.3 and 2.4.²

Computer sciences

Federally-employed computer professionals are divided into several different job classifications (e.g., computer scientist, computer specialist, general physical scientist). This makes it extremely difficult to collect statistics and to make comparisons with the U.S. workforce totals. Consequently, this section will present selected figures which should at least provide a "snapshot" of the current situation in this area.

In 1976, total employment for federal computer specialists showed 24,521 full-time permanent employees. (Full-time permanent employees are those federal workers who have career—three or more years of continuous service—or career conditional—less than three years—appointments and work a full-time—40 hours per week—work schedule.) Of those employed as Computer Specialists, 19.4 percent were women and 11.0 percent were minorities.³ These numbers are up from the corresponding 1974 figures of 19.1 and 10.1 percent.² Table V shows average General Schedule (GS) pay categories ("grades") and percentage figures for all federal employees, women, minorities and non-minorities in several scientific occupations.³ Comparing the statistics for computer specialists with corresponding figures for federal chemists, a field with a General Schedule employment of 7,599, we find a somewhat lower percentage representation for women (15.2 percent) and a slightly higher representation for minorities (12.8 percent). Average grade for all employees in chemistry was 11.63. The average for women and minorities was (10.37 and 11.27), respectively.³

In the computer-related occupations, as well as for many other federal scientific fields (e.g., chemistry, physics, meteorology, metallurgy, astronomy), the average pay grades for women and minorities are less than the average for all employees. Furthermore, the average grade for women is less than that for minorities in each field examined.³

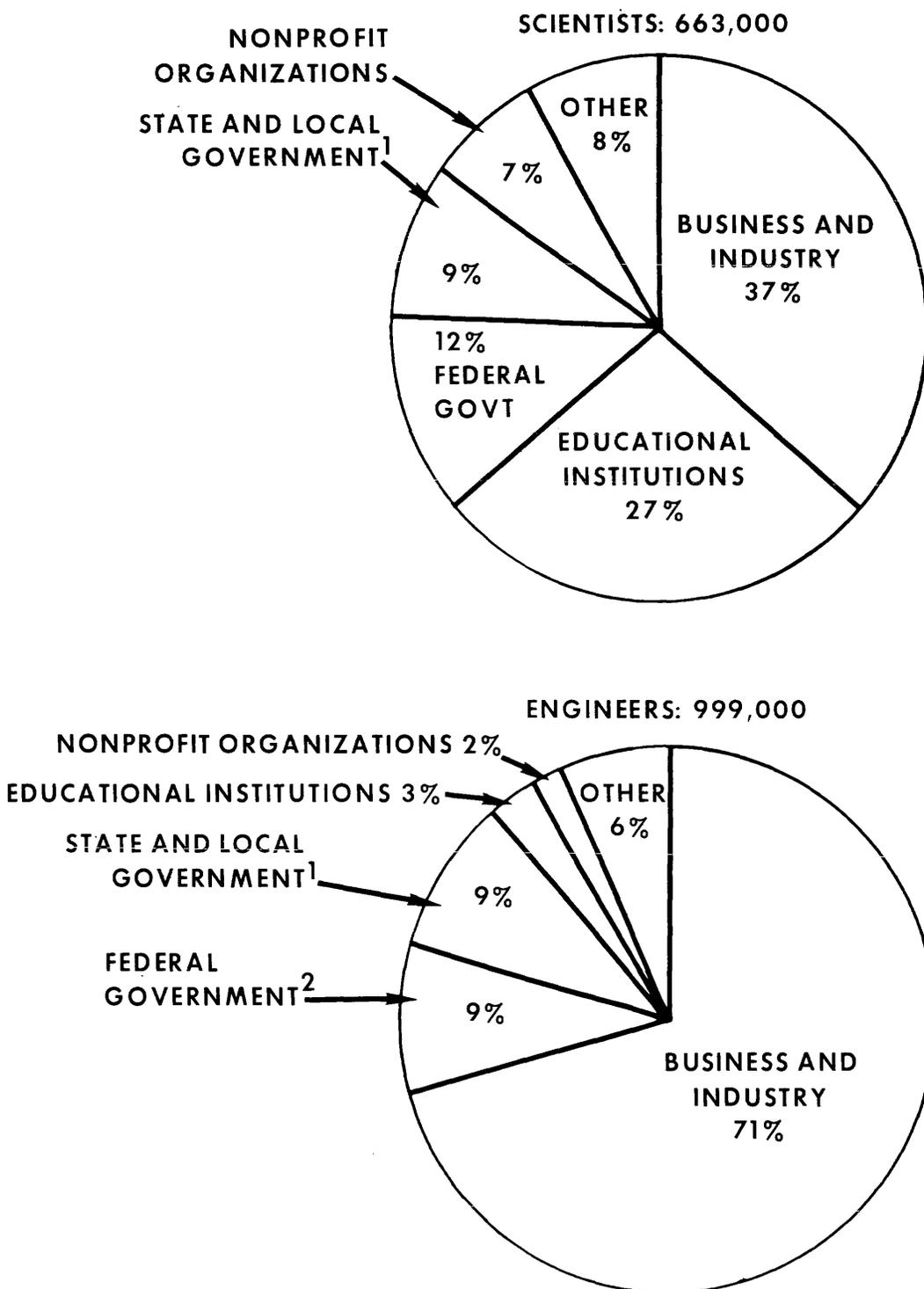
As was brought out in the earlier discussion of factors affecting salary differentials, more information is needed in order for the full implications of these figures to be judged. Age, education, availability and job experience, for example, must be viewed in conjunction with average grade and representation.

SUMMARY AND CONCLUSIONS

The available data do not provide sufficient information to support any surprising conclusions about the current status of women and minorities in computer-related professions. Meaningful analysis of the data is also hampered by the fact that more information was available relating to the employment of women than for minorities. Certainly it would appear, based upon the previously-cited employment rate and other statistics, that the computer field is in general a "healthy" field. Accordingly, one can expect to find better

TABLE V.—1976 Federal White-Collar Employment—Selected Fields

	Average Grade				
	All	Women	Minorities	% Women	% Minorities
Digital Computer Administrator	12.80	10.79	11.88	11%	9.3%
Computer Specialist	11.39	10.58	10.99	19.4%	11.0%
Chemistry	11.63	10.37	11.27	15.2%	12.8%
Mathematics	11.69	10.82	11.38	19.1%	11.0%
General Physical Science	13.51	11.26	13.16	4.2%	5.0%



¹ INCLUDES "OTHER" GOVERNMENT, I.E., DISTRICTS, INTERSTATE ORGANIZATIONS, ETC.

² INCLUDES MILITARY PERSONNEL.

SOURCE: NATIONAL SCIENCE FOUNDATION

Figure 1—Scientists and engineers by type of employer: 1974.

employment opportunities for all, and hence for minorities and women, there than in other scientific and engineering occupations.

Recent reports of the National Science Foundation (NSF) on the status of women and minorities in the sciences and engineering^{13,14} have concluded that there is and has been an under-utilization of women in the sciences and engineering, both in terms of their entry into these fields and their utilization within the scientific workforce. For minorities, data indicate that Blacks and Hispanics appear not to have developed those background skills considered important for careers in the sciences to the same extent as other racial groups and women. Consequently, it is suggested that this may have caused an under-utilization of these minorities in science and engineering. Information on utilization of women and minorities in the computer field was not cited. Further analysis in this area would be useful.

As noted by the NSF report on women and minorities in science and engineering,¹⁴ "... available data indicate that there are relatively few women scientists and even fewer engineers, and considerably fewer minority scientists and engineers of either sex." Data gathered also indicate that relatively few women and/or minority scientists and engineers are unemployed. However, data on labor force participation and salary differentials show possible problem areas. The fact that the labor force participation rate of female scientists and engineers is significantly below that for males in S/E may reflect actual or perceived lack of job opportunities. If one considers the histories of women, both minority and non-minority, who are considered "successful" in the sciences, then it may be more easily understood how some women might become discouraged and "drop out" or lower their expectations.⁸

Many women in the federal government do not feel that there is much difficulty in progressing to the top of a career ladder once they get into the career series. However, they perceive attitudinal barriers preventing them from advancing into supervisory and managerial positions. The fact that real obstacles to advancement remain is illustrated by a recent Civil Service Commission study which reported that women with college degrees are one to three grades behind men with the same educational level.¹⁰ Recent reports indicate, however, that the federal workforce is changing and that some progress is being made in the employment of both women and minorities.¹⁸ We hope future analyses of em-

ployment and utilization statistics will reflect accomplishments of current equal employment opportunity and affirmative action programs of both the public and private sectors.

REFERENCES

1. Causey, Mike, "U.S. Work Force Changing," *The Washington Post*, May 5, 1978, p. C2.
2. U.S. Civil Service Commission, "Minority Group Employment in the Federal Government," November 1974.
3. U.S. Civil Service Commission, "Minority Group Employment in the Federal Government," November 1976.
4. Civil Service Commission Press Release, November 29, 1977
5. Civil Service Commission Press Release, August 23, 1978.
6. Gilford, Dorothy M., and Joan Snyder, "Women and Minority Ph.D.s in the 1970s: A Data Book," National Research Council, National Academy of Sciences, 1977.
7. Krepes, Juanita, *Sex in the Marketplace: American Women At Work*, The Johns Hopkins Press, Baltimore, Md., Policy Studies in Employment and Welfare Number 11, 1971.
8. Kundsins, Ruth B., "Successful Women in the Sciences: An Analysis of Determinants," *Annals of the New York Academy of Sciences*, Vol. 208, March 15, 1973.
9. Mamrak, Sandra A., and R. G. Montanelli, Jr., "Computer Science Faculties: The Current Status of Minorities and Women," *Communications of the ACM*, Vol. 21, No. 2, February 1978.
10. Mendenhall, Janice, "Roots of the Federal Women's Program," *Civil Service Journal*, Vol. 18, No. 1, July-Sept 1977, pp. 21-24.
11. Montanelli, R. G., Jr., and S. A. Mamrak, "The Status of Women and Minorities in Academic Computer Science," *Communications of the ACM*, Vol. 19, No. 10, October 1976, pp. 578-581.
12. National Science Foundation, "Science and Engineering Doctorate Supply and Utilization: 1968-1980," NSF 69-37, 1969.
13. National Science Foundation, "U.S. Scientists and Engineers: 1974," NSF 76-329, 1976.
14. National Science Foundation, "Women and Minorities in Science and Engineering," NSF 77-304, 1977.
15. Ramaley, Judith A. (ed.), *Covert Discrimination and Women in the Sciences*, Westview Press for American Association for the Advancement of Science, AAAS Selected Symposium 14, 1978.
16. Rosenberg, Marcy, "Lack of Women DPers Seen 'Severe Loss'," *Computerworld*, November 20, 1978.
17. "Scientific Engineering Manpower Comments," Scientific Manpower Commission, Vol. 15, No. 6, July-August 1978.
18. Sugarman, Jule M., "Thinking Ahead in Reorganization," *Civil Service Journal*, October-December 1977, pp. 12-15.
19. U.S. Commission on Civil Rights, "Statement on Affirmative Action," U.S. Government Printing Office, Washington, D.C., Clearinghouse Publication 54, October 1977.
20. Weber, Richard E., and Bruce Gilchrist, "Discrimination in the Employment of Women in the Computer Industry," *Communications of the ACM*, Vol. 18, No. 7, July 1975, pp. 416-418.

Computers in judicial administration

by CHARLES L. AIRD and BARBARA H. TODD

Supreme Court of Virginia
Richmond, Virginia

INTRODUCTION

The number of computer installations in the courts has increased rapidly in the past ten years. This trend indicates an enthusiasm in the courts to acquire the scientific and industrial technology that the business community has successfully used. The computer is recognized as being a major tool of technological development and for its tremendous capability for upgrading the quantity and quality of a wide variety of functions and services. However, while looking in ecstasy at the new horizons of computer technology, we must realize the computer's dependency upon man. Computers are unlike any other machine. Switch on a water pump and water is pumped, but switch on a computer and nothing happens unless precise instructions have been prepared.

Today's courts are similar to businesses. They are complex organizations which store large amounts of detailed information. Court files require precise methods for processing and retrieving this information.

Modern computer technology is well suited to the task of accomplishing the repetitive job of maintaining accurate up-to-date information that is easily retrievable. The computer can also create access to information which was previously too costly to assemble.

The courts now using automation experience improved record-keeping efficiency. Computers provide timely and effective management information to individual courts as well as to the Supreme Court and other criminal justice agencies.

The potential capabilities of systems technology can provide the courts with standardized reports and services. Computers can provide this wide range of services in far less time than manually possible. The services and reports that can be produced for the courts are numerous.

Basic docketing and indexing services are typical services. The docket, listing cases to be heard in specific courts on specific dates, can be generated. The computer can retrieve cases or records based on a particular piece of data such as the name of the defendant, docket number, summons number, type of case, date, or other piece of identifying information. This permits the court to make direct inquiry into case records and to update records by visual display terminals located in individual courts.

Cases can be listed by judge, by attorney, and by docket number to keep track of case and schedule conflicts. Particular cases which require follow up or review for specific reasons are listed as exceptions. Notices are automatically prepared as required by the court. These capabilities improve case flow management and reduce the amount of paperwork needed to maintain records and administrative activity. Periodic statistical reports are generated on caseload and courtroom activities for individual courts.

Improved statewide caseload information can be provided to judges, clerks and the Supreme Court on a regular basis. Information on caseload, case age, pending cases, hearings, dispositions and workload forecasting are by-products of computerized record-keeping. This, too, can be used for effective judicial management.

Successful implementation requires careful study by local courts and the Supreme Court. Implementation depends upon long-term cooperation during the necessary trial-and-error process in the initial stages of development.

CONTRIBUTIONS AND PROBLEMS

The trend toward computers is due to the increased workload of the court. Traditionally, the measure of judicial workload has been filings. As filings increase, the complexity of the court and clerk's office operations increases. Computers were solicited to cure inefficiencies. The computer is solving major problems by reducing backlog and delay, improving scheduling and reducing redundant tasks.

Computers will continue to make substantial contributions to court administration. However, automation is not a total panacea. Technology can do little to reduce backlogs and delays stemming from strategy, not procedure. It will not expedite courtroom proceedings established by due process. The computer is an aid to management but is not a substitute for such things as judicial selection, discipline and sentencing.

Even though there are many admirable automated judicial systems throughout the country, judicial computerization on the national, state and local levels faces difficulty. Several reasons exist which indicate the depth of the problem. Vendors have oversold their products. Sales techniques have been more successful than applications. The judges who run

the courts are not businessmen. They are easily influenced by a smooth sales pitch. However, once stung by an expensive, inefficient application, they will not be sold again.

Funding has been a major problem also. Computerization costs money and budgets are usually controlled by the executive branch at all levels of government. The judiciary is sensitive to invasion by the other government branches and may not pursue funding. The most popular solution is to share the expertise and hardware of the executive branch. Court processing is not easily understandable. Lack of communication and scheduling priority conflicts often doom joint projects. The judicial branch is very sensitive with respect to privacy and security.

If these problems are resolved, and actual design of a system begins, the lack of in-house expertise becomes a hurdle. More failures than successes were produced in early judicial data processing projects. This was due to inadequate planning, poor scheduling and insufficient testing. Projects were generally too big and not implemented in stages.

Even with growing caseloads, the computer is not necessarily the solution. Most courts have not reached the point where adding more people does not solve the problem. Only the urban courts are overburdened. The practical solution to increased caseload may be better manual techniques, forms design, specialization of tasks, elimination of duplication and word-processing equipment.

The final problem encountered in the courts has been resistance to change. The courts are the most conservative of the three branches of government. Many courts use the same record-keeping procedures implemented in the English courts and imported to the colonies.

CATEGORIES OF AUTOMATION

Considering the problems, it is surprising that the courts have considered any automation. In many instances, the

courts have not had a choice. Caseload in urban-centered courts has reached overwhelming proportions and so has the paperwork. Therefore, three categories of systems at two levels have been developed. The three categories are administrative systems, case record and trial systems and legal research systems. The two levels are at the court and the administrative.

Administrative systems are payroll, personnel, budget, inventory and financial record systems, that is, the routine business systems. Statistical systems allow for caseload management of the courts. Case record and trial systems deal with case tracking, docketing, indexing, case scheduling, jury selection and management and information systems exchange with other criminal justice agencies. Some systems are useful at only one level. Others perform at both and still others serve as a means of communication between the two levels.

VIRGINIA'S SYSTEMS

In reviewing Virginia's computerized systems, it is beneficial to keep in mind the structure of the judicial system. The District Courts were unified under the state in 1973. Prior to that time, they were municipal courts and independent. These are the lowest level courts and have the greatest volume, most cases and the need for computerization. The Circuit Court clerk's offices are local and totally independent. There is no standardization and low volume. The average number of commenced cases per circuit judge in 1977 was 1,100 or approximately one case every two hours. The Supreme Court stands at the top of the judicial branch. There are numerous computer applications at the District Court and Supreme Court. Only one Circuit Court currently has or is developing a system.

The Office of the Executive Secretary (OES) of the Vir-

FUND 114 01 001		FINANCIAL ANALYSIS AS OF AUGUST 1, 1976					9/17/76		
DISTRICT 16		CHARLOTTESVILLE GENERAL DISTRICT COURT							
OBJECT	(A) FY77 BUDGET	(B) JULY EXPENSES	(C) Y-T-D EXPENSES	(D) % B/A	(E) % C/A	BALANCE (A-C)	% TIME LEFT	% BUDGET BALANCE	
1110 SALARIES	42,080.98	3,506.75	3,506.75	8.0	8.0	38,574.23	92.0	92.0	
1120 WAGES	190.30	0.00	0.00	0.0	0.0	190.30	92.0	100.0	
1243 TRAVEL-MILEAGE	375.00	50.00	50.00	13.0	13.0	325.00	92.0	87.0	
1261 POSTAL SERVICES	300.00	100.00	100.00	33.0	33.0	200.00	92.0	67.0	
1270 PRINTING	1,300.00	0.00	0.00	0.0	0.0	1,300.00	92.0	100.0	
1340 OFFICE SUPPLIES	400.00	23.75	23.75	6.0	6.0	376.25	92.0	94.0	
LOCATION TOTAL	44,646.28	3,680.50	3,680.50	8.0	8.0	40,965.78	92.0	92.0	

ALASKA COURT SYSTEM - CASE HISTORY (CRIMINAL)

CASE NUMBER _____ PAGE _____ OF _____

JUDICIAL DISTRICT DIST SUP COURT AT _____

DATE COMMENCED _____ DATE TERMINATED _____
CHANGE OF VENUE FROM COMPLAINT INFO. INDICTMENT REOPENED APPEAL OTHER

FUNDS		BAIL	FILING FEE	OTHER	RECEIPT #	CHECK #
DATE	AMOUNT \$					

1 DEFENDANT	DISTRICT COURT CASE #		CONSOLIDATED WITH	
2 TYPE OF ACTION	1. FELONY MISD.	2. STATE MISD.	3. MUNICIPAL MISD.	4. OTHER MISD.
3. COMPLAINT		DATE ARRESTED	DATE OF OFFENSE	1. OFFENSE
OFFENSE (CONT)		STATUTE OR ORDINANCE		2. OFFENSE
2. STATUTE OR ORDINANCE		4. SUMMONS	DATE ISSUED	DATE SERVED
5. WARRANT		DATE ISSUED	SERVED	UN-SERVED
6. DISTRICT COURT ARRAIGNMENT		DATE	JUDGE/MAGISTRATE	DATE DISQUALIFIED
JUDGE/MAGISTRATE		DATE	JUDGE/MAGISTRATE	RESULT
7. BAIL		IN CUSTODY	1. DATE	AMOUNT
JUDGE		CASH SECURED SURETY	NON-SECURED SURETY	OR DATE EXONERATED
3. DATE		AMOUNT	JUDGE	CASH SECURED SURETY
NON-SECURED SURETY		OR DATE EXONERATED	3. DATE	AMOUNT
JUDGE		CASH SECURED SURETY	NON-SECURED SURETY	OR DATE EXONERATED
8. HEARINGS		DATE SET	JUDGE	DATE HELD
9. PRELIMINARY HEARING		DATE SET	JUDGE	DATE HELD
RESULT		DIS-MISSED	HELD TO ANSWER	REDUCTION OF CHARGE
DISCH. ARGED		10. GRAND JURY	YES	NO
NO TRUE BILL		11. INDICTMENT	INDICT-MENT	INFORM-ATION
CHARGE		DATE RETURN		
12. FOUR MONTH RULE		DATE COMMENCE	DATE EXPIRED	WAIVERS
DATE		NO. DAYS	DATE	NO. DAYS
13. SUPERIOR COURT ARRAIGNMENT		DATE	JUDGE	DATE DISQUALIFIED
JUDGE		DATE	JUDGE	RESULT
14. PSYCHIATRIC EXAMINATION		DATE ORDERED	DATE FILED	15. PRE SENTENCE REPORT
DATE ORDERED		DATE FILED	16. OMNIBUS	DATE
17. PRE-TRIAL CONFERENCE		DATE	JUDGE	18. PRETRIAL DISPOSITION
DIS-MISSAL		DATE	PURSUANT TO RULE	BY COURT
BY DA		BY OTHER (SPECIFY)		
CHANGE OF PLEA		DATE	ORIGINAL CHARGE	LESSER INCLUDED
NEW CHARGE		NEW CASE #		
19. TRIAL		JURY REQUESTED	JURY WAIVED	DATE SCHEDULED
DATE START		DATE FINISH	TOTAL DAYS IN TRIAL	TRIAL JUDGE/MAGISTRATE
20. VERDICT		DATE	GUilty	NOT GUilty
MUNG JURY		OTHER	LES IN OFF	
21. JUDGE-MENT		DATE	JUDGE/MAGISTRATE	DAYS
DAYS SUSP.		FINE \$	FINE DUE BY	FINE SUSP.
BAIL FORFEITL \$		PERIOD		
REDUCED CHARGE		DEFERRED SENTENCING	SUSP. IMPOS. OF SENTENCE	SUSPENSE DATE
RESTITUTION		COMMITMENT	PROBATION	OTHER (SPECIFY)
CREDIT FOR TIME SERVED		REMANDED TO DISTRICT COURT	GOOD BEHAVIOR	PERIOD
22. PAROLE		ELIGIBLE	NOT ELIGIBLE	UNIT#
23. LIC. ACTION		AUTO	OTHER	LIMITED
REVOKED		SUSP.	PERIOD	DAY#
D.C.C.		UTC #	CONDITIONS	
24. CHANGE OF VENUE		MOTION DATE	GRANTED	DENIED
TO		25. APPEALS	DATE FILED	BY PLW
BY DPT		DESIGN OF RECORD		
26. ATTORNEY STATE		DATE	GRANTED	DENIED
REMANDEMENT ORDERED		ATTY WAIVED		
27. ATTORNEY DEFENSE		P.B. REQ	DATE	GRANTED
DENIED		REMANDEMENT ORDERED	ATTY WAIVED	PUBLIC DEFENDER
28. REMARKS		DATE	BLOCK	
DATE		BLOCK		

White—Commencement
Pink—Termination
Yellow—File Copy



REPORT TO JUDICIAL ADMINISTRATOR AND JUDICIAL COUNCIL

Judicial District _____ County _____

<p>1. <input type="checkbox"/> REGULAR CIVIL</p> <p>2. Case No. _____</p> <p><u>COMMENCEMENT</u></p> <p>3. Date Commenced _____</p> <p>4. <input type="checkbox"/> Retrial or Reinstatement (If No. 4 is checked, put an asterisk before Case No.)</p> <p>5. Division No. _____ (Multi-judge district only)</p> <p><u>Nature of Action</u></p> <p>6. <input type="checkbox"/> Auto Negligence</p> <p>7. <input type="checkbox"/> Other Tort</p> <p>8. <input type="checkbox"/> 60-1507</p> <p>9. <input type="checkbox"/> Foreclosure</p> <p>10. <input type="checkbox"/> Real Property</p> <p>11. <input type="checkbox"/> Contractual <input type="checkbox"/> Inj., Q. W. & Mand.</p> <p>13. <input type="checkbox"/> Other</p>	<p>1. <input type="checkbox"/> DOMESTIC RELATIONS</p> <p>2. Case No. _____</p> <p><u>COMMENCEMENT</u></p> <p>3. Date Commenced _____</p> <p>4. <input type="checkbox"/> Retrial or Reinstatement (If No. 4 is checked, put an asterisk before Case No.)</p> <p>5. Division No. _____ (Multi-judge district only)</p> <p><u>Nature of Action</u></p> <p>6. <input type="checkbox"/> Divorce</p> <p>7. <input type="checkbox"/> Sep. Maintenance</p> <p>8. <input type="checkbox"/> Annulment</p> <p>9. <input type="checkbox"/> Recip-in</p> <p>10. <input type="checkbox"/> Recip-out</p> <p>11. <input type="checkbox"/> Other</p>	<p>1. <input type="checkbox"/> CRIMINAL</p> <p>2. Case No. _____</p> <p><u>COMMENCEMENT</u></p> <p>3. Date Commenced _____</p> <p>4. <input type="checkbox"/> Retrial or Reinstatement (If No. 4 is checked, put an asterisk before Case No.)</p> <p>5. Division No. _____ (Multi-judge district only)</p> <p><u>Felony</u></p> <p>6. <input type="checkbox"/> Crime Against Person</p> <p>7. <input type="checkbox"/> Crime Against Property</p> <p>8. <input type="checkbox"/> Other</p> <p><u>Misdemeanor</u></p> <p>9. <input type="checkbox"/> D. W. I</p> <p>10. <input type="checkbox"/> Other Traffic</p> <p>11. <input type="checkbox"/> Other Misdemeanor</p> <p><u>Appeal</u></p> <p>12. <input type="checkbox"/> D. W. I.</p> <p>13. <input type="checkbox"/> Other Traffic</p> <p>14. <input type="checkbox"/> Other Offenses</p>
<p><u>TERMINATION</u></p> <p>14. Date Terminated _____</p> <p>15. <input type="checkbox"/> Case was Pre-tried</p> <p>16. <input type="checkbox"/> Dismissed</p> <p>17. <input type="checkbox"/> Not Contested</p> <p>18. <input type="checkbox"/> Contested Court</p> <p>19. <input type="checkbox"/> Contested Jury</p> <p>25. <input type="checkbox"/> Regular Judge, Div. No. _____</p> <p>26. <input type="checkbox"/> Assigned Judge (from another district).</p>	<p><u>TERMINATION</u></p> <p>12. Date Terminated _____</p> <p>13. <input type="checkbox"/> Case was Pre-tried</p> <p>14. <input type="checkbox"/> Dismissed</p> <p>15. <input type="checkbox"/> Not Contested</p> <p>16. <input type="checkbox"/> Contested Court</p> <p>17. <input type="checkbox"/> Divorce Granted</p> <p>18. <input type="checkbox"/> Divorce Denied</p> <p>19. <input type="checkbox"/> Annulment Granted</p> <p>20. <input type="checkbox"/> Annulment Denied</p> <p>21. <input type="checkbox"/> Sep. Mtnce. Granted</p> <p>22. <input type="checkbox"/> Sep. Mtnce. Denied</p> <p>28. <input type="checkbox"/> Regular Judge, Div. No. _____</p> <p>29. <input type="checkbox"/> Assigned Judge (from another district)</p>	<p><u>TERMINATION</u></p> <p>15. Date Terminated _____</p> <p>16. <input type="checkbox"/> Dismissed</p> <p>16a. <input type="checkbox"/> Appeal Dismissed</p> <p>17. <input type="checkbox"/> Guilty Plea</p> <p>17a. <input type="checkbox"/> Other Uncontested Termination</p> <p>18. <input type="checkbox"/> Contested Court</p> <p>19. <input type="checkbox"/> Contested Jury</p> <p>20. <input type="checkbox"/> Convicted</p> <p>21. <input type="checkbox"/> Acquitted</p> <p>27. <input type="checkbox"/> Regular Judge, Div. No. _____</p> <p>28. <input type="checkbox"/> Assigned Judge (from another district)</p>

MAINE SUPERIOR COURT CRIMINAL STATISTICS REPORTING FORM

1. Region _____ 2. County _____ 3. County No. _____

4. Case No. _____ 5. No. of Defendants _____ 6. Date Filed: ____|__|__

A. TYPE OF CASE

New Filings

- 1. Bail Review
- 2. Transfer
- 3. Appeal
- 4. Boundover
- 5. Indictment
- 6. Information
- 7. Juvenile Appeal
- 8. Other

Refilings

- 1. Revocation
- 2. New Trial
- 3. Date Refiled ____|__|__

B. CLASS OF CHARGE

- 1. A
- 2. B
- 3. C
- 4. D
- 5. E

C. ACTION INFORMATION

- 1. Date of First Superior Court Appearance
- 2. Date Capias Issued
- 3. Court Appointed Counsel
- 4. Date Trial Began
- 5. No. of Trial Days
- 6. Jury
- 7. Jury Waived Trial
- 8. Date Plead Guilty

	Defendant #1	Defendant #2	Defendant #3
1. Date of First Superior Court Appearance	____ ____ ____	____ ____ ____	____ ____ ____
2. Date Capias Issued	____ ____ ____	____ ____ ____	____ ____ ____
3. Court Appointed Counsel	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. Date Trial Began	____ ____ ____	____ ____ ____	____ ____ ____
5. No. of Trial Days	-	-	-
6. Jury	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7. Jury Waived Trial	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8. Date Plead Guilty	____ ____ ____	____ ____ ____	____ ____ ____

D. DISPOSITION INFORMATION

- 1. District Court Bail Revised
- 2. District Court Bail Affirmed
- 3. Dismissed by Court
- 4. Dismissed by D.A. R. 48 (a)
- 5. Filed Case
- 6. Juvenile Appeal Denied
- 7. Juvenile Appeal Affirmed
- 8. Juvenile Appeal, New Sentence
- 9. Not Guilty, Reason of Insanity
- 10. No Bill
- 11. Probation Revoked
- 12. Convicted
- 13. Acquitted
- 14. Mistrial
- 15. Date Disposed
- 16. Justice Initials

	Defendant #1	Defendant #2	Defendant #3
1. District Court Bail Revised	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. District Court Bail Affirmed	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. Dismissed by Court	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. Dismissed by D.A. R. 48 (a)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. Filed Case	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6. Juvenile Appeal Denied	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7. Juvenile Appeal Affirmed	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8. Juvenile Appeal, New Sentence	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9. Not Guilty, Reason of Insanity	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10. No Bill	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
11. Probation Revoked	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
12. Convicted	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
13. Acquitted	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
14. Mistrial	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
15. Date Disposed	____ ____ ____	____ ____ ____	____ ____ ____
16. Justice Initials	_____	_____	_____

E. SENTENCE AND COMMITMENT INFORMATION

- 1. Probation
- 2. Correctional Center
- 3. Youth Center
- 4. State Prison
- 5. County Jail
- 6. Unconditional Discharge
- 7. Fine
- 8. Mental Health Commitment
- 9. Partially Suspended Sentence
- 10. Suspended Sentence
- 11. Date Sentenced

	Defendant #1	Defendant #2	Defendant #3
1. Probation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. Correctional Center	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. Youth Center	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. State Prison	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. County Jail	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6. Unconditional Discharge	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7. Fine	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8. Mental Health Commitment	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9. Partially Suspended Sentence	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10. Suspended Sentence	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
11. Date Sentenced	____ ____ ____	____ ____ ____	____ ____ ____

1. TYPE OF CASE
 (CR) — CRIMINAL
 (CV) — CIVIL
 (JU) — JUVENILE

2. COUNTY — 0 _____

3. CASE NUMBER — _____

10 FILING

4. DATE OF FILING: Month Day Year
 5. TRAFFIC RELATED: (01) Yes (02) No
 6. SOURCE OF CASE: (03) Original Action (04) Reopened Case

7. Charge/Type Of Action	CRIMINAL	JUVENILE	CIVIL
	<input type="checkbox"/> (05) Felony A <input type="checkbox"/> (06) Felony B <input type="checkbox"/> (07) Felony C <input type="checkbox"/> (08) Misdemeanor A <input type="checkbox"/> (09) Misdemeanor B <input type="checkbox"/> (10) Infraction <input type="checkbox"/> (30) Special Remedy <input type="checkbox"/> (11) Appeal <input type="checkbox"/> (12) Other (Explain Below)	<input type="checkbox"/> (13) Delinquency <input type="checkbox"/> (14) Unruly <input type="checkbox"/> (15) Deprived Child <input type="checkbox"/> (16) Special Proceedings <input type="checkbox"/> (17) Termination of Parental Rights <input type="checkbox"/> (18) Other (Explain Below)	<input type="checkbox"/> (19) Damages <input type="checkbox"/> (20) Action on Debt <input type="checkbox"/> (20) Real Estate Matter <input type="checkbox"/> (21) Divorce <input type="checkbox"/> (22) Reciprocal Support <input type="checkbox"/> (23) Adoption <input type="checkbox"/> (24) Appeal-Admin. Hearing <input type="checkbox"/> (25) Appeal—Other <input type="checkbox"/> (26) Special Remedy <input type="checkbox"/> (27) Trusts <input type="checkbox"/> (27) Other (explain below)

8. Remarks _____

20 DISPOSITION

9. DATE OF FINAL DISPOSITION: Month Day Year

10. TRIAL/ HEARING	CRIMINAL	JUVENILE	CIVIL
	<input type="checkbox"/> (01) Jury <input type="checkbox"/> (02) Non-Jury <input type="checkbox"/> (03) Not Contested	<input type="checkbox"/> (22) Referee Hearing <input type="checkbox"/> (23) Court Hearing	<input type="checkbox"/> (30) Jury <input type="checkbox"/> (30) Non-Jury <input type="checkbox"/> (40) Not Contested
11. Judgment	<input type="checkbox"/> (04) Felony A—Guilty <input type="checkbox"/> (05) Felony B—Guilty <input type="checkbox"/> (06) Felony C—Guilty <input type="checkbox"/> (07) Misdemeanor A—Guilty <input type="checkbox"/> (08) Misdemeanor B—Guilty <input type="checkbox"/> (09) Infraction—Guilty <input type="checkbox"/> (10) Acquittal <input type="checkbox"/> (11) Dismissal <input type="checkbox"/> (12) Uniform Post Conviction Procedures Act <input type="checkbox"/> (12) Change of Venue to _____	<input type="checkbox"/> (24) Judgment after Hearing <input type="checkbox"/> (25) Waive to Adult Court <input type="checkbox"/> (26) Acquittal <input type="checkbox"/> (27) Dismissal	<input type="checkbox"/> (41) Judgment after Trial <input type="checkbox"/> (40) Divorce Decree <input type="checkbox"/> (50) Adoption Decree <input type="checkbox"/> (42) Default Judgment <input type="checkbox"/> (42) Summary Judgment <input type="checkbox"/> (44) Special Remedy Judgment <input type="checkbox"/> (45) Voluntary Dismissal <input type="checkbox"/> (46) Involuntary Dismissal <input type="checkbox"/> (51) Termination of Trust <input type="checkbox"/> (47) Change of Venue to _____ <input type="checkbox"/> (48) Other (Explain below)

12. JUDGE/REFEREE RESPONSIBLE: JUDGE _____

13. Sentence/Placement	JUDGE RESPONSIBLE	JUDGE
	<input type="checkbox"/> (14) County Jail <input type="checkbox"/> (15) State Penitentiary <input type="checkbox"/> (16) State Farm <input type="checkbox"/> (17) Deferred Imposition <input type="checkbox"/> (18) Suspended Sentence <input type="checkbox"/> (19) Fine/Costs <input type="checkbox"/> (20) Restitution <input type="checkbox"/> (21) Other (explain below)	<input type="checkbox"/> (29) State Industrial School <input type="checkbox"/> (30) Private Institution <input type="checkbox"/> (31) Adoptive Agency Placement <input type="checkbox"/> (32) Probation to Parents <input type="checkbox"/> (33) Court Supervised Probation <input type="checkbox"/> (34) State Youth Authority <input type="checkbox"/> (35) Foster Home <input type="checkbox"/> (36) Group Home <input type="checkbox"/> (37) Other (Explain Below)

14. Remarks _____

COURT NAME _____ 114- _____ FILING DATE _____
 COURT ID. NO.

MONTHLY SUMMARY REPORT
 UNIFORM DOCKETING/CASELOAD REPORTING SYSTEM

STATISTICS FOR THE MONTH AND YEAR OF:

GENERAL DISTRICT COURT STATISTICS

CRIMINAL STATISTICS:

NUMBER OF REVIEW PROGRESS HEARINGS	_____	(CODE 0)
NUMBER OF BOND/ARRAIGNMENT/COUNSEL APPT. HEARINGS	_____	(CODE 1)
NUMBER OF EXTRADITION HEARINGS	_____	(CODE 2)
NUMBER OF PRELIMINARY HEARINGS	_____	(CODE 3)
NUMBER OF ADJUDICATORY HEARINGS WITH TESTIMONY	_____	(CODE 4)
NUMBER OF ADJUDICATORY HEARINGS WITHOUT TESTIMONY	_____	(CODE 5)
NUMBER OF DISPOSITIONAL HEARINGS	_____	(CODE 6)
NUMBER OF HEARINGS WAIVED	_____	(CODE 9)
TOTAL NUMBER OF HEARINGS & HEARINGS WAIVED	_____	(CODES 0-9)
NUMBER OF 'NEW' TRANSACTIONS	_____	('NEW' CIRCLED)
NUMBER OF 'CONTINUED' TRANSACTIONS	_____	('CONT' CIRCLED)
TOTAL NUMBER OF TRANSACTIONS	_____	(NEW + CONT)
NUMBER OF TRANSACTIONS BEARING FINAL DISPOSITIONS	_____	('**' CODED)

TRAFFIC STATISTICS:

NUMBER OF REVIEW PROGRESS HEARINGS	_____	(CODE 0)
NUMBER OF BOND/ARRAIGNMENT/COUNSEL APPT. HEARINGS	_____	(CODE 1)
NUMBER OF PRELIMINARY HEARINGS	_____	(CODE 3)
NUMBER OF ADJUDICATORY HEARINGS WITH TESTIMONY	_____	(CODE 4)
NUMBER OF ADJUDICATORY HEARINGS WITHOUT TESTIMONY	_____	(CODE 5)
NUMBER OF DISPOSITIONAL HEARINGS	_____	(CODE 6)
NUMBER OF HEARINGS WAIVED	_____	(CODE 9)
TOTAL NUMBER OF HEARINGS & HEARINGS WAIVED	_____	(CODES 0-9)
NUMBER OF 'NEW' TRANSACTIONS	_____	('NEW' CIRCLED)
NUMBER OF 'CONTINUED' TRANSACTIONS	_____	('CONT' CIRCLED)
TOTAL NUMBER OF TRANSACTIONS	_____	(NEW + CONT)
NUMBER OF TRANSACTIONS BEARING FINAL DISPOSITIONS	_____	('**' CODED)

CIVIL STATISTICS:

NUMBER OF CIVIL HEARINGS WITH TESTIMONY	_____	(CODE 7)
NUMBER OF CIVIL HEARINGS WITHOUT TESTIMONY	_____	(CODE 8)
NUMBER OF CASES DISMISSED PRIOR TO COURT	_____	(CODE 9)
TOTAL NUMBER OF HEARINGS & DISMISSED CASES	_____	(CODES 7-9)
NUMBER OF SUITS IN DEBT	_____	(SUFFIX D)
NUMBER OF GARNISHMENTS	_____	(SUFFIX G)
NUMBER OF UNLAWFUL DETAINERS	_____	(SUFFIX U)
NUMBER OF OTHER CIVIL CASES	_____	(SUFFIX O)
NUMBER OF 'NEW' TRANSACTIONS	_____	('NEW' CIRCLED)
NUMBER OF 'CONTINUED' TRANSACTIONS	_____	('CONT' CIRCLED)
TOTAL NUMBER OF TRANSACTIONS	_____	(NEW + CONT)
NUMBER OF TRANSACTIONS BEARING FINAL DISPOSITIONS	_____	('**' CODED)

CRIMINAL + TRAFFIC + CIVIL:

WARRANTS WRITTEN	_____	CHECKS WRITTEN	_____
APPEALS PROCESSED	_____	MENTAL COMMITMENT HEARINGS	_____
RECEIPTS WRITTEN	_____	HELD BUT NOT DOCKETED	_____

ginia Supreme Court handles the administration of the General District and Supreme Courts in total and the Circuit Court Judges. The OES is also responsible for 426 magistrates. Personnel and payroll systems are centrally handled by the OES. The budgets of each court with revenues and expenditures are tracked by an automated budget tracking system (ABTS). The ABTS produces a financial analysis statement for each court on a monthly basis, generates a summary of state and local revenues and state expenditures, and provides the basis, justification, and documentation for the judicial budget (Exhibit 1).

Certain individual courts keep automated financial records. These are the earliest and most successful applications. These courts track fine payments, support arrears, support check writing and support accounting.

The OES maintains statistical systems. These systems

summarize caseload and measure judicial workload. The results are used to obtain federal funding from the Law Enforcement Assistance Administration (LEAA), determine the number of judgeships and personnel for individual courts and to justify budgets.

There are two schools of thought concerning statistical systems. One school supports case-by-case statistics. This involves the collection of all critical information on a particular case plus any decision points and change of status. All statistics needed are generated from the data bank. Such a system is very flexible and produces many results; exception reporting, sentencing disparities, continuance studies, and case tracking. These are balanced against the negative consequences of tremendous amounts of paperwork, time consumption for clerical personnel, very large computer files, and great expense. The developmental cost is several million

COURT _____		GENERAL DISTRICT COURT DOCKET				PAGE _____ OF _____	PAGE _____ / _____	DATE _____		COURT # _____
JUDGE _____		CRIMINAL DIVISION				DATE _____	TIME _____	COURT # _____		
FILE #	DEFENDANT	CHARGE OFF DTE OFFICER/ IMPLINT	BOND \$	PLEA	DISPOSITION	TRANS TYPE	NEW	DISP	BENCH MINUTES	
			TYPE		CCRE DATE	APPEAL DATE				
1	COUNSEL	CA P W					1	1		
2		CA P W					2	2		
3		CA P W					3	3		
4		CA P W					4	4		
5		CA P W					5	5		
6		CA P W					6	6		
7		CA P W					7	7		
8		CA P W					8	8		
9		CA P W					9	9		
10		CA P W					10	10		
11		CA P W					11	11		
12		CA P W					12	12		
13		CA P W					13	13		
14		CA P W					14	14		

UDS-1 JAN 77

Exhibit 7

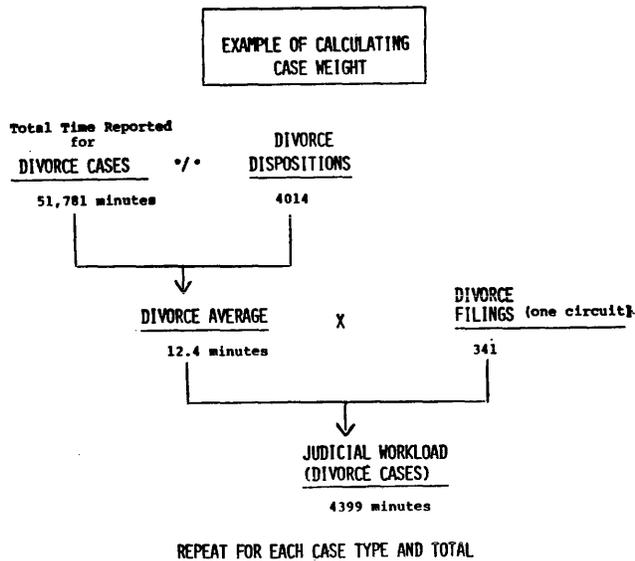


Exhibit 8

minutes for each case type are sampled. Using a statewide average time for each category of cases multiplied by the number of cases for a particular court, the amount of judicial work is calculated for that court (Exhibits 8 & 9). The "normal judge year" is administratively determined and used to calculate the number of judges needed to staff the court. Clerical workload uses a similar procedure. Weights are determined by spot-checking the time spent on case-related work.

The Circuit Court Caseload Reporting and Judicial Workload System is also a case-weighting system. The basic caseload summary gives the categories of civil and criminal cases. Also included are jury data and case aging information (Exhibits 10 & 11). The basic caseload counts of filings and dispositions will be weighted based on the average judicial time per case. Judges are timed to determine the amount of time each of the general categories of cases takes for disposition. Exhibits 12 and 13 are the forms used for sampling workload. Weights are based on both bench time and case-related chambers time (Exhibit 14).

dollars plus operational maintenance costs of \$500,000 annually.

The second method is to collect summary statistics. The design incorporates what is needed and is collected on a summary basis from the courts. The advantage of summary systems is the reduction in paperwork and expense. A summary statistical system can be developed for \$150,000 and operated for \$25,000 per year. Neither approach is successful unless a strong and supportive Chief Justice and Court Administrator control the Courts. The capture of the data elements must be an integral function of the clerk's office. Exhibits 2, 3, 4, and 5 give an indication of the large amount of data that must be included for case by case statistics.

Virginia has three caseload reporting systems:

1. Uniform Docketing & Caseload Reporting System—District Courts.
2. Circuit Court Caseload Reporting and Judicial Workload Systems.
3. Magistrate Statistical System.

These are batch-operated systems. They are programmed in COBOL and run on an IBM 370/158 (2) system under OS-VS2 with 1024k storage.

The Uniform Docketing System is divided into two sub-systems. The first part is used in the clerk's office to standardize case scheduling and indexing in the District Courts. The second part of the system has three purposes: to count basic cases, to determine judicial workload and to indicate clerical workload. Monthly summary reports (Exhibit 6) are completed using the tearoff from the Uniform Docket (Exhibit 7).

Counts of cases are not enough to determine workload. The different types of cases do not take equal amounts of time to process or hear in the courtroom. There is a need to weight cases depending upon their complexity. The bench

CASE RECORD AND TRIAL SYSTEMS

Case record and trial information systems at the court level aid clerical staff with redundant clerical efforts. Such systems increase the accessibility of records and provide the most complete records possible. These systems have certain common characteristics. Each case is one record including basic case and individual information and status. Multiple indices are established which are typically manual, single directional. Automated indices expand the court index to systems similar to those in a library. The number, name, attorney, judge, status, offense, or disposition may be used

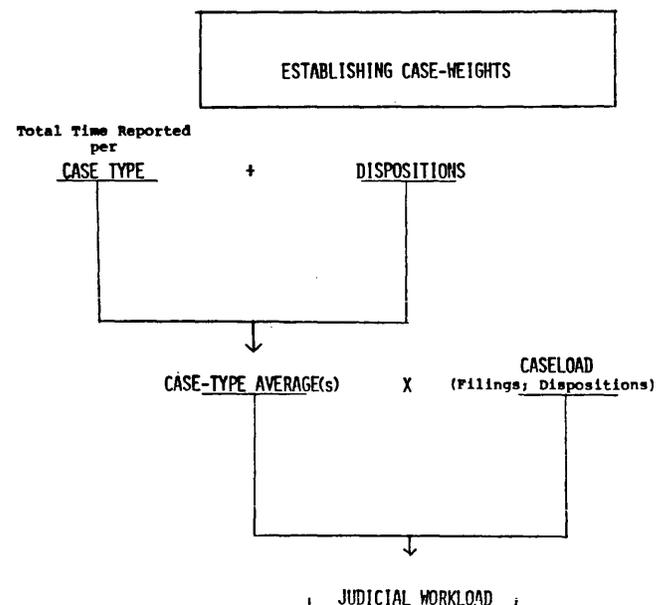


Exhibit 9

MONTHLY CASELOAD REPORT CIVIL CASES

2				
Circuit I.D. Number				

Month	Year

C
13

STATISTICS FOR THE MONTH OF:

City/County of:

For Office Use Only
501

Cases on Docket	Condemnation	Appeals	All Other Law	Divorce	All Other Chancery
Cases Commenced During Month By:					
1. Initial Filing	21				
2. Supplemental Petition			49		
3. Other	51				
4. Total Cases Commenced During Month	81				
Cases Terminated During Month By:					
5. Settlement/Voluntary Dismissal Prior to Trial	101				
6. Default Judgment	121				
7. Trial – JUDGE (with witnesses)	141				
8. – Decree on Depositions				173	
9. – Recommendation by Commissioner	181				
10. Trial – JURY	201				
Purged per Virginia Code Section 8-154					
11. – after two years	221				
12. – after five years	241				
13. Other	261				
14. Total Cases Terminated During Month	281				

Cases Set For Trial	Law	Chancery
15. During Month, Number of Cases Assigned Trial Dates	301	
16. Of the Cases Reported in Line 15 above, the Number of Cases Set Ninety (90) Days or More From Date of Setting	321	
Jury Trial Days		
17. Number of Days Spent in Jury Trials During Month	341	

History of Terminated Cases Number of cases terminated during month which were filed:	Condemnation	Appeals	All Other Law	Divorce	All Other Chancery
18. This Year	361				
19. One Year Ago	381				
20. Two Years Ago	401				
21. Three Years Ago	421				
22. Four Years Ago	441				
23. Five or More Years Ago	461				
24. TOTAL – The Sum of these should equal the sum of Line 14.	481				

_____ Date

_____ Circuit Court Clerk

_____ Report Preparer

MONTHLY CASELOAD REPORT CRIMINAL CASES

2			
Circuit I.D. Number			

Month		Year	

R
13

STATISTICS FOR THE MONTH OF:

City/County of:

For Office Use Only
621

Cases on Docket	Class 1 & 2 Felony	Other Felony	Misdemeanor	Post-Conviction	Habeas Corpus
Cases Commenced During Month By:					
1. Indictment/Presentment/Information	21				
2. Appealed from General District Court			49		
3. Appealed from J&DR Court	61				
4. Reinstatement	81				
5. Other	101				
6. Total Cases Commenced During Month	121				
Cases Terminated During Month By:					
7. Withdrawal Prior to Trial			149		
8. Dismissed by Court Without Trial	161				
9. Nolle Prosequi by Commonwealth Attorney	181				
10. Guilty Plea	201				
11. Trial—JUDGE (with witnesses)	221				
12. Trial—JURY	241				
13. Other	261				
14. Total Cases Terminated During Month	281				
15. Number of Fugitives Apprehended During Month	301				
16. Number of Fugitives Added During Month	321				
17. Total Cases in Abeyance at End of Month	341				

Cases Set For Trial	Felony	Misdemeanor
18. During Month, Number of Cases Assigned Trial Dates	361	
19. Of Those Cases Reported in Line 18 above, the Number of Cases Not Set for Trial During Present Term	381	
Jury Trial Days		
20. Number of Days Spent in Jury Trials During Month	401	

History of Terminated Cases	Class 1 & 2 Felony	Other Felony	Misdemeanor	Post-Conviction	Habeas Corpus
Number of cases terminated during month which were filed:					
21. This Term	421				
22. Prior to This Term, but not more than Five Months Ago	441				
23. From Five to Nine Months Ago	461				
24. More Than Nine Months Ago	481				
25. TOTAL — The Sum of these should equal the Sum of Line 14.	501				

_____ Date

_____ Circuit Court Clerk

_____ Report Preparer

DAILY COURTROOM ACTIVITY SHEET

2						T					
1	Circuit I.D. Number					7	9	Month	Day	Year	

ACTIVITY CODE			
CIVIL CASES		7. Docket Day	12. Trial-Jury
1. Default Judgment			13. Other
2. Trial-Judge (w/witnesses)		CRIMINAL CASES	
3. Trial-Jury	8. Dismissed by Court w/o Trial	14. Pre-trial Evidentiary Hearings	
4. Other	9. Nolle Prosequi By Commonwealth Attorney	15. All Other Pre-Trial Hearings	
5. Pre-Trial Hearings	10. Guilty Plea	16. Sentencing Hearings	
6. Post-Trial/Judgment Hearings	11. Trial-Judge (w/witnesses)	17. All Other Post-Conviction Hearings	
		18. Docket Day	

COURTROOM CLOCK TIME (08:30 to 09:20; 09:30 to 10:30; 10:40 to 10:45, etc.)				CIVIL								CRIMINAL					NUMBER OF ACTIVITIES	NUMBER OF CASES TERMINATED	ACTIVITY (See Code Above)
				CONDEMNATION	APPEALS	ALL OTHER LAW	DIVORCE	ALL OTHER CHANCERY	CLASS 1&2 FELONY	ALL OTHER FELONY	MISDEMEANOR	POST-CONVICTION	HABES CORPUS						
				15	20	25	F	G	H	I	J	K	L	26-27	28-29	30-32			
Sample Entry	0	8	30	A	M	to	1	2	30	P	M						3	3	115
				M		to													
				M		to													
				M		to													
				M		to													
				M		to													
				M		to													
				M		to													
				M		to													
				M		to													
				M		to													
				M		to													
				M		to													

Exhibit 12

DAILY CHAMBERS ACTIVITY SHEET

ACTIVITY CODE	
101.	Hearing Motions
102.	Pretrial Conferences
103.	Researching/Writing Opinions
104.	Other Case-Related Work
105.	All Other Work
106.	Travel Time

2			
---	--	--	--

1 Circuit I.D. Number

T

7

9	Month	Day	Year
---	-------	-----	------

CHAMBERS CLOCK TIME (E.G., 08:30 to 9:20; 09:20 to 10:00; 10:50 to 11:30)				CIVIL								CRIMINAL								NUMBER OF ACTIVITIES		NUMBER OF CASES TERMINATED		ACTIVITY (See Code Above)
15	20	25	F	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	26-27	28-29	30-32		
0 8	3 0	A M	to	1 2	3 0	P M														3	1	101		
		M	to			M																		
		M	to			M																		
		M	to			M																		
		M	to			M																		
		M	to			M																		
		M	to			M																		
		M	to			M																		
		M	to			M																		
		M	to			M																		
		M	to			M																		
		M	to			M																		

Exhibit 13

COURT OR CIRCUIT**24TH CIRCUIT**

-- CIVIL -----	CONDEM- NATIONS	APPEALS	OTHER LAW	DIVORCE	CHANCERY	TOTAL	CIVIL & CRIMINAL ---- TOTALS ----
COMMENCED	10	4	99	204	125	442	779
CONCLUDED	1	3	85	190	103	382	676
PENDING	32	22	314	174	472	1014	1643
#COMMENCED/CONCLUDED	10.00	1.33	1.16	1.07	1.21		
WEIGHT	30	1	8	4	8		
WEIGHTED COMMENCED	300	4	792	816	1000	2912	4345
WEIGHTED CONCLUDED	30	3	680	760	824	2297	3567

-- CRIMINAL -----	1 & 2 FELONY	OTHER FELONY	MISD	HAB. CORP. POST-CONV	TOTAL		
COMMENCED	20	188	127	2	337	#JUDGES	4
CONCLUDED	18	167	108	1	294	JUDICIAL STANDARD	1000
PENDING	114	183	331	1	629	WGHT. COMMENCED /	
#COMMENCED/CONCLUDED	1.11	1.13	1.18	2.00		JUD. STANDARD	4.35
WEIGHT	18	5	1	3		WGHT. CONCLUDED /	
WEIGHTED COMMENCED	360	940	127	6	1433	JUD. STANDARD	3.6
WEIGHTED CONCLUDED	324	835	108	3	1270		

Exhibit 14

to find the case records. These systems feed statistical systems and transmit data to other agencies.

The TRACER system in the Norfolk Court is an excellent system. It is an on-line criminal justice information system functioning for multiple agencies including Police, Jail, Commonwealth Attorney and Probation/Parole Department, as well as the District and Circuit Courts. The components of the system are a person master file, arrest/offense trailer records, and the index accessible by name, alias, social security number, Police I.D., etc. The reports generated by this system are docket, custody status, jail history, arrest record, and court caseload statistics (Exhibits 15 & 16). Other court information systems are in Fairfax County for all courts, Portsmouth—an automated traffic system, and Richmond—arrest system and juvenile records. The Juvenile Justice Information System, a time-sharing system, was a failure in the Tidewater region. Richmond Juvenile and Domestic Relations Court has installed a minicomputer which tracks support cases and performs docketing tasks. Frederick, Winchester, Roanoke, Portsmouth and Virginia Beach have implemented systems to automate some functions of the clerk's office and at the circuit court level to provide for juror management. Juror management includes selection, notification, and payment of trial juries.

CRIMINAL JUSTICE INFORMATION SYSTEMS

LEAA has established its Comprehensive Data Systems (CDS) Program to coordinate and accelerate the development of comprehensive state criminal justice information systems. Two components of the CDS are Computerized Criminal History (CCH) files and Offender-Based Transaction Statistics System (OBTS). The CCH files form a central report source for the important events in the cases of individuals charged with serious crimes. Virginia has developed

the Central Criminal Records Exchange (CCRE) for the state-level reporting. The CCRE is maintained by the State Police. Courts contribute disposition information, but although extensive files exist, little is done with the information except for police use (Exhibit 17). Since 1973, the OBTS has been in the development stage under the Secretary of Public Safety.

Many states are in a similar position to Virginia's. Partial systems are underway, yet none are fully satisfactory. There are three primary reasons. While the cost of development is funded by LEAA, operating costs are not. The annual running costs of such systems may amount to several million dollars. For the same cost, there may be alternatives which are more flexible and less risky. The second problem is that the more comprehensive a system, the more the quantity and quality of data input. This takes large amounts of time. Current laws also prohibit the replacement of the manual systems now in existence. If the manual systems are not replaced, the workload is significantly increased.

LEGAL RESEARCH

Legal research systems are under development or have been implemented by several commercial vendors. Basically these systems involve computer-assisted text look-up using keyword search and text selection algorithms. The LEAA-sponsored organization, SEARCH Group, Inc., has evaluated the practicality of computerized legal research for multiple criminal justice agencies and the three systems currently on the market. SEARCH is a non-profit consortium of states involved in criminal justice research and "dedicated to improving the administration of justice in the U. S. by applying advanced technology to justice systems." The projects include development of national standards and goals for criminal justice agencies, state judicial information sys-

tems (recently transferred to the National Center for State Courts), strategy for implementing privacy and security regulations, etc.

The three legal research systems studied were LEXIS, WESTLAW, and JURIS. LEXIS was developed by Mead

Data Corporation, WESTLAW by West Publishing Company, and JURIS by the U. S. Justice Department. JURIS was evaluated in Virginia for a six-month period. It was not successful since it contained federal cases not directly applicable to state law.

```

                                T R A C E R - P E R S O N S I N F O R M A T I O N          P A G E :
ACTION:      TRN:                NEXT FUNCTION:
NAME:        SEX:  RAC:  DOB:        HGT:    WGT:
ORI:         HAI:    EYE:   SKN:    POB:        STATE:
ALIAS:       SEX:  RAC:  DOB:        HGT:    WGT:
ORI:         HAI:    EYE:   SKN:    POB:        STATE:
RES ADD *:   DIR:  NAME:                STYPE:  CITY:
ORI:         STATE: ZIP:    PHONE*:(   )
EMPLOYER NAME:                OCC:
ORI:         ADDRESS:
RELATIONS NAME:                RELATIONSHIP:
ORI:         ADDRESS:
IDENTIFIERS: ORI:             LOCAL ID*:                SMT:
SID:         FBI:             SOC:                OLS:  OLN:
LIC:         LIS:  LIY:       MIL BRANCH:        STATUS:  MARITAL STATUS:
JAIL ID*:    J&D COURT*:      EDUCATION:    YEARS DEPENDENTS:
COMMENT:
ORI:         CAUTION IND:
NCIC FPC:                HENRY FPC:
ORI:

```

```

                                T R A C E R - P E R S O N S I N F O R M A T I O N          P A G E :
ACTION:      TRN: 00000001 NEXT FUNCTION:
NAME: JONES,MIKE          SEX: M RAC: W DOB: 030454 HGT: 510 WGT: 150
ORI:         HAI: BLK EYE: BLU SKN: FAR POB: LITTLETOWN    STATE: PA
ALIAS: CRAZY JOE         SEX:  RAC:  DOB:        HGT:    WGT:
ORI:         HAI:    EYE:   SKN:    POB:        STATE:
RES ADD *: 124  DIR: S  NAME: SONNY                STYPE: ST CITY: NORFOLK
ORI:         STATE: VA ZIP: 12345 PHONE*:( 803 ) 345 - 1122
EMPLOYER NAME: ACME BRICK CO                OCC: BRICKLAYER
ORI:         ADDRESS: WATERFRONT DR
RELATIONS NAME: JONES,JACK                RELATIONSHIP: FATHER
ORI:         ADDRESS: 235 S MISSION RD NORFOLK
IDENTIFIERS: ORI:             LOCAL ID*: 33444433  SMT: SCR L HND
SID: VA9886777 FBI: 66677G  SOC: 345872344 OLS: VA OLN: 9978877666
LIC: AXD347  LIS: VA LIY: 76 MIL BRANCH: ARMY STATUS: R MARITAL STATUS: S
JAIL ID*: 977778 J&D COURT*:      EDUCATION: 10 YEARS DEPENDENTS: 0
COMMENT: KNOWN TO RESIST ARREST
ORI:         CAUTION IND: X
NCIC FPC: AATTAA12TT7865PIPOTT HENRY FPC:
ORI:

```

The Virginia State Bar has developed a data base of Virginia Supreme Court decisions and Attorney General opinions to be used for legal research. The project has missed many deadlines and has cost overruns. The funding agency recently recommended discontinuance.

FUTURE DEVELOPMENT

In 1979, the Supreme Court of Virginia will begin the study, "Computer Options of the Virginia Judicial System." The objective of the study is to provide a comprehensive

T R A C E R - ARREST INFORMATION

ACTION: ARREST REPORT * TRN*: FCT: PAGE:
 NAME: SEX: RAC: DOB: ORI:
 ADDR: CITY:
 POA: ADDRESS: AOF: AOF*: PCT: D/STN: SQUAD:
 DOA: TIME: ASSISTING OFFICER 1: 2: PHOTO*:
 GENERAL COMMENT:

CHARGE CIT: C/S: M/F: AON: GOC: WRNT*:
 DOO: TIME: ST*: DIR: NAME: STYPE:
 WRNT BY: COMPLNT: COURT DATE: DOCKET:
 CIR:

CHARGE CIT: C/S: M/F: AON: GOC: WRNT*:
 DOO: TIME: ST*: DIR: NAME: STYPE:
 WRNT BY: COMPLNT: COURT DATE: DOCKET:
 CIR:

AMOUNT OF BOND\$: BOND TYPE: DATE: TIME: MAGISTRATE:
 MORE CHARGES: Y

T R A C E R - ARREST INFORMATION

ACTION: ADD ARREST REPORT * TRN*: 00000001 FCT: PAGE:
 NAME: SEX: RAC: DOB: ORI: VA1170000
 ADDR: CITY: NORFOLK VA
 POA: ADDRESS: 132 E TIDEWATER DR
 DOA: 010176 TIME: 1800 AOF: RANDOLPH, ADAM AOF*: 1234 PCT: 1 D/STN: SQUAD:
 ASSISTING OFFICER 1: 2: PHOTO*: 12345
 GENERAL COMMENT:

CHARGE CIT: 18.2-50 C/S: S M/F: F AON: 0900 GOC: WRNT*: 123456644F
 DOO: 010176 TIME: 1300 ST*: 132 DIR: S NAME: BOSTON STYPE: RI
 WRNT BY: PLG COMPLNT: SMITH, MIKE COURT DATE: 010276 DOCKET: CF
 CIR: DEFENDANT ASSAULTED 2 PERSONS RESULTING IN THE DEATH OF MIKE
 SMITH BY STABBING

CHARGE CIT: 18.2-51 C/S: S M/F: F AON: 1300 GOC: WRNT*: 877655555F
 DOO: TIME: ST*: DIR: NAME: STYPE:
 WRNT BY: COMPLNT: JONES, PAUL COURT DATE: DOCKET:
 CIR: DEFENDANT WOUNDED PAUL JONES BY STABBING

AMOUNT OF BOND\$: 0005000 BOND TYPE: DATE: TIME: MAGISTRATE: PLI
 MORE CHARGES: N

DISTRICT COURT		CIRCUIT COURT	
FILE NUMBER:	INITIAL COURT DATE:	DOCKET NUMBER:	DATE OF FILING:
RETENTION DECISION (CHECK ONE): OWN RECOGNIZANCE <input type="checkbox"/> BAIL SET <input type="checkbox"/> RELEASED ON SUMMONS <input type="checkbox"/>		TYPE OF FILING (CHECK ONE): CERTIFIED FELONY <input type="checkbox"/> MISD. APPEAL <input type="checkbox"/> INFORMATION <input type="checkbox"/> REINSTATEMENT <input type="checkbox"/>	
PLEA (CHECK ONE): NOT GUILTY <input type="checkbox"/> GUILTY <input type="checkbox"/> NOLLE CONTENDERE <input type="checkbox"/> OTHER <input type="checkbox"/> UNKNOWN <input type="checkbox"/>		GRAND JURY FINDING (IF APPLICABLE, CHECK ONE): INDICTMENT <input type="checkbox"/> NO TRUE BILL <input type="checkbox"/>	
TYPE OF COUNSEL (CHECK ONE): COURT APPOINTED <input type="checkbox"/> PUBLIC DEFENDER <input type="checkbox"/> PRIVATE <input type="checkbox"/> WAIVED <input type="checkbox"/>		DATE OF ARRAIGNMENT: RETENTION DECISION (CHECK ONE): OWN RECOGNIZANCE <input type="checkbox"/> BAIL SET <input type="checkbox"/> RELEASED ON SUMMONS <input type="checkbox"/>	
DISPOSITION (CHECK ONE): GUILTY <input type="checkbox"/> DISMISSED <input type="checkbox"/> CERTIFIED TO GRAND JURY <input type="checkbox"/>		INITIAL PLEA (CHECK ONE): NOT GUILTY <input type="checkbox"/> GUILTY <input type="checkbox"/> NOLLE CONTENDERE <input type="checkbox"/> OTHER <input type="checkbox"/> UNKNOWN <input type="checkbox"/>	
CONVICTED OF:		TYPE OF COUNSEL (CHECK ONE): COURT APPOINTED <input type="checkbox"/> PUBLIC DEFENDER <input type="checkbox"/> PRIVATE <input type="checkbox"/> WAIVED <input type="checkbox"/>	
SENTENCE IMPOSED BY COURT:		TRIAL DOCKET DATE:	
SENTENCE DATE:		TYPE OF TRIAL (CHECK ONE): TRIAL BY JUDGE <input type="checkbox"/> TRIAL BY JURY <input type="checkbox"/>	
NAME OF COURT:		DISPOSITION (CHECK ONE): GUILTY <input type="checkbox"/> NOT GUILTY <input type="checkbox"/> DISMISSED <input type="checkbox"/> NOLLE PROSEQUI <input type="checkbox"/> OTHER <input type="checkbox"/>	
SIGNATURE OF CLERK:		DISPOSITION RENDERED BY (CHECK ONE): JUDGE <input type="checkbox"/> JURY <input type="checkbox"/> DISPOSITION DATE:	
		CONVICTED OF: MISDEMEANOR <input type="checkbox"/> FELONY <input type="checkbox"/>	
		SENTENCE IMPOSED BY COURT: SENTENCE DATE:	
		NAME OF COURT:	
		SIGNATURE OF CLERK:	

Exhibit 17

plan to identify the areas in which computer development should be achieved. The plan will consider what systems currently exist, what functions of the court can be effectively automated, hardware needs of the judicial system, and the cost/benefits and feasibility of a statewide judicial computer network.

New court computerization will result from continual balancing between the ideal system and the practical constraints of the judicial branch of government. Each system will be carefully weighted and subjected to critical analysis in light of decreased federal funding. Contending needs will be addressed—needs for increased quality of and improved

information, simplicity and usability of systems, consideration of the judiciary's independent status, and flexibility to accommodate future needs. New systems must meet enunciated goals. They must have a sound basis on which more

complex systems can be built. Constant review and evaluation during development, implementation and operation will be required to ensure that the needs of the Courts are met.

Police and computer technology—The expectations and the results

by KENT W. COLTON

Brigham Young University
Provo, Utah

THE EXPECTATIONS—THE CHALLENGE OF THE PRESIDENT'S CRIME COMMISSION

In July, 1965, in the face of dramatic rises in reported crime and delinquency rates, the President's Commission on Law Enforcement and the Administration of Justice (sometimes called the Crime Commission) was created. One area selected for special attention in the Commission's final report was the potential contribution of science and technology in the generally labor-intensive field of law enforcement. Because criminal justice agencies must process enormous quantities of data, the use of computer technology—electronic computers and new techniques such as systems analysis, operations research and computer modeling—seemed particularly promising, and the use of computer technology by the police has expanded significantly since the mid-1960s.

A variety of factors have fueled this growth. The first was the report of the Crime Commission. The recommendations of such a distinctive group drew instant attention and outlined high expectations: "Modern technology can provide many new devices to improve the operations of criminal justice agencies, and particularly to help the police to deter crime and apprehend criminals."¹ The Commission's recommendations were fortified by the addition of large-scale federal resources to the police area through the Law Enforcement Assistance Administration (LEAA). The pressure from vendors to sell their product—enhanced as the Vietnamese War was ending and technology-oriented industries sought to increase their domestic market—also contributed to the expansion of the computer-related innovations. According to one study, \$143 million, or 11.5 percent of the total LEAA block grant budget, was spent for law enforcement telecommunications during the three-and-one-half years between July 1, 1971 and January 1, 1975, and this figure did not include matching money from the states.²

The Crime Commission report was filled with enthusiasm and raised high expectations about the possibilities of such innovations. Advocates felt that computer technology would allow for the rapid processing of information, expand police capabilities and improve law enforcement services, for example by reducing response time. Some hypothesized that

aspects of the technology might even improve apprehension rates, thus deterring criminal activity and reducing crime rates.

The use of computer technology by the police has expanded rapidly since the mid-1960s, undoubtedly aided by the Crime Commission's report and federal funding. However, there is disagreement as to the utility of such computer use. Whereas proponents seek for the benefits noted above, critics claim that much of the money has been wasted, that such innovations have not increased the efficiency or effectiveness of crime control, that the proliferation of such systems represents a potential infringement on civil liberties, and that the money could be better utilized on less technical approaches to the crime problem.³

Although there has been a lot of dialogue regarding the purchase and application of computer technology in law enforcement, there has been relatively little research or evaluation since the Crime Commission concerning the actual uses, difficulties, and diffusion of computer technology by the police. Despite prestigious recommendations, the process of introducing change requires more than directives from the top. Important behavioral and power relationships are involved in the actual implementation of the technology. A decade has passed since the Crime Commission selected computer technology as an area of potential significance. The purpose of this paper, then, is to begin to evaluate what we have learned since the mid-1960s and to address the consequences and diffusion of innovation.

The paper is based on the results of research efforts which have transpired over a period of six years. The research has included two national surveys of U.S. police departments in 1971 and 1974 (designed by the author and administered by the International City Management Association [ICMA]) and a series of seven case studies in different police departments around the country.⁴ The four sections which follow will 1) review the results of "routine" and "non-routine" applications of police computer technology, 2) discuss some of the lessons we have learned and the reasons for the disappointments and problems that have arisen, 3) outline a new focus for the diffusion and application of police computer technology, and 4) discuss the change in expectations for the future.

THE RESULTS—THE EXPERIENCE OF THE PAST DECADE

The use and evaluation of police computer technology

The first real-time police computer system in the U.S. was installed in the St. Louis Police Department in the mid-1960s. Since then the growth of computer technology within police departments has been widespread. However, the surveys conducted as a part of this study in 1971 and 1974 revealed that implementation has been slower than expected. The 1975 survey was mailed to all U.S. police departments in cities with populations over 50,000. Of the 326 (80 percent) that responded, 193 (56 percent) were using computers. Although this was an increase of 12 percent over 1971 responses, it was only about half the growth predicted by the earlier survey.⁵

Some of the difference may be explained by a slight variation in response rate between the two studies and by varying interpretations of survey questions. But, more important, estimates of future growth tend to be overly optimistic. The slower rate may also indicate that some police departments are taking a more careful and sophisticated approach to computer use.

When surveyed, police departments with computers were asked to identify which of 24 applications they were using. The 24 applications were grouped into eight areas: police patrol and inquiry, traffic, police administration, crime statistical files, miscellaneous operations, resource allocations, criminal investigation and command and control.

In evaluating use and impact, it has been useful to draw a distinction between "routine" and "non-routine" applications of computer technology.⁶ Routine applications involve the relatively straightforward, repetitive manipulation and inquiry of prescribed data, often by means of a definite procedure. The same manipulation was usually done by hand before the advent of the computer. Technology simply makes the process quicker and easier. For example, although police patrol and inquiry applications were technically advanced and provide rapid retrieval of information to the field officer, such inquiry systems are relatively straightforward and the tasks can be labelled routine. Other routine application areas comprise traffic files, crime statistical files, police administration and miscellaneous operations.

In non-routine applications the machine becomes a tool for decision-making, strategic planning, and person-technology interaction. There are no absolute methods for handling problems, either because the area is complex or because they require custom-tailored treatment. The human decision-maker plays a vital role in judgment, evaluation and insight. Non-routine application areas in law enforcement include resource allocation, investigation of crime, and command and control—including computer-aided dispatch and automatic vehicle monitoring. (See Figure 1.)

Rather than view routine and non-routine categories as sharply distinct classifications, though, they should be regarded as delimiting the two ends of a continuum. As applications move toward the non-routine end of the continuum, systems design becomes more intricate, and

behavioral, personality and organizational considerations become more significant. Several applications fall between the two extremes. The best example is crime statistical files, which though generally routine in collection and processing, provide the basic data for a number of non-routine activities, such as resource allocation. Command and control applications also have both routine and non-routine dimensions.

As the use of computer technology has evolved since 1960, successful implementation has often been limited to the routine areas. Traffic, police administration, and crime statistical files have all remained important; and the expansion of police patrol and inquiry records—especially in the late 1960s and early 1970s—has been almost phenomenal.⁷ *In the non-routine areas, though, results have been far more disappointing.* For example, in the previously referenced 1971 survey, 61 departments predicted they would implement a computer-aided dispatch system. However, only 15 such systems had been installed by 1974—less than one percent of the computer applications reported in the 1974 survey.

Resource allocation has been the only non-routine computer use where the number of applications actually implemented has exceeded expectations. The 1971 survey results indicated that in three years 12 percent of all computer allocations would be in the resource allocation area; the actual percentage was 16. An additional question in both the 1971 and 1974 surveys asked police departments to rank the relative importance of different computer allocations. There was little shift between the two years, and in both 1971 and 1974, resource allocation applications were ranked first.

Although the actual level of implementation has been below earlier expectations in a number of areas, the computer, with all its interesting implications and problems, has unquestionably become a permanent part of law enforcement technology after a decade and a half of use. The issue now is not will computers be used, but how and with what impact?

Computer impacts

The impact of routine applications. Although experiences vary from city to city, there is evidence that routine computer applications provide a number of benefits, particularly when benefits are defined in a narrow, technical sense. For instance, numerous police patrol and inquiry applications and crime statistical files are working around the country today. Seven-second retrieval of information to the officer in the street has been a reality in Kansas City, Los Angeles, and other police departments for a number of years. In terms of "technical impacts"—benefits resulting from improvements in the input, processing, and output of information—the technology has provided a number of positive advantages. In at least some departments, extensive amounts of new or better information are available more rapidly for broader distribution, although results again vary among police agencies. Further, if one views the service delivery impact of such routine applications from a more narrow "process"-oriented perspective, a number of routine applications have improved service to the public and have been

Routine	Non-Routine
Police patrol and inquiry (including --> warrant, stolen property, and vehicle registration files) ^b	
Traffic applications (including -----> traffic accident, citation, and parking violation files)	
Miscellaneous operations (including ---> intelligence compilation and jail arrest records)	
	←--- Command and control (including computer aided dispatch and automatic vehicle monitoring)
	←--- Criminal investigation (including automated field interrogation reports, modus operandi and automated fingerprint files)
Crime statistical files (including crime offense, criminal arrest, juvenile criminal activity, and offender based files)	
Police administration (including -----> budget analysis and forecasting, inventory control, vehicle fleet maintenance, payroll preparation, and personnel records)	←--- Resource allocation (including police patrol allocation and distribution, police service analysis, and traffic patrol allocation and distribution)

Figure 1—Routine and non-routine uses of police computer technology.* The terms "structured" and "unstructured" have also been used to draw a similar distinction. See, for example, G. Anthony Gorry and Michael S. S. Morton, "Management Decision Systems: A Framework for Management Information Systems," Working Paper No. 458-70, Alfred P. Sloan School of Management, MIT, April 1970. Also, Herbert A. Simon originally used the terms "programmed" and "unprogrammed" to make a related characterization. See Herbert A. Simon, *The Science of Management Decisions*, New York, Harper & Row, 1970, p. 6.

* The dotted arrows reflect the fact that routine and non-routine categories are not sharply defined classifications. Rather, they should be regarded as converging from opposite ends of a continuum.

shown to be cost-effective, though full-scale analysis of costs and benefits were not covered in this project. For example, in Tulsa, Oklahoma, an additional \$180,000 in estimated revenue was returned after the first year's operation of a new automated traffic citation system. In Long Beach, California, membership in an automated want/warrant system in the Los Angeles area increased the number of 1970 warrant arrests 31.5 percent over 1969 figures.⁸ In Kansas City, Missouri, the ALERT (Automated Law Enforcement Response Team) system was installed in 1969, and the num-

ber of monthly inquiries per police officer concerning stolen cars or wanted persons rose from 36 in January to 90 in May 1971, and in 1975 police officers were averaging 250 inquiries per officer per month. In Oakland, California, after digital computer terminals were installed in half the patrol cars in 1971 and 1972, units with terminals in their cars made more than seven times as many information requests, received more than three times as many "possible hits," and were three times as productive in warrant arrests and vehicle recoveries as nonequipped units.⁹

However, when one examines the actual service results or outcomes of such routine applications the benefits of the technology are more uncertain and unexpected impacts and influences begin to emerge. For example, a former Kansas City Chief of Police reported that after installing their ALERT system, one of the most advanced police patrol and inquiry systems in the country, the police department experienced an overload of police officers making stolen car checks, thereby creating a potential manpower drain and shifting concentration from other vital police tasks such as preventive crime patrol.¹⁰

Further, as far as service impacts are concerned, it seems that routine computer uses by the police have almost entirely been devoted to the crime control and law enforcement functions of the police.¹¹ By over-emphasizing the application of technology to crime control, law enforcement agencies may neglect possible applications to social service activities; for example, computer files to assist with referral information, medical assistance, or listings of agencies and names of people who might provide social service assistance.

Finally, large resources from the LEAA have in some cases served as a "seductive stimulant" for police departments to get involved with computer technology in the absence of an intrinsic desire for understanding. As one police data processing manager put it, "Millions of dollars have been spent, but there's still an awful lot of garbage coming out of police computer systems." Although no one knows how much waste and misuse exists, police computer hardware has undoubtedly been sold to police departments that don't know how to use it, or for nonessential applications.

The impact of non-routine applications. Although the service and power shifts of routine computer applications raise certain questions and concerns, overall a number of routine applications have been successful, especially in terms of operational performance and technical impacts. However, non-routine uses of computer technology bring greater complexity both in terms of implementation and evaluation. In this study, case studies have been conducted in two areas of non-routine use—resource allocation and command and control. Each will be discussed.

As noted above, in surveys in both 1971 and 1974, police departments considered *resource allocations* to be their most important areas of computer use. Resource allocation was also the only area in which the number of applications reported in the 1974 survey actually exceeded 1971 predictions. All police departments must make deployment decisions and the interest in the use of technology to aid in this allocation process is growing. However, the interest in automated police deployment should be placed in the context of a realistic understanding of the law enforcement environment. The resource allocation applications noted in surveys generally refer to using tabulations of crime statistics to determine deployment, not to more sophisticated models.¹² Even where modeling work has been tried, many of the efforts have met with only limited success as the three cases examined as a part of this study indicate.¹³

In St. Louis the use of a computer model that was implemented in the late 1960s is purely optional as of 1977, and

district captains no longer request computer-generated reports. The command staff and the Board of Police Commissioners are essentially doing nothing to encourage use of the system by other commanders. In Boston, efforts were made to implement a computer simulation model in the early 1970s but the proposed deployment techniques were dropped entirely in 1973, and questions have even been raised within the police department concerning the manual resource allocation procedures that were implemented in 1974.

Of the three cases reviewed in this paper, the Los Angeles Police Department (LAPD) has the only resource allocation system utilizing computer technology which is actually operating and established as a part of its deployment process. The first level of evaluation—having a working system—has been met. However, even there, the objectives of the resource allocation project were substantially modified. The original LEMRAS/ADAM deployment model¹⁴ was dropped in 1974 to be replaced by the ADAM historical reporting system which was implemented in June, 1975. The current ADAM package no longer includes forecasts of future needs, and deployment recommendations are based on manual calculations using computer-generated reports of historical data. The LAPD has achieved technical benefits in terms of reducing the manpower required to analyze workload and to calculate deployment plans, but many of the service impacts are still unclear. For example, conflicts arose between the strategy for allocation implicit in the deployment model and team policing, an alternative strategy for police work. Finally, one of the original service objectives of the initial allocation system, improved crime prevention, has been virtually abandoned as one of the factors considered in the current ADAM historical reporting system.

Efforts in police departments to utilize computer technology in resource allocation go far beyond the St. Louis, Boston, and Los Angeles case studies. The modeling techniques used in these three cases are now outdated, and improved models have been developed. For instance, a number of projects are currently underway to implement two more recent modeling efforts: the Patrol Car Allocation Model (PCAM) and the Hypercube Model.¹⁵ These models allow the user to identify a wide range of performance measures—i.e. mean travel times to various locations, workload balances, response to call-for-service and other dispatching strategies—and based on the relative importance of these various measures, alternative deployment strategies are provided. As a consequence, some of the objections in St. Louis and Los Angeles—that those modeling efforts did not consider enough of the relevant factors—have been overcome. The actual results of most of these efforts still must be evaluated, though. Further, the implementation problems encountered in the three cases discussed in this paper do not seem to be isolated instances. Rather, there is strong evidence that such difficulties are commonplace.

According to a 1975 report by the RAND Corporation that examined a number of attempts to implement computer models in the criminal justice area: "Through a series of interviews with model builders and personnel in agencies that attempted to implement models, a picture of the implementation process was obtained. In general, criminal justice

models have failed to achieve any notable level of use for policy decisions.¹⁶

The potential for automating aspects of *police command and control* were first pointed out by the Crime Commission in 1967. Computer-aided dispatch (CAD) systems provide the framework for bringing together many of these new tools through the partial automation of the call-answering and dispatch process. Other command and control technological changes that have been considered or tried include mobile and portable digital terminals to allow officers in the street to communicate digitally with headquarters, automatic vehicle monitoring (AVM) systems to keep track of the location and monitor the status of police units, and 911 emergency telephone services. A CAD system may include an AVM system, 911 telephone service, or mobile digital terminals.¹⁷ Some of these innovations in command and control are routine: the technology basically replaces a previously manual activity such as with digital terminals or the automated transfer of information from the telephone operator to the dispatcher. However, CAD also provides the framework for a number of non-routine activities, such as tracking and monitoring vehicle location, automatically timing the lengths of calls and raising a "flag" if a call takes over a specified time (say 30 minutes), or providing new information to be used for management. Command and control as discussed in this report, then, relates not only to dispatch deployment, but to the ability of police administrators to control and modify the manner in which police operations are conducted.

In the study providing the basis for this study, three cases were examined in the command and control area, and in San Diego and New York City working systems have been developed, although in Boston the problems of introducing the new technology have been more significant.¹⁸ The success and failures of these three cases provide certain insights for the future. First, it is possible to establish ongoing, operational CAD systems. The SPRINT system in New York City has been working since 1970, and the CAD system in San Diego has been operating since 1975. Both cities have achieved technical benefits from CAD such as the availability of new and better information, rapidity in matching addresses with geographic location, the effective transfer and recording of data in the dispatch process, and the retrieval of data from the dispatch process.

Secondly, both cities have experienced positive service impacts in terms of process-orientated measures. Some of these process service benefits include: telephone calls are answered and serviced more rapidly (telephone talk time in San Diego has dropped from three minutes to 77 seconds, and the average time required to answer the telephone is 2.5 seconds); standards can be set for communications and field backlogs (New York City has met its standard of answering 98 percent of telephone calls within 30 seconds, and radio airtime and field backlogs are monitored and recorded daily); and the workload has been more evenly distributed within communications divisions.

Thirdly, when it comes to measuring the actual service "results" attributed to CAD, the findings are inconclusive. In the New York City and San Diego police departments

there is a general feeling that dispatch time has been reduced, but the data are inadequate to prove or disprove such a hypothesis. In fact, to the extent that data exist, they seem to show that the impact on response time has generally been negligible or modest at best.¹⁹ Further, the police departments have essentially not analyzed the influence of the CAD systems in such areas as improving police productivity by enabling patrol officers to respond to more calls per shift or providing a better match between police service needs and available resources.

The question remains, then, as to whether the benefits of CAD justify the costs. Although the expenses of much of this technology seem high, when placed in the overall context of the costs of police operations, the comparative magnitude of the dollars seems to diminish. In New York City, for example, the annualized costs for developing and operating the SPRINT system are about \$2.7 million. Because the 1975 police budget in New York City was approximately \$625 million, only 0.4 percent of the annual budget was devoted to the CAD system. Stated in another way, the costs of operating SPRINT are roughly equivalent to maintaining 10 police patrol units on an annual basis.

In both New York City and San Diego, technical and service benefits have been achieved to help offset such costs, and it seems highly likely that the use of CAD systems will continue to expand. Whether their full potential is achieved, though, will depend on the skills of the management personnel. Both New York City and San Diego provide a wide range of new information for managers. However, police chiefs have seldom considered themselves as managers in the past; rather, their responsibility has been to balance pressures within and without the city and to promote the need for law enforcement and police resources. Consequently, it is still unclear as to whether they or their assistants will be able to channel the potential technological talents of the computer to do more than simply perform routine operations.

THE CRIME COMMISSION REVISITED (OR SOME OF THE REASONS WHY THE RESULTS OF POLICE COMPUTER TECHNOLOGY HAVE BEEN MIXED)

When the Crime Commission issued its report in 1967 it was optimistic about the use of science and technology in law enforcement. It set forth a far-ranging program of application and experimentation. Some of these experiments have worked, but a number of others have failed, and whether explicitly or implicitly, the Commission oversold the potential impact of such innovations on reducing crime and increasing arrests. It also seemed to assume that innovation would occur automatically from the top down, that little attention was required for the diffusion process, and that the only motives for implementation would be altruistic, and that vendors of technology would be neutral and pressure-free in their "unbiased advocacy." Finally, they recommended so many possible experiments that it was difficult to select and focus priorities and to follow through. What

have we learned from our experience over the past decade and what recommendations can be made for the next few years?

Firstly, it should be clear that it is extremely difficult to measure the effectiveness of technological innovations in confronting crime. In a number of cases, particularly as reported in the overall study report, allocation and command and control projects failed to demonstrate clear improvements in a department's patrol performance, particularly in the area of crime control. Perhaps the greater failure was the original expectations which were built in the 1960s that we might be able to establish such linkages. Criminal activities are based on a wide range of factors only a small portion of which are influenced by police activity. Changes in deployment patterns or response rates may have some modest influence, but criminal statistics are far too imprecise to measure these differences or to isolate the portion of the change attributed to police allocation or technology as opposed to changes, for example, in the weather or the unemployment rate.

Secondly, it should be apparent that a number of the original specific objectives of the Crime Commission will not be met, and expectations for the future must be altered. The best illustration of this is related to response time. Based on the evidence to date it would be a mistake to maintain hope that response time benefits will justify command and control and resource allocation technological innovations. As noted earlier in this report, the CAD system did not achieve response time benefits. Further, in St. Louis tests of a Phase I AVM system, it was found that AVM did not bring the expected reduction in response time. In fact, although the question will be examined again closely in a Phase II experiment, current findings lack any evidence to suggest that travel time reductions due solely to AVM will significantly improve police operations or reduce costs.²⁰ The entire response time system includes a number of components, not the least of which is the time it takes the victim to call the police after a crime has occurred. In the past, excessive attention has been focused on the elements of the response system which can be influenced by technology. In fact it seems after reviewing the evidence of this report that response time is primarily a personnel and human issue rather than a technical problem. If response time is to be improved, people who have been victimized will need to call the police more rapidly, or a department will need to reorganize *both* the flow of the technology *and* the flow of people related to their communications system. Technology alone will make little difference.

Thirdly, the experience of police departments in using computer technology to date has forcefully demonstrated the importance for performance guidelines in the diffusion of such innovation. The relationship between the user and the vendor must be clearly defined and performance guidelines specified. In San Diego there was a very clear set of vendor specifications in their request for proposal for the CAD system, and this was invaluable in achieving the desired product. The Boston proposal for CAD lacked the same clarity, and misunderstandings inevitably developed. In the long run, both the police and the vendors of technol-

ogy will benefit from a clear framework and set of standards and specifications. In fact, it is the conclusion of this report that effective implementation necessitates such standards, and the Law Enforcement Assistance Administration, or its sequel, should play a central role in developing such guidelines.

Finally, it seems that at least one of the major reasons for the disappointment of the Crime Commission was its failure to recognize many of the complexities and motivations concerning the implementation of technology and the interaction between the context and nature of police work and the technology. Police organizations have a number of characteristics that are quite different from those of other public and private institutions. In most industrial organizations and public bureaucracies, movement to higher levels of power and status is accompanied by greater discretion or freedom of choice in decision-making. Complexity of task increases with responsibility. By contrast in police bureaucracies, the lowest-ranking officer—the patrol officer—is often given the greatest discretion, being forced to continually make decisions without direction from superiors, and consequently the administrator's ability to control and influence police behavior is severely limited.

A further complication in understanding the police is the local and fragmented nature of law enforcement and the fact that police departments have a variety of different tasks and styles of operation. The popular conception of police work, often supported both by news media and by movies and television, is one which assumes that the bulk of a policeman's time is devoted to the exciting and dangerous job of crime-fighting. In fact, a comparatively small part of a policeman's time is devoted to *crime control and law enforcement*. Instead, *service activities* and *order maintenance* occupy the largest portion of police time,²¹ and different police departments have different styles of operation depending on whether their orientation is, for example, *legalistic* (identified by strict interpretation and enforcement of the law and strong centralized authority), *watchman* (characterized by a more traditional approach, greater discretion and weaker centralized authority) or *service-oriented*.²²

In summary, then, the eventual influence and impact of technology in policing will not come from the technology per se, but from an interaction between police work, the nature of a particular department, and any specific innovation. When the Crime Commission set forth its recommendations in 1967, it apparently assumed, at least in part, that police administrators would have strong centralized control and that the diffusion of innovation in the form of computer (and other) technology would be primarily an act initiated from above with effective communication from higher to lower echelons of the police department providing the linkage for implementation. The primary problem recognized by the Commission was monetary,²³ and in failing to more specifically address the diffusion of technology, many of the obstacles such innovations have met over the past decade were overlooked. Given such factors as the fragmented nature of police work and the variety of police departments around the country, the use of technology may have an important influence on power and prominence within orga-

nizations. Behavioral factors have proved essential in achieving acceptance and success, and the nature of innovation and change is a long-term and deeply-rooted process. With this in mind, the next section of this paper will examine a new focus for the diffusion of police computer technology.

THE DIFFUSION OF POLICE COMPUTER TECHNOLOGY—SOME DIRECTIONS FOR THE FUTURE.

There is a human tendency to seek direct solutions and to try to classify actions as either failures or successes. When it comes to the diffusion of technological innovation there seems to be no single prescription that will guarantee success. It is possible, though, to identify what not to do, particularly with the benefit of hindsight. Based on such hindsight and the analysis of the cases noted above, a series of "necessary-but-not-sufficient" conditions in the implementation process have been identified. The factors can be divided into two categories—those related to the nature of the environment of the innovation, and those related to the project management of the innovation. In essence, they are built upon and serve to summarize many of the common themes which have emerged from the case studies: the need for understanding the environment and motivations for change, the long term nature of innovation, vendor pressures and the temptation to oversell or overestimate a project's potential, the necessity of setting priorities and outlining clear performance guidelines in advance and the importance of human and behavioral considerations such as the continuity of personnel and the involvement of police officers at all levels to the extent possible. Listed in Figure 2, they serve as a "check list" for future consideration—not as a magic formula for success.

Obviously, it is impossible to expect that all of the factors relating to the nature, environment and project management of change can be achieved whenever computer technology is implemented. There is no simple answer to assure success. It is clear, though, that in the past we have failed to devote adequate attention to the implementation and diffusion of innovation not only in law enforcement but in almost all areas of urban service delivery. While trying not to raise our expectations beyond reach, it should be possible to concentrate our efforts at more effective evaluation and transfer, where appropriate.

The diffusion of innovation basically involves four steps:²⁴

Inventing—The creating of ideas, technologies, models, etc.

Informing—Publicizing the technology and educating the law enforcement community concerning the technology and its possible advantages and disadvantages.

Implementing—Introducing the technology into a law enforcement agency.

Integrating—The overall social and economic acceptance and adjustment to the innovation by the agency.

In developing a more realistic and productive outlook and direction for the diffusion of law enforcement technology, and for that matter, diffusion related to all urban services, all four deserve consideration.

Inventing—The need for better technology

Although this report has neither the space nor the capacity to be too specific, "better technology" improvements can and should be made in the quality of law enforcement computer applications. For example, in the modeling area we must build better models. Over the last decade, progress has been made. The Hypercube and PCAM Models offer better options to police users than those available six or seven years ago. Further, it may be possible, within the professional community of computer technology, engineering and operations research, to establish high standards and criteria by which inappropriate innovations can be weeded out.

Informing—The need for "truth in technology"

One of the greatest failings related to computer technology in the past decade is the tendency to overpromise. Expectations have been raised only to be dashed, due to a whole range of technical and behavioral factors. The primary change agents in law enforcement technology are vendors. However, they have a vested interest in selling their product and this interest has sometimes tended to focus sales literature on the advantages of technology as compared to the drawbacks. As noted earlier, the time is ripe to develop realistic performance guidelines and to try to assure that in the informing and educating process that the costs of technology, as well as the benefits, receive ample publicity.

Implementing—The need for "policy management"

The implementation process is not simply a matter of policy choice, but a process of conflict resolution requiring the understanding and management of different values and perspectives. It has become apparent in analyzing the implementation of law enforcement technology, that a new breed of police officers is emerging. These are officers who have "come up through the ranks" and have, therefore, "paid their dues" and are respected within the policy community. At the same time, they have had some experience with both the advantages and limitations of new technology and may be helpful in this process of conflict resolution. Rather than try to teach outside engineers about police work, it may be profitable to cultivate this inside set of "police technology experts." As long as they maintain this independence they could become a "pool of resources" to aid in the diffusion process.

1. Conditions related to the nature and environment of the innovation:

A clear and realistic understanding at the outset of the project of the policy issues involved. Multiple, even conflicting objectives are often involved. For example, when Los Angeles first began the LEMRAS project, they failed to appreciate the policy conflict between the model and team policing.

A perceived need for change among those influenced by the innovation -- both police administrators and officers in the street. Effective change must usually build from within an organization. If innovation becomes an "idea in good currency," its chances for success will rise significantly. One of the indicators of this perceived support is a willingness to pay for change. Both San Diego and New York City "used their own money," when installing CAD systems. Although projects funded from the outside may still succeed, often there is less commitment and support than in self-funded efforts.

Effective timing and system design so as to meet user needs and resist the temptation to oversell and therefore build impossible expectations. The first attempt at CAD in San Diego failed miserably because those involved in the design failed to identify the needs of users. The second effort focused special attention on user concerns and was implemented at a time when change seemed essential. The outcome was far more successful.

The proper selection of priorities in implementing computer technology. The most important formula seems to be to start with routine innovations that assist the officer in the street; more nonroutine innovations can be developed later to serve a more narrow range of officer needs. Also, the focus has been on crime and law enforcement activities. Perhaps if greater attention were devoted to service or order maintenance objectives, acceptance would increase.

2. Factors related to the project management of innovation:

Establishment of a clear set of performance guidelines at the beginning of a project. Such guidelines serve as a framework for clear understanding between the vendor and user. They were invaluable, for example, in San Diego, and their absence in other cities has been at the root of many difficulties.

A long-term framework and perspective. Eight years were spent in the implementation of the ADAM historical reporting system in Los Angeles, and the New York City SPRINT CAD system has evolved significantly within a seven year period. Such projects inevitably take longer than initially planned, and if an adequate time-frame is not allowed, frustration and rejection will ensue.

Emphasis placed on human-computer interaction. There is sometimes a tendency to consider computer technology as a replacement for people. This is both unrealistic and inefficient. One of the most critical variables for the efficient operation of any computer system is the development of the proper balance in the interaction between people and machines.

Effective training, education, and information dissemination. The process of communication is often at the heart of effective innovation. Carefully designed training programs provide an important link in such communication. However, innovators must be careful not to oversell and be prepared to listen to feedback. The dialogue process must be two way.

Continuity of personnel. Experience has shown that, as advocates for technological innovation move, the innovation often dies. Change in personnel is inevitable, but at the same time, a certain degree of continuity must be maintained.

Involvement and quality of top-level leadership. Police departments tend to be fairly rigid organizations with well established chains of command. Understanding, involvement and support from the top is essential if technological innovations are to be implemented and used. More than support from the Chief is required, though. In addition, a core of agency leaders is necessary if commitment is to be maintained over time.

Involvement of other police personnel. Besides the top commanders, police at the operating level must be involved in the design and development of computer technology. One reason the resource allocation system faltered in St. Louis was because the field officers strongly resisted a shift of only one hour in their daily schedules because it would have required them to commute to work during the normal rush hour traffic.

Caliber of computer systems and technical staff. Individuals are required who have both technical skills as well as a broad perspective which will allow them to see beyond computer technology to law enforcement needs and to communicate successfully with the police department. In order to attract such individuals, cities must be willing to pay competitive wages.

Unbiased evaluation. A careful (and, if possible, independent) evaluation should be an integral part of any implementation effort.

Figure 2 (continued)

Integrating—The need for the internal motivation and integrity of change

One of the most critical elements for implementation success is that the desire for change must come from within, not without. Better evaluation and guidelines for performance can help educate police departments as to the advantages and limitations of technology, and "pools of resources" from within and without the law enforcement community might establish a two-way communication to facilitate diffusion. Still, the final desire for change and the specific design and implementation of alternatives must come from within the police department involved. Openness and meaningful communication are required, and although it is difficult to maintain such behavior constantly, it is essential in helping to bring about effective innovation.²⁵

CONCLUSIONS—CHANGING EXPECTATIONS FOR THE FUTURE

There are a range of views about the use of computers and technology in our society. At one extreme are those who see the increasing movement towards a technological society as dangerous, a movement that will take us away from the "good life." Scientific rationality and technological progress may have questionable results and set up a chain reaction which we may not be able to reverse. At the other extreme are the technologists and the vendors who sell their products. They argue that the benefits of technology outweigh the costs and tend to oversell their products and to promise more than they can deliver. It is the opinion of this author that the truth lies somewhere in between. On the one hand computer technology has become a part of law enforce-

ment activity. Given this reality, the most useful orientation is to realistically evaluate current needs and progress and to promote change where it is appropriate. On the other hand, we must admit that many of our efforts at technological innovation have failed. Promises have been overextended, expectations have not been met, and resources have been wasted. The answer to our problems does not lie in hardware; it lies in basic value judgments and in people. In talking about a computer application in his police department, one police sergeant astutely remarked:

"The computer terminal in the car is an effort by the police department to professionalize from a hardware approach. This is O.K., but the more we concentrate on hardware, the farther we move from the basic people issues. The real police problems don't have technical solutions. Instead, it's the people who are screwed up, and we need more people-to-people type efforts in police departments, such as improvements in communication, improved motivation, productivity modifications, better interpersonal relations, etc. In short, instead of hardware solutions, we need policy resolutions of the basic issues of the police force. The result of the computer may be to take our minds off what are the more important issues."²⁶

In summary, most arguments against the computer are made on the grounds that too much money is currently being spent on law enforcement technology, particularly when it is not clear that the benefits of such technology justify the costs. However, this study has found that in many routine applications the benefits *can* justify the costs, particularly if benefits are defined in narrow, process-oriented terms. Further, this efficiency may continue to develop with time as computer technology becomes more sophisticated and police departments get better at handling the organizational and behavioral problems which often accompany the introduction of technology and the implementation of change.

More importantly, though, there are other issues surrounding the use of the computer that have greater significance than questions of costs and benefits. The use of computer technology by the police must be placed in perspective. Although many aspects of computer technology in law enforcement are well established and expanding, it would be a mistake to think such innovations will play a major role (at least in the short run) in revolutionizing the police or many of the issues they face. Police work, to a large extent, is determined by the conditions of our society and its people. Crime and law enforcement have a momentum of their own. Computer technology may have a marginal role in influencing and shifting relationships, but the major law enforcement issues must be resolved in the context of society as a whole. For example, some of the most pressing law enforcement questions at this time are to define the basic task of the police, to identify how the officer's time is really being spent, to determine the correct allocation of resources and to determine if current recruiting and training practices complement the basic needs and priorities of the police. The computer (along with proper analysis) may help in a small way to resolve these sorts of issues, but until such

questions are addressed the implementation of the computer may also serve to reinforce the status quo, to lock in and substantiate our present approach, and to indirectly countermand other innovation.

The greatest strengths of computer technology seem closely related to its greatest weaknesses. Computers have the potential to aid in criminal justice activities through rapid communication, accurate and complete information, and perhaps a more rational approach to decision-making. We must realize that there are limits to this technology, though, and not overestimate the potential. These very benefits, if not properly controlled or planned, may result in misuse, unintended consequences, wasted resources and frustrations. Expanded computer use by the police is at a crucial point and now is the time to point to a new direction, one slanting toward attention to evaluation and implementation, stressing performance standards and transfer, and realizing that police play a broader role in society than simply fighting crime. Such a new direction requires careful consideration so that the strengths of technology can be judiciously marshalled and the weaknesses and potential risks prudently forestalled.

NOTES

1. *The Challenge of Crime in a Free Society*, Report by the President's Commission on Law Enforcement and Administration of Justice (U.S. Government Printing Office: Washington, D.C., 1967), p. 246.
2. Kavenagh, Donal D., "Planning Guidelines for Law Enforcement Telecommunications Systems, Product of Project 13, Executive Summary," *Government Data Systems*, July-August 1976.
3. See for example, Casey, Sarah, *Law and Disorder IV*, Center for National Security Studies, Washington, D.C., and *Twentieth Century Fund Task Force Report, Law Enforcement: The Federal Role*, Twentieth Century Fund, New York, 1976.
4. For a detailed review of the survey and case study material, see Colton, Kent W., editor, *Police Computer Technology: Implementation and Impact*, Lexington Books, Lexington, Massachusetts, 1978.
5. The 1971 and 1974 ICMA surveys were designed by the author and administered by the International City Management Association (ICMA). For more detailed descriptions of the results of the surveys see: Colton, K. (Ed.), *Police Computer Technology: Implementation and Impact*, Lexington Books, Lexington, Massachusetts, 1978. Colton, K., "Computers and the police: police departments and the new information technology," *Urban Data Service Report*, Vol. 6, No. 11, International City Management Association (ICMA), Washington, D.C., November 1974.
6. The terms "structured" and "unstructured" have also been used to draw a similar distinction. Gorry, A. and Morton, M. S. S., "Management Decision Systems: A Framework for Management Information Systems," Alfred P. Sloan School of Management Working Paper No. 458-70, Massachusetts Institute of Technology, 1970. Also, Herbert A. Simon originally used the terms "programmed" and "unprogrammed" to make a related characterization. Simon, H. A., *The Science of Management Decisions*, Harper & Row, New York, 1970.
7. For a detailed discussion and documentation of the routine and non-routine applications referred to in this paper, see Colton, K. (Ed.), *Police Computer Technology: Implementation and Impact*, Lexington Books, Lexington, Massachusetts, 1978; especially Chapter 3 for routine applications and Chapters 4-11 for non-routine applications.
8. Since warrants in the Los Angeles automated want/warrant system include both traffic and criminal warrants, it is likely that a majority of the 31.5 percent increase in warrant arrests was for traffic offenses.
9. Information concerning the computer system in Tulsa, Long Beach, Kansas City, and Oakland are found as a series of "mini-case studies" on

- routine applications in Chapter 3 of Colton, K. (Ed.), *Police Computer Technology: Implementation and Impact*, Lexington Books, Lexington, Mass., 1978.
10. Phone conversation in 1973 between the author and Robert McNamara, former Chief of Police of the Kansas City Police Department.
 11. Only a small portion of police time is devoted to law enforcement activities such as burglary in progress, check on car, make an arrest, etc. Rather, the large majority of police time is devoted to service (personal requests, animals, ambulance calls, utility problems, accidents, lost or found property, etc.) or order-maintenance activities (family trouble, gang disturbances, neighborhood trouble, fights, etc.) See for example: Wilson, J. Q., *Varieties of Police Behavior*, Atheneum, New York, 1970; and Webster, J. A., "Police Task and Study Time," *Journal of Criminal Law and Police Science*, March 1970, pp. 94-102.
 12. Of the police departments surveyed in 1974, 48 percent indicated that they use no mathematical techniques in deployment, 34 percent said they relied on some version of a hazard or straightforward quantitative formula, and only 18 percent responded that they used an advanced mathematical method, such as a computer simulation or computer-aided resource allocation approach. For a more detailed discussion of police deployment techniques see Larson, R. C. (Ed.), *Police Patrol Deployment: New Tools for Planners*, Lexington Books, Mass., 1977.
 13. For a complete documentation and analysis of the three resource allocation cases see Colton, K. (Ed.), *Police Computer Technology: Implementation and Impact*, Lexington Books, Lexington, Mass., 1978. Chapters 4 (St. Louis), 5 (Boston), and 6 (Los Angeles).
 14. LEMRAS stands for Law Enforcement Manpower Resource Allocation System. ADAM stands for Automated Deployment of Available Manpower.
 15. For a discussion of a hypercube model see Larson, R. C. (Ed.), *Police Patrol Deployment: New Tools for Planners*. Lexington Books, Lexington, Mass., 1977. For a review of the PCAM model see Chaiken, J., and P. Dormont, *Patrol Allocation Model: Executive Summary*. The New York Rand Institute, R-178611-HUD, DOJ, New York City, September 1975.
 16. See Chaiken, J., T. Crabill, L. Holliday, D. Jaquette, M. Lawless and E. Quade, *Criminal Justice Models: An Overview*, Rand Corp., Santa Monica, California, 1975. p. xii. For a further discussion of problems in implementing models and technology in public organizations see Brewer, G. D., *Politicians, Bureaucrats, and the Consultant*, Basic Books, New York, 1973; and Greenberger, M. A., and B. L. Crissey, *Models in the Policy Process: Public Decision Making in the Computer Era*, Russell Sage Foundation, New York, 1976.
 17. For a discussion of the components of command and control and computer-aided dispatch systems see Sohn, R. L., *Application of Computer-Aided Dispatch in Law Enforcement*, Jet Propulsion Laboratory, Pasadena, California, 1976.
 18. The three complete cases are found in Colton, K. (Ed.), *Police Computer Technology: Implementation and Impact*, Lexington Books, Lexington, Mass., 1978, Chapters 9 (Boston), 10 (New York City), and 11 (San Diego).
 19. *Ibid.*, especially chapters 10 and 11.
 20. See Larson, Richard C., Kent W. Colton and Gilbert C. Larson, *Evaluation of a Police-Implemented AVM System: Phase I*. Op. cit., pp. 15-33.
 21. For a discussion of the actual allocation of police time, see Webster, John A., "Police Task and Time Study," *Journal of Criminal Law and Police Science*, March 1970. Also see Wilson, James Q., *Varieties of Police Behavior*, Op. cit., p. 18.
 22. For a characterization of these three groupings of police style see Wilson, James Q., *Varieties of Police Behavior*, Op. cit., especially pp. 140-226.
 23. *The Challenge of Crime in a Free Society*, Op. cit., pp. 269-271.
 24. See, for example, Hough, Granville W., *Technology Diffusion, Federal Programs and Procedures*, Lomond Books, Mt. Airy, Maryland, 1975.
 25. Space and the focus of this report preclude a full discussion of the importance and process of communication and integration in professional practice. For a thought-provoking and worthwhile treatment of this subject, see Argyris, Chris, and Donald A. Schon, *Theory in Practice, Increasing Professional Effectiveness*. Jossey-Bass, San Francisco, 1974, especially Chapters 4 and 5.
 26. Interview between Kent W. Colton and a police sergeant in Oakland, California, 1974.

Distributed algorithms for global structuring*

by RAPHAEL A. FINKEL and MARVIN SOLOMON

University of Wisconsin
Madison, Wisconsin

and

MICHAEL L. HOROWITZ

Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

In the search for speed and computing power, many researchers in computer science have turned to networks of computers as a possible solution.^{1,5,8,14,9} These networks consist of minicomputers connected by links across which communication between processors occurs. In homogeneous networks, the computer at each node is identical to the others, with the possible exception of peripherals. Each processor has its own local memory, does not share memory with any other processor, and communicates with other processors via message passing. In order to fully utilize the speed and power inherent in a network, emphasis must be placed on the development of parallel (as opposed to sequential) algorithms.

In this paper, we will investigate concurrent algorithms designed to impose a logical structure on top of the physical computer network, such as a pairing of the processors along communication lines. These algorithms are interesting not only for their relationship to parallel algorithms in general, but also because the resulting structure may be used as a basis for writing other parallel algorithms. We restrict our attention to algorithms that have the following properties:

1. Initially, each processor knows only of its neighboring processors in the network.
2. All processors have the same program to execute.
3. Messages sent between neighbors may take an arbitrarily long amount of time to arrive.
4. Messages between any two connected processors will always arrive in order.
5. No assumptions are made about the physical interconnection pattern except that it is connected.

The algorithms we will discuss form three types of structures on the network. These three problems have been chosen because their results are dependent upon the physical

network configuration and because they apply to the solution of other network problems. *Pairing* algorithms match each node of the network with a direct neighbor. Since a pairing is not always possible, as in the case of an odd number of nodes, a good solution should leave a minimum number of "single" processors at the end of the algorithm. *Spanning tree* algorithms impose a tree on the network so that a unique path exists between any two processors in the network. The last problem considered is that of forming *hierarchies* of processors. One step in developing a hierarchy is the formation of processor cliques. Each clique should be of a certain size and one processor in each clique is designated as the "leader." A good solution should try to minimize the average radius of each clique.

The assumptions made earlier about the behavior of messages and about the program each processor runs give rise to several problems. The problem of *agreement* is that each processor must make consistent decisions with regard to the rest of the network, possibly based on widely differing local information. (For example, in the pairing algorithm, two nodes should not simultaneously believe they are paired to the same third node. In particular, the final state of the computation must not exhibit this behavior.) Each processor makes independent decisions about its future role in the resulting structure. Therefore, any decisions a processor makes that might affect its neighbors' decisions must eventually be communicated to those neighboring processors.

Synchronization is the problem each processor encounters when it is about to finish a phase of the algorithm. Since information about the rest of the network is usually incomplete at each node, the algorithm must be designed so that each processor can make its own decisions based on little information. (For instance, in the pairing algorithm, a processor should not decide to halt its active participation and assume that a neighbor is its mate unless it knows that the neighbor will agree.) In particular, it is hard to decide whether a current local state is final or not. In part, this problem results from the assumption that messages between processors may take a long time to arrive. A processor cannot stop participating unless it knows that no later mes-

* This research was supported in part by United States Army under contract #DAAG29-75-C-0024.

sage can force a change in its state. Although messages never get lost, they may arrive quite late, and provisions for handling or preventing these messages must be made.

The above two problems just mentioned have an easy solution if we allow ourselves the luxury of a central controlling processor. Such a central control would, however, become a bottleneck as the size of the network grows. We feel that more general solutions can be derived by avoiding this central control. Certainly, any known methods for sequential solution of these problems could be carried out by the central control.

In developing these algorithms, we have totally ignored low-level aspects of message passing. Low-level protocols and the problems of lost or garbled messages are the responsibility of the underlying network implementation. Some research has already been invested into these problems.^{7,10} It is our intent, however, to study the fundamental problems of parallel algorithms themselves.

We have examined the performance of these algorithms by simulating a network in order to obtain sample results. Different network configurations were used, and comparative performance between different configurations varied, but the relative performance of each algorithm remained the same for each configuration. The configuration used most often was a square grid of varying sizes. We schedule the time of the receipt of a message on a time line, and assume that the computation at each node takes negligible time. The time for a message to be delivered is selected from an exponential distribution with mean 1, truncated to lie between 0.50 and 1.50 time units. In this way, many actions may occur "simultaneously" in simulation. The programs were written in Pascal⁶ and executed on a PDP 11/40 and a PDP 11/45.

Each of the following algorithms operates in the following manner: Upon receipt of a message, a processor executes a program segment that depends upon the processor's state and the message received. In this program segment, the processor may save whatever information it wants from the message, send out new messages, and change its internal state.

PAIRING ALGORITHMS

To achieve a pairing, each processor must agree either to become paired with a neighbor or to become single. A correct solution is one in which a paired processor and its mate agree that they are paired, and no neighboring processors are both single. An optimal solution is one in which there are a minimum number of single processors, which may require some analysis of the network configuration to derive.

We will start with a simple algorithm, called Algorithm A, and then suggest improvements. The basic pairing algorithm has four states. A processor in the *idle* state has not yet started the algorithm. When it is *waiting*, a processor expects a reply from its chosen mate. *Paired* means the processor considers itself paired, and *single* means it considers itself single. There are five messages—*awake*, which starts an idle processor; *query*, which indicates that the sender

wishes to pair with the recipient; *agree*, which tells a waiting processor that the sender agrees to become paired with it; *disagree*, which indicates that a processor's chosen mate is itself awaiting an answer from its own intended mate; and *refuse*, which indicates that a processor's intended mate is already paired with another processor.

At the start, each processor is idle, and one awake message has been sent to it. (It is not important to this discussion how awake messages are generated.) Initially, all direct neighbors are potential mates. If the processor receives the awake message in its idle state, it chooses a random neighbor from its list of potential mates as its intended mate, sends it a query and changes state to waiting. This query may reach the neighbor before any awake message. In this case, the recipient chooses to become paired with the sender and returns an agree message. This case, when the recipient of a query is in its idle state, is the only one in which an agree message is sent. Since agree, disagree and refuse messages are sent out only in response to a query, and queries are not sent out by idle processors, we only have to consider these two cases when the processor is idle.

If a processor is in its waiting state, many different actions may occur. Any processor in the waiting state must already have seen the one awake message directed to it. If a query is received, a disagree is sent back if the query is not from the intended mate; otherwise, the processor becomes paired. No agree need be sent in the latter case; since the sender of the message is this processor's intended mate, a query was sent to it and the mate will take the same action. If an agree is received from the intended mate, then the processor also becomes paired. If a disagree is received from the mate, then the processor chooses a new intended mate, sends it a query and remains in its waiting state. The new intended mate may be the same as the first one. If the processor receives a refuse from its chosen mate, it removes the sender from its list of potential mates, chooses a new intended mate and sends it a query. If there are no more potential mates, the processor becomes single. A single processor should not receive any messages at all, since no query is outstanding and all neighbors have refused, implying that they are paired. Finally, if the processor is paired, it can receive either a query, in which case it sends back a refuse, or an awake, which it ignores.

If we examine Algorithm A, we see that once all of the processors are started, a processor and its intended mate must choose to query each other before they can become paired. Many false starts may happen before this pairing actually occurs. In an effort to reduce contention, Algorithm B begins by sending only one processor an initial awake message. If an idle processor receives an awake or query message, it sends out awake messages to all of its neighbors except the sender of the received message. Simulations show that the number of singles left remains about the same for Algorithm B as for A regardless of the network structure, and the elapsed time (simulated time, not running time) becomes progressively worse as the size of the network increases. This behavior implies that contention is not really time-consuming. Also, the time needed to activate the processors across the network from the originally started pro-

cessor begins to override any extra time that contention might produce. Concurrency, it seems, has definite, although not dramatic, time advantages.

Algorithm C is a modification to Algorithm A designed to provide each processor with earlier information concerning the state of its neighbors so that it can make a better choice of intended mate. Since the refuse message indicates that the sending neighbor is already paired, queries can be forestalled by broadcasting refuse messages to all direct neighbors (except the mate and those neighbors known to be already paired) as soon as a processor becomes paired. This modification has three parts. First, whenever a processor becomes paired, it must send out refuse messages to the appropriate neighbors. Second, if a processor receives a refuse from a processor other than its intended mate, it removes the sender from its potential mate list. Finally, a paired processor that receives a query need not respond, since it has already sent out a refuse. The observed saving in elapsed time is quite dramatic, and the number of singles left also seems to decrease. The amount of information available, then, appears to make a considerable difference in performance.

Since concurrency and early information both improve the speed of the algorithm, we tried Algorithm C, activating all of the processors at the same "time." The previous algorithm sent the broadcast awake message to all processors at the same time, but that message arrived according to our message-delay distribution. Again, the elapsed time performance improves, but no real conclusions can be made concerning the number of singles left. In fact, the number of singles may increase for larger networks.

Of the two, more information makes a greater improvement in the performance of the algorithm than concurrency. In order to follow this idea further, Algorithm D provides an individual processor with still more information about its neighbors. A new message, *neighbor-list*, is introduced. This message contains the size of the potential mate list of the sender. These neighbor-list messages are sent out to all potential mates upon receipt of a refuse message, which changes this number. When a processor receives a neighbor-list message, it enters the new data in a table. When it comes time to choose a mate, each processor picks a random neighbor from among those with the fewest potential mates. One expects that the number of singles will decrease, because better choices can be made. For the smaller networks, the new approach seems to make no difference, but for the larger networks, significantly fewer singles result. Algorithm D also performs faster than Algorithm C for all network sizes.

Perhaps we could do better with a different choice of a neighbor to query. Instead of choosing a neighbor with the fewest potential mates, Algorithm E chooses a neighbor with the most. The results justify the intuition that choosing the neighbor with the fewest potential mates leaves fewer singles. Surprisingly, however, Algorithm E is still an improvement over Algorithm C, in which an intended mate is randomly chosen, both in elapsed time and number of singles. Again, increased information creates an algorithm that performs better.

Since none of these algorithms guarantees an optimal solution (that is, one in which the number of singles is minimal), we introduce a second phase to the basic algorithm during which singles migrate, eventually to meet and pair. All the previous algorithms have the property that each processor knows when its active participation in the algorithm is over. In the subsequent pairing algorithms, termination is not locally discernable.

Algorithm F introduces a *break* message that is sent out by a single to a randomly chosen neighbor. This neighbor, if still paired, will then send back an agree, send an *eloped* message to its old mate to indicate that they are no longer paired, and become paired with the single that sent the break message. The recipient of a break message may be waiting or single, though, since processors can now become paired and unpaired an arbitrary number of times. If the recipient is waiting or single, and its intended mate is not the sender of the break message, it then sends back a disagree. Otherwise, the recipient becomes paired, much as in the previous case when two processors send queries to each other. If a disagree message is received by a single processor, then the sender of the disagree is added to the potential mate list, the processor enters the waiting state and a new query is sent out. When a single processor receives an agree, query or break message from its intended mate, it becomes paired. Eloped messages are ignored if they do not come from one's mate, since the message can be late, but must be handled otherwise. They are treated as a refuse in response to a query, which means that the receiving processor must now become single or waiting, depending upon the state of its potential mate list. An appropriate message (break or query, respectively) is also sent. The algorithm has halted (in simulation) when all of the processors have become paired, or the minimum number of singles is left for the given network.

The simulated time to execute phase two is much higher than that for phase one, and varies widely in different test runs. Of course, this phase allows little concurrency, since only those processors near singles are involved. Moreover, networks that cannot be completely paired fare better during phase two than networks of similar size that can be paired, since more singles are migrating throughout the network. We tried Algorithm F in conjunction with both Algorithms A and C, that is, with refusals both broadcast and not broadcast. The results were a bit puzzling at first—Phase Two time increases when refusals are broadcast. The cause is probably that the potential mate list becomes more of a hindrance than an aid in the second phase, since it restricts the choices a processor can query before it must become single. In addition, the average path length, or number of break messages per single at the end of Phase One, also increases when refusals are broadcast.

In an effort to reduce the time spent by singles that migrate randomly throughout the network, Algorithm G includes a homing signal for the singles so that they might find each other more easily and directly. Whenever a processor becomes single, instead of immediately breaking a random neighbor's pairing, it broadcasts a homing signal containing some random number. This message is passed on by every processor until some other single receives the signal. This

single will then break the pairing of the neighbor who relayed the signal. In this way, one hopes that single processors will meet and pair much sooner, since they can move toward each other directly. One also hopes that the signal broadcast messages will not seriously affect performance.

Unfortunately, the phase two performance of Algorithm G turns out to be seriously worse than random migration. First, broadcast messages take up some time, since every processor must receive each signal twice before the message is squelched. More importantly, breaking the pairing of a neighbor in the direction of another single does not necessarily place the new single any closer to the originator of the signal. In a square matrix configuration, for example, there is a two-out-of-three chance that the new single would be just as far away, as shown in Figure 1. Only a better understanding of the configuration of the network at each processor will allow a more intelligent break message to be sent after a processor becomes single.

The pairing algorithm is an excellent medium for studying parallel structure-producing algorithms. We discovered that concurrency, even if it does increase conflict among the processors, decreases the elapsed time of the algorithm without seriously affecting other aspects of performance. Also, gathering greater amounts of information leads to faster and more accurate results. Finally, the network configuration cannot be totally ignored in the development of an algorithm; an algorithm that ignores configuration information will do worse than one which incorporates the data into its "solution."

As a passing note, we also tried our algorithms on different connection patterns for the network. The algorithms perform consistently better on square matrix configurations than on any of the "flake" configurations of similar sizes with respect to the number of singles. (See Reference 3 for a definition of "flake" network configurations.)

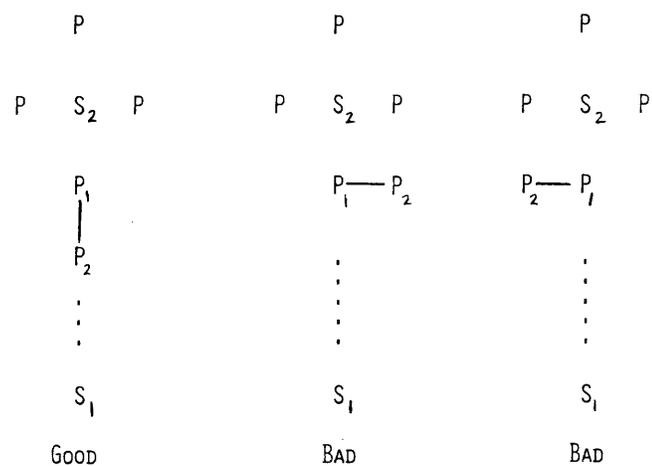


Figure 1— S_1 is a single generating a homing signal detected by S_2 and relayed by P_1 , which is paired to P_2 . In each case, S_2 will pair with P_1 , and P_2 will become a single. Only in the first case is the resulting single closer to S_1 than before.

SPANNING TREE ALGORITHMS

A spanning tree is a subset of the physical links of the network such that there exists a unique path along the selected links from any given processor to any other processor in the network. An algorithm to produce a spanning tree should select appropriate links in such a way that each node agrees with its direct neighbors as to which of the connecting links are in the tree. A useful algorithm by-product is a routing table at each node associating destinations with locally-selected links. We will examine two algorithms for this problem.

In Algorithm H, each processor can be in one of three states—*idle*, *working* or *done*. *Awake* and *known-nodes* are the only messages. *Awake* starts an idle receiving processor. Each *known-nodes* message contains a list of those nodes the sender knows it can reach through selected links other than the one on which the message is sent. Each processor associates each direct neighbor with those nodes it thinks that neighbor can reach, according to the most recent information passed to it via *known-nodes* messages. A list of selected links is also kept.

Upon starting, each processor tells its direct neighbors that it can reach itself. When a processor receives a *known-nodes* message, if the connecting link is currently selected, and nodes newly reachable through it could be reached through other links in the spanning tree, then the link to the sending neighbor is de-selected. If the newly reachable nodes do not conflict with any current information, then the link to the sending processor remains in the tree. On the other hand, if the connecting link is not currently selected, and if the reachability set the neighbor has sent does not conflict with the current set of reachable nodes, then the connecting link is selected. Whenever a *known-nodes* message arrives, it may show that some links have been removed from the spanning tree, so the receiving processor also checks if any of the other connecting links can now be selected. Finally, the new information about its status of reachable nodes is sent to all of the receiving processor's direct neighbors, unless no new information is to be reported (the last message sent to each neighbor is saved to facilitate this decision). A processor can determine when its role in the algorithm is finished when it chooses not to send any messages to its neighbors, and all nodes in the network are reachable through selected links. In order to perform this test, each processor must know the names (but not the locations) of all nodes in the network, which can be supplied in the *awake* message.

It is easy to prove that this method will produce a correct solution. First, if a link is selected by a given processor when it has terminated, then the set of nodes reachable through that link is the complement of the nodes reachable through the other selected links, by the termination condition. If the neighbor on the other side of the link has not selected the link, and has also finished, then it must think it can reach the first processor through another selected link. If this is the case, it would have told the first processor so. The first processor could not then have selected that

link, because it causes a conflict (a processor can always reach itself without going through any links). So the neighbor must also have selected that link. Second, the sets of nodes reachable through each of the selected links will be disjoint, so any path along the tree will be unique. Finally, all of the processors are in the tree when the algorithm halts. A major drawback to Algorithm H, however, is that the computing time required for each action and the amount of space required to maintain the appropriate information at each node are both large.

A second method, Algorithm I, sacrifices some concurrency for ease of computation. The idea is based on the pairing algorithm. During the algorithm, every processor belongs to a partial spanning tree, and each tree is controlled by one of the member processors. Partial trees merge using a version of the pairing algorithm, with the controlling processors representing the trees.

At the start, each processor starts as its own spanning tree. Partial trees are built by repeatedly asking other processors to join. Each controlling processor keeps track of which nodes are in its tree, and asks only those nodes not in the tree. If the controlling processors of both trees agree (i.e., the "trees" have queried each other as in the pairing algorithm), a combined tree is formed, and one of the two controlling processors is chosen as the new controller. Each controlling processor has an associated random number; whichever has the higher one becomes the controller for the combined tree. Processors that no longer control a tree relay messages to their controlling processor. As soon as one transaction is complete, the surviving controller initiates the next one, until the set of neighbors not in the current tree is empty. This algorithm does not require any advance knowledge of which processors are in the network, since the set of tree neighbors can be calculated from the set of nodes in the tree and the set of neighbors of nodes in the tree. When the set of nodes neighboring those that are in the tree is a subset of the nodes in the tree, the entire tree has been formed.

Algorithm I can be sped up in two ways, resulting in Algorithm J. First, when one controlling processor becomes chosen by another, it informs all of the nodes in its tree which processor is now controlling the tree, so that messages can be relayed directly. The second idea provides a major improvement, because it increases the concurrency of transactions. When a controlling processor, α , receives a query from another controller, β , that has a smaller associated random number, and α is also waiting for an answer from a third tree, γ , then α sends an acceptance, not a disagreement, to β . Processor β is destined to lose its position as controller in any case. Update messages are now required, however, since the controller γ may be the surviving controller of the transaction between α and γ . That controller will not be aware immediately that other nodes have joined α 's tree in the interim, since such information is normally passed along with the queries and γ may therefore generate an indirect query to itself, delayed by some relays. Update messages enable this processor to rectify such a situation.

These two improvements together decrease the elapsed time of Algorithm J until it is comparable to the time of Algorithm H. In addition, it seems that the simulated time increases more slowly with the size of the network for Algorithm J than for Algorithm H.

It is obvious, however, that Algorithm H is much cleaner and that it derives its speed from its inherent concurrency. In Algorithm J, all chosen processors are idle except when relaying messages, and the speed is derived primarily from the lack of conflict. If Algorithm J could be changed to distribute the decision process, then perhaps it could become a more powerful solution. On the other hand, the number of messages sent in Algorithm H grows significantly faster with the size of the network than in Algorithm J. Therefore, for larger networks, Algorithm H may prove impractical since message passing may become a bottleneck. In general, though, it is much better because it does not rely on random numbers or shortcuts to derive its speed, although a good way to terminate the algorithm may be harder to develop.

The previously-mentioned algorithms were developed solely under the assumption that more than one processor can begin within a short amount of time (much shorter than that required to complete the algorithm). If only one processor is started, then Algorithm K, a much simpler method, may be used: When a processor receives an awake or query message and it is idle, it selects the link over which the message came and responds with agreement. It then sends out its own queries along the other links. It only selects those links across which agreements are received. In order to allow each node to determine termination, disagreements can be sent to all subsequent requestors. When all neighbors are accounted for, the node is finished. This procedure is clean and simple, but requires that exactly one processor be started. Running time is directly proportional to the diameter of the network and to the maximum degree of the nodes.

HIERARCHY ALGORITHMS

The final class of algorithms investigated are those designed to form a hierarchy of processors. The major step is to form cliques of processors, with one processor chosen as the head of the clique. The hierarchy can then be built by repeating the clique-forming algorithm on the heads of the cliques formed in the previous step. Therefore, we shall only look at algorithms that form cliques. A correct solution is one in which each clique is well formed—there is exactly one head, and all of the processors in the clique agree that they are in the clique. An optimal solution is one in which the average radius, as measured from the head of the clique in message links, is minimal. A constraint is the acceptable range of sizes (number of nodes) per clique.

We will discuss two algorithms. The first assumes that all the processors are started within a short time of each other, whereas the second assumes that a central controller directs the algorithm. In Algorithm L, each processor is in one of two stages—*head* and *chosen*. Each processor begins as a

head and tries to form its own clique. It keeps asking processors not in the clique to join. Each head also keeps track of how close it is to its goal. Chosen processors are those that think that they are members of some clique. When a chosen processor receives a query from another clique's head, it relays the request to its own head. The head either denies the request or tells the processor to switch allegiance to the other head. The chosen processor then sends back a disagree or an agree, respectively, to the requesting head. If a head is itself queried, it decides either to abdicate or to disagree. If it abdicates, the head tells its chosen processors to start all over and sends an agree to the requestor; otherwise it sends a disagree. The head stops asking new processors to join its clique when it feels that the clique is of the correct size. In order to minimize the average radius of each clique, the head prefers to query those processors that are closer. This algorithm requires decision algorithms for selecting potential members, freeing current members and abdication. We could not produce very good hierarchies with Algorithm L with the various decision algorithms tried.

Algorithm M is a variant of one developed by Larry Wittie.¹³ In this method, one clique at a time is formed, and a central control is needed to ensure that the algorithm terminates. The algorithm operates in two stages, the first of which informs each processor of all the other nodes in the network and their distances in message links. This preparation can be done by a straightforward modification of Algorithm H.

The second stage of Algorithm M forms the cliques. The central control sends a message to an arbitrary processor in the network ordering it to form a clique. This processor chooses and queries the right number of nodes, sending along the calculated radius of its proposed clique. If one or more of these chosen processors can form its own clique with a smaller radius, it responds to that effect. Otherwise, the chosen processor sends back an agree. If, when all of the chosen processors have replied, at least one has sent back a better radius for its proposed clique, the querying processor chooses one from among those with the best radius for a proposed clique, and tells it to try and make a clique. If all processors respond with agreements, then the querying processor becomes the head of the clique and sends messages to the chosen processors indicating that they are now members of the clique. It then chooses some processor not in any clique and tells it to form its own clique from among the other processors that are not in any clique. When as many cliques as possible are formed, the algorithm halts. Algorithm M is not very concurrent, but does produce fairly good hierarchies. It is also possible to devise a network for which this algorithm cannot produce an optimal solution, since the algorithm uses local hill climbing. If concurrency could be added, this algorithm would be very powerful indeed.

It appears to be more difficult to design good distributed algorithms for forming processor cliques. Either some heuristics are needed so that the different cliques forming simultaneously do not interfere with each other, or concurrency needs to be sacrificed. Some further study into the

basic nature of the problem appears to be required in order to find a good, concurrent clique-forming algorithm.

CONCLUSION

We set out to develop parallel algorithms for globally structuring networks of processors. We discovered that although concurrency is an important aspect of the performance of these algorithms, the amount of information about the problem available at each processor plays an essential role. We also found that the nature of the particular problem has to be examined for sources of concurrency.

In future research, we hope to examine not only other problems and algorithms for concurrent solutions, but also problems for which the associated solutions will use the imposed network structures produced by the algorithms in this paper. Since processor networks appear to be emerging as an important computing resource, it is essential that research into the design of distributed algorithms be continued and expanded.

REFERENCES

1. Arden, B., and H. A. Lee, *Multi-Tree Structured Network*, Princeton University Electrical Engineering and Computer Science Technical Report #239, January 1978.
2. Feldman, J. A., *A Programming Methodology for Distributed Computing (among other things)*, University of Rochester Computer Sciences Department Technical Report #TR9, September 1976.
3. Finkel, R. A., and M. H. Solomon, *Processor Interconnection Strategies*, University of Wisconsin-Madison Computer Sciences Department Technical Report #301, July 1977.
4. Finkel, R. A., and M. H. Solomon, *The Roscoe Kernel*, University of Wisconsin-Madison Computer Sciences Department Technical Report #337, September 1978.
5. Huen, W., P. Greene, R. Hochsprung, and D. El-Dessouki, "TECH-NEC, A Network Computer for Distributed Task Control," *Proceedings of the First Rocky Mountain Symposium on Microcomputers*, August 1977.
6. Jensen, K., and N. Wirth, *Pascal User Manual and Report*, Springer-Verlag, 1974.
7. McKenzie, A., *Host/Host Protocol for the ARPA Network*, Stanford Research Institute, January 1972.
8. Mills, D., "An Overview of the Distributed Computer Network," *Proceedings of the National Computer Conference*, Vol. 45, AFIPS Press, 1975, pp. 523-531.
9. Solomon, M. H., and R. A. Finkel, "ROSCOE: A Multi-microcomputer Operating System," *Proceedings of the Second Rocky Mountain Symposium on Microcomputers*, August 1978, pp. 291-310.
10. Sunshine, C. A., "Factors in Interprocess Communication Protocol Efficiency for Computer Networks," *Proceedings of the National Computer Conference*, 1976.
11. Tischler, R. L., M. H. Solomon, and R. A. Finkel, *Roscoe User Guide*, University of Wisconsin-Madison Computer Sciences Department Technical Report #336, September 1978.
12. Tischler, R. L., R. A. Finkel and M. H. Solomon, *Roscoe Utility Processes*, University of Wisconsin-Madison Computer Sciences Department Technical Report #338, September 1978.
13. Wittie, L. D., and A. Van Tilborg, *Control Hierarchies for Arbitrarily Connected Microcomputer Networks*, State University of New York at Buffalo Department of Computer Science Technical Report #126, May 1977.
14. Wittie, L. D., *Micronet: A Reconfigurable Microcomputer Network for Distributed Systems Research*, State University of New York at Buffalo Department of Computer Science Technical Report #143, April 1978.

The tree-structured distributed network front-end processor architecture

by ROBERT M. MONROE, RONALD J. SRODAWA and FRANKLIN H. WESTERVELT

Wayne State University
Detroit, Michigan

INTRODUCTION

Contemporary front-end processors, such as the IBM 3705,¹ Memorex 1380,² Burroughs Data Communications Processor³ and the MERIT Communications Controller,⁴ share the same general architecture depicted in Figure 1. Communication lines are associated on a one-on-one basis with line adapters. These are connected in small groups (clusters) to the local processor, and perhaps to the local memory, through line interface bases. In a similar manner, the host system input/output channels are associated on a one-on-one basis with channel adapters which are connected to the local processor and memory. The architecture can be thought of as a minicomputer system in which the channel adapters and line interface base/line adapter subsystems are input/output devices. In fact, many front-end processors are configured in precisely this fashion.

There are many design parameters which can be varied to affect the maximum aggregate data rate, cost, and services-rendered characteristics of front-end processors. The number of line adapters per line interface base and their relative complexities are an example. The line interface bases might interrupt the processor for every character or transfer entire records by means of direct memory accesses. The channel adapters might connect directly to the host system channels or through some other standard interface such as a channel-to-channel adapter or high-speed communication line. Generally channel adapters utilize direct memory access features because of the high aggregate data rates they sustain.

All of the intelligence of a typical front-end processor is located in the single local uni-processor. This uni-processor must handle all the interrupts from the line and channel adapters. In effect it is multiprogrammed across all of the data streams to provide front-end services such as line editing, character code translation, and message formatting. Saturation of the local processor limits the kinds of services that can be provided and the aggregate data rate that can be sustained.

We describe here a front-end processor architecture in which the single local uni-processor, memory, line interface bases and line adapters are replaced with a network of microprocessor systems as depicted in Figure 2. The network

is tree-structured with the root microprocessor system connected to the channel adapters and local direct access storage devices (DASD). At the same time the leaf microprocessor systems replace the line interface bases and line adapters. The low cost of microprocessor systems makes such a network economically feasible. Each communication line has a dedicated microprocessor and the computational power of the network easily exceeds that of the local uni-processor in the traditional front-end processor configuration. It is reasonable to expect that more sophisticated services might be provided or that higher aggregate data rates might be sustained. Data movement between the communication lines and the host system channels is accomplished by store and forward message routing through the tree of microprocessors. The tree-structure network organization was chosen because it correlates well with the hierarchical organization of the hardware system.

The remainder of this paper begins by describing the network topology and the configuration of the nodes in abstract terms. Next, a description of the prototype configuration is given followed by a description of the proposed production configuration. The cost of a typical production configuration is established and compared to that of a similar configuration using a commercially available front-end processor. Finally, arguments are given supporting the advantages of a tree-structured distributed network architecture for a front-end processor.

NETWORK TOPOLOGY

As briefly described earlier, the topology of our front-end processor is a distributed network of microprocessor systems arranged as a tree-structure depicted in Figure 2. Each node (microprocessor system) of the tree is loosely coupled only to those nodes to which it is connected by edges. The network is a loosely-coupled microprocessor system—the interconnections consist of loose input/output couplings rather than tight couplings such as shared busses.

There are three functional types of nodes. The leaf nodes connect on a one-on-one basis to the communication lines.

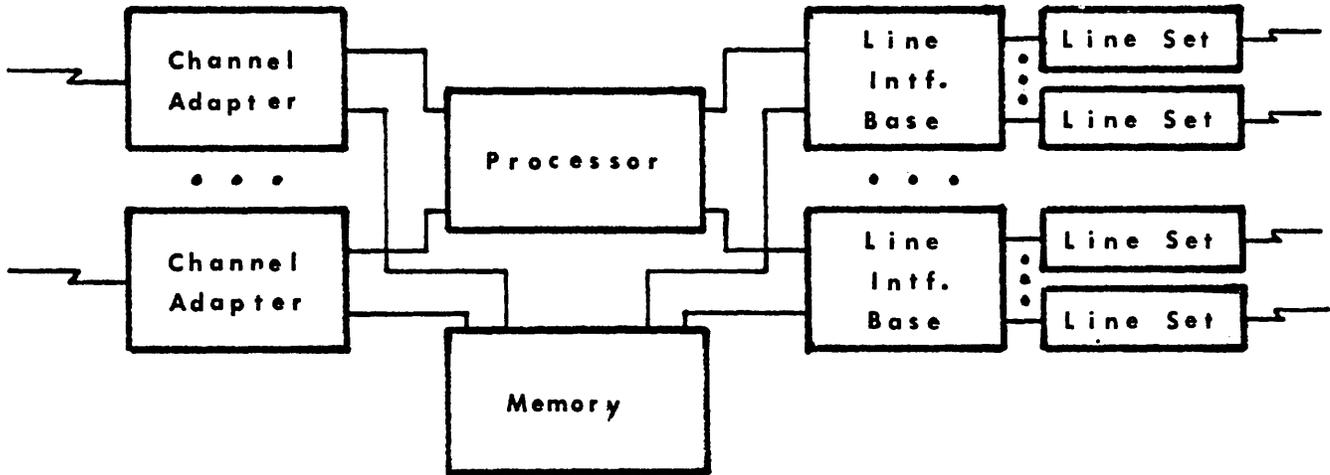


Figure 1—Front-end processor general architecture.

These are the nodes which take over the function of the line adapters. A leaf node provides the front-end services-rendered to the associated communications line. To the user, the leaf node appears to be a microprocessor system located within the communication link between his terminal and the host system and dedicated to his personal use. For this reason it is called the personal processing link (P²L) and the entire front-end processor is called the Personal Processing

Front-end Processor (P²FEP). The root node connects to the host system channel adapters and local DASD. It gives the network its voice to the host system and maintains a local database. Each of the intermediate nodes connects the subtree below it to the next level closer to the root. The intermediate nodes function as store and forward message handlers. The term link control (LC) refers to a node which is either the root node or an intermediate node.

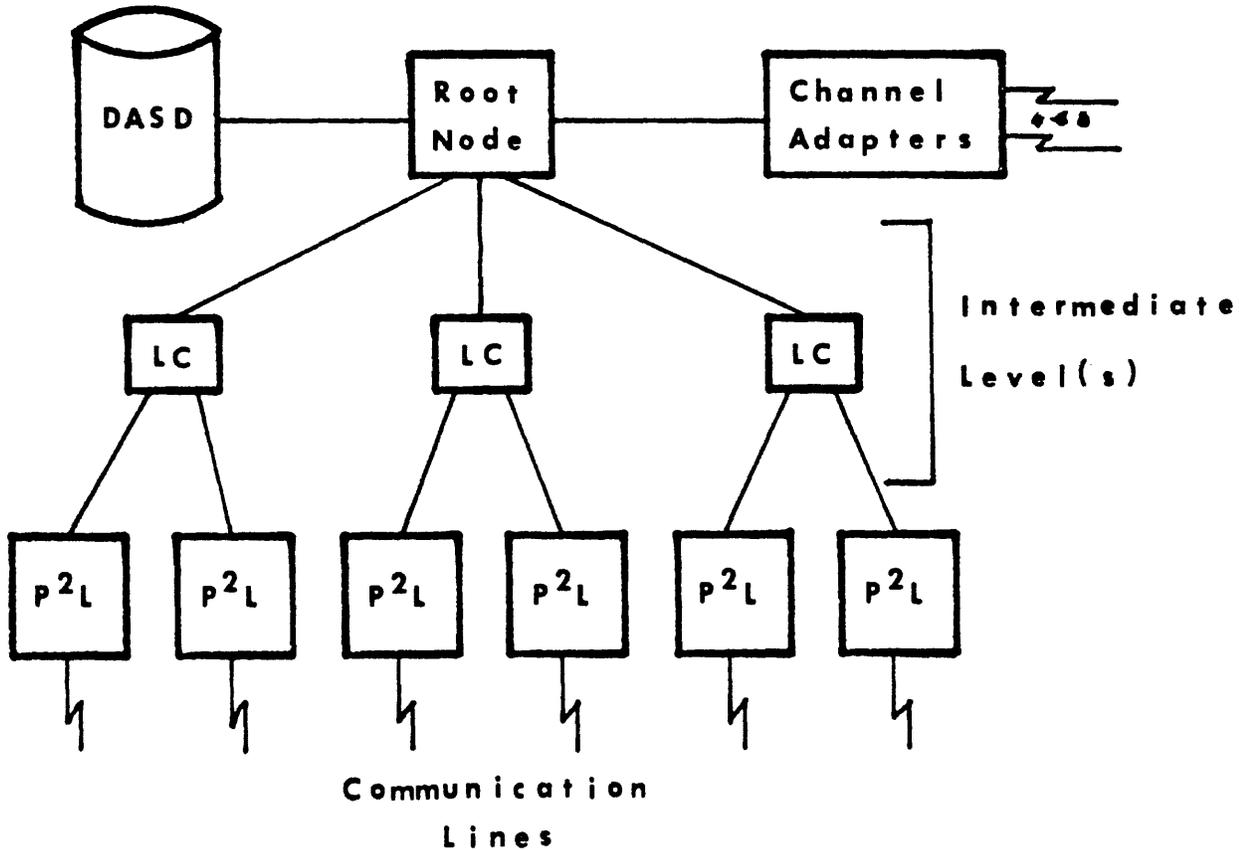


Figure 2—Personal processing front-end processor.

Leaf node (P²L) architecture

Each P²L consists of a microprocessor system with four interfaces as depicted in Figure 3. The communication line interface connects the P²L to the communication line associated with the leaf. Generally this is a serial interface following either RS-232-C⁶ or 20 ma current loop⁷ conventions. Adequate support of modems on the dialable telephone network implies a careful adherence to the full RS-232-C set of conventions. The *break/disconnect monitor* (BDM) interface allows the adjacent link control to monitor the communication line for break and disconnect status. This interface is necessary to assure system integrity if users are allowed to execute arbitrary programs in their personal processing links. Most microprocessor systems contain protection mechanisms insufficient to prevent arbitrary programs from capturing the P²L. Furthermore, an original design goal was to allow the user of a P²L to execute any code that could be executed on a comparable microprocessor system under the complete control of the user at his/her local site. This includes being able to completely "wipe out" one's local program and then recover by a local "reset." The BDM interface provides the ability to perform the "reset" even though the P²L is, obviously, not local to the user's site.

The *non-maskable interrupt* (NMI) control line allows the adjacent link control to interrupt the P²L into its ROM program nucleus even in the presence of a non-cooperating user program. This together with the BDM interface assures the integrity of the P²L.

Finally, the *data* interface allows data to be shared between the P²L and its adjacent link control. The *data* interface could be either serial or parallel depending upon the

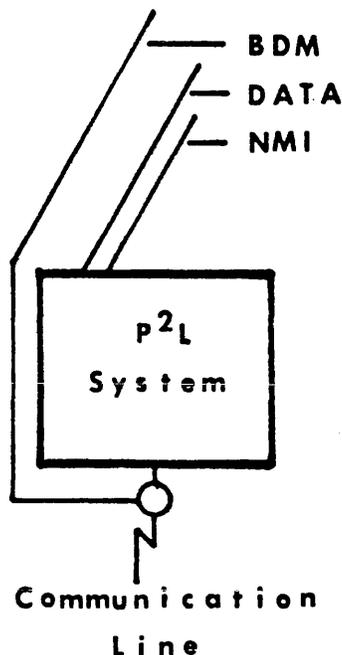


Figure 3—Leaf node (P²L).

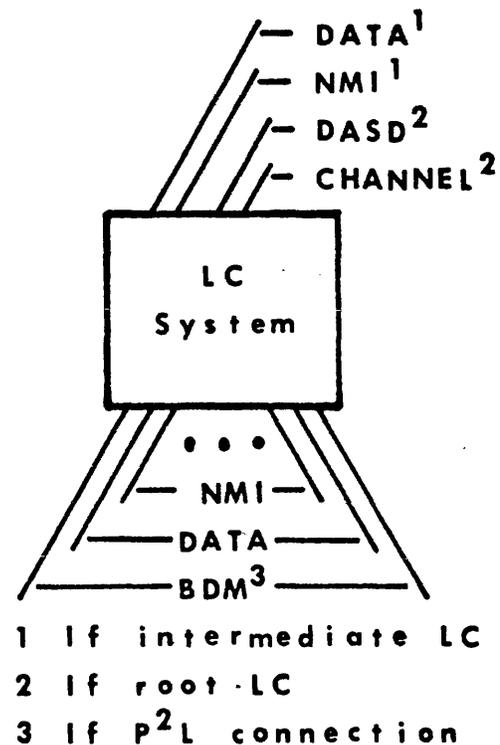


Figure 4—Link control node (LC).

manufacturer's standard card configurations. A full duplex *data* interface allows simultaneous data movement in both directions and reduces the complexity of message protocols.

Link control (LC) architecture

Each link control node (intermediate node or root node) consists of a microprocessor system with a varying number of interfaces as depicted in Figure 4. The precise configurations of interfaces depend upon whether the link control functions as the root node, intermediate node, or intermediate node adjacent to the P²Ls. In small configurations more than one function can be supported by a single node.

Each link control connects to all of the nodes immediately below it on its branches. There are two types of interface bundles, depending upon whether these branch nodes are leaf nodes or other link controls. If the link control in question connects to P²Ls, then each interface bundle consists of the BDM, NMI, and *data* interfaces described above. If the link control connects to other link controls, then each interface bundle consists of NMI and *data* interfaces similar to the interfaces of the same name described for the P²Ls. In either case, there are as many interface bundles as branches from the node in question.

Each link control (other than the root node) connects to one link control in the next level closer to the root. This connection is made through a NMI interface and *data* interface. The NMI interface is used to propagate gross system

events such as resets. The *data* interface is used to move messages comprising the data streams one level between the P²Ls or root node.

The root node interfaces to local DASD and the channel adapter instead of a link control at the next level. The root node replaces the NMI and *data* interface bundle with interfaces to the channel adapters and local DASD. The channel adapter interface connects the root node to the host system channels thus giving the front-end processor a voice to the host system. The DASD interface gives the front-end processor and indirectly its users a local file system independent of the host system.

PROTOTYPE SYSTEM

The prototype system consists of two P²Ls and one link control node. The link control node serves both as the root node and the intermediate node adjacent to the P²Ls. The prototype is constructed from standard Zilog cards. MCB, RMB, SIB, and IOB cards are used.

Zilog MCB cards consist of a Z-80 CPU, a serial universal synchronous/asynchronous receiver/transmitter (USART) with both RS-232-C and 20ma current loop interfaces, a parallel interface supporting 16 bi-directional lines, and a combination of ROM and RAM memory.⁸ Zilog RMB cards supplement the ROM/RAM on the MCB card to provide up to 64K bytes of memory.⁹ Up to 16K bytes may be non-volatile ROM, PROM, or EPROM components. Zilog SIB cards provide four serial USART components with RS-232-C interfaces.¹⁰ Zilog IOB cards provide four parallel interfaces, each supporting 16 bi-directional lines.¹¹

Each P²L of the prototype system consists of two cards—one MCB and one RMB. The total memory configuration is 64K bytes with 8K bytes non-volatile. The USART supports the communication line interface. The *data* interface is supported by the parallel lines, with eight lines in each direction to provide full duplex transmission.

The root node of the prototype system consists of four cards—an MCB, RMB, SIB, and IOB. The USART of the MCB connects to the host system through a high-speed RS-232-C interface. (This interface was chosen to remove the need for a channel interface hardware design). The *data* interfaces from the P²Ls are supported by 32 parallel lines of the IOB card. The NMI interfaces of the P²Ls are each driven by a parallel line from the IOB card. The BDM interface from the P²Ls are supported by two USART components from the SIB card. The prototype system contains no direct access storage in its original configuration. It is wired to accept the Zilog floppy disk controller card (MDC).

The prototype system consists of eight cards assembled in a Zilog unwired 9-slot card cage. Each card contains a 122-pin edge which plugs in the backplane. All pertinent interface signals are available on the backplane. The backplane is wired as three independent systems—the root node and the two P²Ls. For example, there are three independent address busses, each interconnecting the cards of the respective systems. All loose couplings between systems are likewise implemented through backplane wiring. External

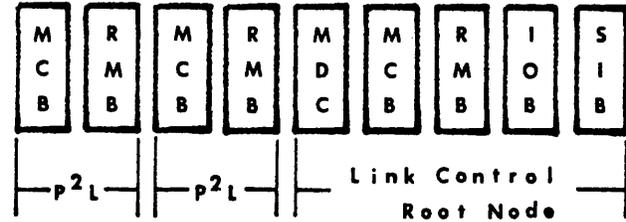


Figure 5—Prototype P²FEP card arrangement.

interfaces (three RS-232-C interfaces and the disc controller interface) are accomplished by wiring from the individual card pins to interface areas along the periphery of the cage. Ribbon cables with 100 mil spacing pin connectors at one end and standard RS-232-C connectors on the other effect the interface. Figure 5 depicts the cards of the prototype system in the card cage.

PROPOSED PRODUCTION SYSTEM

At the time of this writing the production system configuration is not yet frozen. Factors such as the recent announcement of new card configurations with improved component mixes and the anticipated availability of Z8000 components and cards in the near future account for this indecision. We shall describe here the general design philosophy, using the subsystems of the prototype configuration as building blocks.

The production system will be structured around 18-slot unwired rack mountable card cages. Since rack mounted modems are also available, one can anticipate a system in which the front-end processor card cages and modems are mounted in the same rack as in Figure 6. This configuration minimizes the wiring between modems and personal processing links.

One 18-slot card cage can hold a link control (six cards) and six P²Ls. The P²Ls are configured as in the prototype system. The link control is augmented with another SIB card to handle the BDM interfaces and another IOB card to handle the *data* and NMI interfaces of the six P²Ls. The link control nodes, in turn, are connected to either the root node or intermediate nodes through 17-line parallel interface (full duplex *data* and NMI).

The root node (or intermediate link control node, depending upon total configuration size), is housed in a separate card cage within the cabinet. It must have sufficient IOB cards to support the 17-line parallel interfaces of its branches. If it is an intermediate link control, it must have an additional 16-line parallel interface to the next level link control. However, the root node requires interfaces to the local DASD and the host system channel adapters.

COST ANALYSIS

To estimate typical system costs, let us assume a configuration supporting 96 low-speed asynchronous lines. Assume that rack mountable modems are packaged 16 modems

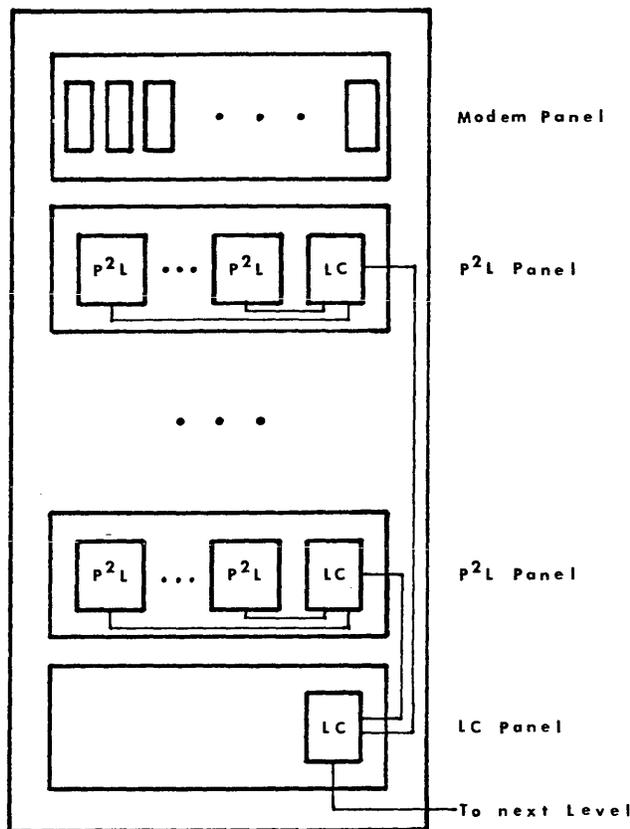


Figure 6—Typical production P²FEP cabinet.

per rack panel. It makes sense to package one modem panel, three personal processing link panels, and one link control panel per cabinet. (This leaves two personal processing links without modems—P²Ls without modems can be assigned to monitor an interactive line under program control when a user disconnects. This provides a very powerful extension of the VM/370-CP *disconnect hold* command¹² in which automatic responses can be generated under P²L control.) Ninety-six lines are housed in six cabinets. A seventh cabinet (or extra space in the original six) houses the root node, channel adapters and DASD storage and controllers.

The configuration just described is a tree containing four levels. The root (Level 1) is connected to each of the link control panels in the six cabinets (Level 2), which are connected to the link control systems in each personal processing link panel (Level 3), which are connected to the personal processors of the panel (Level 4).

The system contains 25 card cages, 133 MCB cards, 133 RMB cards, 44 IOB cards, and 36 SIB cards. The system cost is estimated to be approximately \$350,000.

Memory is the most expensive component of the given configuration—each of the 133 microprocessor systems has 64k bytes of memory, giving a system-wide total of over eight megabytes. Halving each system memory size to 32k bytes reduces the system cost to approximately \$225,000. Quartering each memory size to 16k bytes removes the need

for the RMB cards altogether and reduces system costs to approximately \$150,000. (For comparison, a 96 asynchronous line IBM 3705II lists at approximately \$150,000). A mix of memory sizes would have a system cost somewhere between the two extremes and would have unusual processing services available to its users.

ADVANTAGES

The Personal Processing Front-end Processor offers several advantages over traditional front-end processors. These generally accrue from the compute power inherent in the multi-processor network, the network architecture, and the properties of the components. Each advantage is as follows:

Host system offloading

One advantage offered by all front-end processors is offloading of the host system. Generally offloading is in the areas of code translation, record editing (e.g., backspace interpretation), recovery from transmission errors, and record formatting. Offloading with the P²FEP is expected to be significantly higher due to the additional services provided by the P²L including file editing, input/output record filtering, source program compilation for some languages, and execution or interpretation of programs.

It is quite clear that a significant load can be removed from the host facility when one considers the following facts:

1. The density function of the CPU requirements over all interactive user service requests shows a typical exponential decay curve.¹³ This indicates that a significant amount of CPU time is expended to satisfy small requests for a large number of users.
2. The frequency of use of host system components such as the context editor and BASIC, which are strong potential candidates for offloading to the P²Ls, is quite high—indicating that a significant CPU load could be in fact offloaded.
3. A significant amount of CPU time is expended processing each input or output record between the host system and the interactive terminals—this CPU time vanishes when the user is conversing with a program resident in his P²L.

Offloading of whole interactive conversations has a greater beneficial effect than just the cumulative processor time required for the offloaded computations. Input/output record transfers between the host system and the front-end processor are eliminated which also eliminates the many trips through the host system interrupt handlers that would have been necessary otherwise. This causes the operating system overhead to be reduced. Since interrupts have a deleterious effect on hardware instruction lookahead algorithms, host system performance may increase with sophisticated host hardware possessing this feature.

Time-sharing systems incorporate short quanta times in

their CPU scheduling algorithms to reduce the average response times to requests requiring low CPU requirements. This biases system response time to favor requests requiring low CPU requirements. Since many such requests are processed in the P²L rather than the host system, it may be reasonable to increase the length of the quanta used in scheduling the host system. This further reduces the number of trips through the host system interrupt handlers.

Cost reductions

Programs offloaded onto a personal processor cost less to use than their counterparts on the host system. It is appropriate to charge users for *connect time* to the personal processor regardless of the work performed. These connect charges, since they need only recoup the front-end processor acquisition and operating costs, are relatively low.

Uniform short response times

Response time to user interactions with offloaded programs will not depend upon total system load and should be significantly less than response time with equivalent host-system programs under heavily loaded conditions. Again, the offloading of the processing of trivial requests from the host system ought to improve the responsiveness of the host system to those requests that are not filtered out by the P²Ls.

Increased availability

The P²FEP may be available even when the host system is unavailable. At these times users can still accomplish useful work provided it can all be done by the P²L. Examples include local file editing, local language program development and execution and the staging of host system jobs for automatic submission when the host system becomes available.

Microprocessor program development

As the usage of microprocessors increases, the user community will need to develop programs for eventual execution upon offline microprocessor systems (e.g., in laboratory apparatus). If the microprocessor hardware systems are compatible, it is possible for users to develop these programs utilizing services provided by the P²FEP and perhaps even unit tests program modules in the P²Ls.

General communication protocols

The LSI devices which microprocessor manufacturers market for driving data communication equipment are very general. These drivers generally adapt to synchronous and asynchronous protocols; switched lines, leased lines, or di-

rect attachment of equipment; a variety of character or message formatting conventions; and any practical rate of transmission under program control. It is possible with these devices to dynamically determine the proper communication parameters at the time the user initiates his session with the system. Many commercially available front end processors do not possess this flexibility.

Input/Output filtering

It is possible for the user to interject a program in his P²L between the host system and his terminal. This program can then function as a data filter. A general macro-processor is an example of such a program. This is reminiscent of the chained processes concept in UNIX.¹⁴

Job staging

The local DASD and editor can be used to prepare jobs or portions of jobs for submission to the host system. This can be done whether or not the host system is available at that moment. With this feature the P²FEP can act as a programmers' workbench.¹⁵

Maintainability

The P²FEP is easy to maintain on a card replacement basis. It is reasonable to stock replacement cards on-site because of its construction from a few card types, each occurring in large quantities.

Hard faults are relatively easy to isolate because of the inherent tree structure. By sending echo requests and timing responses, the root node can isolate a fault to a single node. In a similar manner a P²L can isolate a fault on its path to the root to a single node. Once a fault is isolated to a single node, a complete card swap of that node can be performed to put the P²FEP back in operating condition. The faulty cards can be tested later in a standalone configuration.

CONCERNS

There are three concerns due to the tree-structured architecture of the system. One is that the aggregate data rate increases as one moves toward the root node. The second is that the fraction of the system affected by an error increases as one moves toward the root node. The third is that the amount of work that must be performed by the root node might saturate it.

The potential increase in aggregate data rate poses no problem. Offloading, since it eliminates the movement of data between the P²L and the host system, reduces the aggregate data rate. Furthermore, the maximum aggregate data rate of contemporary front-end processors such as the IBM 3705 (500,000 bytes/sec) and the Memorex 1380 (100,000 bytes/sec) is within the capabilities of microprocessor systems.

A failure in a P²L affects only the single user serviced by the failing P²L. A failure in a LC affects all nodes in the subtree beneath the failing LC. The fraction of the system affected by a failure increases the closer the failure is to the root node. The only effect of a LC failure is to sever communications between the nodes in the subtree beneath it and the root node. Message timeouts can be used to detect a failure in a node along the path to the root. The previous section outlined the maintenance strategy for the P²FEP, which is to isolate the failure to a single node quickly, then repair immediately with a complete card replacement of the node. We anticipate that this policy will yield acceptable reliability. If this should prove not to be the case, it is readily possible to provide redundancy distributed throughout the tree structure to obtain the necessary "fail-soft" characteristics.

The root node is obligated to provide service for high levels of activity. Besides performing store and forward message processing between the host system and its branches, the root node maintains the local data base. It is possible that the level of activity might saturate a single Z80 in larger configurations. There are, fortunately, at least two alternatives. One is to use faster components such as the Z8000 in the root node. Another is to distribute the workload of the root node across more than one loosely-coupled microprocessor system. The division of effort might be along functional lines. For example, the root node might be structured as three systems, one communicating with the other nodes of the P²FEP, the second servicing the channel adapters connecting the P²FEP to the host system, the third maintaining the local database.

ALTERNATIVES

An alternative to the P²FEP described here is a number of decentralized microprocessor systems, each with local DASD and terminals, and perhaps a communication link to the host system. Decentralized systems of this sort might be user-programmable or supported by the manufacturer for a special purpose, such as word-processing. Many vendors, including Sykes, Wang, and Vydec, for example, market such products. There are good justifications for a centralized FEP rather than a number of decentralized systems. These are as follows:

1. *Higher utilization.* The P²FEP naturally experiences a higher utilization because it is shared among all the host system users. Decentralized systems are generally associated with a single individual or group of individuals. The "macro cost" is lower for the P²FEP because a smaller number of units are required to satisfy peak demands. The cost to individual users is decreased because a simple connect-time charge replaces the significant cost of a personal system.
2. *Economy of scale.* The P²FEP with its large number of loosely-coupled microprocessors located in close proximity to one another leads to certain economies, such as rack-mounted systems, shared power supplies, and quantity discounts. These tend to make the single large system cost significantly less than the equivalent number of smaller systems.
3. *Sharing of peripherals.* The close proximity of the individual P²L systems allows peripherals such as DASD to be shared among all the P²L systems. This again results in a more economical system. It also may lead to the use of larger, more reliable devices than those normally used with microprocessor systems.
4. *Centralized software responsibility.* Just as their larger cousins, microprocessor systems require expensive nurturing by a competent systems programming staff. Centralized hardware also makes it possible to centralize the software responsibility effectively. In contrast, a decentralized set of (possibly incompatible) microprocessor systems will lead to competition for the available systems programming expertise to the detriment of all.
5. *Franchise.* The P²FEP is available to all host system users, even though budgetary limitations might prevent them from securing their own microprocessor systems. (It is a fact of life that decentralized systems are often budgeted by the local unit.)

CONCLUSIONS

The tree-structured distributed network front-end processor architecture appears to offer several intriguing advantages and few significant disadvantages. It may provide a most reasonable method for realizing the advantage of distributed logic or intelligence to a large number of users without incurring the significant disadvantages of distributed maintenance and software development costs. The prototype module is nearing the system test phase and the results of these tests should prove to be most interesting. If successful, it may be possible to provide very low cost, transparent access to a hierarchy of increasingly powerful computing resources for a very large number of users in a novel and highly reliable way. The results of these tests will be reported later.

REFERENCES

1. *IBM 3704 and 3705 Communications Controllers Principles of Operation.* IBM Systems Reference Library. Form No. GC30-3004. IBM Library Services Dept., White Plains, New York.
2. *Memorex 1380 Communications Processor Introduction Manual.* Publication No. 1380.03-01, Memorex Marketing Distribution Center, Santa Clara, California.
3. *Burroughs Field Engineering Technical Manual B5700/B6700 Data Communications Processor.* Form No. 1041639, Burroughs Corporation, Detroit, Michigan.
4. Cocanower, A. B., W. Fischer, W. S. Gerstenberger and B. S. Read, *The Communications Computer Operating System—The Initial Design.* Manual 1070-TN-3. Merit Computer Network, Ann Arbor, Michigan, 1970.
5. Srodawa, R. J., "The Tree-Structured Distributed-Network Front-End Processor Characteristics."
6. *EIA Standard RS-232-C for the Interface between Data Terminal Equip-*

- ment and Data Communication Equipment Employing Serial Binary Interchange*, EIA Engineering Dept, Electronics Industries Association, Washington, D.C.
7. McNamara, J. E., *Technical Aspects of Data Communication*, Educational Services Dept., Digital Equipment Corporation, Maynard, Massachusetts, 1978.
 8. *Z80-MCB Hardware User's Manual*, Form No. 03-0007-03, Zilog Corporation, Cupertino, California.
 9. *Z80-RMB Hardware User's Manual*, Form No. 03-3016-01, Zilog Corporation, Cupertino, California.
 10. *Z80-SIB Hardware User's Manual*, Form No. 03-0051-00, Zilog Corporation, Cupertino, California.
 11. *Z80-IOB Hardware User's Manual*, Form No. 03-0045-01, Zilog Corporation, Cupertino, California.
 12. *IBM Virtual Machine Facility/370: CP Command Reference for General Users*, IBM Systems Reference Library, Form No. GC20-1820, IBM Library Services Dept., White Plains, New York.
 13. Moore, G. C., *Network Models for Large-Scale Time-Sharing Systems*, Report No. 1971. 71-1, ISDOS Research Project, University of Michigan, Ann Arbor, Michigan.
 14. Ritchie, D. M., and Thompson, K., "The UNIX Time-Sharing System," *Comm. ACM*, Vol. 17, No. 7, July 1974, pp. 365-375.
 15. Ivie, E. L., "The Programmer's Workbench—A Machine for Software Development," *Comm. ACM*, Vol. 20, No. 10, Oct. 1977, pp. 746-753.

Analysis of real-time control systems by the model of packet nets

by MOHAMED GAWDAT GOUDA

*Honeywell Corporate Technology Center
Minneapolis, Minnesota*

INTRODUCTION

During the planning stages of a real-time control system, the planning team is more concerned about the global aspects of the system rather than the details of its different components. The first questions that have to be resolved are, What are the basic processes (or modules, or tasks) in the system? What are the data flow rates between these processes? and What are the memory and processing rate requirements of each process? Also of prime concern is how to select a suitable computer architecture to host the system and how to decompose the system on a selected distributed architecture.

In order to be able to address these questions in a systematic and formal way, we introduce a model, called Packet Nets, for real-time control systems. The model is based on the concepts of data flow;^{4,12,11,3} but unlike other data flow models, this model focusses on the global system characteristics and hides the details which are not needed during system planning. The model is intended as a tool to represent real-time control systems in an implementation-independent fashion. Moreover, it can be used in estimating the values of some basic system parameters such as data flow rates between system components and the processing rate requirement of each system component.

The model of packet nets is introduced in the next section following the introduction. In the third section, we present a systematic method to compute data flow rates between the different processes in a packet net. Then in the fourth section, a technique to estimate the processing rate requirement of each process in the net is outlined. In the fifth section, the memory requirements of packet nets are discussed. In the final section, we discuss some issues concerning the decomposition of packet nets on distributed computer architectures.

PACKET NETS

A process in a real-time control system performs three basic operations in a cyclic fashion—receiving (or input operation), execution and sending (or output operation). The process first receives a ‘‘packet’’ of data from another proc-

ess in the system or from the external world (e.g., sensor data coming from a sensor). Then the process executes some internal functions on the received data and finally it sends the results as a data packet to the next process in the system or to the external world; then the cycle repeats. The process is halted temporarily if its input queue is empty during a receiving operation, or if its output queue is full during a sending operation.

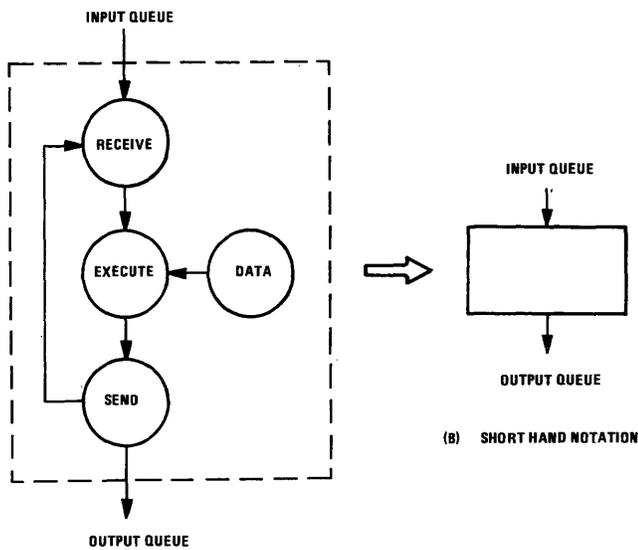
Graphically, a process can be presented as a directed cycle (Figure 1a) with three types of nodes—a receiving node, an execution node and a sending node. Attached to the receiving node is an input queue via which the incoming data packets are received. Similarly, an output queue is attached to the sending node to send the result packets to the next process. The internal memory of the process is represented as a data node attached to the execution node. Using a short-hand notation, a process can be also represented as a single node (Figure 1b) with input and output queues.

The model in Figure 1 is restrictive as each process can receive data packets from only one process and can send data packets to only one process. To remove these restrictions, we extend the model such that a process can have more than one input and one output queue. As shown in Figure 2, the different input queues of a process are controlled by an OR-receiving operation or by an AND-receiving operation. Similarly, the different output queues are controlled by an OR-sending or an AND-sending operation. These four sending/receiving operations are defined (informally) as follows.

To execute the OR-receiving operation, the process waits until one of its input queues has at least one packet. Then, the process removes one packet from this input queue and proceeds to process it. To execute the AND-receiving operation, the process waits until there is at least one packet in each input queue. Then, the process removes one packet from each input queue and proceeds to process them.

To execute the OR-sending operation, exactly one of the associated output queues is selected (arbitrarily) to put the sent data packet in it. To execute the AND-sending operation, one data packet is put in each output queue. The sent packets are not necessarily identical.

A process with an OR-receiving operation and an AND-



(A) DETAILED DEFINITION

Figure 1—A single-input, single-output process in a packet net.

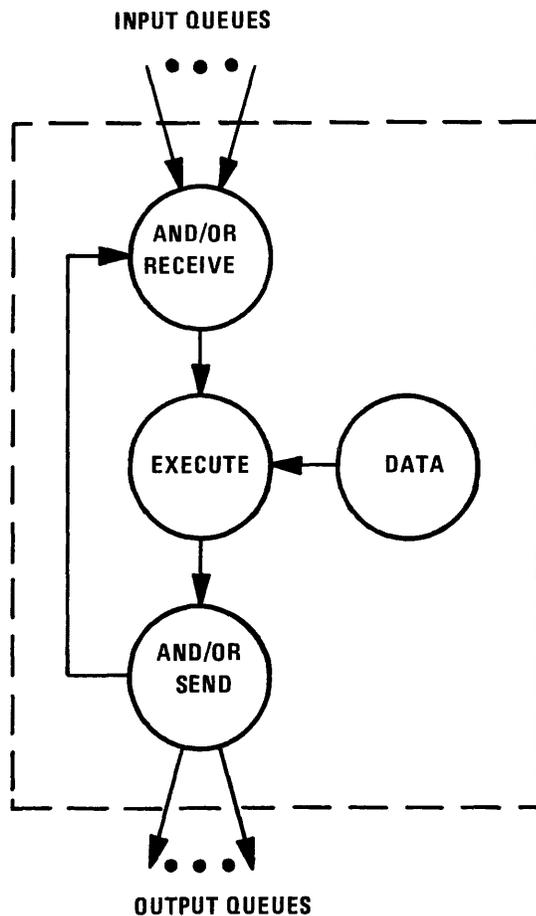


Figure 2—A multi-input, multi-output process in a packet net.

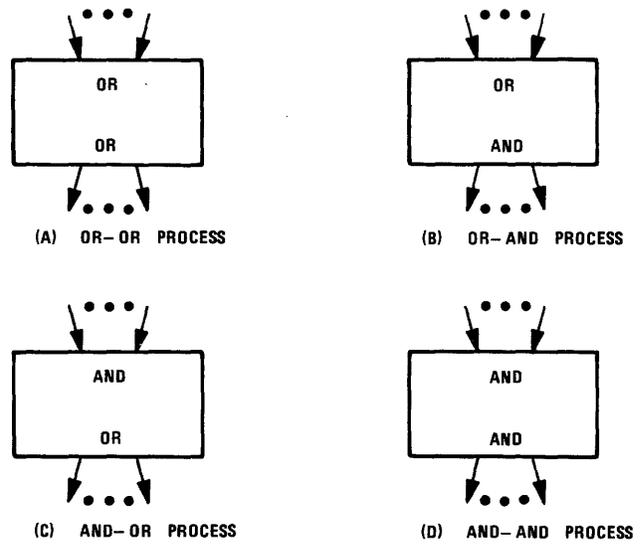


Figure 3—Different classes of processes in a packet net.

sending operation is called OR-AND process. Similarly, OR-OR, AND-AND and AND-OR processes can be defined. A short-hand notation for the different processes classes of processes in a packet net is shown in Figure 3. Observe that if a process has a single input queue, then an OR-receiving operation becomes equivalent to an AND-receiving operation. Therefore, we use the convention that if a process has a single input (or a single output) queue, then it has an OR-receiving (or OR-sending respectively) operation.

To give an example of a real-time control system and its packet net, consider a radar scheduling system.⁶ The system receives requests to use the radar. It orders these requests based on some priority scheme, then delivers the ordered sequence (called frame) of requests to the radar to serve them in order. But since the radar cannot accept any frame unless it satisfies some constraints, the system has also to examine the generated frame against all the constraints (imposed by the radar) before delivering the frame to the radar.

Figure 4a shows the outline of a radar scheduling system. Arriving requests are of three different classes. Each request has its own local priority such that requests of the same class are ordered according to their local priorities. The partially sorted requests are sent to the framing process to be ordered in a time frame based on some global priority. The frame is then sent via a frame modification process to a number of checking processes. Each checking process examines (in parallel with other checking processes) whether or not the frame meets some radar constraint. The individual decisions for the frame are then sent to a final decision process to decide whether the frame is accepted or rejected. Accepted frames are sent to the radar, while rejected frames are sent back to the frame modification process to be corrected before being examined once more. The packet net for this system is shown in Figure 4b.

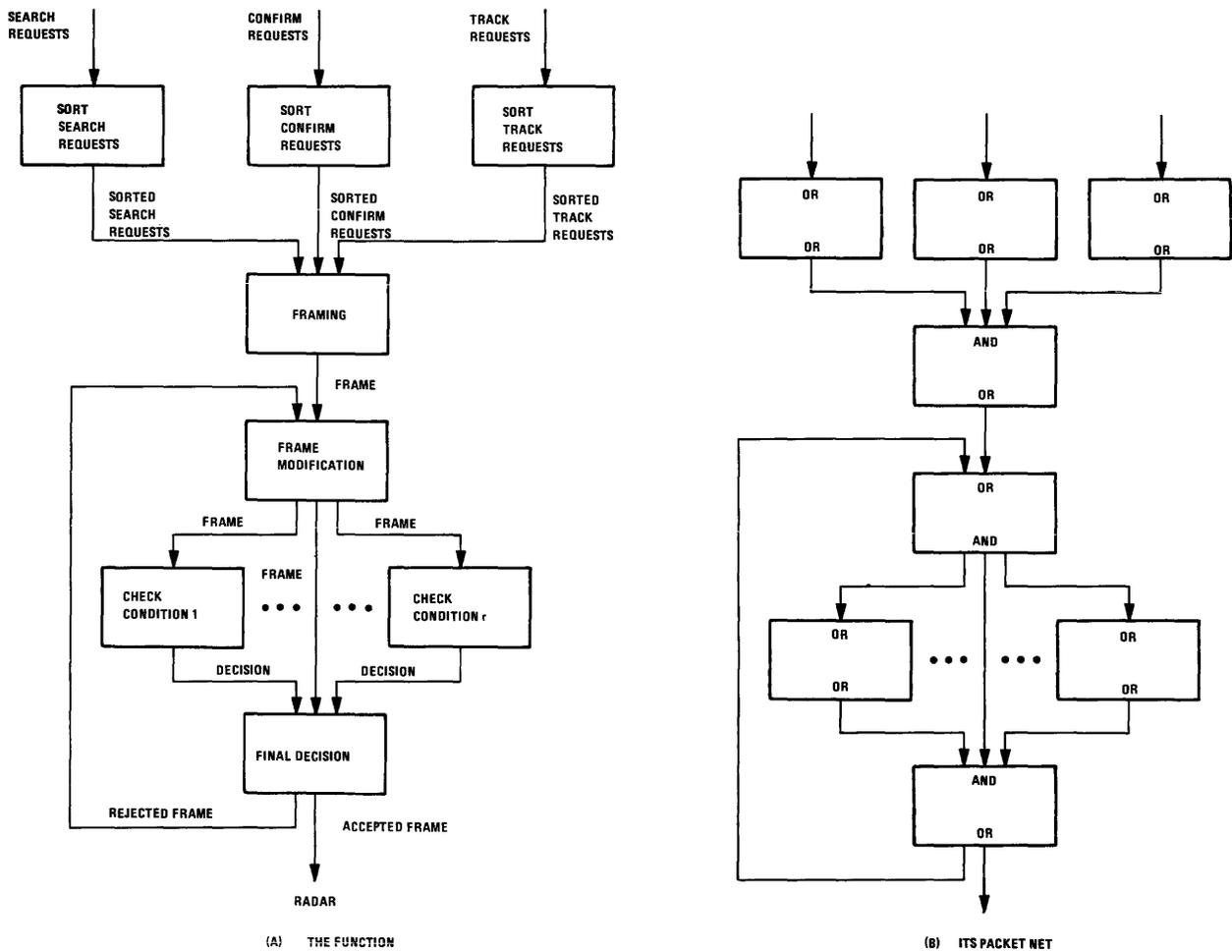


Figure 4—A radar scheduling system.

PACKET FLOW RATES

Part of the analysis of a real-time system is to compute the rates with which the data flow between the different system components. In the packet net model, this corresponds to compute the packet flow rate in each queue in the net. To make this computation, the flow rate (in packets/sec.) in each external input queue should be estimated from the application (e.g., the rate with which the sensor data arrive to the system). If an input flow rate changes with time, an upper bound for its value should be estimated. Also, the probabilities with which each OR-sending operation distributes the output packets among its output queues should be estimated from the application.

As an example, consider the packet net in Figure 5. The external input flow rate to the net is estimated to be 10 packets/second. The output queues of OR-OR Process 1 are labelled with the probabilities .2, .1 and .7, while the output queues of OR-OR Process 2 are labelled with the probabilities .5 and .5. The variables t , u , . . . , and z are associated

with the different queues in the net; each variable is defined to be the packet flow rate in its associated queue. The problem, now, is to evaluate these variables. Next, we present the equations to evaluate the packet flow rates in any well formed packet net.

Consider the OR-OR process shown in Figure 6a. The input packet flow rate to that process equals $\sum_{j=1}^m x_j$. This flow is distributed among the n output queues according to their associated probabilities. Thus, the packet flow rate y_i in the i th output queue is evaluated as follows:

$$y_i = p_i \sum_{j=1}^m x_j \quad i = 1, \dots, n$$

Similarly, the packet flow rate y_i in the i th output queue of an OR-AND process (Figure 6b) is evaluated as follows:

$$y_i = \sum_{j=1}^m x_j \quad i = 1, \dots, n$$

Consider the AND-OR process shown in Figure 6c. Be-

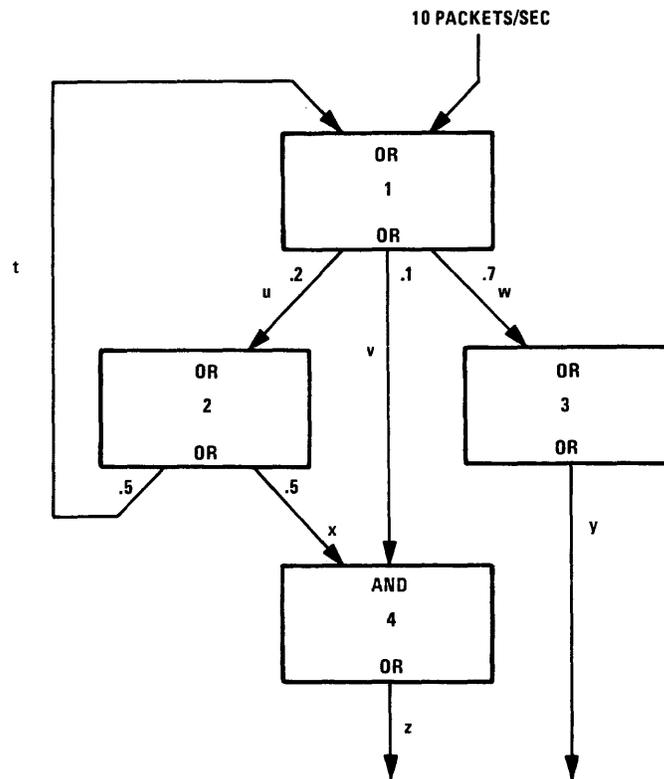


Figure 5—A packet net.

cause of the AND-receiving operation in this process, the flow rates in all input queues should be equal. Assume that the flow rate in any input queue equals x . This flow is distributed among the n output queues according to their associated probabilities. Thus, the packet flow rate y_i in the

i th output queue is evaluated as follows:

$$y_i = p_i x \quad i=1, \dots, n$$

Similarly, the packet flow rate y_i in the i th output queue of an AND-AND process (Figure 6d) is evaluated as follows:

$$y_i = x \quad i=1, \dots, n$$

For any packet net, we can write the flow equations using the general forms just discussed. By solving these equations simultaneously, the packet flow rates in each queue in the net can be determined. For example, the flow rate equations for the net in Figure 5 can be written as follows:

$$\begin{aligned} t &= .5 u \\ u &= .2 (t+10) \\ v &= .1 (t+10) \\ w &= .7 (t+10) \\ x &= .5 u \\ y &= w \\ z &= x = v \end{aligned}$$

Thus, the flow rates can be determined: $t=v=x=z=1.11$, $u=2.22$, $w=y=7.78$. All dimensions are in packets/second. If the packet sizes (in bits/packet) for each queue can be estimated, then the flow rates can be computed in bits/seconds.

For many packet nets, the flow equations are inconsistent; thus produce no solution. For example, if the output probabilities of Process 2 in the packet net in Figure 5 are

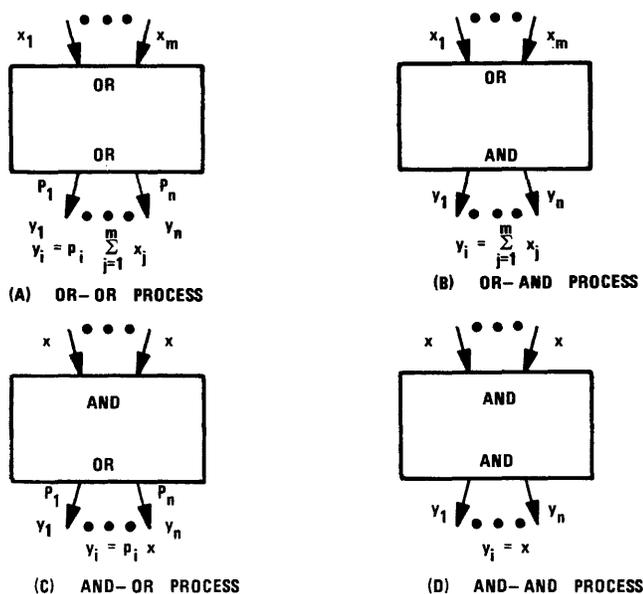


Figure 6—Data flow rates in the output queues of a process.

changed from .5 and .5 to .4 and .6, we get the following flow equations:

$$\begin{aligned}
 t &= .4 u \\
 u &= .2 (t+10) \\
 v &= .1 (t+10) \\
 w &= .7 (t+10) \\
 x &= .6 u = .12 (t+10) \\
 y &= w \\
 z &= x
 \end{aligned}$$

From the third and fifth equations, we have

$$v \neq x$$

This contradicts the requirement that v and x should be equal since they are AND inputs to Process 4 in the net.

A packet net is said to be well formed only if its flow equations are consistent and produce a single solution to the flow rates in the net. Obviously, the packet net in Figure 5 is well formed; but if we change the output probabilities of Process 2, the resulting net is not well formed.

The consistency of flow equations for a packet net depends on the net structure, and the estimated output probabilities for the processes in the net. Therefore, inconsistent flow equations imply that either the net "structure" is intrinsically inconsistent, or that our estimation of the output probabilities is inconsistent. In the previously-mentioned example, we illustrated how an inconsistent estimation of the output probabilities can lead to an inconsistent set of flow equations. Next, we give some examples of intrinsically inconsistent packet nets.

Figure 7a shows a part of a packet which consists of a cycle with n OR-AND processes. The n flow equations for such cycle are as follows:

$$\begin{aligned}
 x_1 &= x_n + I \\
 x_2 &= x_1 \\
 &\vdots \\
 x_n &= x_{n-1}
 \end{aligned}$$

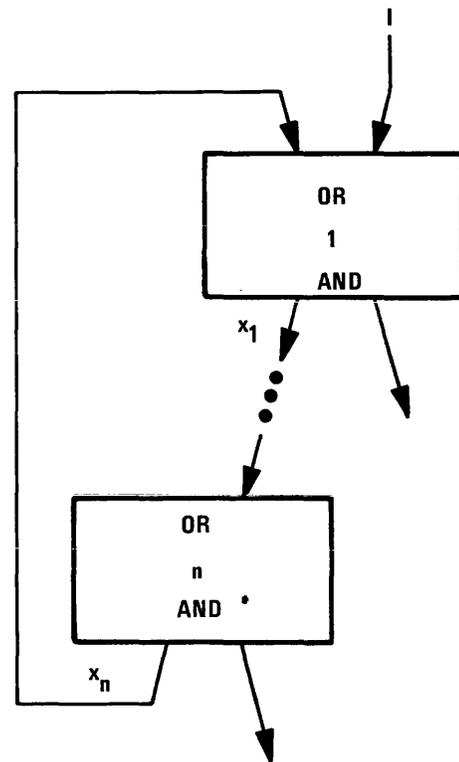
From these equations, we get $x_n = x_n + I$ (which can be true iff $I=0$). Thus, in Case $I \neq 0$, this cycle is intrinsically inconsistent.

As another example, consider a cycle of n AND-OR processes (Figure 7b). The flow equations for this cycle are as follows:

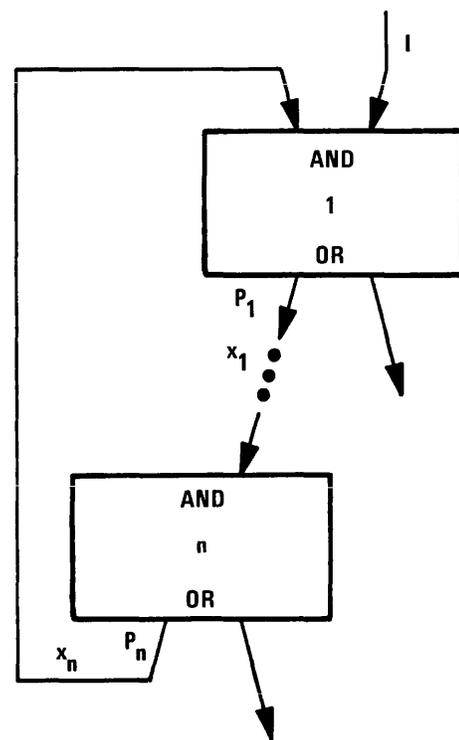
$$\begin{aligned}
 x_n &= I \\
 x_1 &= p_1 x_n \\
 x_2 &= p_2 p_1 x_n \\
 &\vdots \\
 x_n &= p_n p_{n-1} \dots p_2 p_1 x_n
 \end{aligned}$$

which can be satisfied iff $p_1 = p_2 = \dots = p_n = 1$; i.e., each process should have exactly one output queue. If any process has more than one output queue, then the cycle is intrinsically inconsistent.

From these examples it is clear that a packet net should not contain a cycle whose nodes are all OR-AND processes,



(A) A CYCLE OF n OR-AND PROCESSES



(B) A CYCLE OF n AND-OR PROCESSES

Figure 7—Examples of packet subnets.

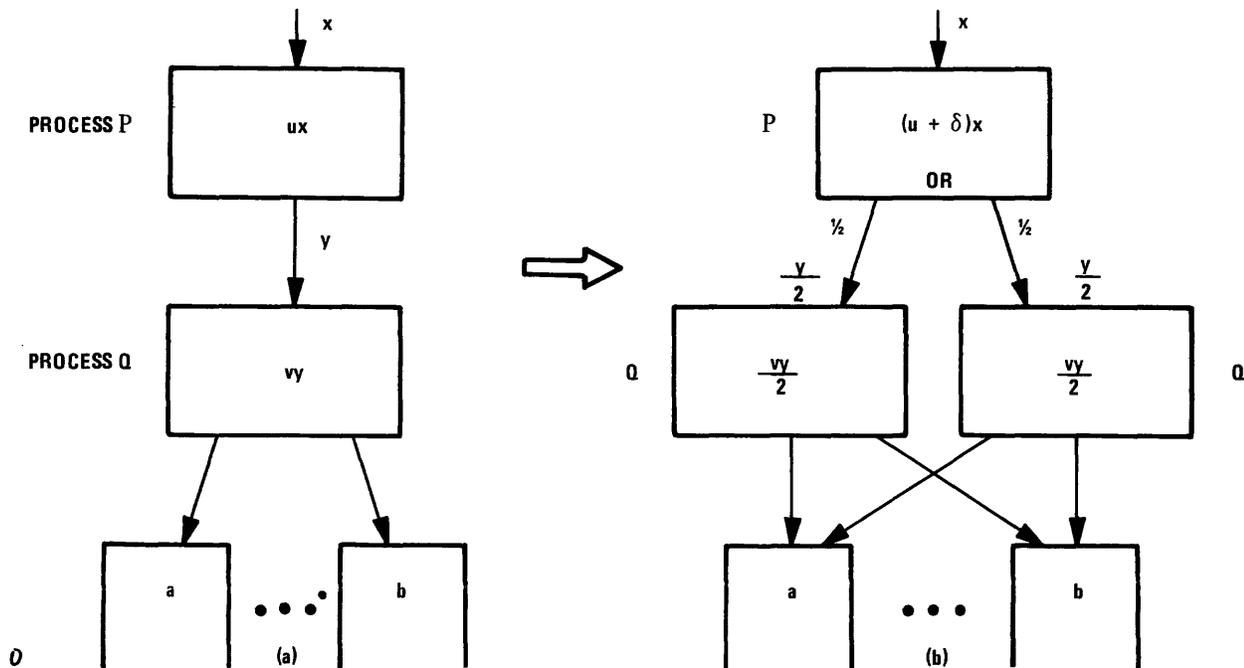


Figure 8—An example of adding extra copies of a process.

or whose nodes are all AND-OR processes. It is also clear that flow equations are useful in designing well formed packet nets.

EXECUTION RATES

Another part of the analysis of a real-time system is to compute the execution rates (in inst./sec.) for all the processes in the system. In order to make this computation, the number of instructions in the main cycle of each process (see Figures 1a and 2) in the net should be estimated.

Assume that the input packet flow rate to a process P is $\sum_{i=1}^m x_i$ packets/second. Therefore, the cycle of P should be executed $\sum_{i=1}^m x_i$ times per second. If the cycle of process P has r instructions, then the required execution rate of P is $r \sum_{i=1}^m x_i$ inst./sec. Observe that the above computation does not depend on the process type (e.g. OR-OR, OR-AND, . . .).

The required execution rate of a process can be reduced by adding extra copies of the same process to the packet net. For instance, Figure 8a shows an arrangement of two processes P and Q connected by a queue (from P to Q). Assume that the input packet flow rates to P and Q are x and y respectively. Thus, the required execution rates for P and Q are ux and vy , where u and v are the number of instructions in the main cycles of P and Q respectively. Assume that an extra copy Q' of process Q is added to this arrangement, as shown in Figure 8b. This requires that process P becomes responsible for equally distributing its output packets to both Q and Q' . In other words, P becomes an OR-OR process with output probabilities of .5 and .5. The

input packet flow rate to Q (or to Q') becomes $y/2$; and the execution rate of Q (or Q') becomes $vy/2$ half the original amount. On the other hand, process P has more responsibilities; and the number of instructions in its cycle increases with the amount of δ . Hence, its execution rate will increase to become $(u + \delta)x$ inst./sec.

In the previous example, the distribution of data packets between the process copies is handled by an already existing process in the net. In some cases, however, a new process is added to the net to perform this distribution function. We call such a process a distributor. Figure 9 shows how $(k-1)$ additional copies of one process and a distributor are added to a packet net to reduce the process execution rate by a factor of $1/k$. Observe that the distributor execution rate δy is "small" since there is a "small" number of instructions (namely δ) in its cycle.

Another advantage of adding extra copies of the same process to a packet net is to increase the overall system reliability and availability. At any rate, whether the extra process copies are added to decrease the execution rate requirements or to increase the system reliability, the implied assumption is that no two (or more) copies of the same process should be assigned to the same processor in the host hardware. The problem of assigning processes to processors is discussed in more detail later on in the sixth section; but first we need to discuss the memory requirements of packet nets.

MEMORY REQUIREMENTS

Memory is required for a packet net to store its processes and its queues. The required memory to host one process

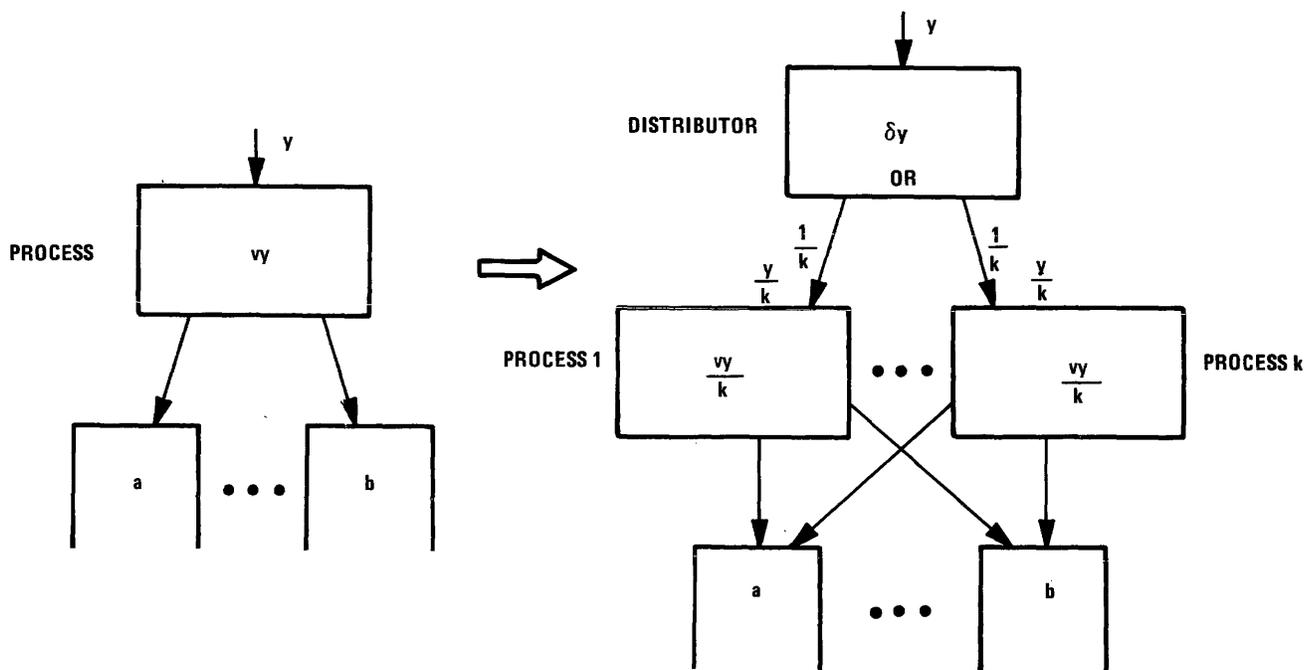


Figure 9—Adding an extra $(k-1)$ copies of a process to decrease its execution rate by a factor of $1/k$.

in the net depends on the control and data structure of that process; i.e., it depends on the size of its program and its data file(s), if any. An upper bound for the required memory of each process should be estimated from the application so that the total memory requirements can be calculated.

The required memory to host the queues of a packet net depends primarily on how this memory is organized and how it is managed. We next describe one scheme for the organization and management of such a memory. Other attractive schemes are not discussed in this paper because of the space limitation. For convenience, we define an OR process to be either an OR-OR process or an OR-AND process, i.e., it is a process with an OR-receiving operation. Also, we define an AND process to be either an AND-OR process or an AND-AND process.

Consider an OR process P with m inputs (Figure 10a). To allow both P and its input processes to proceed concurrently, the input queues of P should be implemented as one packet pool which can hold up to $m+1$ packets (Figure 10a). When one input process (say, Q) finishes the assembly of a new packet (say, p_1) in the packet pool, and P finishes the processing of an old packet (say, p_2) from the packet pool, then P initiates a packet interchange with Q . Thus, Q starts to assemble a new packet in place of p_2 , while P starts to process the packet p_1 .

Similarly, if P is an AND process with m inputs, then the packet pool between P and its input processes should hold up to $2m$ packets (Figure 10b). When each input process finishes the assembly of a new packet, and P finishes the processing of the m old packets, then P initiates a packet interchange which causes P to get the m new packets and assigns each input process a space for one packet.

In this static memory allocation scheme, the storage capacity of any cycle in the packet net is fixed. Thus, it is possible that packets may continue to flow into some cycle in the net until the cycle is completely full. At this instant, the cycle processing stops and a deadlock situation arises. The remainder of this section examines such deadlocks in more detail. In particular, we introduce some sufficient conditions for deadlock avoidance.

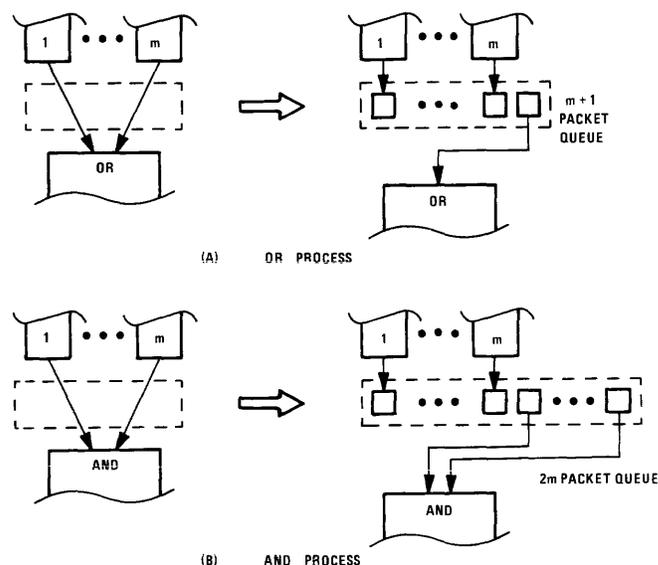


Figure 10—Queue requirements in packet nets.

Deadlocks arise in a packet net whenever one or more cycles in the net become full of packets. But, because AND processes can never increase the number of packets in any cycle (as illustrated in Figure 11), they can never contribute to deadlock situations. Therefore, we need only to consider OR processes. In general, if a cycle contains one or more OR processes in a packet net, the OR processes can fill the cycle with packets causing a deadlock. In order to avoid such a deadlock, we need sufficient conditions to prevent the OR process from filling their cycles with packets. Next we present such a set of conditions; but first we need to define "cycle queues." A cycle queue is a queue which exists in one or more cycles in the packet net.

The following three conditions on the structure and operation of OR processes are sufficient to avoid deadlocks in packet nets:

cond1—Each OR process has at most one input cycle queue (Figure 12a). A packet which arrives to the OR-process via the input cycle queue is called a cycle packet; otherwise it is a non-cycle packet.

cond2—If there are one cycle packet p_1 and one space p_2 for a new packet in the packet pool of an OR process P , then P initiates a packet interchange such that p_2 is assigned to the input cycle queue to assemble a new cycle packet, and p_1 is assigned to P to be processed. However, the processing of

p_1 will not start until there is one packet space in each output queue of P .

cond3—If there are one non-cycle packet p_1 and one space p_2 for a new packet in the packet pool of an OR-process P , and if there is one packet space in each output of P , then P initiates a packet interchange such that p_2 is assigned to the input non-cycle queue (via which p_1 has arrived), and p_1 is assigned to P for processing. The processing of p_1 can start immediately after the packet interchange.

Observe that both *cond2* and *cond3* do not restrict the structure of packet nets in any way; they merely enforce some order on the packet processing by the OR processes. On the other hand, although *cond1* does restrict the structure of packet nets, we feel that the restriction is not severe; the packet net in Figure 4b satisfies *cond1*.

The processing of cycle packets can never increase the number of packets in any cycle in the net; thus, it can never lead to a deadlock. For this reason, *cond2* implies that the processing of cycle packets should proceed whenever possible (Figure 12b).

The processing of one non-cycle packet by an OR process can generate at most one extra packet in every cycle which contains the OR process. Therefore, the third condition *cond3* implies that the processing of one non-cycle packet

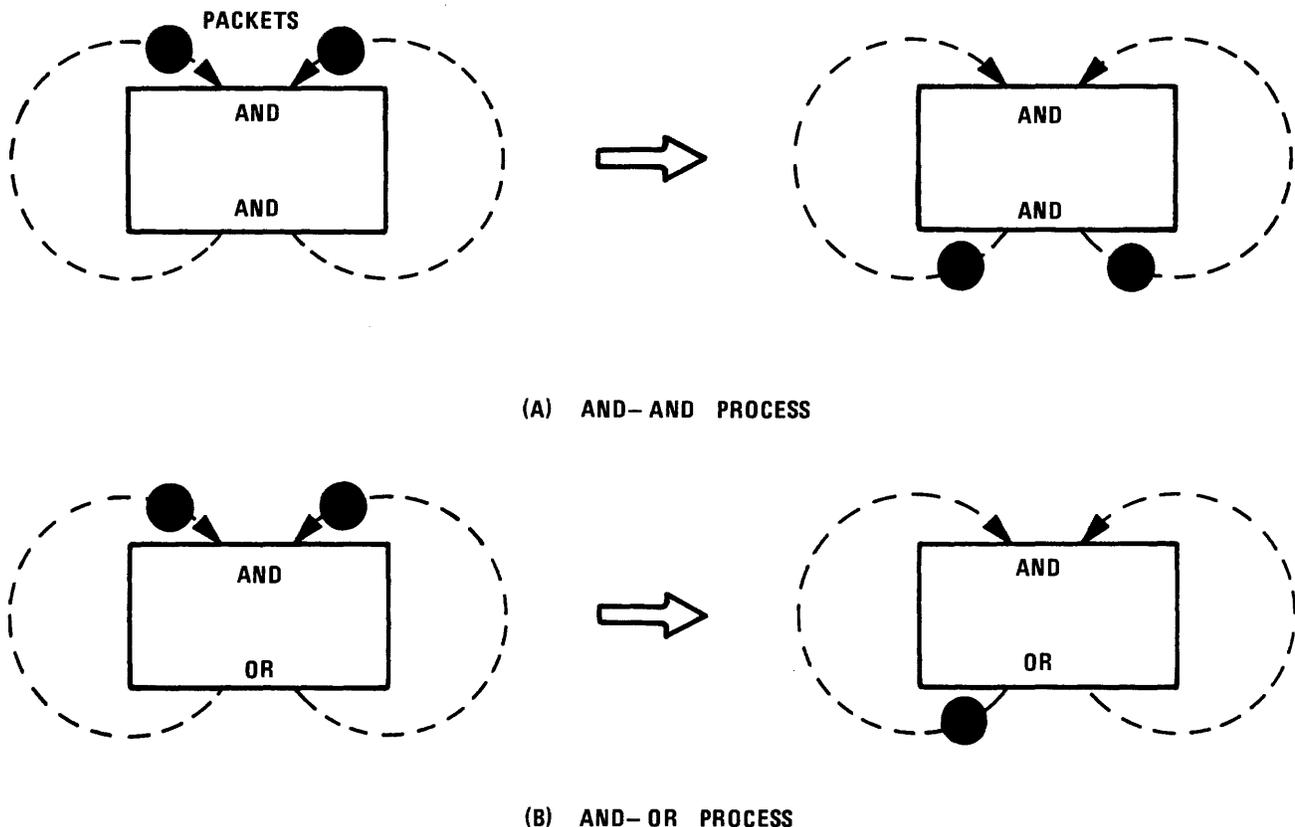


Figure 11—AND-processes do not increase the number of packets in cycles.

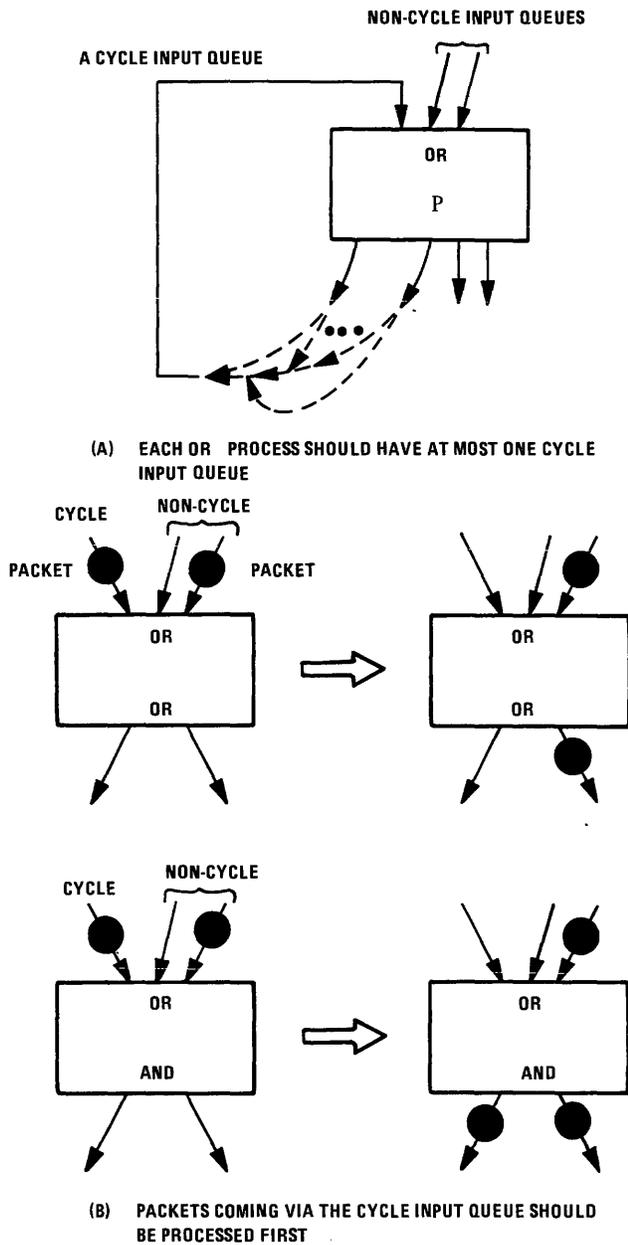


Figure 12—Avoiding deadlocks in packet nets with OR-processes.

shouldn't start unless the OR process "sees" two available packet spaces in each cycle which contains the OR process. This is the case when the OR process has one packet space in its input pool, and has one packet space in any of its output pools. Under this condition, the processing of non-cycle packets can never make any cycle full of packets.

Because of the space limitation, the discussion in this section has been limited to static memory allocation where each queue has been assigned a fixed memory size. We acknowledge, however, that static memory allocation can be unaffordable in some applications. The solution for these applications lies in dynamic memory allocation where

queues can grow and shrink in size as their needs change with time.

DISTRIBUTED COMPUTER SYSTEMS FOR REAL-TIME CONTROL

In this section, we investigate some of the problems associated with designing distributed computer systems for real-time control. Our bias toward distributed computer systems stems from our conviction that these systems provide high degrees of extensibility, integrity and performance¹⁰ which are most needed in real-time environments.

In the previous sections, we have discussed how to calculate the following quantities:

1. The data flow rates (in packets/second, or equivalently in bits/second) between the net processes.
2. The execution rate (in instructions/second) of each process in the packet net.
3. The memory requirements (in words) of each process in the net.

These quantities are needed for the design of a distributed computer system to host a packet net (i.e., a real-time application). For that reason, these quantities are recorded on a graph, called the system graph of the packet net. The nodes in the system graph correspond to bi-directional communication between processes in the packet net. As an example, Figure 13b shows the system graph for the packet net in Figure 13a.

Each node i in the system graph is labelled with two quantities (e_i, m_i) , where e_i (in instruction/second) is the execution rate of the corresponding process, while m_i (in words) is the memory requirement for the corresponding

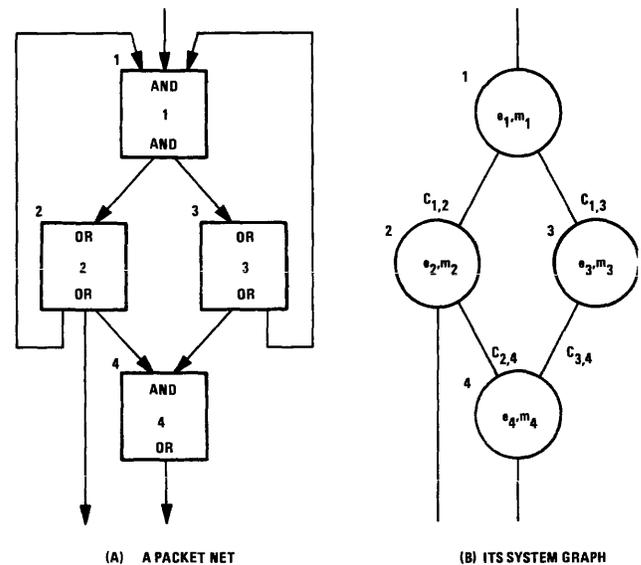


Figure 13—The relationship between a packet net and its system graph.

process and its input buffer. Each edge (i, j) in the system graph is labelled with a quantity $c_{i,j}$ (in bits/second) which is the sum of the data flow rate from process i to process j and the data flow rate from process j to process i .

The system graph of a packet net contains all the needed information to design an "optimum" distributed computer system to host the packet net. In Reference 7 we outline a design methodology of distributed computer systems based on the model of system graphs. In this methodology, a system graph is used to generate a set of possible distributed architectures which can host the packet net. Then, from these generated architectures, an "optimum" architecture is chosen based on cost and reliability considerations. This approach still needs more research and more design automation support before it can be utilized in a practical way.

A more practical approach is to begin with a ready-made distributed computer system which can host the packet net. Then, find the "optimum" assignment of the packet net processes to the system processors.^{2,5,8,9} This optimization problem can be expressed and solved in terms of system graphs as illustrated by the following example.

Consider the shared bus system shown in Figure 14. It consists of some processors; each of them has its own private memory and its own Bus Interface Unit (BIU). The processors communicate only by sending and receiving messages via the global bus which they control in a decentralized fashion. In this architecture, the global bus is the limiting resource.¹ Thus, on assigning processes to processors in this architecture, the object should be to minimize the bus traffic.

Assume that a shared bus system with r processors is to host a packet net. The optimum process assignment can be reached by finding at most $r-1$ minimum cuts for the system graph of the packet net. These cuts should satisfy the following two conditions:

cond1—The cuts partition the nodes in the system graph into at most r partitions such that for each partition:

$$\sum_{\text{partition}} e_i \leq E$$

$$\sum_{\text{partition}} m_i \leq M$$

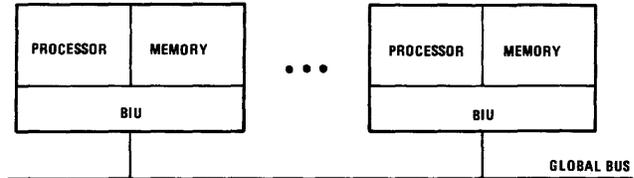


Figure 14—A shared bus system.

where E is the processing rate which can be devoted to the application by one processor in the system, and M is the memory which can be devoted to the application by one processor in the system.

cond2— $\sum c_{i,j}$ is minimum
 i in one partition,
 j in another partition

Finding these minimum cuts may require, in general, an exhaustive search among all possible cuts.

It has been mentioned in the fourth section that for reliability reasons, different copies of the same process should be assigned to different processors. To make sure that the minimum cuts technique will produce this assignment, we add an edge to the system graph between any two nodes which correspond to copies of the same process. These edges will be labelled with $-\infty$, as in Figure 15; thus, the minimum cuts must cut through all of them.

Sometimes it is required (e.g., for input/output considerations) to assign some processes to specific processors. To make sure that the minimum cuts technique will produce these assignments, the following additions should be made to the system graph:

- 1.—A node labelled $(0,0)$ should be added for each processor which needs to host specific process(es).
- 2.—An edge labeled $-\infty$ should be added between any two processor nodes.
- 3.—An edge labelled ∞ should be added between a processor node and a process node iff it is required to assign the process to the processor.

An example is shown in Figure 16.

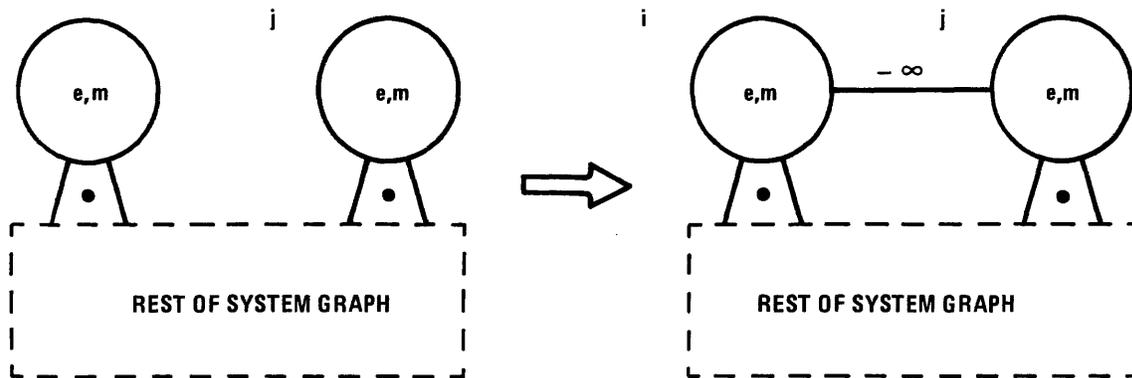


Figure 15—Assigning two copies i and j of same process to different processors.

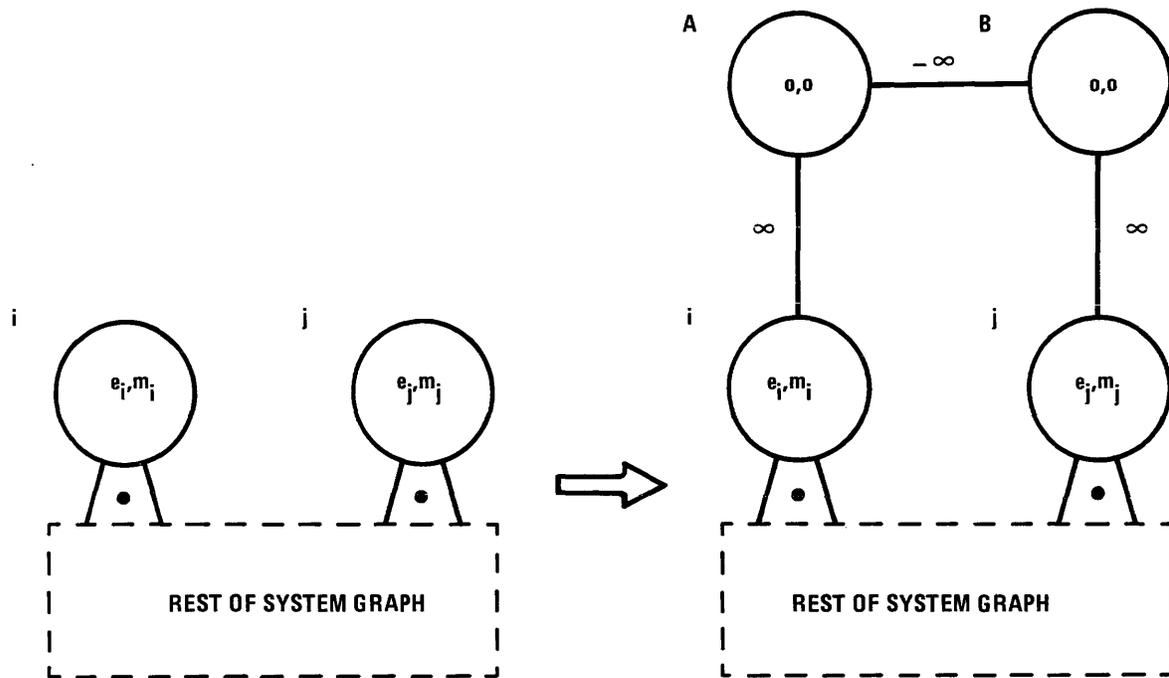


Figure 16—Assigning process i to processor A , and process j to processor B .

CONCLUSIONS

A simple model to represent a wide class of real-time control systems is introduced. The model can be used in the early stages of system planning to represent the system in a gross form while hiding the fine details till later stages. The model can be also used to compute the values of different parameters which are critical to the planning of real-time systems. The required computations are simple and can be constructed and executed in a systematic way. The model is also useful in planning the computer system to host the application.

It is conceivable that for specific applications, more features (or restrictions) can be added to (or imposed on) the model to "tune" it to that particular application. In this presentation, however, we have only discussed those features that seem common to a wide class of applications.

ACKNOWLEDGMENTS

Some of the ideas in this paper have been motivated by discussions with R. G. Arnold and E. D. Jensen. I am also thankful to the referees for their helpful comments.

REFERENCES

1. Arnold, R. G., and L. L. Kinney, "Analysis of a Multiprocessor System with a Shared Bus," *Proc. of the Fifth Annual Symposium on Computer Architecture*, April 1978.
2. Brantley, W. C., et. al., "Decomposition of Data Flow Graphs on Multiprocessors," *Proc. of the National Computer Conference*, 1977, pp. 379-388.
3. Denning, P. J., "Operating Systems Principles for Data Flow Networks," *Computer*, July 1978, pp. 86-96.
4. Dennis, J. B., *First Version of a Data Flow Procedure Language*, Project MAC, Computation Structure Group Memo 93-1, August 1974.
5. Forss, B., et. al., "Distribution of Logical Processes in Telephone Systems," *Proc. of National Telecommunication Conf.*, December 1977.
6. Gouda, M. G., R. Y. Kain, et. al., *Distributed Data Processing Technology, Vol. 4 - Application of DDP Technology to BMD: Architectures and Algorithms*, Honeywell Systems and Research Center Report, September 1977.
7. Gouda, M. G., et. al., "Towards a Methodology for Distributed Computer System Design," *Proc. of the Sixth Texas Conf. on Computing Systems*, November 1977.
8. Hofri, M., and C. F. Jenny, *On the Allocation of Processes in Distributed Computing Systems*, IBM TR RZ905 (#30471), October 1978.
9. Jenny, C. J., "Process Partitioning in Distributed Systems," *Proc. National Telecomm. Conf.*, December 1977.
10. Jensen, E. D., "The Honeywell Experimental Distributed Processor—An Overview," *Computer*, January 1978, pp. 28-39.
11. Kodres, U. R., "Analysis of Real-Time Systems by Data Flowgraphs," *IEEE Trans. on Software Engineering*, Vol. SE-4, No. 3, May 1978.
12. Rumbaugh, J., "Data Flow Languages," *Proc. of the 1975 Sagamore Computer Conf. on Parallel Processing*, pp. 217-219.

Performance and economy of a fault-tolerant multiprocessor*

by JAYNARAYAN H. LALA and CHARLES J. SMITH

The Charles Stark Draper Laboratory, Inc.
Cambridge, Massachusetts

INTRODUCTION

The FTMP (Fault-Tolerant Multiprocessor) is one of two central aircraft fault-tolerant architectures now in the prototype phase under NASA Langley Research Center sponsorship.^{1,2} The intended application of the computer includes such critical real-time tasks as "fly-by-wire" active control and completely automatic Category III landings (zero visibility and zero ceiling) of commercial aircraft. The life-critical nature of these tasks and the profit-oriented attitude of airlines translate into some very challenging and sometimes contradictory computer requirements. For example, a candidate computer should be able to execute tasks on time, without significant delays, be extremely reliable, and yet be cost-effective. At a first glance these requirements seem to be in conflict with each other. After all, to meet the performance criteria some sort of parallelism such as multiprocessing capability would be in order. Second, to meet the reliability and safety requirements it would be necessary to use redundancy since no simplex computer could possibly meet the "extremely improbable" failure criterion of the Federal Aviation Administration. The result is a computer that could be very expensive. Yet it may still be cost-effective. The reason is that it is the balance between the benefits and costs that determines the cost-effectiveness of a product; cost alone is not a pertinent criterion. In this paper we intend to show that the FTMP architecture is a viable solution to the multi-faceted problems of safety, speed, and cost.

The next section briefly describes the FTMP architecture. The third section is devoted to the performance analysis. Three job dispatch strategies are described, and their results with respect to job-starting delays are presented. The first strategy is a simple First-Come-First-Serve (FCFS) job dispatch executive. The results of FCFS were obtained by running a representative job-mix on a multiprocessor and through Markov modeling techniques. The other two schedulers are an adaptive FCFS and an interrupt driven scheduler. The results of these schedulers were obtained by a

GPSS simulation of a representative job-mix. The fourth section briefly discusses the impact of FTMP involvement on aircraft safety and its survival probability in the face of random hard failures. The fifth section outlines the benefits to airframe manufacturers and airlines of using a fault-tolerant computer, and estimates the life-cycle-cost to airlines of operating the FTMP. A comparison is also made on a flight-hour basis to the life-cycle-cost of operating a civil transport aircraft.

The emphasis in this paper is on results rather than means of getting those results. Simulation, analytical techniques, and empirical formulas are some of the methods and techniques used to arrive at these results. However, the methods have not been dealt with here so that more attention can be devoted to the results and their implications.

FAULT-TOLERANT MULTIPROCESSOR DESCRIPTION

The FTMP is to be constructed of ten identical line replaceable units (LRUs). Each LRU contains one processor/cache module, one mass memory module, one I/O port, one clock generator, and related peripheral support and control circuitry. Each LRU will be packaged in one one-half-ATR-long box (approximately 19.6"×4.9"×7.6"). The processor/cache modules operate in groups of three called triads. Processor triads are formed by assigning any three modules to work together in tight synchronism. A triad functions as if it were a single processor. The failure of a module within a triad does not impact the correct execution of its instruction stream because voting is used to mask the effects of the failed module. A spare module, if available, can be used to replace the failed element. Otherwise, the damaged triad is retired from service with the surviving elements being turned into spares. Memory triads are formed in much the same way as processor triads, with the three memory modules of a triad assigned to work together in tight synchronism. All elements of the multiprocessor operate using a common time reference. This time base is provided by four clock generator modules operating together and phase-locked to each other. Figure 1 illustrates the functioning of this system from a software viewpoint. Three processor triads function as the

* This work was supported by the NASA Langley Research Center under contract NAS1-13782 and National Science Foundation Grants GJ-36255 and DCR74-24116.

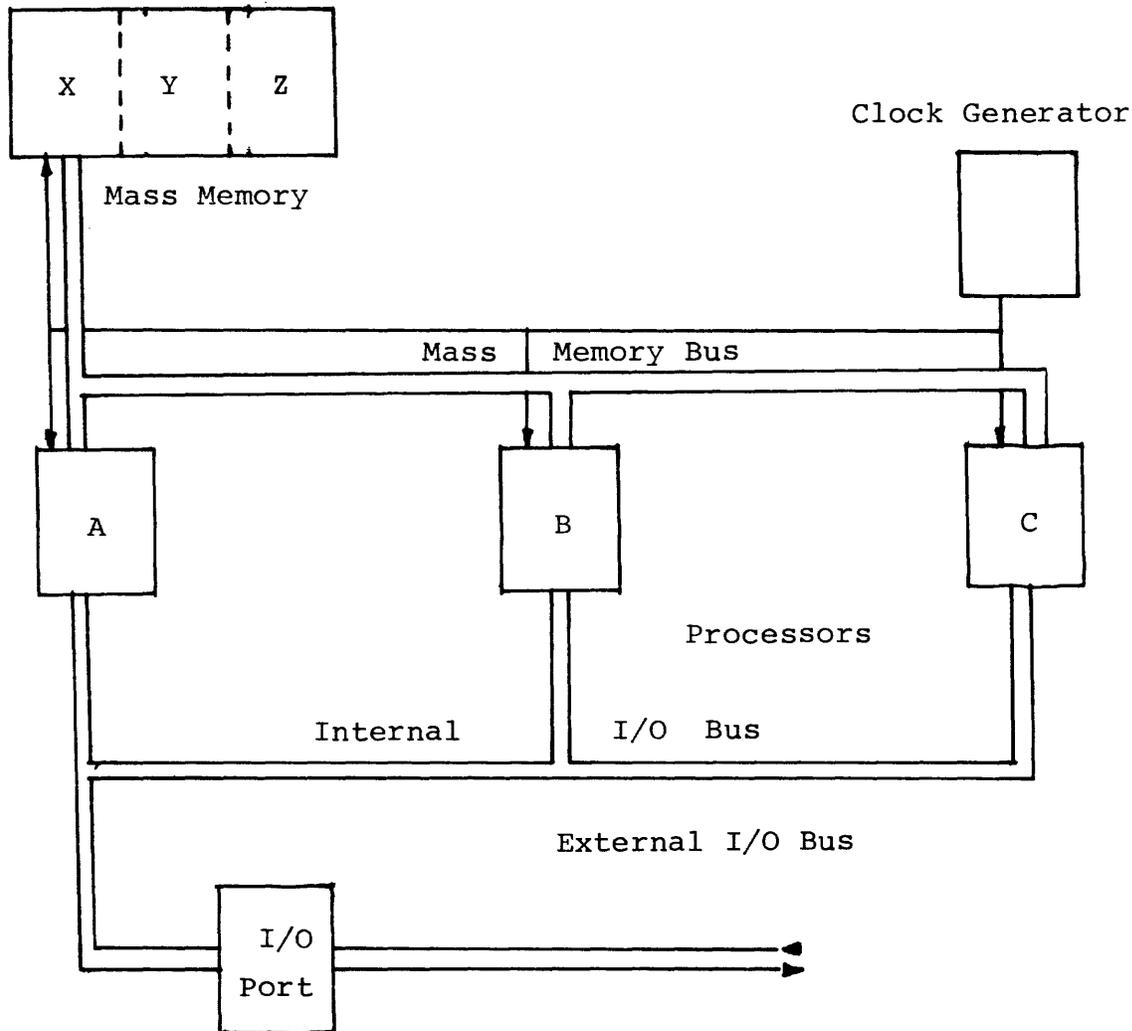


Figure 1—System architecture—Programmer's view.

logical equivalent of three simple processors with shared access to a single mass memory and a common logical I/O bus. The actual redundancy underlying the system is invisible to the programmer. The system is completely symmetric and non-discriminatory in that all processor triads operate at the same level. There are no master or slave processors or triads. Each triad is capable of performing every task in the system, be it the executive programs or applications programs, and does indeed perform every task in turn. The system executive whose performance is to be analyzed in the next section may thus be called a floating executive.

MULTIPROCESSOR PERFORMANCE IN AN AIRCRAFT ENVIRONMENT

The mission of a central computer in an aircraft can be described as the compensation and closure of a number of simultaneous sampled data control loops. Typical examples include navigation, guidance and control, load alleviation

and flutter control. The loop control jobs must be invoked at specified times or by specified events. Several studies of workload expected in an aircraft environment^{3,4} show that such a job-mix can be closely approximated by a number of purely periodic jobs, with execution frequencies ranging from several times per second to as much as a hundred times per second. In addition to the workload imposed by the applications programs there are a number of housekeeping tasks that must be carried out in a fault-tolerant multiprocessor, specifically in the FTMP. These tasks range from routine diagnostic programs to system configuration control and other internal redundancy management and system management programs. All of these jobs are also periodic in nature, and can be handled in a fashion identical to the handling of the applications programs. Such a treatment of system and applications tasks removes most of the burden and complexity from the central or the core executive. The only important function left to the executive is to dispatch jobs at appropriate times and monitor their progress to completion. In essence, the core executive is just a job sched-

uler. In conventional data processing applications, it is common to find complex operating systems in which the objective is to support a variety of simultaneous demands for resources. In this application, by contrast, we desire to have a simple, straightforward job dispatch process that is reliable, error free and easily verifiable, so that the resultant software is of a quality commensurate with the hardware reliability.

Several philosophies exist for implementing such a job scheduler. For example, the periodic jobs may be completely prescheduled with each job occupying a certain time slot in the schedule. This is the synchronous job scheduler. Its advantages include the on-time execution of jobs with practically no delays, while at the same time keeping a rather high load factor. However the overwhelming disadvantage of such an algorithm is the total inflexibility and the complexity of preassigning jobs to processors in a three-unit multiprocessor. In addition, a degradation in system capacity, to two processors, for example, or changes in job parameters such as iteration rates, may require a totally new schedule. The synchronous scheduler is therefore not considered here.

Another scheduling strategy is the First-Come-First-Serve (FCFS) executive. All jobs are simply arranged in a waitlist ordered by their starting times. Each processor triad that is idle examines this list and picks up a job that is due next. Such an executive is simple, straightforward, and easy to implement. However, it is not obvious what the impact of this job dispatch strategy on system performance would be. One of the important performance criteria in our application is the job-starting delay. The next section details the results of a simulation of the FCFS executive carried out on a breadboard multiprocessor.⁵ As shall be seen in that section, although the multiprocessor's performance may be quite satisfactory using the FCFS executive and its minor variations for most real-time applications, no guarantees can be made regarding maximum job-start delays. Therefore an interrupt-driven job scheduler was also evaluated. A major advantage of this scheduler is that the more important jobs can be given higher priority by interrupting the execution of the less important jobs. A later section describes this scheduler and the simulation results.

FCFS scheduler results

The results of the FCFS scheduler were obtained by running a representative workload of 25 periodic jobs under a FCFS scheduler on a three-processor bus centered multiprocessor which emulated the FTMP architecture from the software viewpoint. To discuss the results it is convenient at this point to define a few parameters for this section. First of all, the percent system load is defined as (100 minus percent idle). In other words, the percent load includes the percent of time a processor is running a job step or the scheduler, waiting for the scheduler or memory access, and fetching instructions/data from the shared memory. At the saturation point there is no idle time, and the load equals 100 percent. Another important parameter for this section

is the ratio, R , of the mean job step length to the mean executive/scheduler run time. This is a normalized measure of job starting delay. We define normalized delay as the delay measured in units of mean job step length.

The importance of job-starting delay as a performance criterion in a real-time system cannot be overemphasized. Extensive measurements regarding the probability distribution of the delay were made in the simulation experiments. Figure 2 shows the normalized mean job starting delay as a function of the load. The normalized delay decreases as the ratio of the mean job step-length to executive running time is increased. This is due to the fact that the higher the ratio R , the lower is the probability of executive access conflict between processors. For each value of R the mean delay function increases without bound as the load approaches saturation. In some real-time applications the important performance criterion may be the absolute rather than the normalized measure of mean starting delay. Figure 3 shows the mean delay, measured in milliseconds, as a function of the mean job step length for three values of the system load. It is seen that a linear relationship exists between these two parameters, with the slope of the function increasing rapidly with the system load.

Figures 4 and 5 show the probability distribution function (PDF) of the delay for $R=1$ and 50 respectively. For low system loads the delay distribution can be described as a monotonically-decreasing function with a very sharp initial

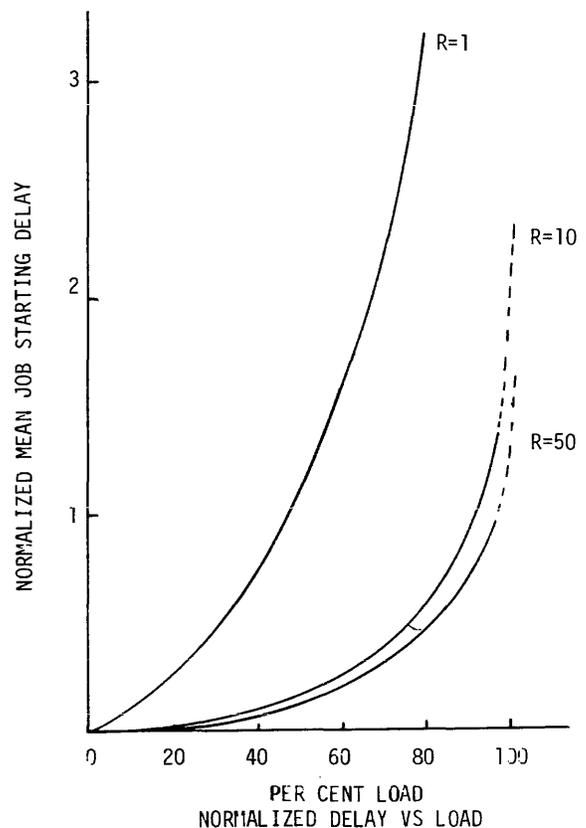


Figure 2

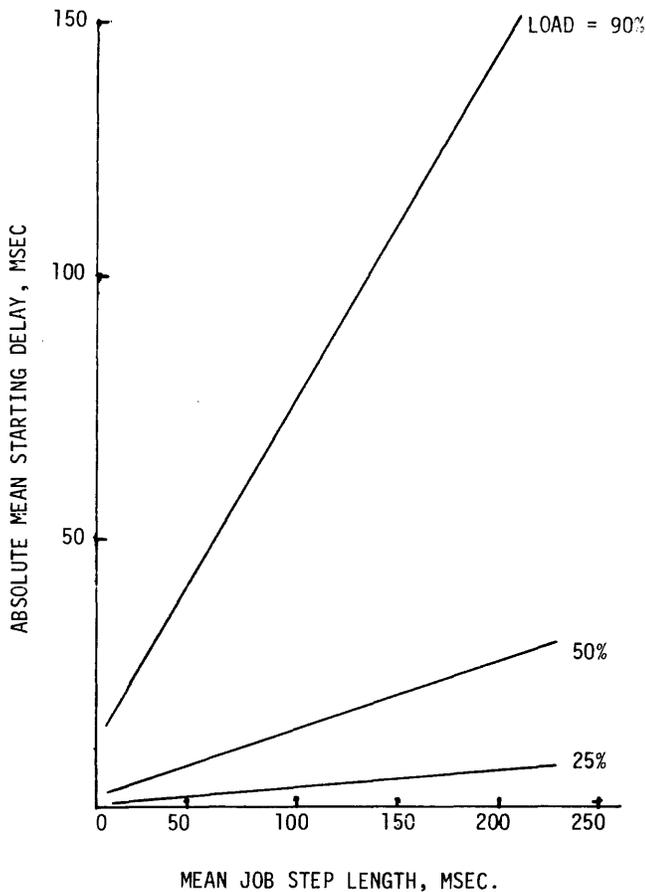


Figure 3

decline, indicating that a large percentage of jobs start without an appreciable delay. These characteristics are mostly retained for loads approaching 90 percent. Beyond that, however, the function shape changes considerably and for 99 percent load the probability function, in fact, increases slightly before gradually decreasing with a long tail. A monotonically-decreasing function implies that the probability of a job starting without a delay is higher than with any other value of delay. The distribution function for $R=50$ is slightly more complex as shown in Figure 5. The high load curves have three distinct phases; a rapid initial decline followed by a plateau and then a final asymptotic descent.

Let us turn our attention now to results regarding maximum job-starting delay. In principle, the probability of any delay, no matter how large, occurring in practice is non-zero. That is, maximum delay is always infinite. However, it is instructive to take a look at various percentiles of delay. Figure 6, for example, shows for $R=1$ the 80 percentile delay as a function of the system load. That is, 80 percent of all delays fall below this function. This function has the same characteristics as the mean delay of Figure 2. The 80 percentile delay doubles, for example, as the load is increased from 50 to 75 percent. The 99 percentile curve, however, has a slightly different nature. It is seen that the same increase in the load factor results only in a 50 percent

increase in the 99 percentile delay. This has some important consequences. For example, the penalty to be paid in going from a low to a high load factor operating point is not as great as the mean delay function implies. Figure 6 also shows points corresponding to the 99.99 percentile curve. The scattering of points is due to the small number of samples obtained. At any rate, the function tends to show a linear behavior in the 5 to 95% load range. In other words, cost incurred in terms of worst case delays increases only linearly with the system load. This has also been found to be true for the higher values of R .

Figure 7 shows the probability of the job-starting delay exceeding a given value as a function of the system load for $R=1$. It is seen that this probability approaches a linear function for zero delay. That is, the percent of jobs that start right on time decreases linearly with the load. This figure indicates that large system loads may be quite acceptable in real-time systems if moderate delays are permitted. Notice, for example, that if a 51 msec starting delay is permissible, then a system load of 85 percent will produce delays in excess of 51 msec only one percent of the time.

The results discussed so far have been obtained with the First-Come-First-Serve job dispatch strategy. It can be shown with the help of an example that by using the shortest-job-first strategy the mean delay can be reduced, while the spread of the delay is widened.⁵ Simulation results shown

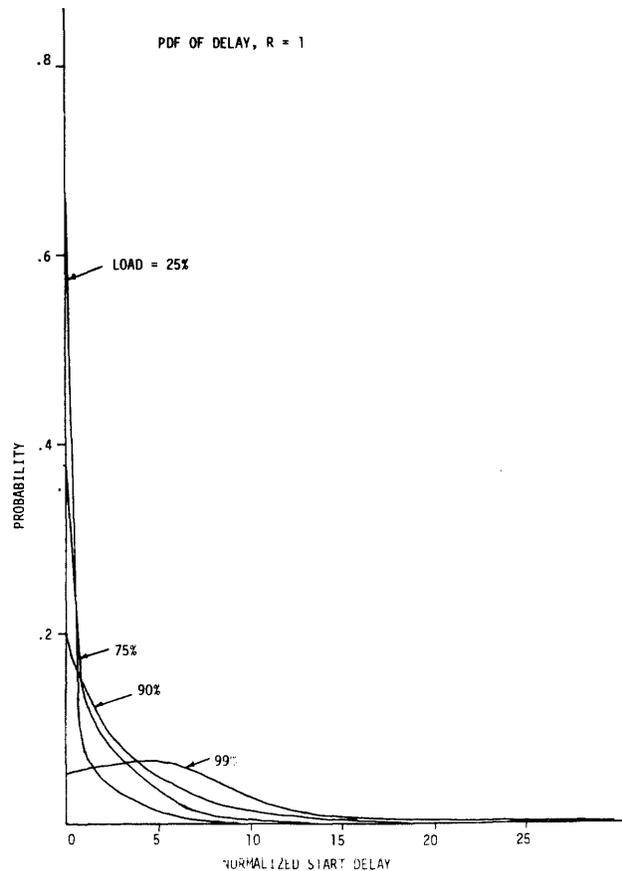


Figure 4

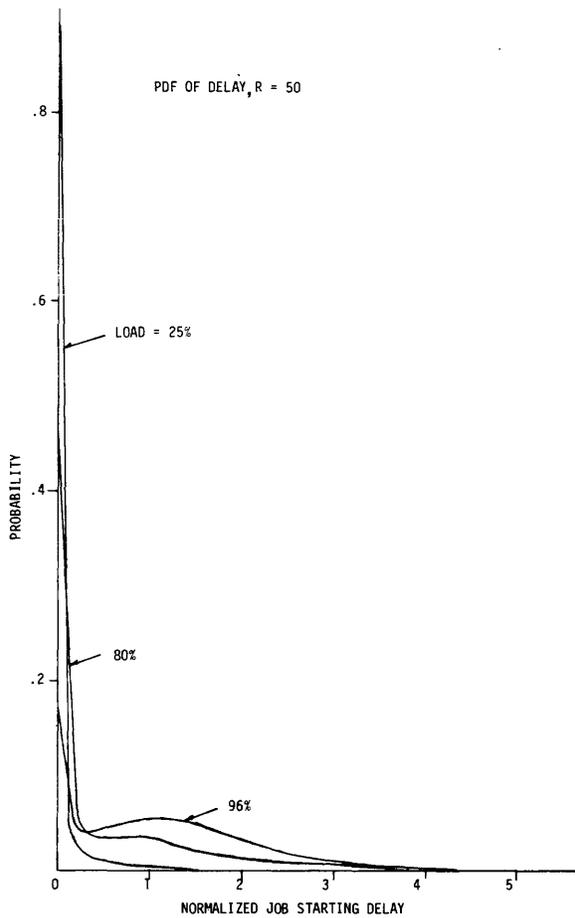


Figure 5

in Figures 8 and 9 confirm this. Figure 8 compares the normalized mean job starting delay for $R=1$ for the two strategies as a function of the system load while Figure 9 shows the PDF for 50 percent load for the same value of R . It is seen that the delay for some jobs is reduced, thereby increasing the probability of short delays. However, this is done at the expense of the longer jobs, thereby increasing the probability of longer delays. The range of delays is increased considerably while the mean delay is reduced only slightly. Also for $R=1$ the processor time devoted to the executive is up to two percent more for SJF than for FCFS.

One other variation of the FCFS strategy is the adaptive FCFS algorithm. In this case the starting times of jobs are adjusted dynamically based on past performance measured in terms of job start delay. The idea is to find a time slot that best suits each job and creates the least amount of interference with the other jobs. This strategy shows some improvement over the FCFS strategy. However, in applications such as ours where it may be necessary to certify that a certain subgroup of jobs will never be delayed more than a given amount, neither the FCFS nor the adaptive FCFS scheduler are particularly good candidates. Therefore an interrupt driven scheduler was also simulated. The next section describes the results of adaptive and interrupt driven schedulers which were evaluated using simulation.

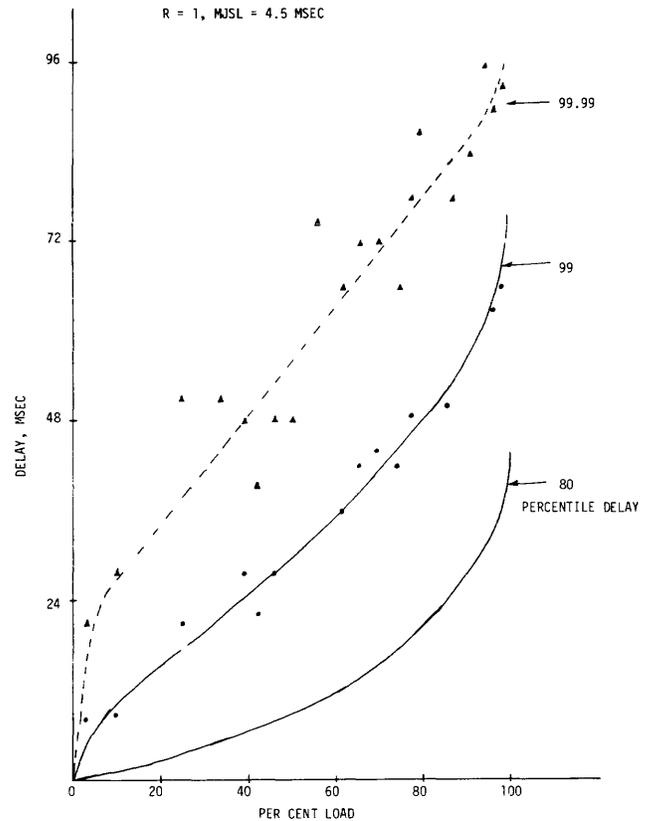


Figure 6

FCFS scheduler ("B" scheduler)

The *B* scheduler algorithm is an implementation of a FCFS within-priority-class method for determining which task a triad is to select. The set of all eligible tasks is divided into four priority classes, according to the criticality of the func-

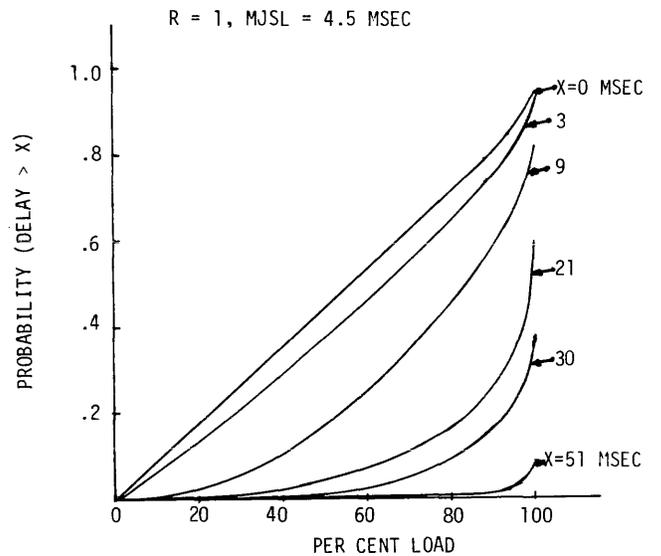


Figure 7

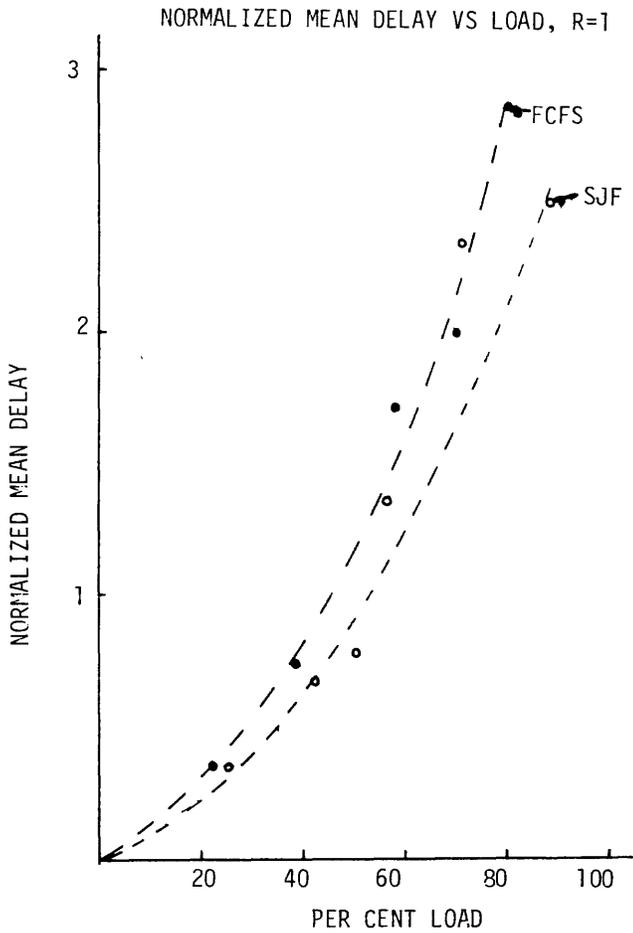


Figure 8

tion to be performed by the task. The criticality of the task determines its dispatching priority. The adaptive model uses delay information feedback in order to minimize job-starting delays. The result of this feedback is a modification of the next scheduled arrival time of the previously delayed task. If the starting delay for the previous iteration of a task exceeds a certain minimum value, the next iteration is delayed by a fraction of the inter-arrival time. The adaptive technique as implemented was stable, and led to significant reductions in mean starting delays for all tasks. However, because of the lack of pre-emptability, all tasks must execute for only a short period of time, or high iteration rate jobs will not be executed often enough. This and other weaknesses of the algorithm would preclude its use, even if the delay and jitter could be controlled.

Preemptive scheduler ("X" scheduler)

This model incorporates a preemptive interrupt-driven scheduling algorithm. Interrupts are periodically generated to cause activation of high-priority tasks and suspension of low-priority tasks. All tasks are assumed to be periodic, executing at rates of 5, 20, or 40 iterations per second. The

high iteration rate tasks are of higher priority than the lower-rate tasks, and intra-group dispatching constraints can be specified.

Results

In this section, results of three experiments are presented. In each experiment, the architectural parameters were held constant and the job mix used in each experiment was also identical. The experiments consisted of executing 1000 tasks, using the following test configurations:

1. The basic *B* model, not adaptive.
2. The adaptive *B* model.
3. The preemptive (*X*) model.

The 1000 job simulation corresponded to approximately 3.25 seconds of FTMP execution time. Based on an analysis of the time history output it appears that the system reached a steady-state condition within this time period.

The statistics of primary interest in this simulation are related to the job-starting delays. Because the jobs are used to implement a sampled data control system, the jitter is a

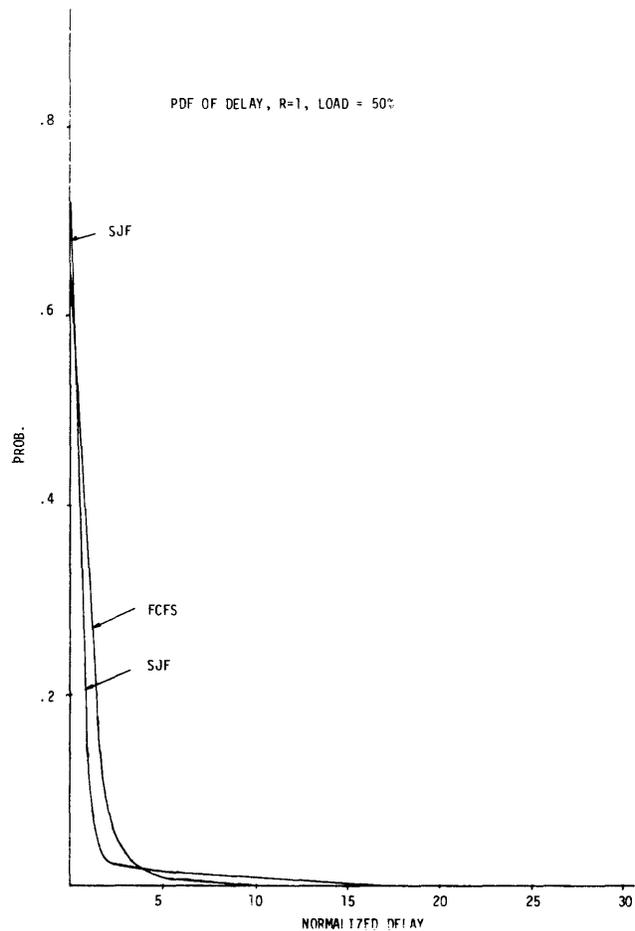


Figure 9

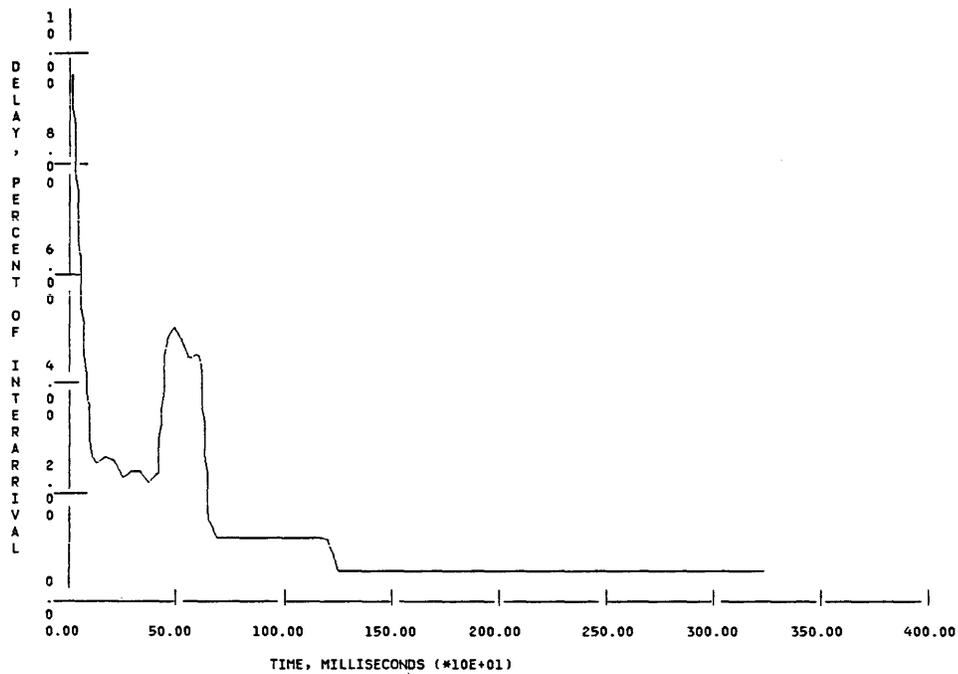


Figure 10—Adaptive FCFS scheduler, autopilot task.

critical parameter with respect to the overall system performance. The measurement of jitter in these models is the standard deviation of the job-starting delay. A relatively long delay which is constant between iterations is preferable to a large deviation between iterations. The absolute delay will not affect the performance of the control system as long as it is constant between update loops.

A time history of delay for sample jobs is presented in Figures 10-13. The delay, normalized by interarrival time, is plotted versus time. These plots represent the envelope of the set of discrete delay data points, and illustrate the changes in delay from one iteration to the next for a particular job.

Figure 10 is the delay plot for an example job, dispatched

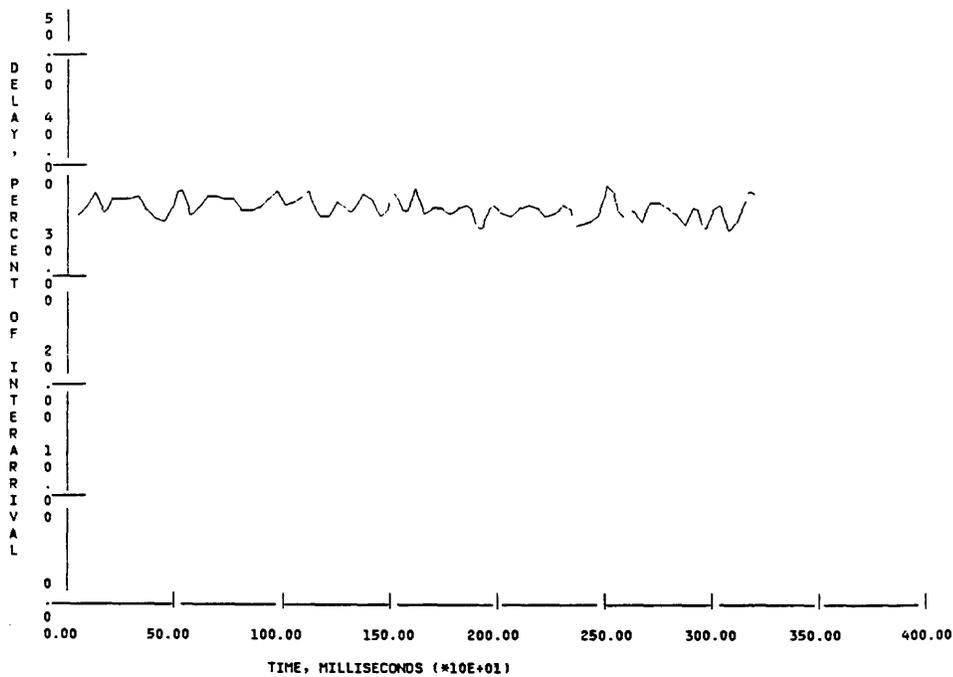


Figure 11—Preemptive scheduler, autopilot task.

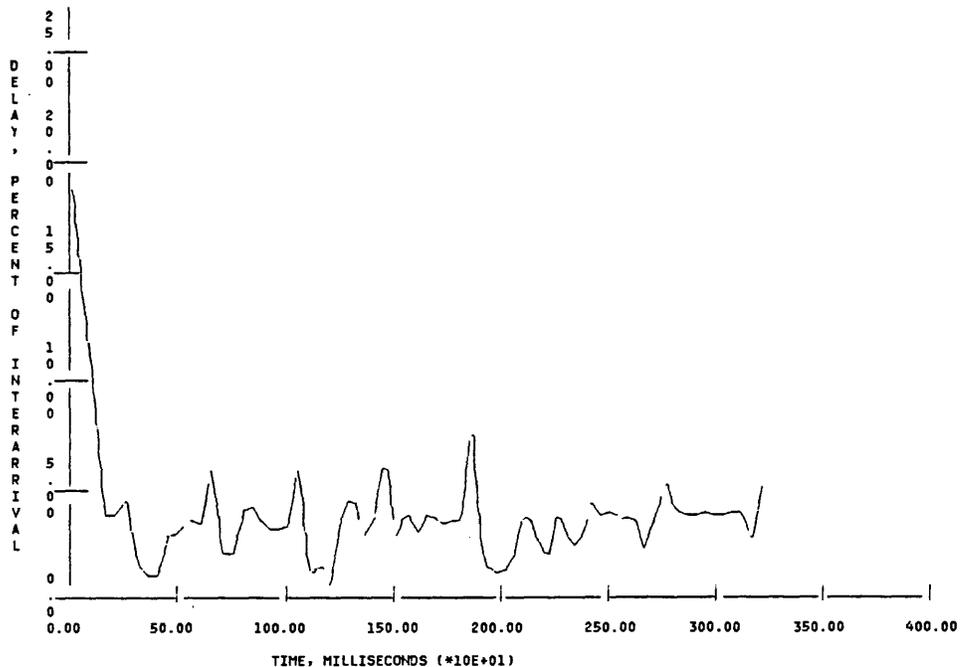


Figure 12—Adaptive FCFS scheduler, NAV and guidance task

by the adaptive FCFS scheduler. Up until approximately 1.25 seconds, there is a relatively large variance in the delay between iterations. This phase corresponds to the time period in which the delay feedback mechanism is active. After this initial phase, the feedback cannot provide any further improvements, and the delay reaches a steady-state value. For this task, the jitter is reduced to near zero. For other

tasks, however, a periodic change in delay is observed, because there is no interference-free time slot that can be found for each and every task.

Figure 11 is the delay plot for the same sample job, but dispatched by the preemptive scheduler. The delay is relatively constant throughout the simulation, but at a higher level than the FCFS delay. For this particular task, the

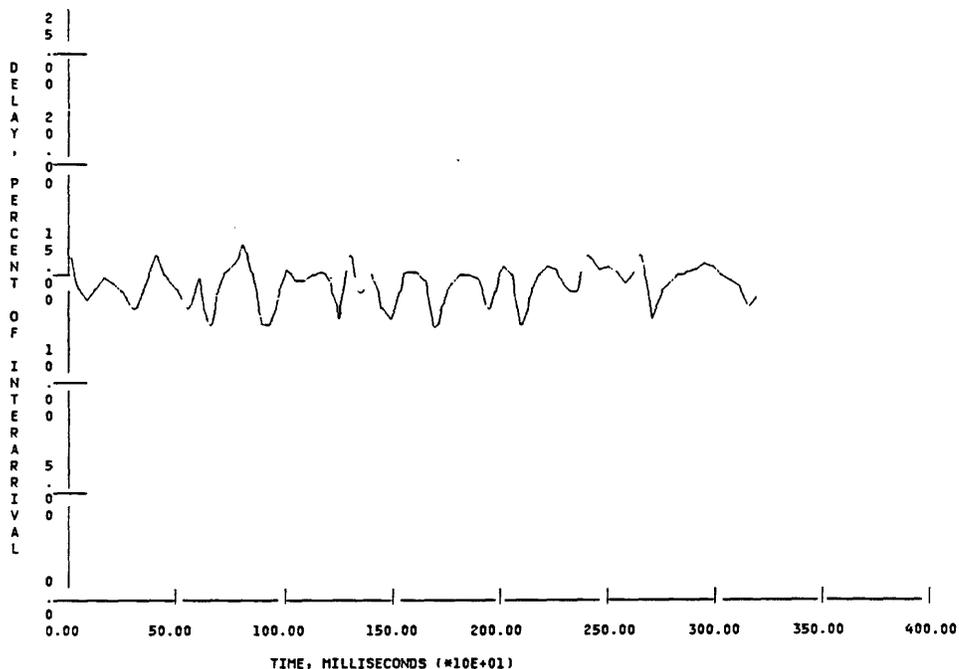


Figure 13—Preemptive Scheduler, NAV and guidance task.

adaptive FCFS model yields better performance than the preemptive model.

Figures 12 and 13 are the delay plots for another job, using the adaptive FCFS and preemptive schedulers. In this case, the preemptive model yields less jitter, both in the initial phase and in the steady-state phase of the simulation.

The three strategies have been compared with respect to performance of individual jobs. Table I lists the mean delay and jitter for each task for each of the three strategies. The following results were obtained with respect to aggregate performance.

The non-adaptive *B* model had a mean delay of 6.2 milliseconds, and the adaptive *B* model had a mean delay of only 1.5 milliseconds. However, the mean jitter was approximately 6.9 milliseconds in the adaptive model and was only 2.4 milliseconds in the non-adaptive model. Thus, the delay information feedback scheme used in the adaptive scheduler reduced the mean delay, but at the expense of the jitter.

In the adaptive experiment, the average jitter is approximately 6.9 milliseconds; in the preemptive experiment, it was approximately 1.2 milliseconds. Therefore, with respect to jitter, the preemptive model yields better performance on an aggregate basis.

Performance conclusions

Three major scheduling strategies *viz.*, FCFS, adaptive FCFS and preemptive schedulers have been evaluated. Each has certain advantages, and a selection must be made depending on the applicable performance criteria for the intended application. The FCFS strategy is a simple, straightforward algorithm that is suitable for most real-time applications. With this scheduler, high load factors can be maintained without making maximum jobstart delays inordinately high. The adaptive version of this algorithm can decrease the average delays, at the expense of being slightly more complex to implement. FCFS and adaptive FCFS schedulers can be used in those applications where the maximum delay can be allowed to exceed a certain value, say, only 10 percent of the time. Neither algorithm, however,

guarantees that the maximum start delay for a job will never exceed a given value. In those applications, such as ours, where this performance guarantee is of paramount importance, one must give up the simplicity and ease of verifiability in favor of guaranteed performance. In light of these requirements, the interrupt driven preemptive scheduling philosophy has been selected for the FTMP.

MULTIPROCESSOR RELIABILITY

The likelihood of a catastrophic failure of a central computer in a civil transport aircraft is required by the Federal Aviation Administration to be "extremely improbable." This requirement has been interpreted to mean a failure probability of the order of magnitude of 10^{-9} per hour. There are a number of ways in which random failures can result in a catastrophic failure of the FTMP. For the sake of analysis, these have been grouped under three categories as follows.

Lack of perfect coverage

In the FTMP some time is required to detect, isolate and recover from any failure. Depending upon the subtlety of the failure this time may range upward from several milliseconds. During this time the system is exposed to a second failure which may arrive in such a place as to be catastrophic. It may be mentioned here that by design there is no single point failure mechanism in the system. It takes two or more failures to bring the system down. Given that there is one fault in the system, the probability of successfully recovering from the failure is not 100 percent. This lack of unity coverage is the first failure mode.

Exhaustion of Spares

When a failure is detected in a module, that unit is replaced by a spare, if one is available. A succession of failures

TABLE I—Job Starting Delay Statistics

JOB NO.	PRIORITY (GROUP)	FCFS		FCFS ADAPTIVE		PREEMPTIVE	
		MEAN (μ sec)	STD DEV (μ sec)	MEAN (μ sec)	STD DEV (μ sec)	MEAN (μ sec)	STD DEV (μ sec)
01	005	17,160	401	8,963	2,478	36,844	771
02	005	26,371	3,476	4,308	8,448	35,633	809
03	005	31,777	460	9,102	7,040	32,617	1,108
04	005	33,271	1,318	10,940	7,628	22,236	553
05	005	36,382	803	8,154	10,265	20,115	397
06	020	4,904	212	675	803	18,087	568
07	020	6,355	337	1,143	1,019	8,519	307
08	020	9,250	194	1,948	1,382	7,244	499
09	020	15,029	616	1,715	2,440	5,390	226
10	040	375	250	480	588	4,908	297
11	040	758	491	647	892	3,188	201
12	040	1,133	689	692	716	1,227	69
13	040	4,030	342	757	697	889	56
14	040	4,577	534	698	760	601	42

can cause the system to degrade to a point where it cannot keep up with the minimum workload critical to the safety of the flight. This second failure mode is the result of a lack of throughput and/or memory capacity.

BGU enable mode failures

In the FTMP, there are two Bus Guardian Units (BGU) in each LRU that allow each processor, memory, and oscillator within that LRU to transmit on the memory and I/O buses. Both BGUs must be in agreement before any module may transmit on a bus. When a module fails, the BGUs can be commanded to disable the failed module from transmitting on the bus. It is evident that if the two BGUs fail to perform their duty when a module in their LRU fails, a number of buses may be rendered useless, thereby causing a system catastrophe. This is the third failure mode.

Figure 14 shows the system failure probability as a function of time due to each of the three failure modes described above.^{1,4} Figure 15 shows the composite failure probability of the FTMP due to all random failures. The following conclusions can be drawn from these two figures:

1. During a typical commercial flight of one-to-ten hours, the most likely threat of the FTMP failure is due to an arrival of two failures so close that system reconfiguration is not possible. The probability of this event, however, is acceptably low (about 3×10^{-10} per hour) because of high component MTBFs and fast reconfiguration times.
2. There is very little chance that the FTMP computer will run out of spares during a ten-hour flight, assuming that the system initially has all ten LRUs fully operational. In longer flights, however, failure would be quite possible, as evidenced by the sharply rising failure probability curve after 50 hours.
3. Finally, the system failure rate due to BGU enable mode failures is substantially lower than other system failure modes. Therefore, it does not contribute significantly to the overall system failure probability.

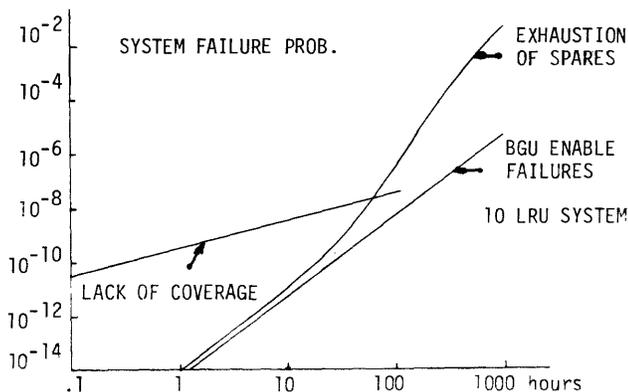


Figure 14

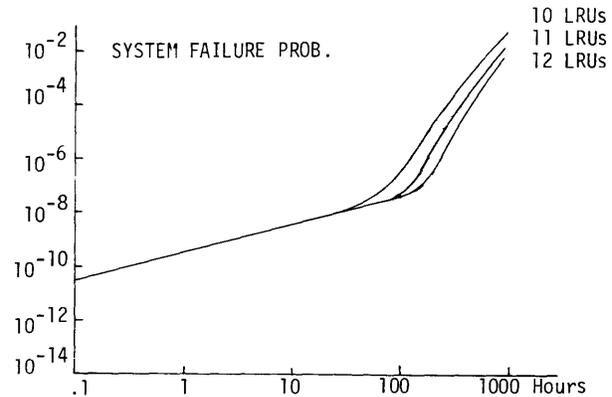


Figure 15

FTMP ECONOMICS

The viability of the FTMP depends not only on its technical merits but also on its cost effectiveness. For the FTMP to be economically attractive to the airframe builders and airlines, the direct and indirect benefits such as fuel savings and improved safety should be shown to be commensurate with its cost of ownership. The next two subsections explore these two facets of the FTMP economics.

Life-cycle-costs

The cost of ownership of the FTMP can be measured effectively in terms of its life-cycle-cost (LCC). The LCC includes all the cost items associated with the computer over its total life time. It can be broken down into the following four major categories: (1) acquisition cost, (2) maintenance cost, (3) spares inventory cost and (4) added fuel cost.

Each of these is self-explanatory except for the last item. The added fuel cost is the cost of extra fuel that must be burned to carry the computer on-board. Each of these cost items is determined by a number of different variables. For example, one of the variables that affects the acquisition cost is the initial purchase price, which in turn is determined by hardware cost, software development cost, certification cost, company profit, etc. The number of variables determining LCC is increased even further when normalized costs such as per flight-hour cost or per passenger-mile cost are considered. The normalized costs can be directly compared to airlines' present operating costs since all the data reported to the Civil Aeronautics Board by the airlines is normalized by flight-hours, block-hours or passenger-miles. Using the empirical formulas developed by the Boeing Commercial Airplane Company⁶ for cost estimation, and with a reasonable estimate of input variables, the FTMP absolute and normalized costs were computed. These costs are summarized in Tables II and III, respectively. According to these figures, it would cost \$138,500 to acquire one FTMP unit and each spare LRU would cost about \$13,000. An airline with a fleet of 100 airplanes, each equipped with two FTMPs, would need to maintain a spare inventory of 112

TABLE II—FTMP Absolute Cost Summary
(\$)

LRU Cost	12,960
Unit Hardware Cost	135,861
Software Development Cost	1,340,000
Unit Software Cost	2,680
FTMP Cost	138,541
Spare Inventory	112

LRUs. The reason for having two FTMPs (connected through an I/O node network) per aircraft is to be able to survive localized physical damage to either computer.⁷ The normalized cost of operating two FTMPs is found to be \$37 per flight-hour. Most of this is about evenly split between acquisition (\$17) and maintenance (\$16) costs. The normalized costs of carrying the added weight and maintaining the spare LRU inventory are found to be negligible in comparison to the other two components.

To put these numbers into proper perspective, the cost of operating a jetliner as reported to the Civil Aeronautics Board is itemized in Table IV. The numbers shown are the flight-hour costs (1974 dollars) of operating a Boeing 747, averaged over all the U.S. carriers operating such an aircraft. The total cost is \$2708 per flight-hour or \$2415 per block-hour. The largest single component of this is the fuel and oil at \$795. By comparison, the additional cost of operating two FTMPs would be \$37 per hour. This is about five percent of fuel expense or 1.5 percent of the overall 747 cost. Thus, if the FTMP can help make the aircraft just five percent more fuel efficient it can pay for itself. This and other FTMP benefits are discussed in the next section.

As stated earlier, a large number of variables are involved in this study. Uncertainties in the estimation of values of these variables can lead to erroneous cost figures. However, not all the variables are equally important in that the FTMP cost is not equally sensitive to all the parameters. Variations in some parameter values have direct impact on daily operating costs of the FTMP, while others have none. A sensitivity analysis found that only four items have important bearing on the FTMP cost. Table V lists the percent change in the hourly operating cost of the FTMP due to a 100-percent change in each of the four variables. Not surprisingly, the LRU reliability and repair time have a direct impact on the maintenance cost, which is one of the recurring cost items. The parts and labor cost to build the FTMP are the major factors in determining the FTMP acquisition cost. Lastly, the FTMP life-time is important because it is used to normalize all the fixed costs to obtain the hourly operating cost. The effect of all other factors is five percent

TABLE III—FTMP Flight-hour Cost
Summary (\$/hr)

Total Cost	37
Acquisition Cost	17
Spare Inventory Cost	1
Added Weight Cost	3
Maintenance Cost	16

TABLE IV—Boeing 747 Operating Cost
(1974)
(\$/Block-Hour)

Flying Operations	1183
Crew	363
Fuel and Oil	795
Insurance	25
Maintenance	570
Depreciation and Rental	662
Total/Block-hour	2415
Total/Flight-hour	2708

or less. As an example of this, if the software development cost \$2.7 million rather than the \$1.35 million assumed, the hourly operating cost of the FTMP would be increased from \$37 to only \$39. On the other hand, if the LRU mean time between failure (MTBF) turned out to be only 925 hours rather than 1850 hours assumed, the hourly cost would increase from \$37 to \$46.

FTMP benefits

As pointed out in the previous section, the hourly operating cost of two FTMPs is less than five percent of just the cost of fuel burned per hour by a Boeing 747. The potential of the FTMP to make future aircraft fuel efficient is enormous. The performance advantages of statically-unstable aircraft where the tail shares the wing load, rather than opposing it, are well known. Similarly, load alleviation can be used to build a lighter wing structure. All of these applications, however, require some form of active controls, the failure of which can have financial and/or fatal consequences. Lockheed is already flight testing an advanced version of the L-1011 Tristar with increased wingspan supported by active wing tip ailerons. The wing fatigue life in the absence of active controls would be only 40 hours, compared to 30,000 hours or more for a conventional wing. The economic penalty of the control system failure would be catastrophic in this case. These are some of the areas where airframe builders can use a central fault-tolerant computer to advantage.

The next generation of jet aircraft will be making extensive use of digital computer technology. The Boeing 767, for example, will employ more than six different digital computers, ranging in functions from air data to flight control.⁸ By consolidating guidance, navigation, control and other functions into one single computer, there are obvious savings to be obtained in terms of commonality of design, implementation, spares, and maintenance overheads.

TABLE V—Effect on Flight-hour Cost

FTMP Life-time	24%
LRU and FTMP	
Parts Cost & Assembly/Test Time	43%
LRU MTBF	24%
LRU Repair Time	33%
All Others	≤5%

There are other recurring benefits of the FTMP to the airlines, such as fewer false removals of LRUs, which presently run at about 50 percent. A fault in the FTMP can be traced to the LRU level quickly and with almost 100 percent certainty. The FTMP can also extend the aircraft flight envelope by giving the airlines an all weather operational capability, such as Category III operations. There also are some intangible benefits such as increased safety. The general increase in automation can reduce the crew workload, which indirectly contributes to safety.

CONCLUSIONS

From the results presented in this paper, it can be predicted that the FTMP can do the job for which it is designed, and do it reliably and economically. Using the interrupt driven job scheduling strategy, it can respond to critical job requests promptly. It has a very low failure probability, and at the same time the hourly cost of operating two FTMPs in a transport aircraft can be as little as one-to-two percent of the total flight-hour cost of the aircraft. The benefits of the FTMP, in any case, far outweigh its costs. It is our conten-

tion, therefore, that the FTMP will be a viable and competitive option for the future generation of transport aircraft.

REFERENCES

1. Hopkins, A. L., Jr., T. B. Smith, III, and J. H. Lala, "FTMP—A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," *Proceedings of the IEEE*, Vol. 66, No. 10, October 1978.
2. Hopkins, A. L., Jr., and T. B. Smith, III, "The Architectural Elements of a Symmetric Fault-Tolerant Multiprocessor," *IEEE Transactions on Computers*, Vol. C-24, No. 5, May 1975.
3. Ratner, R. S. et. al., "Design of a Fault-Tolerant Airborne Digital Computer, Vol. II—Computational Requirements and Technology," NASA CR-132253, 1973.
4. Smith, T. B., III, et. al., "A Fault-Tolerant Multiprocessor Architecture for Aircraft," *NASA Contractor Report 3010*, Vol. I, July 1978.
5. Lala, J. H., "Performance Evaluation of a Multiprocessor in a Real-Time Environment," Ph.D. Thesis, Dept. of Aeronautics and Astronautics, MIT, February 1976, C.S. Draper Lab Report T-622.
6. Eberlein, A. J., and P. G. Savage, "Strapdown Cost Trend Study and Forecast," Honeywell Report for NASA Ames Contract NAS 2-8065, 1975.
7. Smith, T. B., III, "A Damage- and Fault-Tolerant Input/Output Network," *IEEE Transactions on Computers*, Vol. C-24, No. 5, May 1975.
8. *Aviation Week and Space Technology*, "767 Digital Avionics Stress Flexibility," September 4, 1978.

Serviceability features of the HP 300 small business computer

by CURTIS R. GOWAN

Hewlett-Packard
Santa Clara, California

INTRODUCTION

As computing system prices go down due to technological advances in hardware, more emphasis is being placed on the overall cost of ownership. To meet this challenge, serviceability features are designed into the HP 300—intertwined throughout the system hardware and software.

THE HP 300

The HP 300 is a broad-capability system for dedicated business data processing applications. It includes general-purpose features found on much larger systems, yet it is designed for direct use in daily business activity.

A typical HP 300 configuration consists of the System Unit (Figure 1), a printer and several application terminals.

The System Unit

The central element in every HP 300 system is the HP 300 Mainframe, or "System Unit." The System Unit combines the HP 300's processing, storage and control functions into a single compact package:

Integrated Display System—A keyboard and display screen form the Integrated Display System, or IDS. The display can be broken into multiple windows, each of which is directly attached to a data file. Eight "softkeys" bordering the right side of the IDS screen provide a push-button choice capability.

Processor—The HP 300 is based on a Silicon-on-Sapphire (CMOS/SOS) LSI processor. The processor has a stack architecture and provides hardware support for virtual memory operation.

Main storage—A standard HP 300 system includes 256 KBytes of semiconductor memory expandable to 1024 KBytes in 128 KByte increments.

Flexible disc drive—The HP 300 System Unit includes a flexible disc drive for offline storage and data exchange with other HP 300 systems.

Online disc storage—A fixed disc provides 12 million bytes of storage for system and user programs and data. It is

also used as secondary storage for virtual memory operation. In larger configurations, the fixed disc can be replaced by a stand-alone disc drive.

The 25-slot card cage in the System Unit contains not only the CPU, memory and I/O boards but also circuit boards for the console (IDS), flexible disc and 12 MByte integrated system disc. Except for the circuitry which must be near either the display tube, the disc mechanisms, or the power supply, every mainframe PC board is in the System Unit card cage—with all configuration switches accessible on the board edges. This simplifies diagnosis by making it very easy to check the mainframe's hardware configuration for human error.

Amigo/300 operating system

The HP 300's operating system, Amigo/300, is an advanced virtual memory operating system, with extensive data management and online processing facilities. In dedicated, terminal-oriented applications, Amigo/300 supports terminal response and interactive processing in addition to concurrent non-interactive jobs.

Some of the key features include:

- Multiprogramming
- Multitasking
- Large addressing space—A potential addressing space of over 260 million bytes for each program.

CLASSES OF SERVICEABILITY FEATURES

Serviceability features exist either to contain faults or to diagnose them.

- Fault containment features correct or detect faults to minimize their effect on the system. The next section covers these features in detail.

The diagnosis features can be split into two sets—Monitoring tools and stimulus-response tools.

- Monitoring tools (fourth Section) allow you to see what

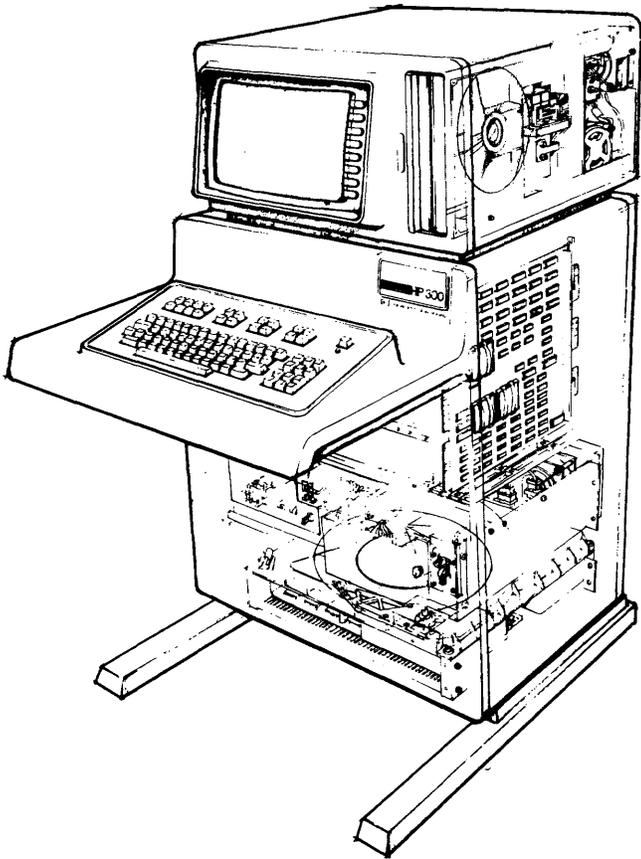


Figure 1—HP 300 system unit.

is happening or what has just happened leading up to an error. The tools must disturb the system as little as possible.

- Stimulus-response tools, on the other hand, by their nature perturb the system under test by providing simple, controlled, repeatable stimuli instead of the complex stimuli provided by normal operation. These are covered in the fifth section.

FAULT CONTAINMENT FEATURES

The HP 300 hardware and software have built-in provisions which correct or detect certain faults when they occur. Whether the result of hardware failure or human error, these faults are of the type that cause multiple, obscure (and often delayed) system faults.

Error-correcting memory

The memory subsystem corrects all single-bit memory errors; plus, it detects all double-bit and the vast majority of multi-bit errors. Each memory word consists of 16 data bits plus five Hamming-encoded correction/detection bits and a parity bit.

Privileged mode instructions

Programs may execute in one of two modes—privileged or user. In user mode, a program is confined to its own code and data areas and is prevented from destroying the operating system or directly performing I/O.

Instruction bounds checking

The CMOS/SOS LSI processor performs bounds checking on memory reference instructions, in both user and privileged code, via hardware limit registers. The executing user code segment is kept separate and distinct from the data segments.

Error-detecting disc subsystems

The disc subsystems record cyclic-redundancy error-detecting codes with the data; the controller and driver automatically retry if the data read fails the CRC check.

MONITORING TOOLS

Monitoring tools show what is happening—the current state—or what just happened—a state history. The key to the design of monitoring tools for diagnosis is that they be as non-invasive as possible, i.e., the tool should not affect the circuit under test.

The first group of monitoring tools are intrinsic to the HP 300—they are always present, monitoring the system during normal operation. The last two monitoring tools are extrinsic—they are external instruments connected to the system when a fault is suspected.

Trace-LEDs—HP-manufactured light-emitting diode arrays (with integrated current-limiting resistors) are built into most boards. All of these LEDs are observable by opening the rear door. They provide non-invasive monitoring of normal activity—to check certain key nodes without tools. The Trace-LEDs are, in effect, a low-cost built-in maintenance panel. For example, one LED is connected to the last stage of the CRT timing chain; its once-per-second blink shows that the chain is working. Another LED is connected to the system power-fail-warning line; this circuit is checked by turning the system on and off while watching the LED.

System Error Log—The error logging facility records errors, both user-transparent and fatal, as time-stamped entries in a disc file. The number of I/O requests and errors are tallied for each device, then regularly recorded on the disc. Fault trends may be detected by analyzing this data.

System Trace Table—The operating system keeps a list of the most recent events in a circular table in memory. The data can be dumped via the System Debug facility.

Console Log—A list of the most recent console commands is kept in a disc file so that one can check the sequence of commands which led up to some event.

System Debug—Running under the operating system, System Debug provides to Specialist CEs interactive control, examination and modification of both code and data.

There are two key external tools which Customer Engineers use in monitoring—the Processor Maintenance Panel and state analysis instrumentation.

Processor Maintenance Panel—The Processor Maintenance Panel (PMP) is used by Specialist CEs for detailed tracing of HP 300 microcode and macrocode. The PMP consists of a printed circuit board which plugs into a reserved slot in the system card cage and an HP 9825 Calculator which provides a user interface independent of the system under test. The PMP provides non-invasive breakpoints plus micro-level and macro-level manipulation of the system hardware.

State Analysis Instruments—HP 300 Customer Engineers are trained in the use of the HP 1602A Logic State Analyzer and the HP 1640A Serial Data Analyzer—both standard portable instruments—to trace I/O transactions on the system busses and data communication ports.

STIMULUS-RESPONSE TOOLS

The key to the design of stimulus-response diagnosis tools is that they not only identify that a problem exists but also

isolate it to a replaceable module. To do this, tools must:

- Depend as little as possible on portions of the system other than the portion under test.
- Build outward from a small kernel, layer by layer, using already-tested circuit blocks to test additional circuitry.

Figure 2 gives a block diagram of the system hardware. This diagram is then annotated to show the outward progression of the stimulus-response tools. The layer-by-layer sequence of tests starts at the processor, directly attacking the diagnostics-won't-load problem.

CPU Self-Test

The CPU Self-Test is a ROM-resident microdiagnostic which is invoked on power-on or by depressing a switch on the edge of the CPU Bus Interface board. Its purpose is to check the hardware required to cold-load the system. CPU Self-Test results are displayed on two five-LED arrays next to the CPU TEST switch. The test may be set up to loop continuously, either halting on the first failure or continuing regardless of failure.

The CPU Self-Test first performs a thorough test of the processor, including checksum tests on each of the micro-code ROMs. It then tests the Inter-Module Bus handshake lines, checks communication with the Memory Controller

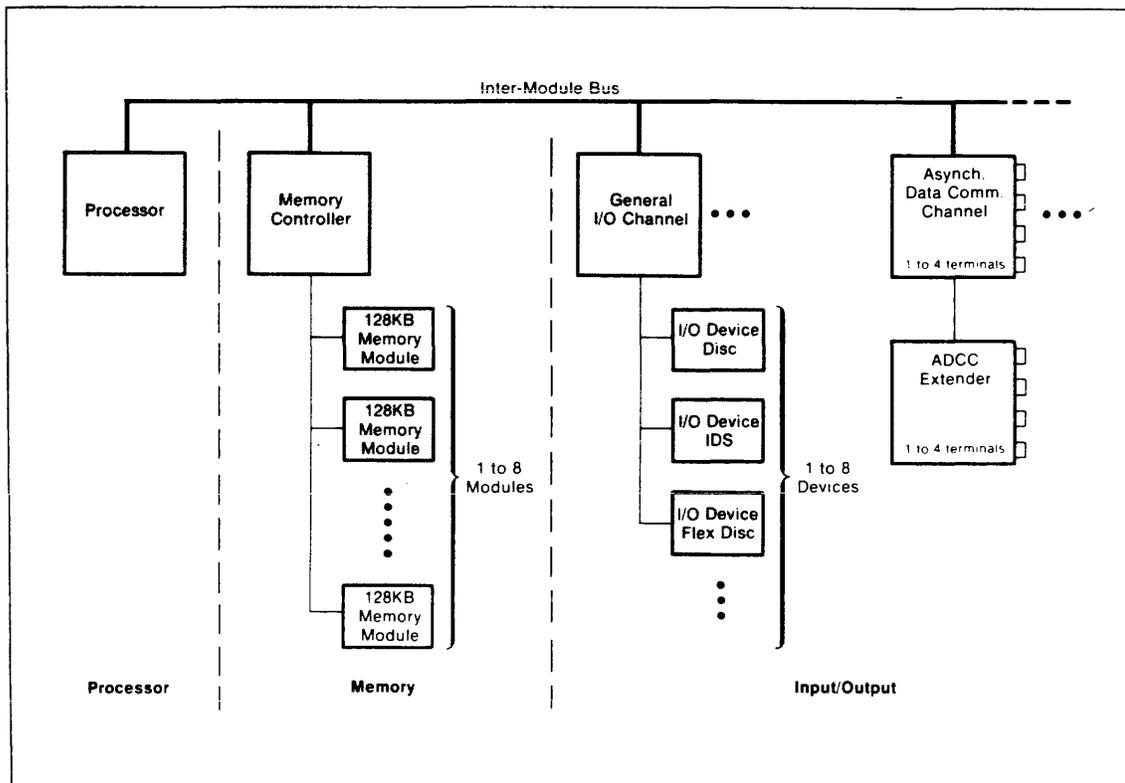


Figure 2—HP 300 logical hardware organization.

and tests the lowest memory module for stuck bits. The General I/O Channel (GIC) is tested—first proving that the CPU can communicate with the GIC, then looping data completely through the board to the output of the LSI device-bus interface chip and back to the memory via DMA.

The device to test is selected by the Control Panel switch used to select the load/dump device. The CPU Self-Test sends a data pattern through the channel to the device controller's memory, commands the device to return the data, and then checks for correctness.

When the Control Panel is set up for a cold-load, the CPU Self-Test checks and diagnoses the entire data path from disc to CPU. By changing the Control Panel switches, the loopback process is also used to check out the links to the Interactive Display System, printers and other peripherals. See Figure 3.

Fixed Disc Self-Test

The built-in 12 MByte fixed disc has its own independent Self-Test. Actuated by a switch on the disc Controller board, it loops and displays errors in the same manner as the CPU Self-Test.

Key steps include a processor test, ROM checksums, device-bus interface chip loopback, actuator arm motion tests and extensive writing/reading on a reserved track.

Interactive Display System Self-Test

An independent self-test is also provided in the Interactive Display System (IDS). It checks for ROM checksums, device-bus interface chip loopback, keyboard scanning and stuck keys, character ROMs in wrong sockets, horizontal oscillator, and correct dot emission to the sweep circuitry. The operator checks the displayed fonts on the display screen.

Flexible Disc Unit Self-Test

A similar Self-Test is provided for the flexible disc unit (FDU). If a flexible disc is present, reading will be tested; if an additional switch is depressed at the start of FDU Self-Test, write/read testing is performed.

At this point in the stimulus-response test sequence, the kernel of the system has been tested by microcode (Figure 4). One can now load and run macrocode test programs.

Diagnostic/Utility System

A memory-based operating system, the Diagnostic/Utility System (DUS), provides file management of test and utility programs on a flexible disc. DUS provides a simple and deterministic base for software testing of the hardware.

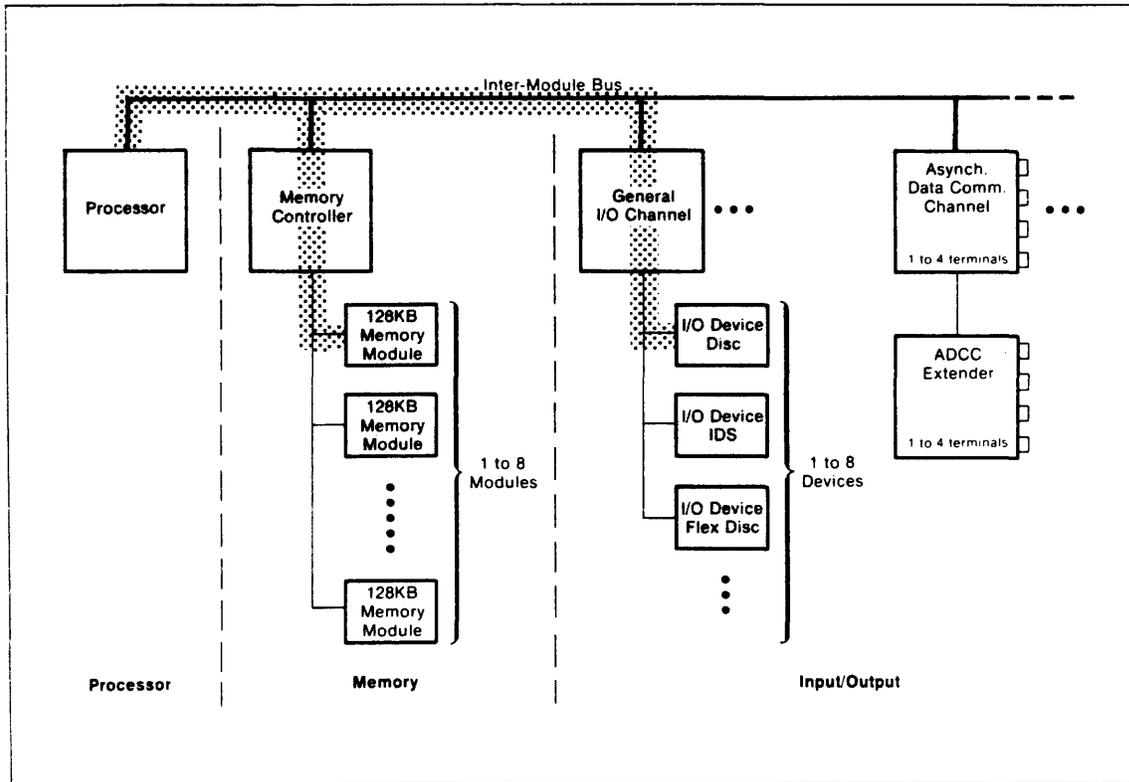


Figure 3—CPU Self-Test data paths.

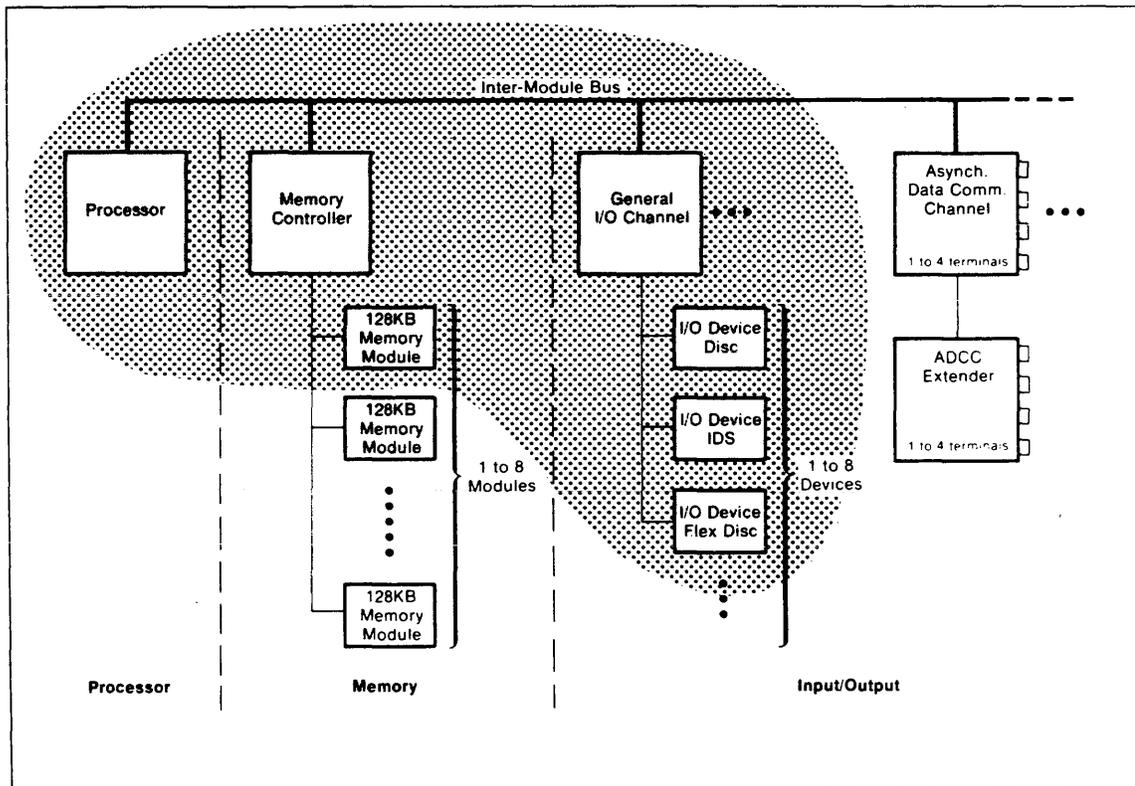


Figure 4—System kernel.

DUS is easily used by a CE or trained user to check system hardware. There is a HELP capability which lists the directory and commands; a program is invoked by simply typing its name.

Processor test program

This program checks certain software-related CPU functions not tested in the CPU Self-Test.

I/O map program

The IOMAP program uses hardware self-identification features to display the type and address of each channel and device in the system. Any number of Identify, Loopback, or Self-Test commands may be sent to any device—for example, a printer's self-test can be invoked to check out both the I/O path and the device.

Device diagnostic and test programs

There is a diagnostic or test program for each device. These are easily used by a CE or trained user—each has step-by-step prompt messages and a default mode which consists of a fast, non-destructive subset of the overall test sequence. If the user suspects a disc problem, for example,

the default mode of the disc diagnostic program runs a subset of the test menu which will give a good confidence check of the disc drive without damage to the user's data base.

Users with more time and knowledge can run these tools at the CE level—this is particularly valuable to OEMs and software houses incorporating the HP 300 into their own products.

SUMMARY

The HP 300 has been described in an overview. The product's serviceability features are broken into three groups—fault containment features, monitoring tools and stimulus-response tools. Each specific feature has been described in detail sufficient to show its role in reaching the overall goal—low cost of ownership for a small commercial system.

* * *

ACKNOWLEDGMENT

Like quality, serviceability cannot be added to a product by an outside group. The design of serviceability into the HP 300 was intertwined with the system design process—thus, the features described are the result of contributions by every member of the project.

Automatic tuning of computer architectures

by KEN SAKAMURA, TATSUSHI MOROKUMA and HIDEO AISO

Keio University
Yokohama, Japan

and

HAJIME IIZUKA

Electrotechnical Laboratory
Tokyo, Japan

INTRODUCTION

One of the most important unsolved problems in the design of a computer system is the automatic optimization or tuning of the computer architecture to better suit the problem under consideration. In particular, it is very important to make an effective mapping of the structure of the problem to be solved to the structure of the computer being used.

The best approach to this problem is to automate the tuning procedure of the computer architecture by the computer itself, in which the computer evaluates its performance and thereby achieves better performance. This seems to be the best approach because the existing computer systems have become too large and too complicated to be handled by manual tuning techniques. It should be noted that tuning iteration is carried out automatically at the Instruction Set Processor (ISP) level utilizing the inherent characteristics of dynamic microprogramming.

The purpose of this paper is to present a detailed algorithm for an automatic tuning of computer architectures using dynamic microprogramming techniques and to show the effectiveness of the proposed ideas by describing the results of some experiments. We have been doing research on this problem since the late 1960s.¹ Very little research has been conducted with regard to the automatic tuning of computer architectures.²⁻⁵

PRINCIPLES AND IMPLEMENTATION MECHANISMS

Basic principles

A simplified model of a mechanism for automatic tuning of a computer architecture at the ISP level is shown in Figure 1. The basic functions required for the optimization of the architecture or tuning are as follows:

Monitoring of programs

Detailed information regarding the dynamic characteristics of both the computer and the program to be solved is necessary to perform the optimization processes. This information must include the relative frequencies of machine instructions, the relative frequencies of sequences of instructions, that is serial dependencies, and the relative frequencies of address and data values. A monitor implemented in hardware, software or firmware collects this information.

Analysis of data

The information obtained from the monitor is processed by an analyzer which may be an independent computer. The function of the analyzer is to identify all possible candidates for instruction patterns that can be tuned up in order to create new instructions to save execution time and storage space. This is performed by a thorough analysis of the application program and its execution profile.

Feedback to computer

New instructions, that were synthesized by the analyzer, are converted into new microprograms which occupy less storage space and have faster execution time than the original instructions. The synthesized microprograms are loaded into the writable control storage (WCS) in the computer through the feedback path during the program execution to form a new enhanced architecture. This results in dynamic modification of the computer architecture and in better performance. It should be noticed that dynamic microprogramming techniques are very effective to realize this dynamic modification.

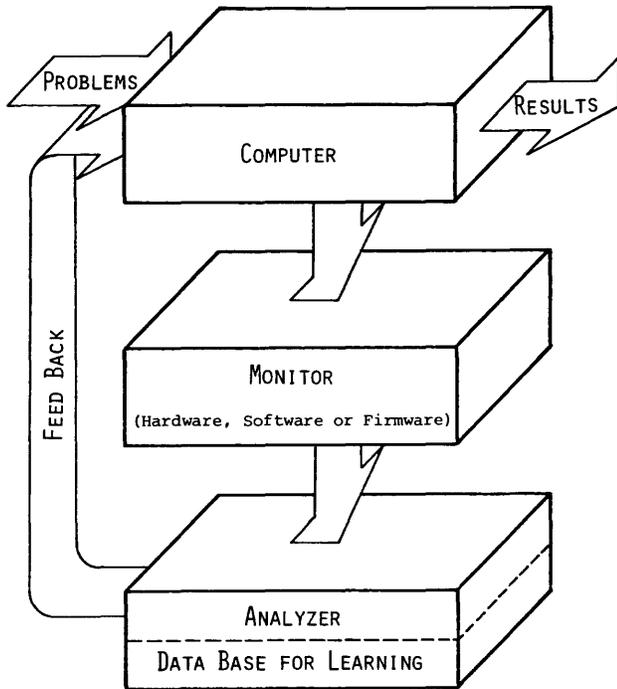


Figure 1—Model of an automatic tuning mechanism.

Learning of tuning behavior

A special feature of our method is the fact that a data base is provided in the analyzer. The above-mentioned tuning processes are usually repeated until the desired performance improvement has been achieved. Therefore, the analyzer keeps track of the tuning behavior in the data base whenever tuning occurs. The analyzer can refer to the contents of the data base so that the number of tuning iterations will be minimized.

Definition of terms

The terms used in the next two sections are defined as follows:

Machine

Each intermediate language (IML) instruction set defines a corresponding *machine*, M_i , which has an IML instruction set IML_i . IML_i is a set of instructions represented as follows:

$$IML_i = (I_1, I_2, \dots, I_k, \dots, I_n)$$

where I_k is an instruction which is a microroutine consisting of a sequence of microinstructions.

Program

A *program* is a sequence of IML instructions.

Block

In order to define a block, IML instructions have to be divided into the following two types:

1. Branch type—Instructions whose next instruction to be executed is not necessarily next one in sequence.
2. Sequential type—Instructions whose next instruction to be executed is always the next one in sequence.

A *block* is a static sequence of instructions which satisfies the following conditions:

1. Its entry point is restricted only to the top instruction.
2. The last instruction is of branch type.

Length of block

The length $\gamma(B)$ of a block B is the number of instructions contained in that block.

Instruction pattern

An instruction pattern (P) is a continuous sequence of instructions in a block. The number of different instruction patterns in a block is not greater than $\frac{1}{2}\gamma(B) \cdot (\gamma(B) - 1)$.

Length of pattern

Let the bit length of pattern p consisting of γ be $\gamma(p)$.

Weight of a pattern

The weight of a pattern p , $\xi(p)$ is defined as follows:

$$\begin{aligned} \xi(p) &= \sum_i^B \frac{\tau(p) \times f(p, i)}{T_0 \times \gamma(p)} \\ &= \frac{\tau(p)}{T_0 \times \gamma(p)} \sum_i^B f(p, i) = \frac{\tau(p) \times f(p)}{T_0 \times \gamma(p)} \end{aligned}$$

where

- $\tau(p)$: Execution time of pattern p
- $f(p, i)$: Number of times that pattern p is executed in the block i .
- $f(p)$: Number of times that pattern p is executed in all blocks.
- T_0 : Total program execution time
- $\gamma(p)$: Length of pattern p .

Improvement ratio

The capacity of writable control storage required to convert a pattern p into a microprogram may be thought of as the overhead of the tuning. It is represented by $\theta(p)$. Fur-

thermore, the improvement ratio ($\mu(p)$), indicating how much improvement of the execution speed has been achieved by tuning, is defined by the following equation:

$$\begin{aligned}\mu(p) &= \frac{T(p) - t(p)}{T_0} \\ &= \frac{\tau(p) \times f(p)}{T_0} - \frac{\tau'(p) \times f(p)}{T_0} = \frac{\tau(p) - \tau'(p)}{T_0} f(p)\end{aligned}$$

where

- $T(p)$: Total execution time of pattern p
- $t(p)$: Total execution time of pattern p after tuning
- $\tau(p)$: One execution time of pattern p
- $\tau'(p)$: One execution time of pattern p after tuning
- T_0 : Total program execution time
- $f(p)$: Execution frequency of pattern p in all blocks

Optimum design of an intermediate language based on dynamic computer behavior

It is well known that better execution efficiency may be achieved through description of the problem in a lower-level language. For instance, better performance can be expected when the problem is written in machine instructions rather than in high-level languages, and much better performance can be expected when the problem is written in microinstructions rather than in machine instructions.

The direct description of a problem in microinstructions is called "Firmware," and this results in the creation of new instructions suitable for the problem to be solved. The improvement of the execution efficiency is carried out at the ISP level, and this procedure is called "Architecture Tuning."

To achieve optimal efficiency, it is clearly desirable to implement all the programs in firmware. However, more than 80 percent of the execution weight is concentrated on at most four to five percent of the total number of program steps.⁶ Therefore, the best performance cost ratio is not achieved by making all the programs into firmware. Parts of the program should be implemented in firmware considering the dynamic behavior of the program execution. In this case, the best strategy is to apply microprogramming to those parts of the program, which are most frequently executed. In order to automatically detect the parts to be tuned up, the program is divided into blocks based on the proposed algorithm described in the next section. Next, the execution weight is measured to decide the blocks that have to be implemented in firmware starting from the block with the highest execution weight.

An algorithm for tuning of IML instruction patterns

An important consideration is that detailed information about the characteristics of programs is necessary to perform the tuning processes. Since the use of each IML instruction is not always uniform at the moment of program

execution, the relative frequencies of both machine instructions and sequences of instructions, that is, serial dependencies, must be included in this information. The sequences of pairs or multiplets of instructions, that create new instructions are called "Instruction Patterns." Thus, the programs can be expressed in terms of a finite number of instruction patterns by static analysis described below.

Weighting of the instruction patterns

In order to give a weight to each of the instruction patterns, the frequencies of sequences of instructions should be measured. In this case, no branch instruction should be contained in the instruction patterns. The program is divided into blocks separated from each other by branch instructions. The frequencies of the blocks are measured by a hardware or firmware monitor. The weight of the corresponding instruction patterns can be calculated from the measured frequencies of the blocks.

Selection of the instruction patterns to be tuned up

Many strategies have been considered to select the instruction patterns to be tuned up. We propose the following two methods:

1. Method to achieve maximum execution efficiency.

Using the equation for estimating the tuning effect, which will be described in the next section, we can estimate the performance improvement ($\hat{\mu}$) by means of the previously found instruction patterns.

First, we select the instruction patterns with the maximum value of $\hat{\mu}$. It turns out that the weight of the other instruction patterns may be changed, since some of the instruction patterns may be overlapping each other and some instruction patterns contain other instruction patterns.

Next, we carry out the static analysis again to weigh the instruction patterns. The value of $\hat{\mu}$ is recomputed to select new instruction patterns. The above steps are repeated until the sum of all the $\hat{\mu}$'s exceeds a certain percentage which has been determined through some experiments.

2. Method to achieve maximum economical effect.

This method is similar to the method described above, but now we use $\hat{\mu}/\hat{\theta}$ instead of $\hat{\mu}$. Here, $\hat{\theta}$ represents the estimated value of the capacity needed for the WCS, which is necessary for the translation into firmware of the instruction patterns. It is also derived from the equations for estimating the tuning effect. Tuning is executed until the following condition is satisfied:

$$\hat{\mu} > \delta \quad \text{or} \quad \hat{\theta} > \delta'.$$

Synthesis of new instructions

We make a selected instruction pattern into a new instruction. Each sequence of instructions is implemented in a single microprogram, which occupies less space and has a faster execution time than the original instructions. The tuning is based on the existing microprogram optimization techniques⁷ which bring about reduction in the number of memory references, efficient use of internal resources and the possibility of parallel processing.

Feedback

The synthesized instructions are stored into the WCS and also registered in the code generation part of the compiler. The codes being compiled are translated into corresponding new codes by applying an editing operation at the machine code level.

Estimation of tuning effects

If no limitation is imposed on the capacity of the WCS, then optimal tuning may be achieved by synthesizing new instructions corresponding to the instruction patterns detected during the program execution. However, the capacity of the WCS is directly related to the cost of the system. It is also necessary to save time and effort in writing the microprograms.

It is difficult to give a quantitative evaluation of this effort. However, if we assume that the amount of work spent on microprogramming is proportional to the number of the microprogrammed steps, then the amount of labor will be proportional to the capacity of the WCS. Therefore, it is desirable to select instruction patterns which may combine maximal efficiency with minimal capacity of the WCS. To accomplish this, a simple method to estimate the firmware effect, that is the ratio of the execution efficiency improvement to the increase of the capacity of the WCS, when instruction patterns are implemented in firmware, must be developed. If such an estimation is possible, the selection of instruction patterns can be done based on the overhead of the tuning.

Here, we assume that the efficiency improvement resulting from the implementation of a microprogrammed new instruction (μ) is a function of the weight of the instruction pattern implemented in firmware (ξ) and the increase of the capacity of the WCS (θ), that is,

$$\mu = f(\xi, \theta). \quad (1)$$

Further, we assume that θ is a function of the length of the instruction pattern (γ), i.e.

$$\theta = g(\gamma). \quad (2)$$

We determined the function f and g , experimentally.

In order to determine f , we determined $\log(\mu/\theta^m)$ as a function of $\log \xi$. The result is shown in Figure 2, indicating an approximately linear relation. From Figure 2, we ob-

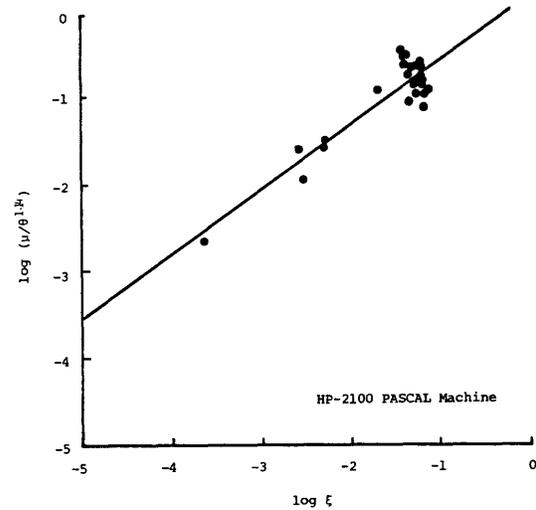


Figure 2—Relationship between $\log \xi$ and $\log(\mu/\theta^m)$.

tained

$$\log(\mu/\theta^m) = n \log \xi + a, \quad (3)$$

hence;

$$\mu = A \theta^m \xi^n, \quad (4)$$

where a , A , m and n are constants determined by the IML instruction set under consideration.

Similarly, we obtained the relationship between θ and γ , which is indicated in Figure 3. The relation is given by a linear expression:

$$\theta = b\gamma + c, \quad (5)$$

where b and c are constants determined by the IML instruction set.

Equation 4 includes that firmware effect can be estimated from the weight of an instruction pattern and the total steps

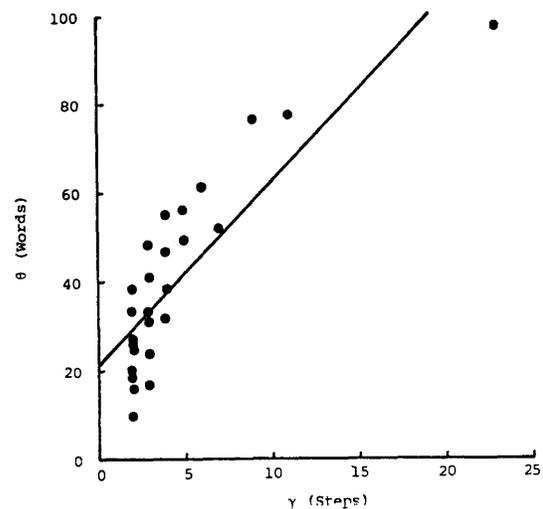


Figure 3—Relationship between γ and θ .

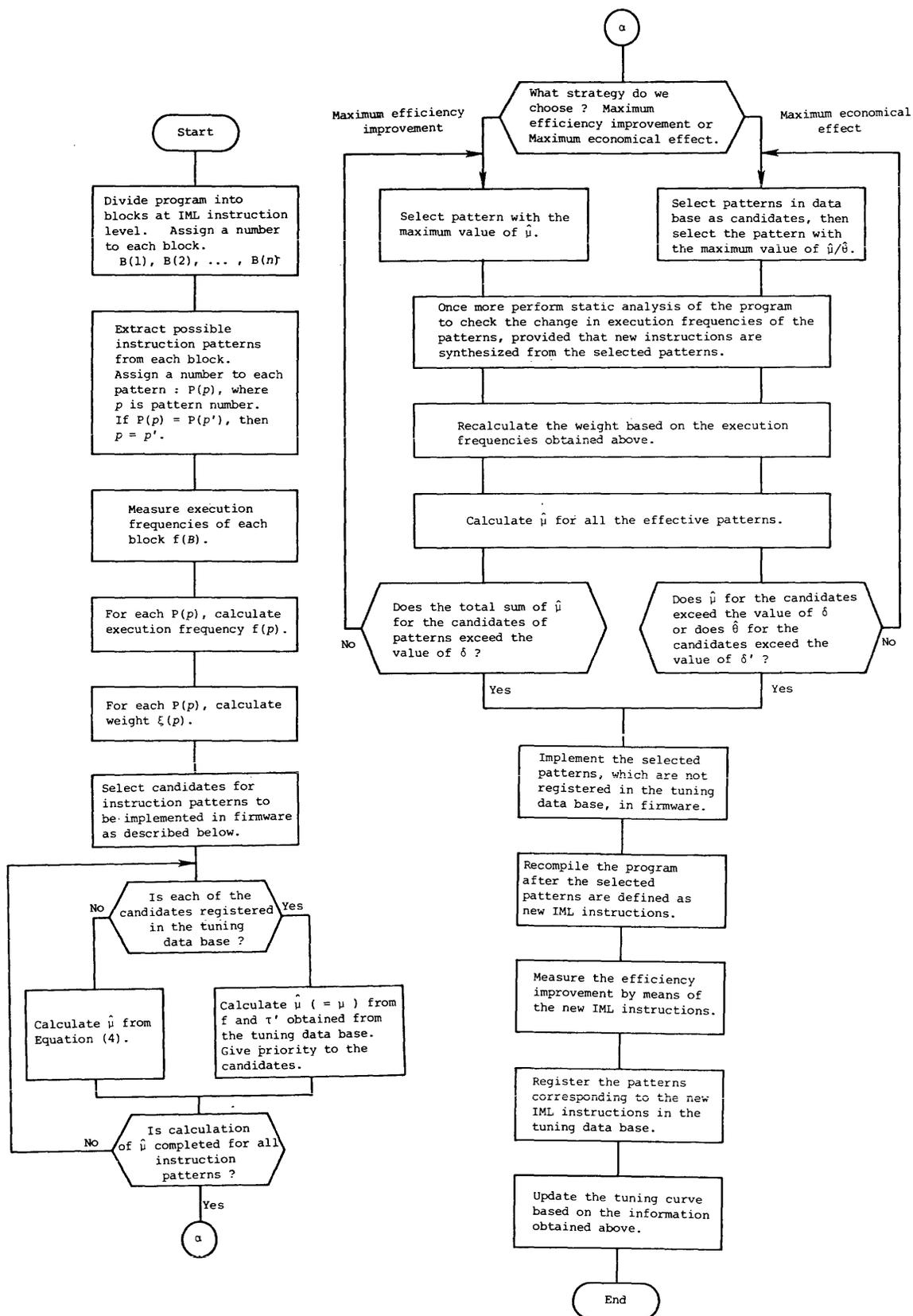


Figure 4—Simplified flowchart of the tuning algorithm.

required for the corresponding microprogram. Therefore, this relationship is called the "tuning curve."

Obviously, the constants a , A , m and n are positive and the larger the weight ξ and the increase of capacity of the WCS θ become, the greater the efficiency improvement will be.

A simplified flowchart of the algorithm just discussed is shown in Figure 4.

Implementation

It should be noticed that the tuning procedure just described can be performed automatically. The following implementation mechanisms are required:

1. Either hardware or firmware monitor is used to determine the weight of the instruction blocks.
2. A dynamic microprogramming technique is employed to modify the contents of the WCS.
3. The compilers for high-level languages should be able to expand the IML instruction codes. A technique used for incremental compilers can be adopted for this purpose.
4. The mechanism for the synthesis of new instructions can be implemented on a minicomputer or microcomputer.

EXPERIMENTAL SYSTEMS

We call the tuning mechanism consisting of a Monitor, an Analyzer and a Synthesizer the "Automatic Performance Evaluator (APE)." We developed two simple APE mechanisms to prove the effectiveness of the principles described in the previous chapters.

Experimental System-1 on a HP-2100 and hardware monitors

The process flow of tuning and learning in the experimental system is shown in Figure 5. We used a HP-2100 computer as the host computer to be tuned up. The HP-2100 computer is a 16-bit microprogrammable minicomputer.⁸ Since the machine instructions are incorporated in HP-2100, we can utilize two accumulators, while six other registers can be used with the micro-instructions.

The control storage consists of a ROM board with a capability of 256 24-bit words in which the microprogram that has to interpret the machine instructions is stored, and three WCS boards for user microprograms. Each WCS board has the same as the ROM.

We employed a DYNAPROBE 7900+8000 hardware monitor of COMPRESS Co. as the APE monitor. We used

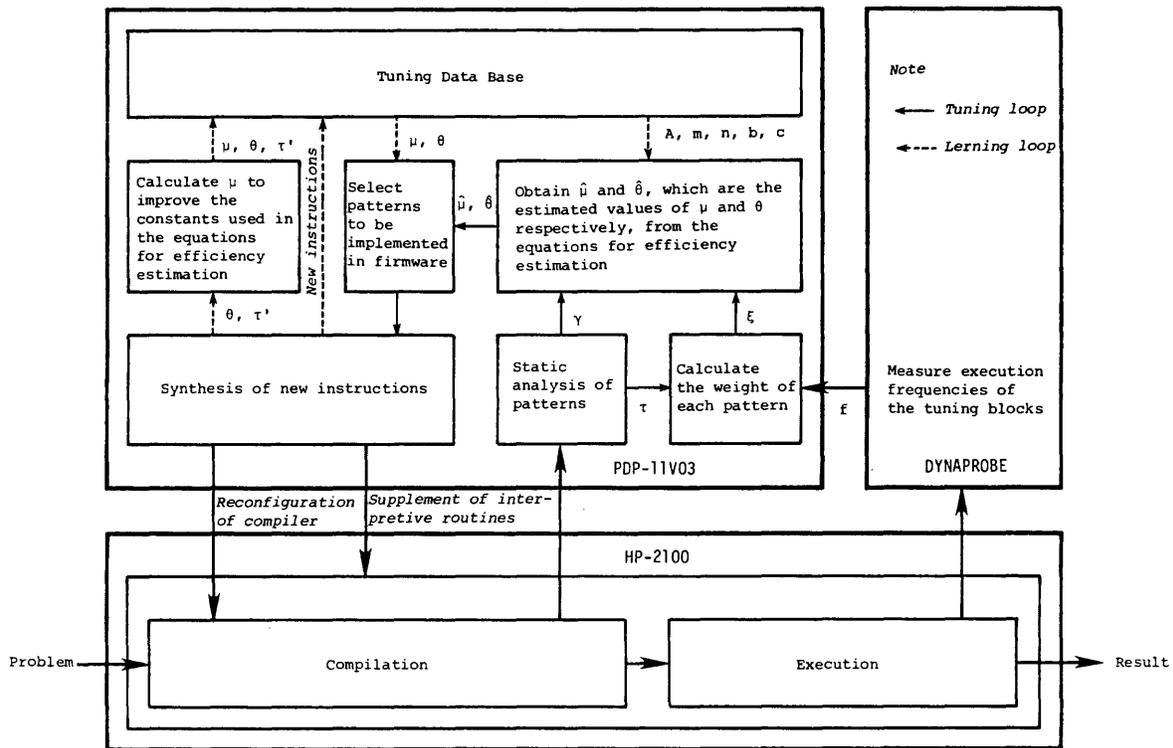


Figure 5—Process flow of tuning and learning in the experimental system.

a PDP-11V03 system to analyze or synthesize the output derived from the hardware monitor and further a D7916 count/time type hardware monitor and a D8028 map/store type hardware monitor to accumulate periods of time during which certain events have happened.

Figure 6 shows the block diagram of the experimental system. The program to be measured is executed on the HP-2100 host computer. The D7916 and D8028 hardware monitors collect signals generated from the host computer through probes in order to measure the weight of the instruction patterns.

The signals collected by the hardware monitors are directly supplied to the PDP-11V03 on which tuning analysis is carried out, the LSI bus of the PDP-11V03 is connected to the I/O bus of the HP-2100 through the I/O interface in order to make a feedback loop.

The LSI-11 is used for the analysis of instruction patterns, the reconfiguration of the IML instruction set and the organization of the data base in the tuning phase. To form a new IML instruction set, the microprograms for the synthesized instructions are stored into the WCS, that is incorporated in the HP-2100 through the I/O interface.

One of the main features of this system lies in the fact that there is no overhead in the monitoring function at all, since high-speed hardware monitors are used, and since the LSI-

11 is exclusively used for tuning analysis. Figure 7 shows several equipments of the experimental system.

Experimental System-2 on a Burroughs B-1700

The Burroughs B-1726 is a microprogrammable computer with a data length of 24 bits.⁹ It provides many internal resources, such as four general registers and thirty two scratchpad registers. It can perform bit addressing and it has access to any sub-field of the registers. The capacity of the WCS which stores microprograms expressed by 16-bit words is 4K words.

We developed several S-code interpreters in which a firmware monitor was implemented. To measure frequencies of instruction patterns, a firmware monitor was incorporated in the microinstruction fetch routine in the S-code interpreter. Therefore, apart from the B-1700 computer no special hardware is necessary. Hereby, the monitoring overhead becomes larger and the time required for the instruction fetch is approximately twice as long as that in the original computer.

However, the tuning operation does not occur very often, and if it occurs, the microprogram for the monitor is removed and the S-code interpreter is reconstructed so that the total overhead does not become very large.

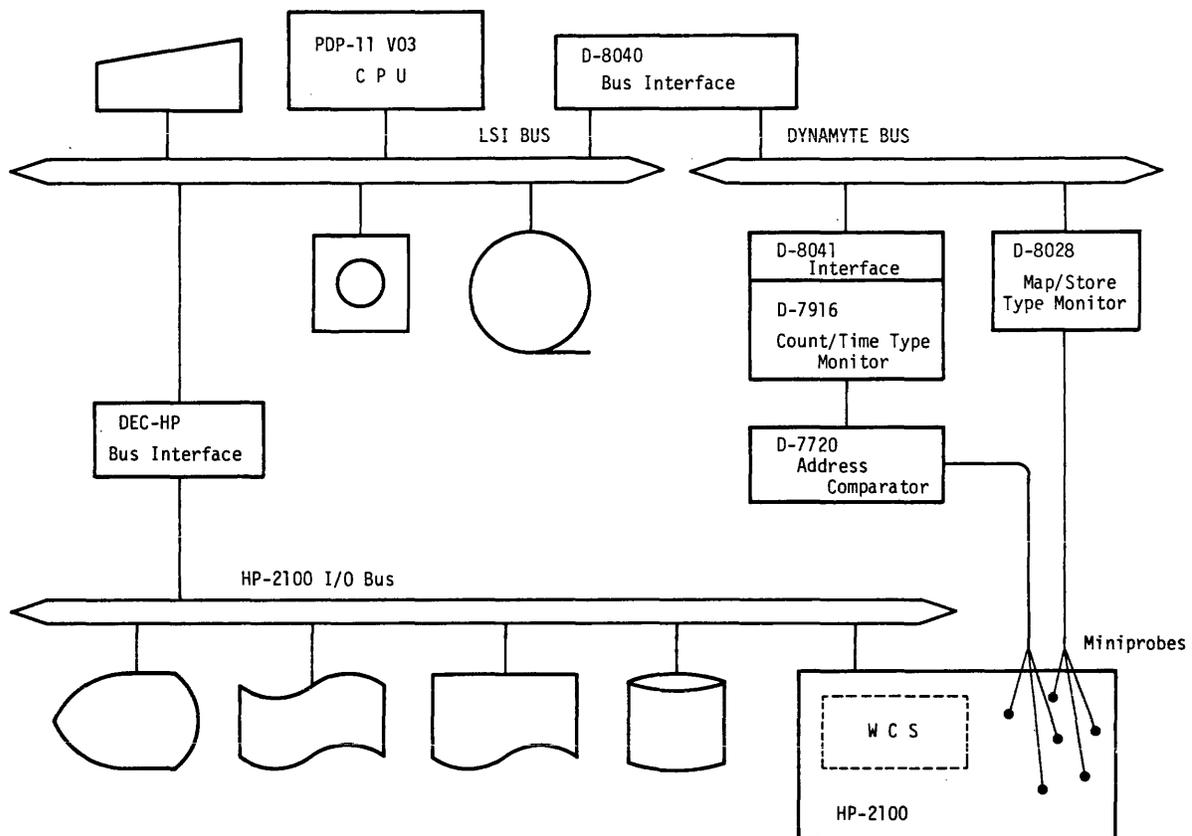


Figure 6—Functional configuration of the experimental system.

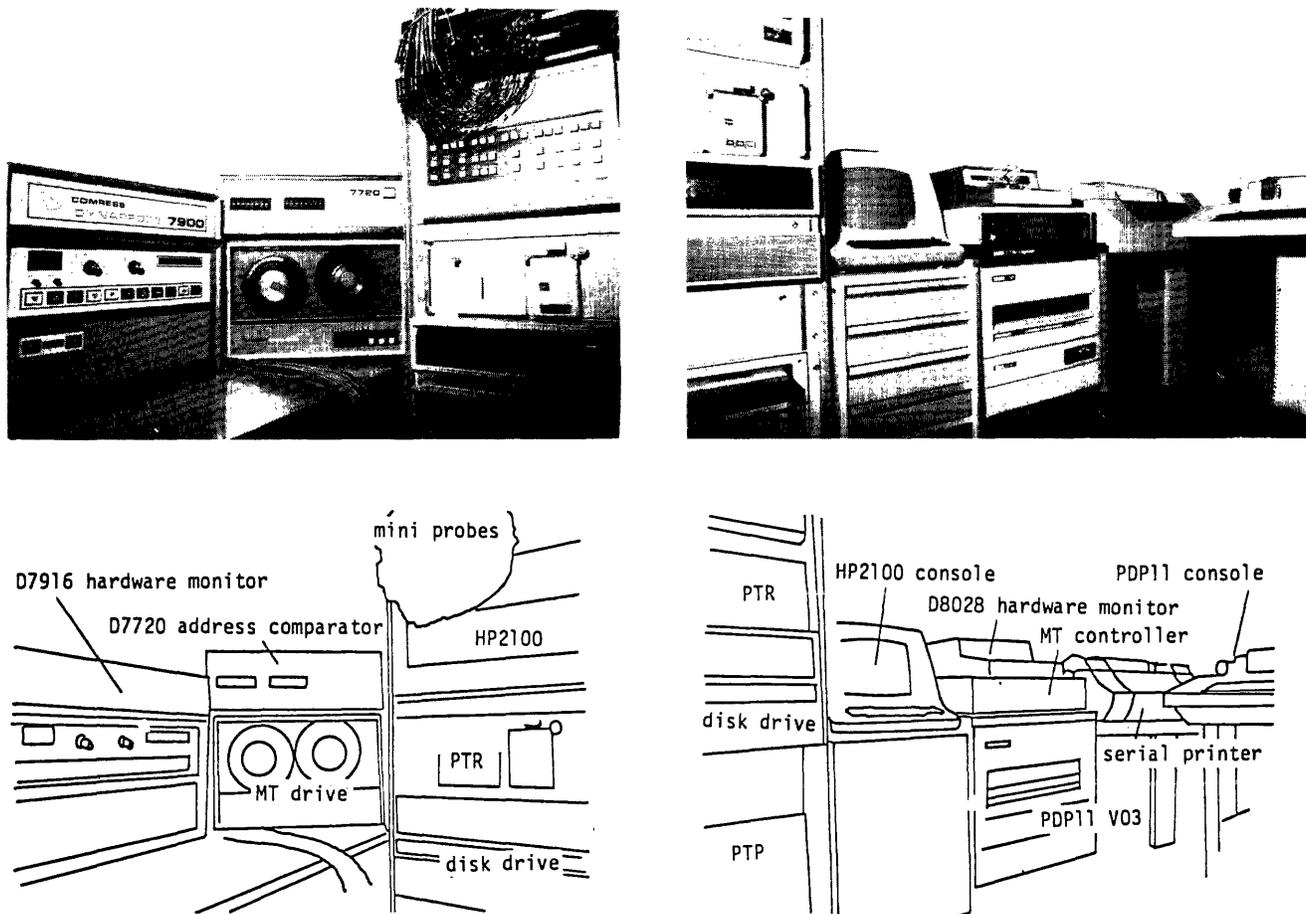


Figure 7—APE experimental system.

IMLs used for the experiments

For the IMLs to be tuned up, we chose the following two languages:

1. The IML for PASCAL¹⁰ which incorporates 60 instructions and is emulated on both the HP-2100 and B-1700 computers.
2. The FORTRAN IML for the HP-2100, i.e. sequences of machine instructions interpreted on the HP-2100 computer.

RESULTS OF THE EXPERIMENTS

Tuning results

Some tuning results are shown in Figure 8-Figure 11. Figure 8 indicates the tuning results when a sorting program is executed on the PASCAL machine emulated by the HP-2100. For the tuning procedure on the B-1700 PASCAL machine, see the Appendix.

If the strategy selected is maximum execution efficiency, it turns out that the execution efficiency is twice as high as

that of its corresponding non-tuned version, while the required capacity of the WCS is increased by 210 words when seven instruction patterns are selected. Further, the amount of the object code is reduced from 392 bytes to 284 bytes. This indicates that an effective code compaction is carried out during the static analysis of the program.

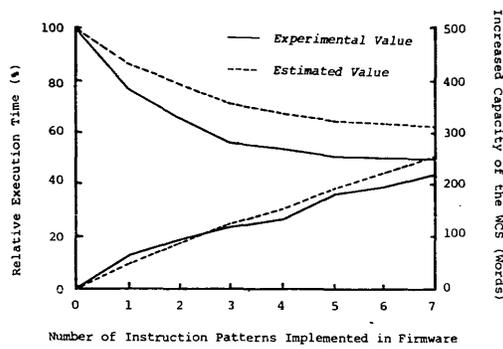
If the strategy selected is maximum economical effect, the capacity of the WCS is increased by 130 words, while the overall execution time of the program is reduced by 30 percent.

These results prove the effectiveness of the principles that we propose for automatic tuning of the computer architecture.

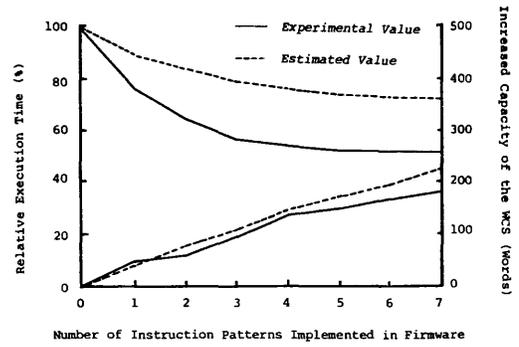
Frequently generated instruction patterns

Table I is a list of the instructions frequently generated in maximum execution efficiency strategy as well as in maximum economical effect strategy. The detected instruction patterns in the former strategy consist of more machine instructions than those in the latter strategy.

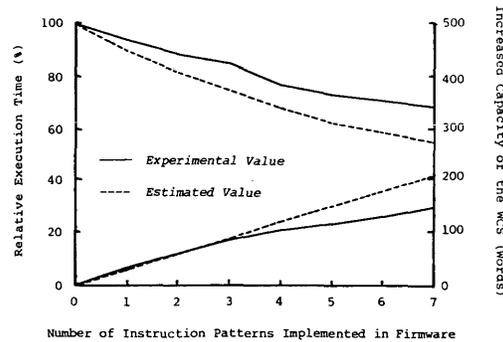
The meaning of most of the long instruction patterns is indicated in the table. For instance, the instruction pattern,



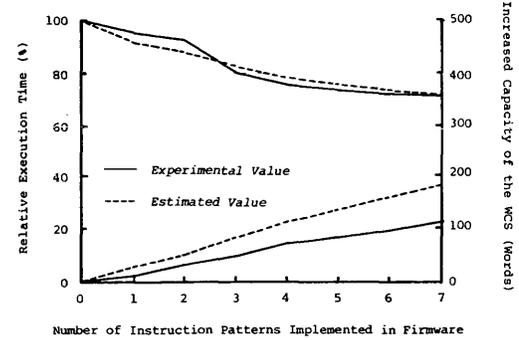
(a) Maximum Execution Efficiency Strategy (Select the instruction patterns so that $\bar{\mu}$ becomes maximal.)



(a) Maximum Execution Efficiency Strategy (Select the instruction patterns so that $\bar{\mu}$ becomes maximal.)



(b) Maximum Economical Effect Strategy (Select the instruction patterns so that $\bar{\mu}/\bar{\delta}$ becomes maximal.)



(b) Maximum Economical Effect Strategy (Select the instruction patterns so that $\bar{\mu}/\bar{\delta}$ becomes maximal.)

Figure 8—Tuning results (bubble sort problem on the HP-2100 PASCAL machine).

Figure 9—Tuning results (bubble sort problem on the B-1700 PASCAL machine).

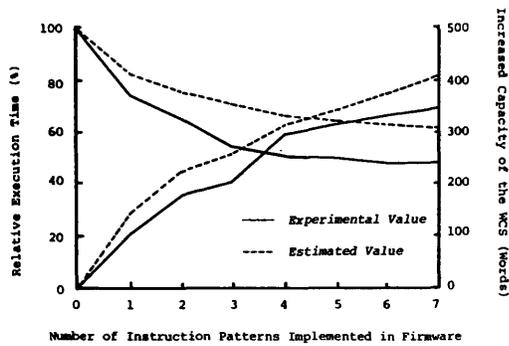
Table I — List of the Instruction Frequently Generated (Bubble Sort Problem on the HP-2100 PASCAL Machine)

Maximum execution efficiency

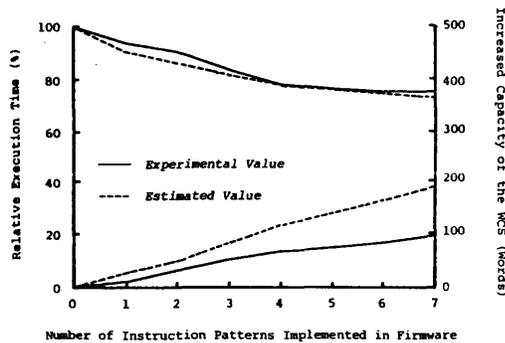
No.	Pattern No.	Instruction Patterns	Meanings
1	88	LAO LDO CHK DEC IXA IND	Load the contents of array elements into stack.
2	13	LDO LOD LEQ FJP	Compare two values. If one is less than other, then do not jump.
3	40	LDO INC SRO UJP	Increase value by q and jump.
4	70	LDO SRO	Trivial.
5	41	LAO LDO CHK DEC IXA	Calculate addresses of array.
6	19	LDO STO	Trivial.
7	6	SRO LDC STR	Trivial.

Maximum economical effect

No.	Pattern No.	Instruction Patterns	Meanings
1	17	DEC IXA	Trivial.
2	15	LDO CHK	Trivial.
3	8	LDO LOD	Trivial.
4	31	LDO INC SRO	Trivial.
5	10	LEQ FJP	Trivial.
6	70	LDO SRO	Trivial.
7	73	IND SRO	Trivial.



(a) Maximum Execution Efficiency Strategy (Select the instruction patterns so that $\bar{\mu}$ becomes maximal.)



(b) Maximum Economical Effect Strategy (Select the instruction patterns so that μ/δ becomes maximal.)

Figure 10—Tuning results (matrix multiplication problem on the B-1700 PASCAL machine).

e.g. the loading of the contents of array elements into a register, is reflected in the environment of the problem to be solved. The characteristics of these instruction patterns will play a very important role in the design of ISP architecture for high-level language oriented computers in the future. On the other hand, the short instruction patterns detected in maximum economical effect strategy do not have any significance.

IML structures and computer organization

The best way to emulate an IML instruction set depends on the structure of the computer, which has a subtle influence on the tuning effect. For instance, the tuning results on the HP-2100 machine instruction is shown in Figure 11, indicating that the tuning effect is not great if we do not take the address patterns as well as the instruction patterns into consideration. This is because the machine instruction and its corresponding operand are simultaneously fetched by a hardware mechanism on the HP-2100, while it is impossible to fetch the microinstruction and its operand simultaneously in user microprogramming.

There are two methods to solve this problem:

1. If the tuning is performed using only instruction patterns, the structure of the HP-2100 has some disadvan-

tages, since the possibility of handling multiple operands is not taken into account. It is desirable to make it possible to expand the operand field at the microprogramming level. The microprogrammable bit addressing function incorporated in the B-1700 and the machine instructions concerning the manipulation of multiple operands provided in the PDP VAX-11/780¹¹ make this problem easier to handle.

2. If the tuning is performed using both instruction patterns and their corresponding address patterns, a better tuning effect is achieved. For instance, the broken line in Figure 11 indicates the tuning results using both the instruction patterns and the address patterns. This result was obtained by a manual procedure. However, it is possible to automate the tuning procedure using address patterns by modifying the proposed algorithm.

IMLs and host computers

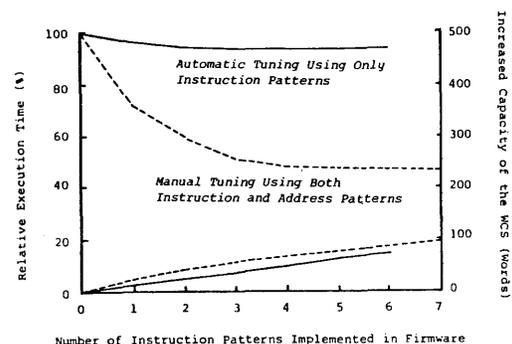
We arbitrarily selected two IMLs, namely the PASCAL and HP-2100 machine instructions. Nevertheless, significant performance improvement is achieved. Therefore, if we would use an IML instruction set which is more suitable for tuning procedure, an even better performance achievement is to be expected.

The HP-2100 is intentionally designed for the machine instructions. Therefore, it does not have effective features for tuning. Although the B-1700 is an emulation-oriented computer, it seems to us that its microprogram capability is designed to integrate the hardware and software on the Master Control Program. Therefore, some of the well organized mechanisms were not utilized in the automatic tuning procedure.

We think an automatic tuning oriented computer should be developed in the future.

Tuning curve and learning

The tuning curve is shown in Figures 12 and 13. The experimental results confirm that the methods to estimate the tuning effect are useful for the proposed algorithm.



Maximum Execution Efficiency Strategy (Select the instruction patterns so that $\bar{\mu}$ becomes maximal.)

Figure 11—Tuning results (bubble sort problem on the HP-2100 machine instructions).

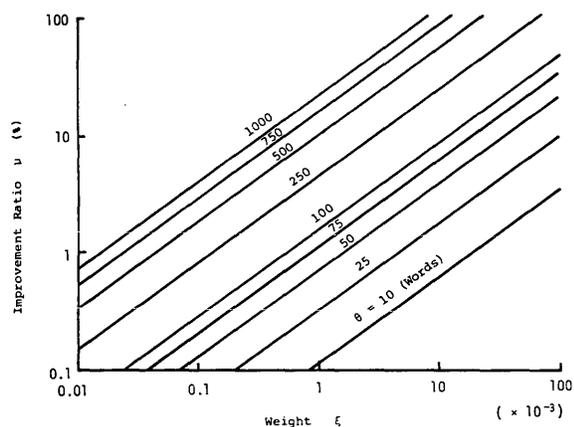


Figure 12—Tuning curves (HP-2100 PASCAL machine).

As shown in Table I, the frequently generated instruction patterns are significant. These patterns may often be generated in other application programs. This proves that the proposed method is very effective to improve IMLs for a wide variety of applications.

CONCLUSION

Development of a computer architecture that is adaptable to the problems to be solved is an important area of research. In this paper, we consider an automatic tuning of computer architectures at the ISP level using dynamic microprogramming techniques. After a brief explanation of the principles of the automatic tuning mechanisms, a detailed algorithm for the tuning procedure is described.

The basic processes of the proposed automatic tuning algorithm are:

1. Monitoring of the dynamic characteristics of the program to be solved.
2. Analysis of the monitored information to create new instructions.
3. Feedback of corresponding microprogrammed instructions to the WCS in the computer to form a new enriched architecture.

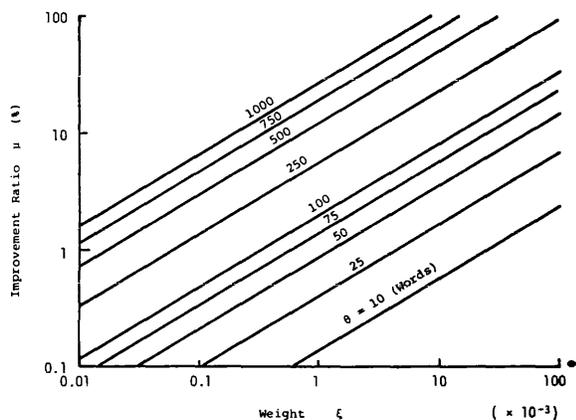


Figure 13—Tuning curves (B-1700 PASCAL machine).

4. A learning process with regard to the tuning behavior to guarantee faster achievement of performance improvement.

The first three processes are automatically carried out until the desired performance improvement is achieved.

In order to prove the effectiveness of the proposal algorithm, we carried out some experiments on a HP-2100 and a Burroughs B-1700 computer. Our experiments show that tuned programs are executed in 30-60 percent less time than the original programs.

It should be noted that dynamic microprogramming techniques are very effective to dynamically modify the computer architectures during the program execution. Our experiments proved that a computer can change dynamically so as to function optimally depending on the status of the problems to be solved, and that the learning effect is significant. We hope that the results of this research will contribute to the future development of adaptive computer systems and learning machines.

ACKNOWLEDGMENT

We wish to thank Mr. Yamaishi, Mr. Kato and our colleagues of Keio University and the Electrotechnical Laboratory for their kind suggestions and assistance. We would like to gratefully acknowledge the support and counsel of many people of Burroughs Co., Ltd. in Japan.

REFERENCES

1. Aiso, H., "On the realization of a Learning Machine," Internal Memorandum, Electrotechnical Laboratory, November 27, 1969.
2. Reigel, E. W., and H. W. Lawson, "At the Programming Language—Microprogramming interface," *SIGPLAN-SIGMICRO Interface Meeting*, June, 1973.
3. Abd-Alla, A. M., and D. C. Karlgaard, "Heuristic synthesis of microprogrammed computer architecture," *IEEE Tran. Comput.*, Vol. C-23, No. 3, August, 1974.
4. El-Ayat, K. A., and J. A. Howard, "Algorithm for a self-tuning microprogrammed computer," *MICRO 10*, October, 1977.
5. Snyder, D. C., "Computer Performance Improvement by Measurement and Microprogramming," *Hewlett-Packard Journal*, February 2, 1975.
6. Knuth, D. E., "An empirical study of FORTRAN programs," *Software—Practice and Experience*, Vol. 1, 1974.
7. Agerwala, T., "Microprogram Optimization: A Survey," *IEEE Trans. Comput.*, Vol. C-25, October, 1976.
8. "MICROPROGRAMMING GUIDE for Hewlett-Packard Model 2100 Computer," Hewlett-Packard Company, November, 1971.
9. Wilner, W. T., "Design of the B-1700," *AFIPS Conference Proceedings, FJCC*, 1972.
10. Nori, K. V., U. Amman, K. Jensen and H. H. Nageli, "The Pascal (P) Compiler: Implementation Notes," Nr. 10 Berichte des Institutes für Informatik, E.T.H., December, 1974.
11. Strecker, W. D., "VAX-11/780—A virtual address extension to the DEC PDP-11 family," *National Computer Conference*, 1978.
12. Organick, E. I., and J. A. Hinds, *Interpreting Machine: Architecture and Programming of the B1700/B1800 Series*, Elsevier North-Holland, 1978.

APPENDIX

An example of the tuning procedure on the B-1700 PASCAL machine

The following steps should be referred to the tuning algorithm shown in Figure 4.

1. Original PASCAL Program

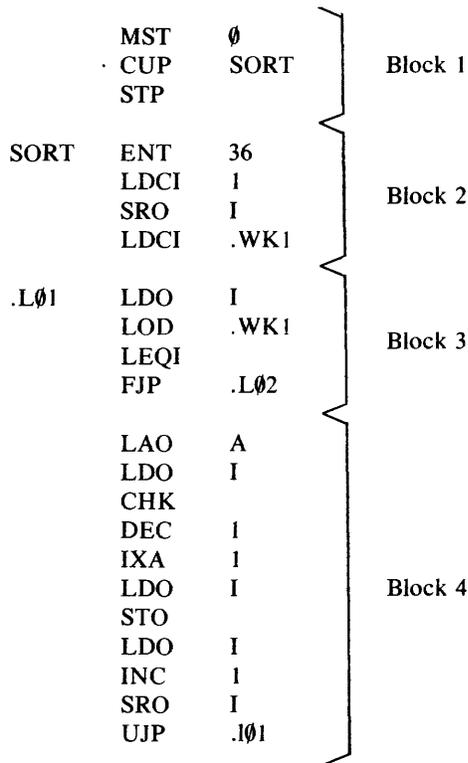
```
? SORT AN ARRAY OF INTEGER ?
CONST N=25;
VAR I,J,K,M: INTEGER;
    A: ARRAY [1..N] OF INTEGER;
BEGIN FOR I:=1 TO N DO A[I]:=I;
      FOR I:=1 TO N-1 DO
        BEGIN K:=I; M:=A[I];
          FOR J:=I+1 TO N DO
            IF A[I]>M THEN
              BEGIN K:=J;
                M:=A[J]
              END;
            A[K]:=A[I]; A[I]:=M
          END
        END
      END.
```

```
9  LOD LEQ
10 LEQ FJP
11 LDO LOD LEQ
12 LOD LEQ FJP
13 LDO LOD LEQ FJP
14 LAO LDO
15 LDO CHK
16 CHK DEC
17 DEC IXA
    :
```

Integral patterns of each block

Block No.	Pattern No.
1	1
2	2 3 4 5 6 7
3	8 9 10 11 12 13
4	14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
5	2 3 5 69
6	8 9 10 11 12 13
	:

2. Divide program into blocks at IML instruction level.



4. Calculate the length and the execution time of each pattern.

Pattern No.	Length of Pattern [γ]	Execution Time of Pattern [τ]
1	2	25.1
2	2	21.5
3	2	21.5
4	2	22.7
5	3	30.2
6	3	35.6
7	5	50.0
8	2	25.7
9	2	27.5
10	2	22.5
	:	

Hardware Monitor
or
Firmware Monitor
↓

3. Extract possible instruction patterns from each block.

Pattern No.	Pattern
1	MST CUP
2	LDC SRO
3	SRO LDC
4	LDC STR
5	LDC SRO LDC
6	SRO LDC STR
7	ENT LDC SRO LDC STR
8	LDO LOD

5. Measure execution frequencies of each block.

Block No.	Frequency [f]
1	1
2	1
3	26
4	25
5	1
6	25
7	24
8	324

9	300
10	156
11	300
12	24
⋮	

6. For each pattern, calculate execution frequencies then calculate weight.

Pattern No.	Frequency [f]	Weight [ξ]
1	1	1.02 × 10 ⁻⁴
2	2	1.75 × 10 ⁻⁴
3	26	2.28 × 10 ⁻³
4	25	2.31 × 10 ⁻²
5	2	1.64 × 10 ⁻⁴
6	25	2.42 × 10 ⁻³
7	1	8.14 × 10 ⁻⁵
8	375	3.93 × 10 ⁻²
9	375	4.21 × 10 ⁻²
10	375	3.44 × 10 ⁻²
⋮		

Note: $\xi(p) = \frac{T(p)}{T_0 \times \gamma(p)}$
 $T(p) = \tau(p) \times f(p)$
 $T_0 = T(1) + T(7) + T(13) + T(40) + T(68) + T(69) + T(95) + T(96) + T(97) + T(98)$
 $= 1.23 \times 10^5 \mu s$

7. Calculate $\hat{\theta}$, $\hat{\mu}$ and $\hat{\mu}/\hat{\theta}$ from Equations 4 and 5. In the case that none of the candidates is in the tuning data base, but the coefficients for Equations 4 and 5 are registered.

$A=0.69$ $b=4.6$ $\hat{\theta}(p) = b \times \gamma(p) + c$
 $m=1.14$ $c=14.7$ $\hat{\mu}(p) = A \times \hat{\theta}(p)^m \times \xi(p)^n$
 $n=0.61$

Pattern No.	[θ̂]	[μ̂]	[μ̂/θ̂]
1	24.1	0.1	0.004
2	24.1	0.1	0.006
3	24.1	0.6	0.03
4	24.1	0.6	0.03
5	28.7	0.2	0.005
6	28.7	0.8	0.03
7	37.9	0.1	0.004
8	24.1	3.6	0.15
9	24.1	3.8	0.16
10	24.1	3.3	0.14
⋮			

8a. Strategy 1—Maximum efficiency improvement

Sorting result of patterns in descending order of $\hat{\mu}$.

No.	Pattern No.	[μ̂]	[θ̂]
1	88	10.6	42.5
2	41	9.9	37.9
3	84	9.7	37.9
4	97	9.6	56.3
5	34	9.3	33.3
⋮			

The rest is omitted but same as in Strategy 2.

8b. Strategy 2—Maximum economical effect

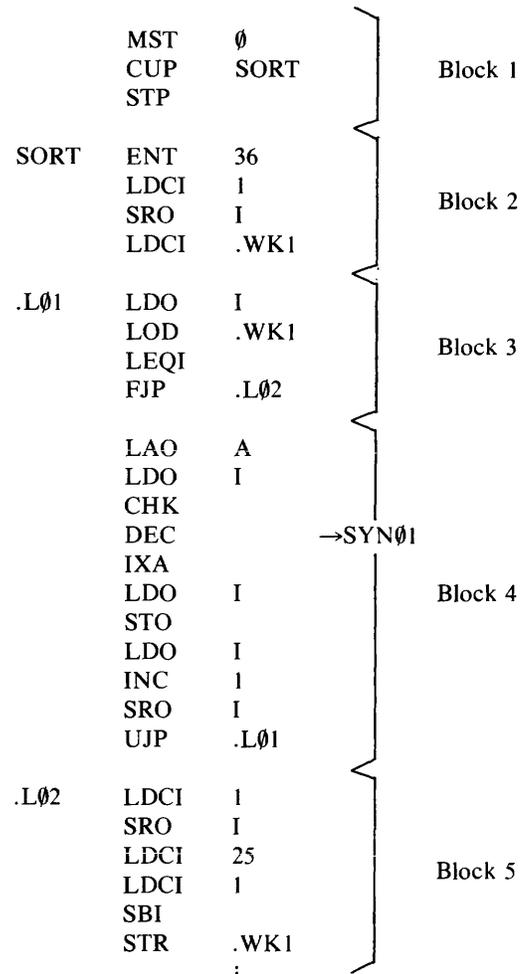
Sorting result of patterns in descending order of $\hat{\mu}/\hat{\theta}$.

No.	Pattern No.	[μ̂/θ̂]	[μ̂]	[θ̂]
1	17	0.33	7.9	24.1
2	72	0.31	7.5	24.1
3	26	0.30	8.5	28.7
4	76	0.28	8.1	28.7
5	34	0.28	9.3	33.3
⋮				

Pattern with the maximum value of $\hat{\mu}/\hat{\theta}$.

17 DEC IXA

9. Once more perform static analysis of the program to check the change in execution frequencies of the pattern, provided that new instructions are synthesized from the selected patterns.



10. If the total sum of $\hat{\mu}$ for the candidates of patterns exceeds the value of δ or if the total sum of $\hat{\theta}$ for the candidates of patterns exceeds the value of δ then go to 11, else go to 6.

11. Original PASCAL IML interpreter written in Burroughs B-1700 MIL.¹²

```

% IML  FETCH AND DECODE
        CLEAR BASE.REG
        MOVE 24 TO CP
        MOVE A TO FETCH.ADDRESS
        %A=ACTUAL ADDRESS OF
        FETCH
FETCH  MOVE FETCH.ADDRESS TO TAS
        MOVE PC TO T
        SHIFT T LEFT BY 5 BITS TO X
        SHIFT T LEFT BY 4 BITS TO Y
        MOVE SUM TO FA
        ADD BASE.REG TO FA
        FA.POINTS TO IML
        READ DL(OP)+DL(P) BITS TO T
        PT.FA Q
        READ DL(Q) BITS TO X
        MOVE X TO Q.FIELD
        EXTRACT DL(OP) BITS FROM T(10)
        TO Y
        MOVE PC TO L
        COUNT L UP BY 1
        MOVE L TO PC
        MOVE Y TO M
        JUMP FORWARD
        GO TO LOD
        GO TO LDO
        :
DEC    CALL POP.UP.BREG.1
        MOVE Q.FIELD TO Y
        MOVE DIFF TO BREG.1
        GO TO PUSH.DOWN.BREG.1
IXA   CALL POP.UP.BREG.1
        MOVE Q.FIELD TO AREG.1
IXA.1 CALL SET.SIGN.AND.ABS
        MOVE AREG.1 TO Y
        MOVE BREG.1 TO FA
        CALL I.MULTIPLY
        IF SIGNS.ARE.DIFFERENT THEN
            BEGIN
            CLEAR X
            MOVE DIFF TO Y
            END
        MOVE Y TO TAS
        CALL POP.UP.BREG.1
        MOVE TAS TO Y
    
```

```

MOVE SUM TO BREG.1
GO TO PUSH.DOWN.BREG.1
    
```

12. Implement the selected pattern in firmware.

```

SYNØ1 %   DEC Q1-IXA Q2
%
%   + - - - - + - - - - +
%   I SYNØ1   I           I
%   + - - - - + - - - - +
%   I           Q 1       I
%   + - - - - - - - - - +
%   I           Q 2       I
%   + - - - - - - - - - +
%   I           DUMMY     I
%   + - - - - - - - - - +
%
%   BUMP(PC,1)
%   READ 24 BITS TO X
%   MOVE X TO AREG.1
%   CALL POP.UP.BREG.1
%   MOVE Q.FIELD TO Y
%   MOVE DIFF TO BREG.1
%   BREG.1=TOP OF STACK
%   AREG.1=Q2
%   GO TO IXA.1
    
```

13. Measure the efficiency improvement by means of the new IML instructions then register the patterns corresponding to the new IML instructions in the tuning data base, and update the tuning curve.

Tuning result for Pattern 17

Routine Name	Cycle	Routine Name	Cycle
FETCH	25	FETCH	25
DEC	48	SYNØ1	276
FETCH	25	Total 301	
IXA	265		
Total 363			

Execution Time [τ] 60.4 μ s Execution Time [τ'] 50.3 μ s
 Required WCS [θ] 9 words

$$\mu(17) = \frac{(\tau(17) - \tau'(17)) \times f(17)}{T_0} \times 100\%$$

$$= 4.8\%$$

The BTI 8000—Homogeneous, general-purpose multiprocessing

by GEORGE R. LEWIS and J. SHIRLEY HENRY

BTI Computer Systems
Sunnyvale, California

and

BRIAN P. McCUNE*

Stanford University
Stanford, California

APPROACHES TO GROWTH IN COMPUTER USAGE

With the price of computer hardware decreasing steadily and the scope of data processing applications ever rising, the problem of upgrading a computer system is omnipresent. The myriad of potential pitfalls includes losing an investment in purchased hardware and software, reprogramming applications, reformatting data files, retraining personnel, operating two different systems in parallel during the conversion period and reoptimizing finely tuned applications.

Computer series

Most computer vendors attempt to ease the pain of upgrades by offering an entire series of similar computers spanning a wide range of price and performance. Such series (e.g., IBM 360^{1,10,28} and its successors, and PDP-11^{8,6}) usually consist of a relatively constant machine architecture with different underlying implementations which yield faster and faster central processing units (cpus) and memories (see Reference 7, Part 6, "Computer Families").

Unfortunately, re-implementation of an architecture may result in forced changes at the user level; for example, a program or even the operating system may run on one implementation and not another. At the least, re-implementation is costly in terms of the additional design and manufacturing efforts required.

In addition, minimal configurations of cpus, memories, and peripherals are designed with the typical user in mind. Thus, it is common for a user requiring much central processing power but little I/O to be forced into a configuration sporting many unneeded data channels in order to get a powerful enough cpu.

Modular multiprocessor

An alternative to the typical computer series with multiple implementations is a multiprocessor architecture which offers systems of one to many identical cpus, memories and data channels, thereby spanning the desired price and performance range. Cpus, memories and channels can be added independently to meet a particular processing need and without changes to user programs.

It costs less to design a simple cpu once and replicate it within an elegant multiprocessor architecture than to design many different, possibly very complicated cpus. Increased reliability because of redundant resources is an additional benefit of such a modular approach. Indeed, we feel that computer systems of the future should not have to depend upon the availability of a single cpu.

BACKGROUND ON MULTIPROCESSING

There has been much interest in multiprocessing computer systems since the first, the Burroughs D 825,³ appeared in 1962. A *multiprocessor* system has more than one cpu, each with its own stream of instructions operating on its own data stream. Such an architecture is often termed "MIMD" for these "multiple instruction, multiple data" streams.¹⁹

Homogeneity of resources

There are many possible ways to classify MIMD architectures (Figure 1, after Reference 16). Perhaps the most useful is by their degree of *homogeneity*, or the degree to which resources of the system (processors, main memory, and peripheral units) can be shared indistinguishably (Reference 18, pages 104-105). This has also been defined as the quality that all resources of a particular type appear *symmetric* to the rest of the system.¹⁷

* Currently at Systems Control, Inc., Palo Alto, California.

Computer	Full Homogeneity?			Operating System & User Programs	Purpose	Cpu-Memory Interconnection
	Cpu	Main Memory	Peripherals			
BTI 8000	Yes	Yes	Yes	Yes	General Purpose Timesharing	Timeshared Common Bus
D 825	Yes	Yes	Yes	Yes	Command & Control	Crossbar
CLC	Yes	Yes	Yes	Yes	Command & Control	Multiported Memory
U 1108	Yes	Yes	Yes	Yes	General Purpose	Multiported Memory
PLURIBUS	Yes	Partial	Yes	Yes	Packet Switching	Multiple Buses
C.mmp	Yes	Partial	No	Yes	Realtime Artificial Intelligence	Crossbar
PRIME	Yes	No	Yes	No	Timesharing	Multiported Memory
T 16	Yes	No	Partial	No	Transaction Processing	None
ARC	No	No	Partial	No	Timesharing	None
IBM 370 ASP	No	No	No	No	General Purpose	None
AP-120B	No	No	No	No	Scientific	None

Figure 1—Multiprocessing aspects of some computers.

Adding capabilities to an existing system is straightforward when it is composed of homogeneous modules. The more homogeneous a system is, the more robust (i.e., less prone to catastrophic failure), because other equivalent resources are still available when one fails. Using only one basic design in each part of a multiprocessor also keeps design, manufacturing and programming costs to a minimum. And homogeneity allows more efficient utilization of resources, since each member of a particular resource pool is unrestricted as to what tasks it can do.

Most multiprocessors of the past have lacked homogeneity in at least one respect. The concept of homogeneous cpus, for instance, breaks down if a system has one general purpose processor and one or more special purpose processors, e.g., the fast floating point AP-120B array processor by Floating Point Systems.³⁰ In this case, programming for each type of cpu is obviously different and must be done explicitly. (Not to be considered a multiprocessor at all in this discussion is the typical computer with one general-purpose cpu and one or more special-purpose I/O processors, Reference 7, Part 5, Section 2, "Computers with One Central Processor and Multiple Input/Output Processors.")

Ideally, all cpus would have access to all of memory to make the sharing of programs and data simpler and more efficient, and thus cpus would not have local private memories (other than perhaps a small transparent cache for performance improvement). This type of cpu is often termed "closely coupled." The Tandem Computers T 16 NonStop computer,²¹ which has nonshared memory for each cpu, must be regarded as a network of computers (each consisting of a cpu, I/O processor and memory) with partial sharing of peripherals. Similarly, the Datapoint Attached Resource Computer (ARC) and IBM's loosely coupled attached support processor (ASP) (Reference 7, page 506) are networks (Figure 2). The Carnegie-Mellon University C.mmp (based on Digital Equipment PDP-11 cpus)^{31,24} and the Bolt Beranek and Newman PLURIBUS (based on Lockheed SUEs)^{20,26,4} are hybrids, having both shared memory and memory private to each cpu.

The sharing of all I/O capabilities is the next aspect of homogeneity, one which is lacking in C.mmp, for example, because an I/O device must be attached to the Unibus of

one of its PDP-11s and is restricted to communicating with a local memory there. This is undesirable because the loss of either the cpu or its local memory isolates the I/O device from the rest of the system.

A final aspect of homogeneity concerns software, including both the operating system and user programs. Ideally, for efficient use of processor and memory resources there should be only one copy of one operating system which can be run on any of the cpus; this is the case with C.mmp. In practice, however, the typical multiprocessor system has resorted to having either separate copies of the same operating system running on each cpu or one copy restricted to always running on the same cpu (the common master-slave mode of many dual processor systems).

For reliability and efficiency it should be possible for any user program to run on any processor. Too often, applications are segregated to run on dedicated cpus. When segregation is done by entire classes of programs (e.g., interactive, batch, data base management), it may be a consequence of the lack of homogeneity in I/O. A common example of this is the restriction of interactive terminals to one cpu.

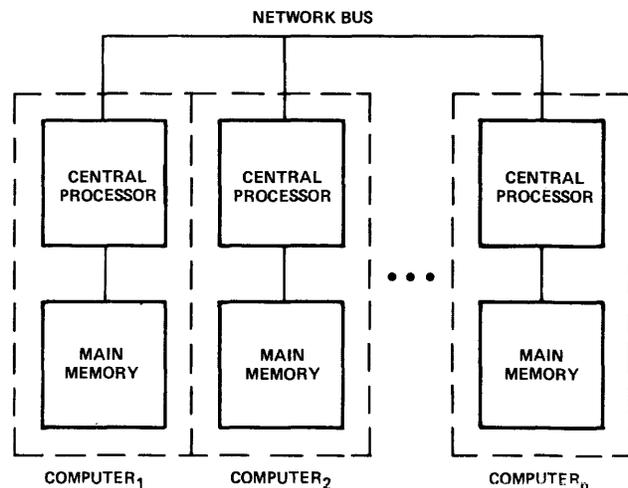


Figure 2—Architecture of a computer network.

Purpose

Another dimension of multiprocessors is their purpose. Most successful multiprocessors have not been designed for the general purpose commercial computing environment. The D 825 and the Bell Telephone Laboratories CLC (part of the SAFEGUARD ABM system)²⁵ were designed for real-time military command and control applications in which maximizing both overall throughput and reliability was paramount. The T 16 was designed for transaction processing applications in industries, such as banking and airlines, in which continuous system availability is required. PLURIBUS was designed as a packet-switching node for the ARPAnet. One-of-a-kind research prototypes have included C.mmp and the University of California at Berkeley PRIME,⁵ designed for real-time artificial intelligence applications and timesharing, respectively.

Many computer manufacturers have introduced multiprocessing to extend the high-end capabilities of a computer line. Examples are the Sperry Rand Univac 1108,²⁷ IBM 370 MP,^{23,15} and DECsystem-10⁹ dual processors. These architectures were not designed from the start with a multiprocessor in mind. So we often observe systems which could theoretically support a number of processors, but in practice never have more than two.

Closely related to the purpose of a multiprocessor is the way in which user programs are expected to take advantage of its multiprocessing capability. There are two basic philosophies.

The first is most useful for special-purpose multiprocessors (e.g., C.mmp, PLURIBUS, CLC) dedicated to running a small number of large, known programs. In this case the user must explicitly segment an application into a number of independent programs. These programs, called *processes*, are designed so that they may run concurrently, thus taking advantage of the multiple cpus available. Segmenting a program currently must be done by hand by the programmer or analyst and is not a simple task.

The alternative used in general purpose multiprocessors, especially in time-sharing, is to rely upon the job mix of a multiprogramming environment to provide enough processes (one per job) to keep all cpus busy. This avoids the programmer headaches of program segmentation and also avoids the system overhead resulting from the interprocess communication needed to coordinate processes. Of course, it is desirable to allow such coordination, while not requiring it.

Processor-memory interconnection schemes

Probably the most critical design decision in a multiprocessor architecture is selection of the method by which main memory is interconnected to the cpus and peripheral processors (channels). The bandwidth of this interconnection switch provides an upper bound on the number of processors which can be handled, and thus on the throughput of the entire system.

Four basic schemes have been used. In decreasing order

of complexity and potential throughput, they are

1. A crossbar switch or matrix connecting each memory unit with each processor.
2. Multiported memory, in which each memory provides a port for a connection from each processor.
3. Multiple buses, each of which connects some cpus, memories and other buses (which may in turn connect to other cpus, memories and buses).
4. One common time-shared bus, over which all data traffic must pass.

Although the first three alternatives provide concurrent transfers between more than one processor-memory pair (Figure 3), analogous to a telephone switching exchange in which more than one conversation can occur simultaneously, they are also relatively expensive and typically only used in large systems. The D 825 and C.mmp used crossbar schemes, while the dual processor IBM 370 MP systems, Univac 1108, and DECsystem-10 (along with most other multiprocessors) rely on multiported memory.

The complexity (and hence cost) of a switch is proportional to the number of hardware switching elements it has, which in turn depends on the number of unique interconnections allowed between processors and memories. If each of p processors is to be capable of "talking" to each of m memories in parallel, then there must be $p \times m$ unique interconnections. If, on the other hand, each processor and memory need only be able to talk to a single common bus, then the number of interconnections is reduced to just $p+m$. For systems with more than two cpus and two independent memories, the difference in the complexity of the two schemes becomes immense.

Until now, no one has developed a bus fast enough to make a multiprocessor using a single timeshared bus feasible (Figure 4), analogous to a telephone exchange in which all subscribers are on a single party line. For example, limited by buses with 200-nanosecond data transfer rates, a PLURIBUS system requires multiple buses, each bus connecting up to two processors and two memories. Up to seven of these buses may then be connected to shared memory and peripherals by still other buses.

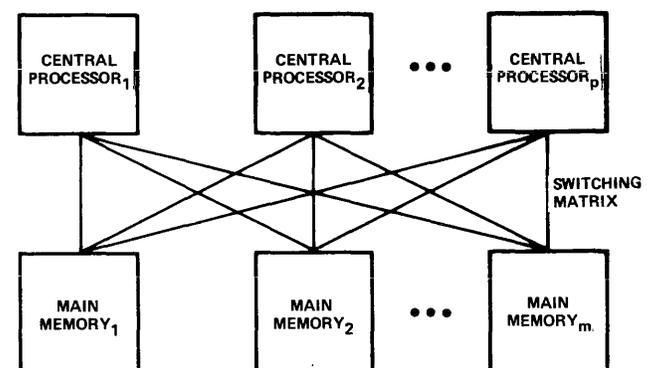


Figure 3—Architecture of a multiprocessor with parallel buses.

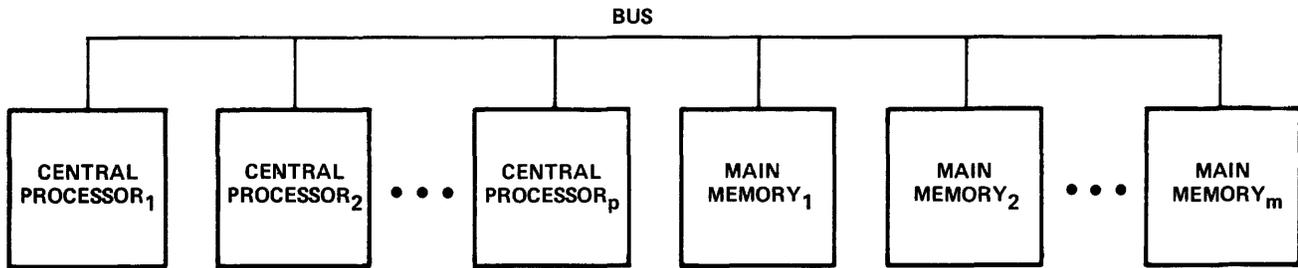


Figure 4—Architecture of a multiprocessor with a single common time-shared bus.

ARCHITECTURE OF THE BTI 8000

The BTI 8000 computer system was designed to be a general-purpose system featuring inexpensive, modular resources for easy upgrading and high reliability, interactive time-sharing, an easy-to-use and constant virtual machine for the user and high-level programming languages.¹¹

In light of these goals, the BTI 8000 was designed from the beginning with fully homogeneous, general-purpose multiprocessing as an objective. To our knowledge it is the first system to meet this goal, while also being the first general purpose multiprocessing minicomputer. The BTI 8000 is homogeneous with respect to cpus, memory, peripherals, operating system and user programs.

A BTI 8000 system (Figure 5) is based on 32-bit words and consists of a number of modular *resource units*.

1. Computational processing units (CPUs). (This paper denotes a computational processing unit of a BTI 8000 by "CPU" and an arbitrary central processor by "cpu.")
2. Memory control units (MCUs) and associated memory.
3. Peripheral processing units (PPUs) and associated I/O peripherals.
4. System services unit (SSU).

Up to 16 of these resource units (each of which is actually a microprogrammed processor) communicate via a single

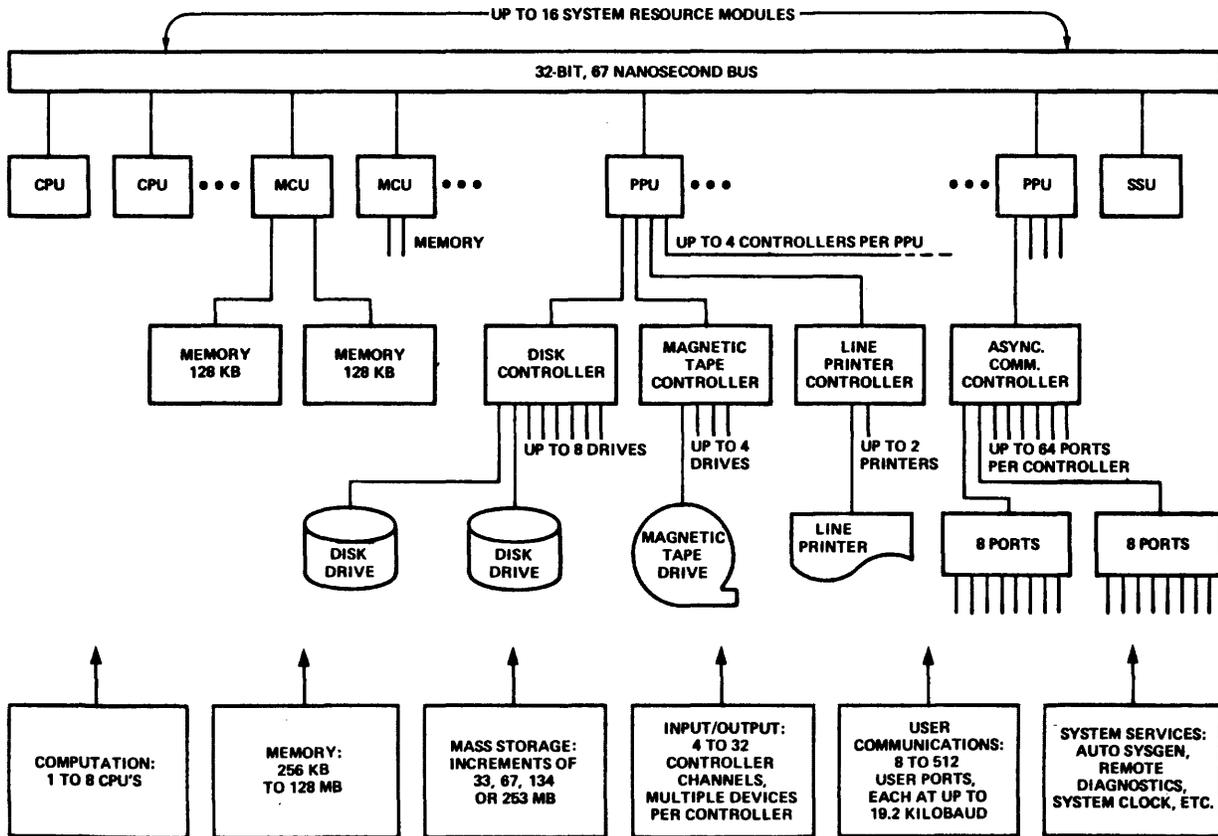


Figure 5—Block diagram of the BTI 8000 architecture.

32-bit-wide directly connected common (or "global"²⁹) time-shared (or functionally and physically "non-dedicated"²⁹) bus. Its speed is the key to the success of the 8000 as a multiprocessor. The bus can transfer one 32-bit word every 66.7 nanoseconds, or 15 million words (60 million bytes) per second. This speed is possible because the passive, synchronous bus relies upon distributed (or "decentralized"²⁹) control logic for fast resolution of the bus contention arising from simultaneous, independent bus requests from two or more resource units.

The hardware modularity (along with an operating system which requires no reprogramming to handle hardware re-configuration) makes possible both easy system enhancement and graceful degradation. A small user may start out with the minimum system configuration, comprising one each CPU, MCU, PPU, and SSU and corresponding to a medium speed minicomputer. As system requirements increase, additional resource units can be added, resulting in a system which has a throughput rivaling that of many mainframes, but which is much more cost effective. For example, an additional CPU costs a fraction of the price of a complete system and consists of one 20-inch-by-23-inch printed circuit board ready to be plugged into the bus (Photograph 1). A typical large 8000 configuration might consist of six CPUs, six MCUs, three PPUs, and one SSU.

Because all CPUs must communicate via the same bus to the same homogeneous memory, the question arises, how badly will bus and memory contention degrade performance in a system with more than one CPU? In many multiported memory multiprocessors, for example, more than two cpus are impractical because the cpus are faster than memory and not enough memory ports are provided. In contrast, the number of CPUs and MCUs in a BTI 8000 can be flexibly chosen based upon the relative speeds of each type of resource. And the bus is fast enough so that contention for it is minor.

Simulation studies of various configurations of CPUs, MCUs and PPUs have been run assuming typical memory accesses for each type of processing unit. An 8000 system with six CPUs, six MCUs, and two PPUs should have the throughput of approximately five separate 8000 systems, each having one CPU, one MCU and one PPU (Figures 6, 7). Each of these five systems would, of course, require its own SSU, bus, and peripherals. So for equivalent throughput, the multiprocessor configuration requires one additional CPU and MCU, but eliminates the redundant three PPUs, four SSUs, four buses, four sets of peripherals, four sets of power supplies, four system cabinets, etc. The multiprocessor will become even more attractive as the cost of CPUs and MCUs continues to drop with respect to the cost of other system components.

Bus protocols

The common bus is a number of parallel lines which may be grouped into

1. Four lines identifying the bus slot to which the current message on the bus is being routed.

Typical Multiported Memory Multiprocessor		
Cpus	Throughput (cpu equivalents)	Additional Throughput from Additional Cpu
1	1.0	
2	1.7	.7
3	2.1	.4
4	*	*
5	*	*
6	*	*
7	*	*

* =not practical

BTI 8000		
CPUs,MCUs,PPUs	Throughput (CPU Equivalents)	Additional Throughput from Additional CPU
1,1,1	1.0	
2,2,2	1.9	.9
3,3,2	2.7	.8
4,4,2	3.5	.8
5,5,2	4.3	.8
6,6,2	4.9	.6
7,7,2	5.5	.6

Figure 6—Throughput of multiprocessors.

2. Two lines identifying the type of message currently on the bus.
3. 32 data lines containing the message.
4. A number of control lines which allow the resource units to resolve bus contention.

All traffic on the bus takes place in terms of messages between a source and a destination resource unit. Furthermore, all resource units use the same types of messages.

There are three message types—command, data and abnormal data. A command message is a request from one resource unit to another to carry out a specified function. A data message has 32 bits of data on the data lines (e.g., a word read out of memory by an MCU and being sent to a CPU). An abnormal data message is sent instead of a data message if an error occurs in trying to provide data. In this case the data lines contain error information.

A command message has a type code on the four high-order data lines, has an address or request code on the lower 22 lines, and leaves the other six lines unused. There are five command types—information request (known as "who are you"), read, write, read/modify/write, and self test (which starts microcode diagnostics).

Depending upon the request code, an information request can ask a resource unit to return a data message identifying its resource unit type, interrupt status, bus error status, error count, etc. This could then be used, for instance, by the SSU to determine the system configuration upon reload (e.g., how many CPUs are now on-line and occupying which bus slots).

A read command specifies the 22-bit address of the requested word to the destination resource unit. The destination responds with either a data message containing the



Photograph 1—BTI 8000 CPU board and chassis for up to 16 resource units.

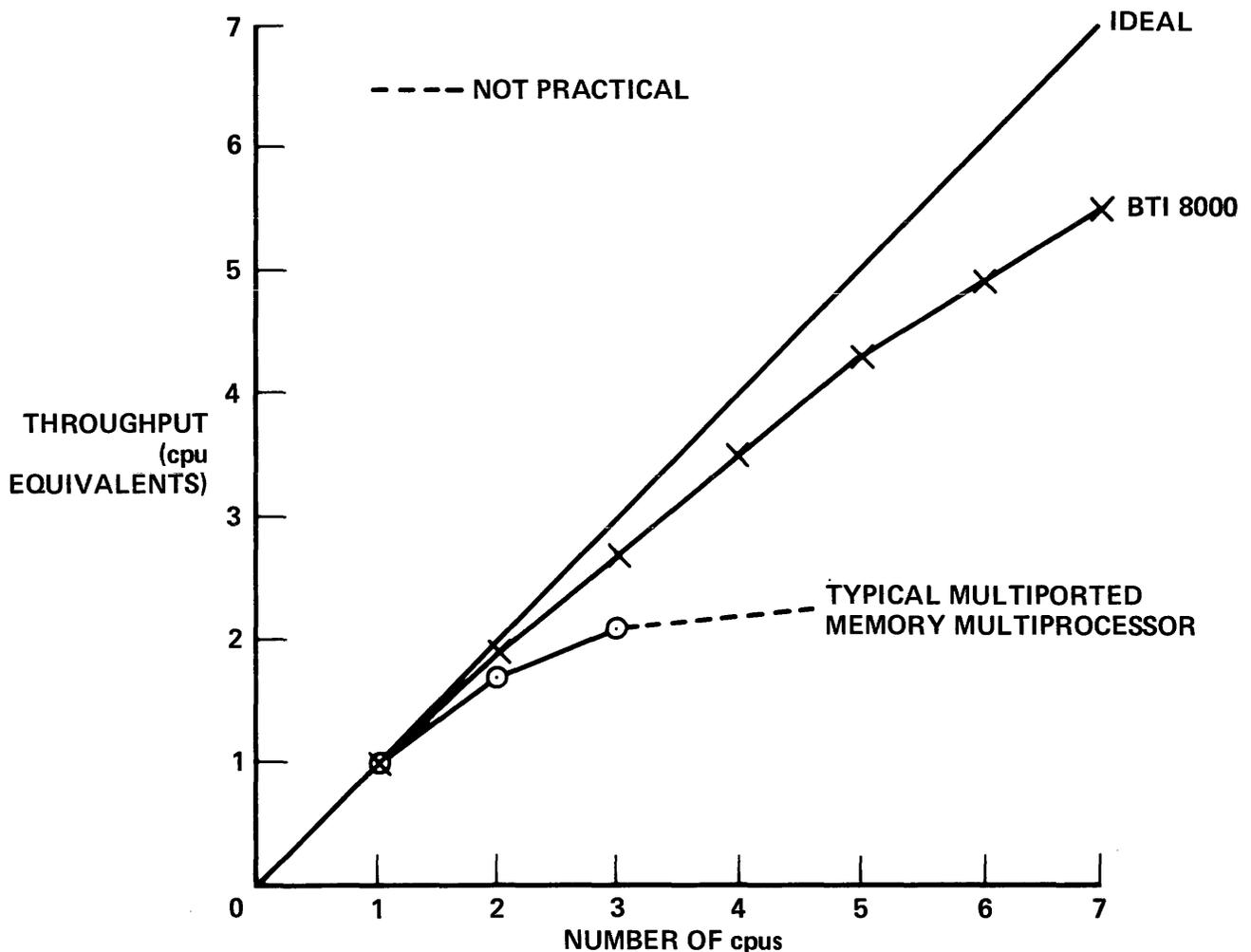


Figure 7—Throughput graph for multiprocessors.

contents of the requested word or an abnormal data message containing error information.

A write command specifies the 22-bit address to be stored into the destination unit. This is followed by a data message containing the word to be stored.

A read/modify/write command is only sent to an MCU. The MCU returns the contents of the addressed word in memory in a data message and then waits for the source resource unit to send another word to be written back into the same location. In the interim the MCU denies other access to that memory location. This is the hardware feature which makes process interlock mechanisms possible on the 8000.

Computational processing unit

Because the system relies upon multiple CPUs working concurrently to achieve high throughput, the CPU design was purposefully kept straightforward and inexpensive. Each CPU is an identical microprogrammed processor with

a program state described by eight 32-bit general-purpose registers, a 17-bit program counter, a 17-bit current console area register, a 15-bit process status register, a 32-bit monitor status register and a page map of 256 20-bit words.

The current console area register points to the 10-word block of memory which stores the user's program state during an interrupt. This register, instructions which load and store the entire console area, and interrupt firmware which stores the area automatically all enhance the speed of context switching.

The process status register contains flags for arithmetic faults and condition bits set by compare operations.

The monitor status register contains information accessible only to the monitor, such as the processor number (i.e., the bus slot the processor is plugged into), interrupt enabling flags and paging control flags.

Every virtual memory address is transformed into a physical address by a calculation involving the page map. The page map is divided into two 128-word halves, one for the monitor and the other for the current user. Which half is used is controlled by bits in the monitor status register.

Memory is divided into pages of 1024 words. Of a 17-bit virtual address, the upper seven bits select a word in the page map and the lower 10 bits a word within the physical page. The physical page is determined by the selected word of the page map. Of this 20-bit word, 12 bits determine the physical page and four bits the bus slot. This scheme allows for many MCUs, each containing 2^{22} ($22=12+10$) words of memory, and means that PPU's and MCUs are addressed identically. The remaining four bits in the page map word determine the legal types of access to this page (e.g., execute only, read only, read/write) and note whether the page has been read or written.

Memory control unit

An MCU is a "slave" resource unit in that it operates only in response to commands from other units, i.e., it does not originate commands itself. Its microprocessor selects one of two memory devices, handles error conditions occurring in memory and locks out other references to the MCU during a read/modify/write cycle (discussed under "Bus Protocols"). The data paths within an MCU are 32 bits wide.

Peripheral processing unit

A PPU handles up to four I/O controllers, two for high-speed devices (disk or high-speed tape drives) and two for slow devices (e.g., low-speed tape drives, line printers, and interactive terminals). Data transfers between a PPU and its controllers occur in eight-bit bytes and are buffered by FIFO queues.

A CPU initiates I/O by writing control information into the appropriate PPU. Some of this information is passed on to the controller being addressed. Other information (e.g., the MCU memory starting address for a block transfer) is stored in the local RAM of the PPU's microprocessor. Once a DMA (direct memory access) transfer is begun between the MCU and PPU, the originating CPU is free to do other tasks until completion of the transfer.

The disk and communications controllers are also microprocessor-based.

System services unit

The SSU performs a number of functions which need only be done in one place in the 8000. Among these are (1) providing the system-wide clock for synchronizing bus operations, (2) providing a real-time clock readable by other resource units, (3) sensing abnormal environmental conditions (e.g., temperature, humidity), (4) providing a local system status and reload capability via an extremely simple front panel and (5) providing remote diagnostic and preventive maintenance capabilities for BTI personnel via a dial-up communications port.

INSTRUCTION SET ARCHITECTURE

The instruction set processor of the CPU was designed to (1) keep bus traffic due to instruction fetches to a minimum and (2) to enhance the efficiency of the operating system, system utilities, compilers and compiler-generated code. These two goals have led to a rather high-powered instruction repertoire—there are 10 data types, approximately 200 different operation codes, and about 50 addressing modes.¹²

Data types

Data types supported include 32- and 64-bit fixed point numbers, 64-bit floating point numbers, eight-bit ASCII characters, one-bit Boolean values, 32-bit pointers, one- to 32-bit bytes, arrays of all of the preceding types, linked lists and pushdown stacks. Arithmetic is two's complement. There is a special "undefined number" value (a leftmost one bit followed by 31 or 63 zero bits) which may cause an error trap when used. A byte may cross the boundary between two words.

Instruction set

All instructions comprise one 32-bit word in which the leftmost 10 bits define the operation code, sometimes specifying a general register to be used, and the rightmost 22 bits specify the operand. The instruction set is fairly complete, providing most useful variants of each instruction, e.g., reverse subtract and reverse divide.

As a precautionary measure to prevent runaway programs, operation codes which arise in common data words are illegal. These include words containing all zeros or all ones, containing the undefined number, or starting with an ASCII space.

Arithmetic, Boolean, and compare instructions all have both register-to-register and register-to-memory modes. Also, certain arithmetic operations store the result into both a register and memory. Arithmetic reverse subtraction and division, as well as Boolean subtraction and reverse subtraction, are provided.

To support process synchronization, a number of instructions can read and write a word of memory in one indivisible step. In this class of instructions are the four Boolean operations to memory, the set and test operation, an operation which exchanges a register with a word in memory, and the seven single-word fixed point add and subtract to memory instructions.

Instructions to enhance systems software

A number of instructions simplify common tasks done by systems software. Two instructions jump based on the value of a particular bit of a register. Other instructions store in memory useful constants such as 0, 1, -1, and the "unde-

defined" value. One instruction jumps to itself without requiring any subsequent instruction fetches from memory, effectively causing the CPU to do nothing until interrupted.

The permutation operation computes the exclusive OR of those words in a contiguous 32-word block of memory which are indicated by a one in the corresponding bit position of a register. Thus, the bits in a register may be permuted according to a pattern defined in the 32-word block of memory. A checksum of a block of memory is computed by initializing the controlling register to all ones. By loading the register with a data word and memory with 32 words containing the integer 1, the parity of the register is computed.

Operations on linked lists

Linked lists are supported by instructions which advance to the next element of a list, testing for the end of the list. Both one-way and two-way lists may be used.

A one-way list consists of a number of list *elements*, or blocks of memory of arbitrary size. The address field (rightmost 17 bits) of the first word of each element contains the address of the successor element in the list, or zero if there is none. A pointer to the current place in the list is kept in a register. If a list is to be left unmodified, this is all the information necessary to traverse it. If, however, elements are to be inserted or deleted, an optional pointer to the predecessor of the current element is maintained in an adjacent register.

In place of the address of its successor, an element of a two-way list has the exclusive OR of the addresses of its left and right neighbors.²² Call this $l \oplus r$. An important property of exclusive OR is that $(l \oplus r) \oplus l = r$ and $(l \oplus r) \oplus r = l$. Thus, given the addresses of an element and one neighbor, the address of the other neighbor can be calculated. So the instruction to move ahead in a two-way linked list *requires* that pointers to both the current and previous elements be in adjacent registers. An example Exclusive OR Link and Jump on Non-Zero Address instruction is

```
HERE: XLJNA i THERE
```

The effect of executing this instruction is defined by the following program, and is shown graphically in Figure 8.

```
temp ← registeri+1;
registeri+1 ← registeri;
registeri ← temp ⊕ contents (registeri+1);
if registeri ≠ 0 then pc ← THERE;
```

Also, an instruction is available which searches a one-way linked list for an element containing a specified key. The key may be a word, character, or byte which is offset a fixed amount from the first word of the list element, as specified in the instruction. One register is loaded with a pointer to the first element to be examined. After a successful search, it points to the desired element. Another register holds the key being searched for.

Subroutine linkage instructions

A group of 27 instructions is provided for subroutine linkage, including parameter passing and type checking and ensuring that a legitimate subroutine is called. The following example shows a typical pair of calling and entering sequences:

```

:
C: CALL S      S: ENTR REG7
   PAR  A1     STP  F1
   PAR  A2     STPV F2
   PAR2 A3     STP2 F3
   PARV A4     STPV F4
   PARL A5     STPL F5
C+6: :         : [subroutine body]
                        LEAVE REG7
```

Here the main program on the left calls subroutine S, passing it five actual parameters, A1 through A5. After type checking and storing the parameters into local locations F1 through F5, the body of the subroutine is executed, and then control is returned to the main program at C+6.

These instructions use general registers R0 for passing single-word parameters, R0 and R1 for double-word parameters and R7 for holding the next address and parameter specifications.

The CALL Subroutine instruction ensures that the instruction located at S is some type of Enter Subroutine (ENTR being the simplest of these), saves the contents of R7 in memory location REG7 and puts S+1 in R7.

The PARAMeter instruction at C+1 stores its type (in this case, single-word call by reference) along with C+2 in R7, loads R0 with the address A1, and jumps to the address previously in R7 (S+1).

The STP Store Parameter instruction at S+1 ensures that its desired type is the same as that of the actual parameter, as defined in R7 (it is). Then the contents of R0 (A1) are stored in F1, S+2 put in R7, and control returned to C+2.

The rest of the corresponding Parameter and Store Parameter instructions are executed in interleaved order until a Store Parameter Last (STPL) is encountered. This instruction marks the last parameter, so control passes into the body of the subroutine instead of returning to C+6 for more parameters.

At the end of subroutine execution, the LEAVE Subroutine instruction returns to C+6 (assumed to still be in R7) after restoring R7 from REG7.

Suffixes "2" and "V," singly or together, on PAR and STP operation codes define double word and call by value types. For example, PARV2 A6 will load the 64-bit value located at A6 and A6+1 into R0 and R1. There are three other type bits available to distinguish, e.g., fixed and floating point operands.

Some type coercion is possible. The PAR at C+2 is call by reference, while the corresponding STPV at S+2 specifies call by value. The STPV is smart enough to load indirect through R0 to get the value stored at A2.

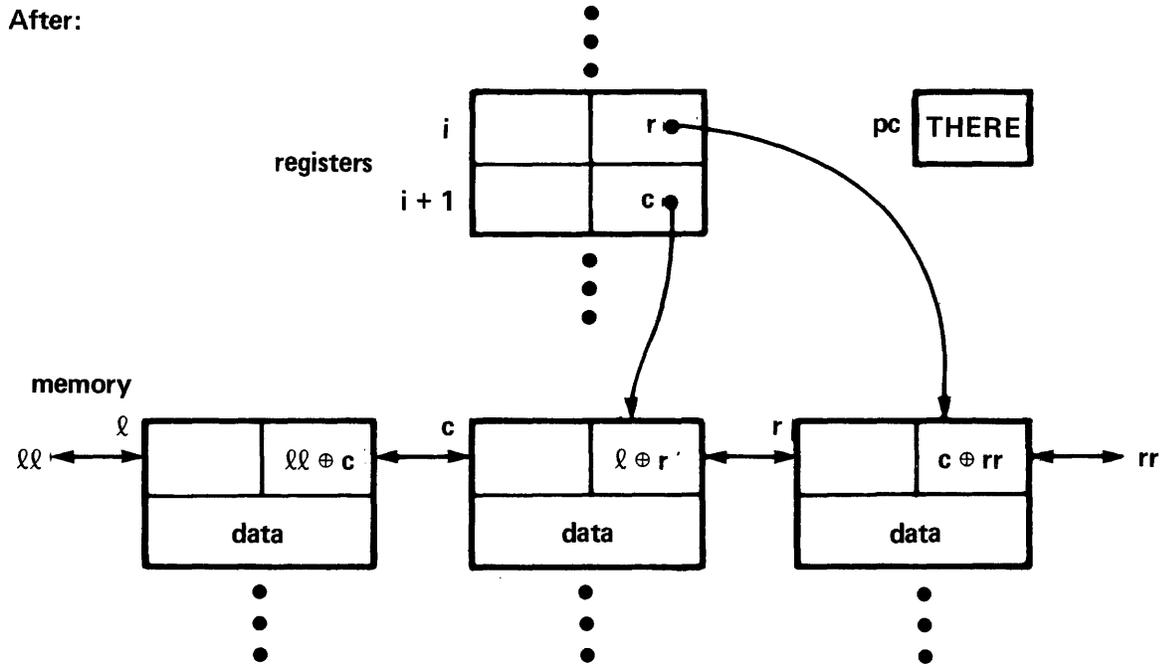
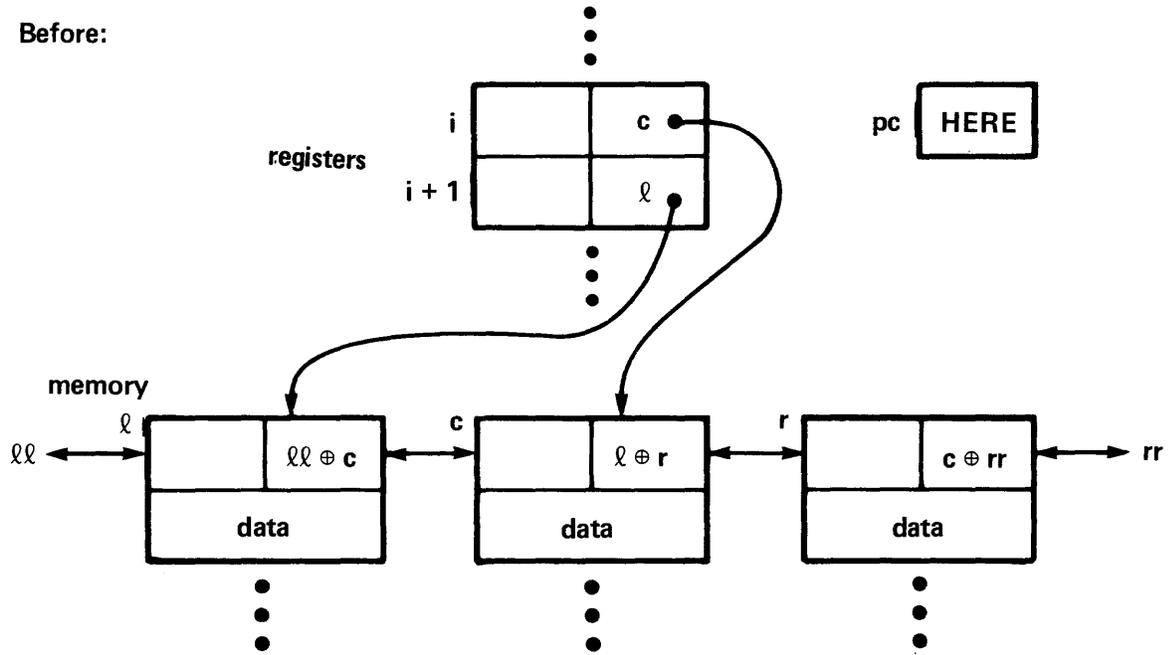


Figure 8—Registers and memory snapshots of "Exclusive OR Link and Jump on Non-Zero Address" instruction.

Instructions for compiled code

To improve compiler efficiency, an instruction exists to load an effective address. It performs the address calculation done for a load, but loads a register with this computed address instead of the value contained there.

Four instructions trap if an array index is out of prescribed

bounds. The instructions handle single word integer, double word integer and floating point indices.

Addressing modes

Much of the power of the BTI 8000 instruction set arises from its wealth of ways to address data. As discussed in the

previous section, each 32-bit instruction has its operation code in the upper 10 bits. This leaves 22 bits to define the way in which the operand, operands and/or the location to store the result will be determined. Two instruction classes don't use the addressing modes. Jump instructions use the lower 22 bits as a five-bit number defining which bit to test and a 17-bit jump address. Character instructions have a five-bit extended operation code, and the Character Fill instruction also has an eight-bit immediate operand.

Completely separating designation of the operation codes from the address modes implies that any operation may utilize any address mode which is well defined for that operation. For example, in a single instruction one could

add to a register an integer which is stored as a three-bit byte, split across the boundary of two words of memory. For a 32-bit add, the three-bit byte would be right justified and padded with 29 zero bits; for a 64-bit add, an additional word of padding would be provided. Conversely, storing a register into such a three-bit byte location in memory would only use the rightmost three bits of the register.

Addressing mode formats

There are six different formats for addressing modes (Figure 9). The format, as well as the specific address calculation

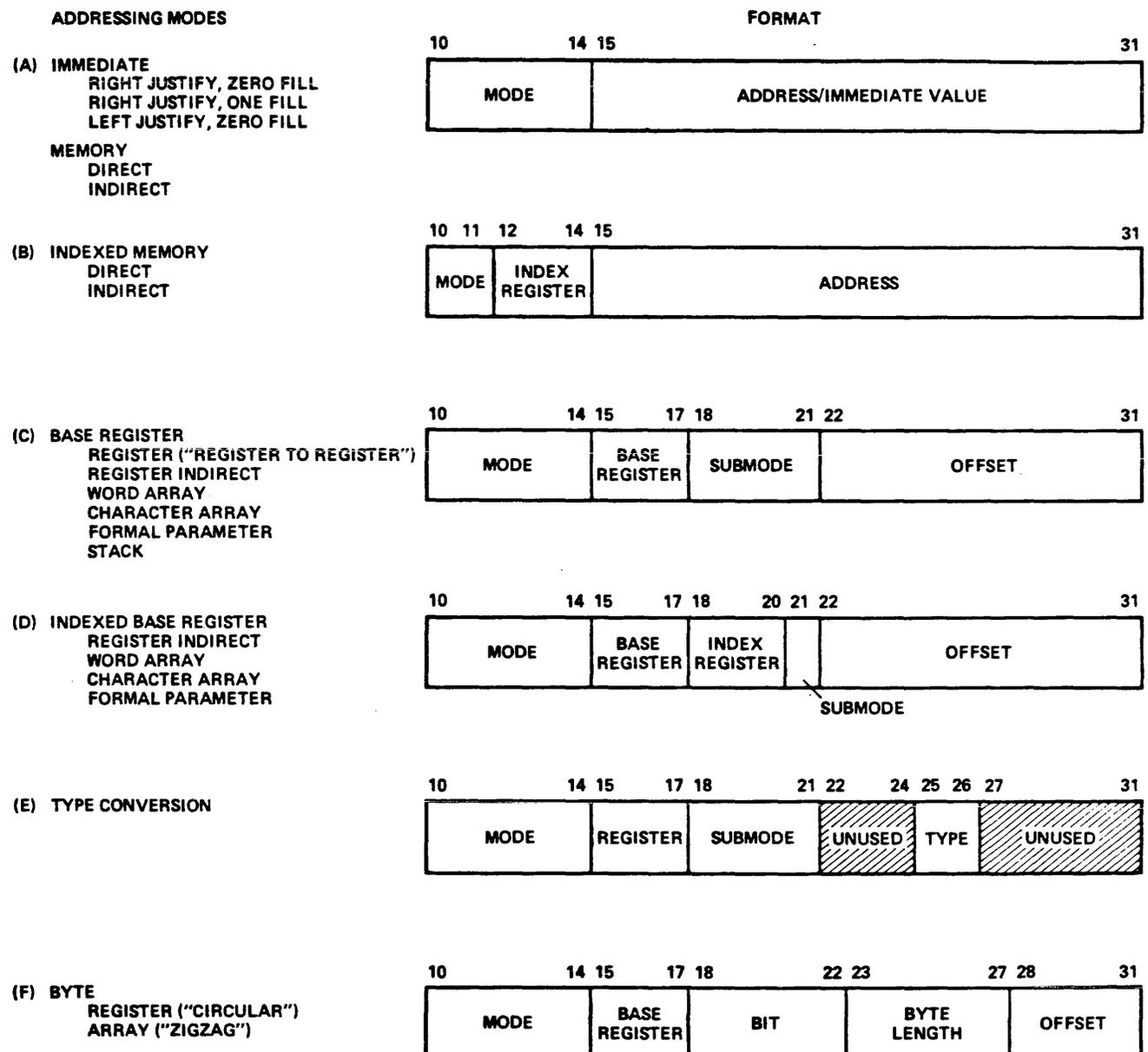


Figure 9—Instruction addressing modes and formats.

to be done, is determined by the mode field of the instruction.

Address mode format A is the simplest one. It is used for direct and indirect access of the memory word specified in the address field. In addition, the address field may be used as an immediate operand in three ways—right-justified with zero fill, right-justified with one fill, and left-justified with zero fill.

Format B specifies a register to be used to index the result of a direct or indirect address calculation. For instructions dealing with 64-bit operands and results, the index register is added twice.

Format C specifies a base register and a constant index or offset. Depending on the submode, six calculations are possible. The simplest of these uses the specified register directly, adding the offset to the contents of the register to form an operand. When addressing a result, the offset is subtracted from the result before the result is stored in the register. The second submode uses the register as an indirect pointer. The next three submodes use the register to point to the base of an array and the offset to specify the desired array element. The array can be of words, characters, or formal parameters, which are indirect pointers to the actual parameters of a procedure. The sixth submode uses the register as a pointer to the top of a stack (pushdown list) growing downward from high memory addresses to low. The register is decremented by the offset prior to storing a result, thus pushing the stack. After fetching an operand from the top of the stack, the register is incremented by the offset to pop the stack. For an instruction which loads an operand and stores a result, the stack is both popped and pushed.

Format D has four calculations analogous to format C, but with the added capability of indexing with a register the final address calculated. An extra mode is used because the submode field is limited to one bit to accommodate the index register field.

Format E specifies type conversion to be done upon loading or storing one or two consecutive registers. For loading an operand from the register(s), three submodes specify converting either a 32- or 64-bit integer into either a 64-bit integer or floating point quantity. The register(s) are unchanged. When storing a result into the register(s), the inverse conversion is done. The type field has bits specifying whether integers are considered signed or unsigned and whether to round or truncate upon conversion.

Format F is used for two byte-accessing modes which allow the 8000 to omit shift instructions. In the first the

register is considered to be a circular list of bits, and a byte is referenced starting at the specified bit and extending for the specified length. The offset is unused in this mode. The second mode uses the register as the bit address of a bit array. The desired byte in this array starts with the bit indexed by 32 times the offset plus the bit field. The size of the byte is specified by the byte length field. The byte may cross a word boundary, hence the name "zigzag byte" for this addressing mode.

Format of pointer words

Words used as pointers to fields in memory use the format shown in Figure 10. A base or index register for a word array uses only the address field. A base or index register for a character array in addition uses the character field to select a particular character of the word addressed. The base register for a bit array also uses the bit field to specify a bit within the character.

The mode field is used in pointer words in a register or memory which are specified by instructions in an indirect addressing mode. Four of the pointer's own modes specify direct addressing and three forms of immediate operands, as discussed earlier under address mode format A. In these modes the character, bit and byte length fields are unused. The fifth pointer mode uses the character field to select a character in the word addressed. The sixth pointer mode references a word of memory and a zigzag byte within that word. The byte starts at the bit specified by the bit field within the character specified by the character field. Indexing a pointer in zigzag byte mode causes the index register to be multiplied by the length field before being added in, thus allowing indexing of an array with elements of arbitrary byte size (not greater than 32 bits).

IMPLEMENTATION

Little has been said about the implementation of the 8000. This has been deliberate. Because memory and logic circuits are continuing their well established trend toward smaller size, lower cost and higher performance, the design philosophy has been to separate specific implementation decisions from the system architecture as much as possible. Thus, a future implementation of a system component should be able to take advantage of advances in integrated circuit technology.

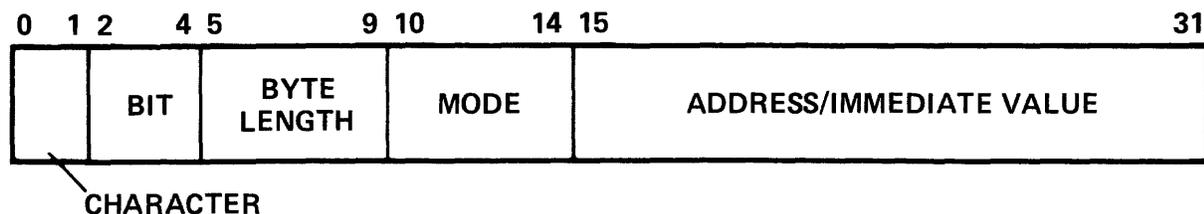


Figure 10—Format of pointer words.

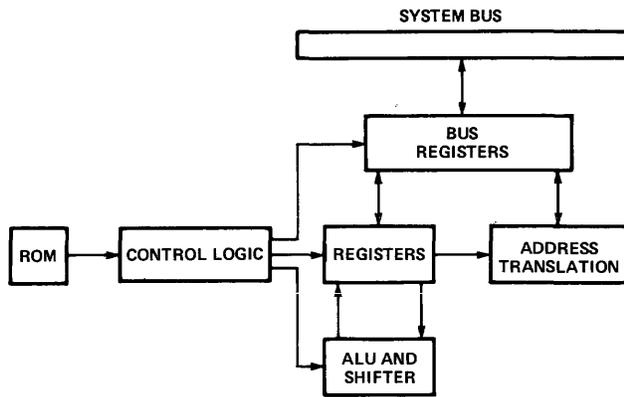


Figure 11—Computational processing unit.

Especially important in this regard was the decision to make every resource unit be a microprogrammable computer with its own internal microprocessor, memories, data paths and I/O connections (current architecture shown in Figures 11-14). From the viewpoint of the resource unit, these I/O connections include the common system bus plus main memory (in the case of an MCU), standard I/O controllers (in the case of a PPU) and special devices such as a thermometer (in the case of the SSU). Each of these computers may now be treated as possessing a flexible ar-

chitecture, easily changed when desirable, as long as the I/O characteristics of each computer do not change.

SYSTEMS SOFTWARE

A user of the BTI 8000 is presented with a virtual machine which is independent of a particular physical configuration.^{13,14} For example, if an additional CPU is required to meet increasing system load, it is merely plugged into the bus and the operating system is reloaded at the push of a single button. The system automatically recognizes the hardware reconfiguration; no user reprogramming is required. In fact, a user never knows—and need not care—on which CPU his program was run. Similarly, a failing CPU in a multiprocessor configuration is merely unplugged from the bus, followed by the same reload. The only difference noticed by a user may be a degradation of response time. Because of virtual memory mapping, physical memory can be similarly reconfigured without affecting the user's access to 131,072 (128K) words (512K bytes) per user process.

The 8000 is designed to facilitate the use of high-level languages, including PASCAL, FORTRAN, BASIC, COBOL and RPG. The use of vendor-supported PASCAL is promoted because of its applicability to structured programming and the specification of concurrent processes. Unlike uniprocessor systems, a software process on the BTI

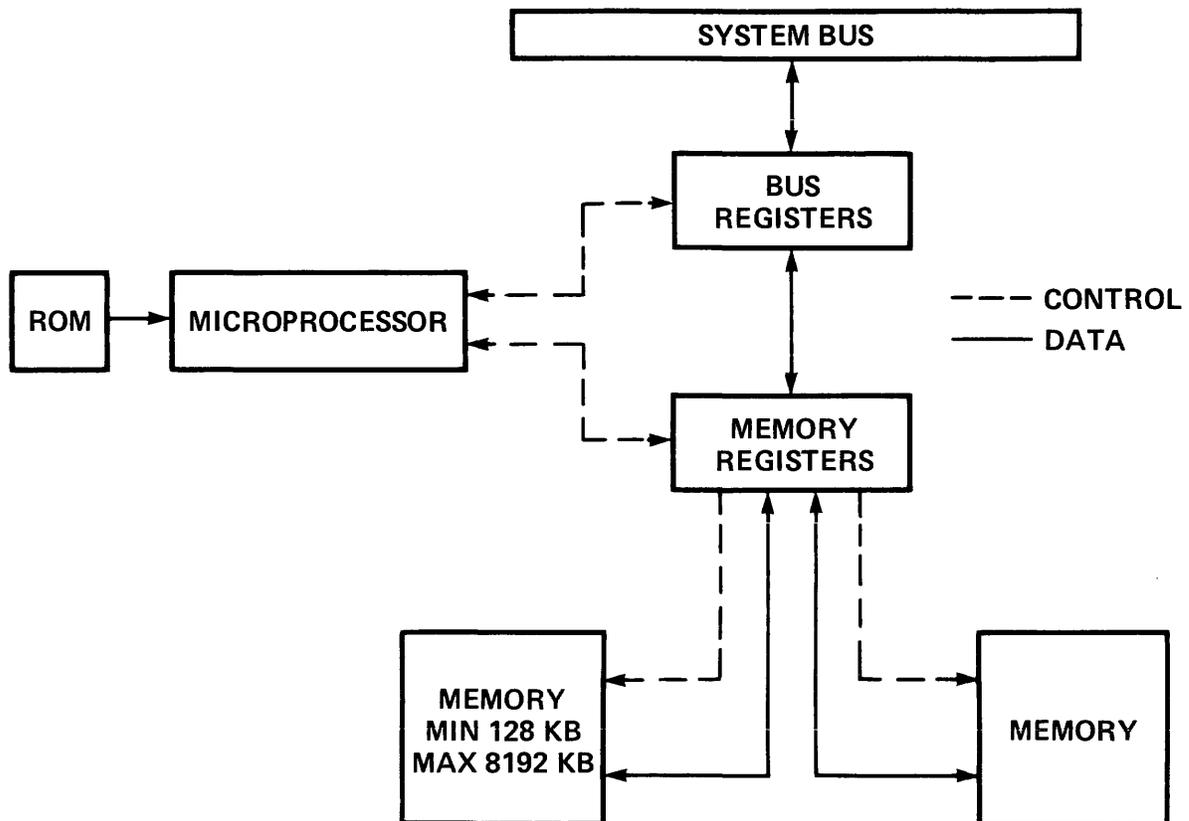


Figure 12—Memory control unit.

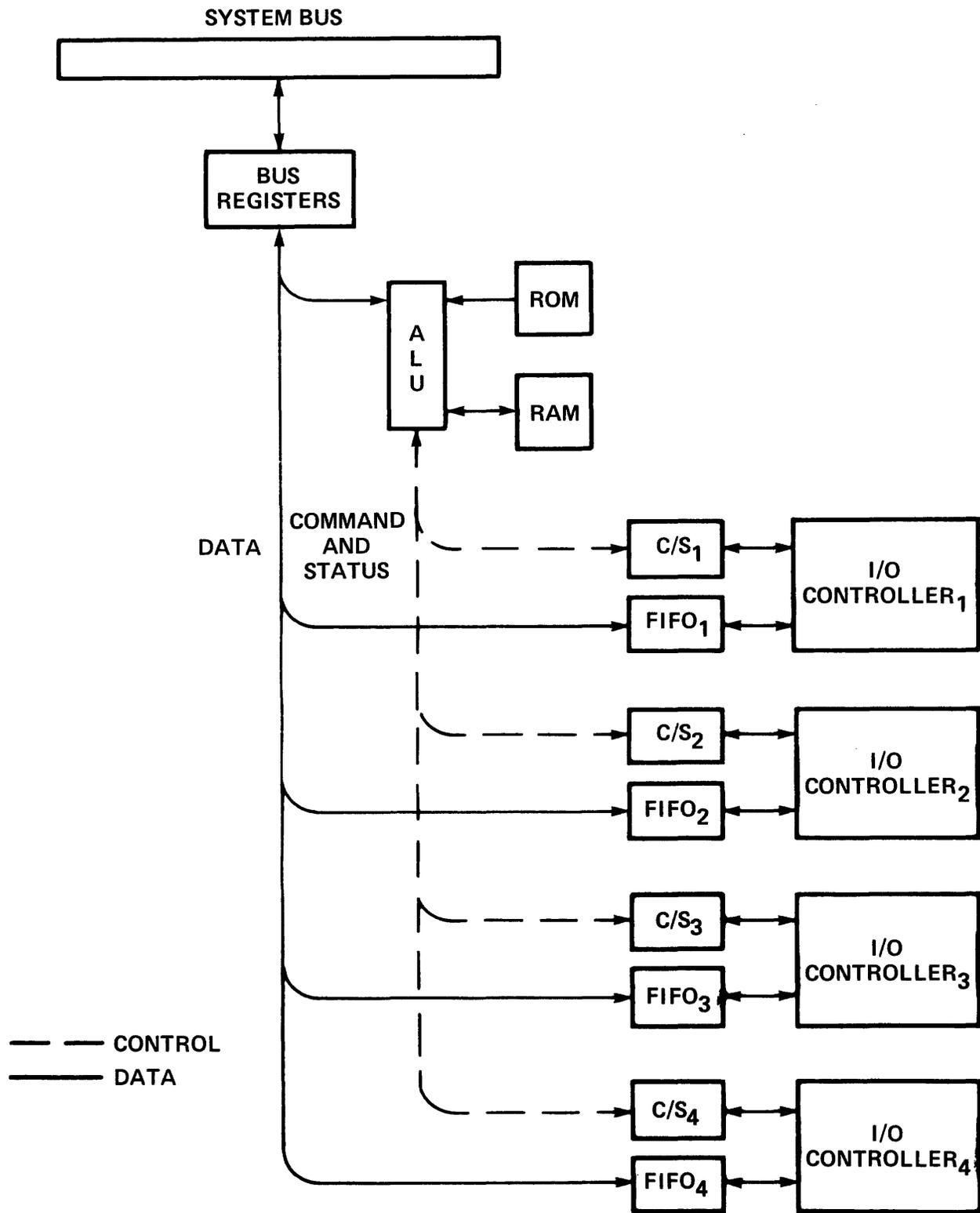


Figure 13—Peripheral processing unit.

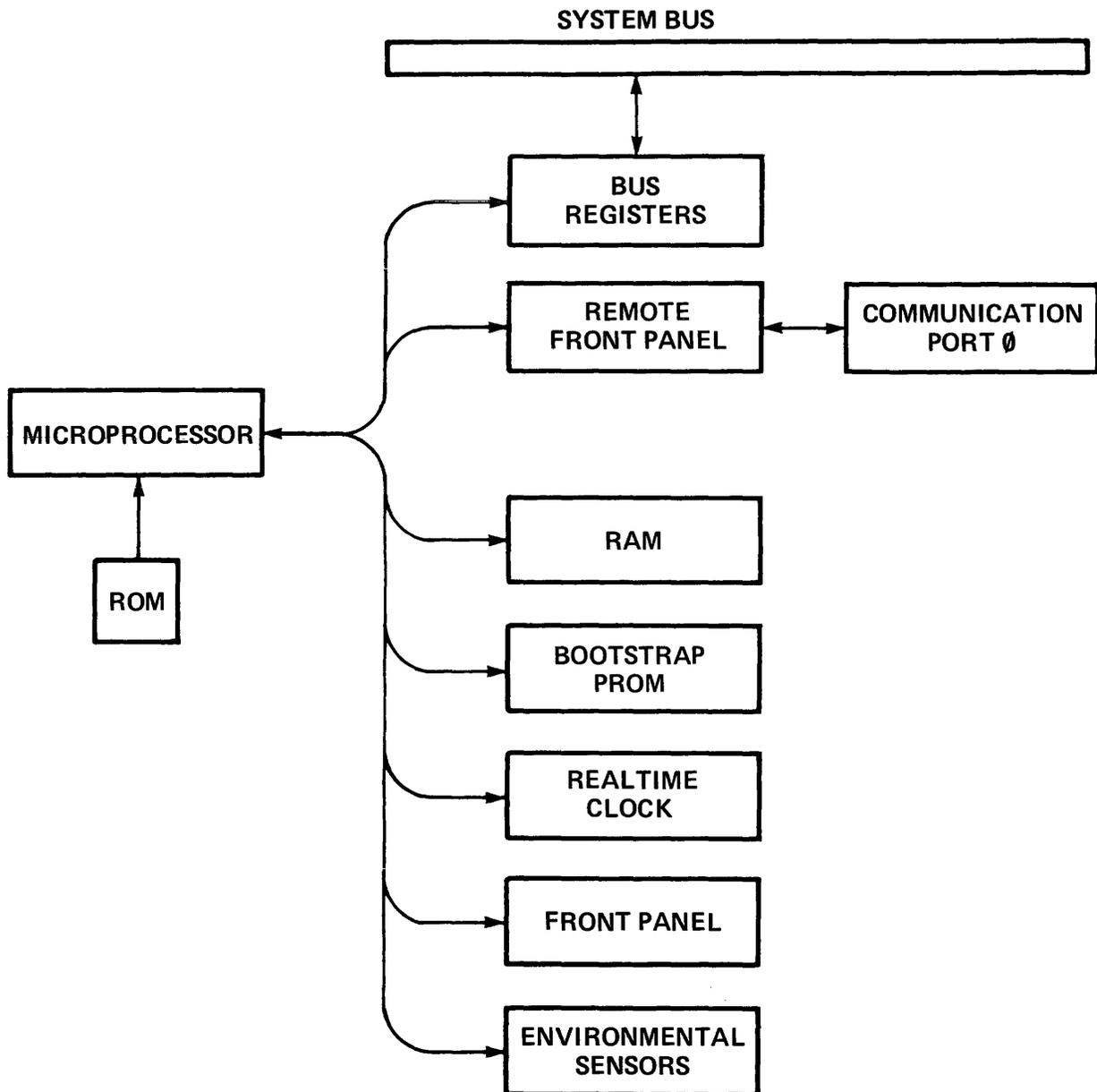


Figure 14—System services unit.

8000 may actually be run concurrently, communicating via shared synchronization regions which are kept in main memory. To promote the use of advanced high-level languages, the PASCAL compiler is the only one bundled with the hardware. By contrast, the assembler is available only to educational institutions for instructional purposes.

POTENTIAL PROBLEMS

The high reliability and availability of the BTI 8000 depend strongly on the unique elements of the system. For example, each system has only one active SSU, which includes the

system clock. However, a spare SSU can be kept ready to go online if the primary SSU fails.

There is only one bus, the failure of which would be catastrophic. By distributing all of its active logic to each of the connected resource units, most of the potential for such a failure is alleviated. However, distributing the bus logic to resource units means that each such device is more complicated.

Software complexity is usually higher on multiprocessors than uniprocessors. Most of the added complexity has been handled in the 8000 instruction set and operating system. The issue will be further ameliorated by the availability within high-level languages of a capability for writing concurrent processes.

A final problem is the continual desire for larger and larger contiguous addressing spaces for user programs. Because each machine instruction takes one 32-bit word and many bits were allocated to the operation codes and addressing modes of the 8000, virtual addresses are 17 bits. Thus, 128K words (512K bytes) are directly addressable by each user process. While this is actually a rather large address space for a minicomputer, it may be extended even further in the future. In addition, user tasks can be divided into a large number of communicating concurrent processes, each of 128K.

ACKNOWLEDGMENTS

We acknowledge the inspiration and hard work that the BTI Research and Development group has contributed to the 8000 project. We are especially grateful to the management of BTI Computer Systems for its continuing faith in a project of such high risk. Robert Adams, Ron Crandall, Bill Quackenbush, and Don Quaintance provided important details of the 8000 and advice on their presentation during the writing of this paper. Finally, special thanks are due Carol Hjerpe for countless hours spent preparing this document.

REFERENCES

- Amdahl, G. M., G. A. Blaauw and F. P. Brooks, Jr., "Architecture of IBM System/360," *IBM Journal of Research and Development*, Vol. 8, No. 2, April, 1964, pp. 87-101.
- Anderson, George A., and E. Douglas Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," *COMPUTER SYSTEMS ARCHITECTURE*, *Computing Surveys*, Vol. 7, No. 4, December, 1975, pp. 197-213.
- Anderson, James P., Samuel A. Hoffman, Joseph Shifman and Robert J. Williams, "D 825: A Multiple Computer System for Command and Control," 1962 Fall Joint Computer Conference, *AFIPS Conference Proceedings*, Vol. 22, Spartan Books, Washington, D.C., 1962, pp. 86-96 (reprinted in Reference 7, Chapt. 36, pp. 447-455).
- Barker, W. B., *A Multiprocessor Design*, Ph.D. thesis, Harvard University, Report 3126 Bolt Beranek and Newman Inc., Cambridge, Massachusetts, October, 1975.
- Baskin, Herbert B., Barry R. Borgerson and Roger Roberts, "PRIME: A Modular Architecture for Terminal Oriented Systems," 1972 Spring Joint Computer Conference, *AFIPS Conference Proceedings*, Vol. 40, AFIPS Press, Montvale, New Jersey, May, 1972, pp. 431-437.
- Bell, C. Gordon, "What Have We Learned from the PDP-11?," in *Perspectives on Computer Science*, Anita K. Jones (ed.), Academic Press, Inc., New York, New York, 1977, pp. 7-38.
- Bell, C. Gordon, and Allen Newell, *Computer Structures: Readings and Examples*, McGraw-Hill Book Company, Inc., New York, New York, 1971.
- Bell, G., R. Cady, H. McFarland, B. Delagi, J. O'Laughlin, R. Noonan and W. Wulf, "A New Architecture for Minicomputers: The DEC PDP-11," 1970 Spring Joint Computer Conference, *AFIPS Conference Proceedings*, Vol. 36, AFIPS Press, Montvale, New Jersey, May, 1970, pp. 657-675.
- Bell, C. G., A. Kotok, T. N. Hastings and R. Hill, "The Evolution of the DECsystem-10," Special Issue on Computer Architecture, *Communications of the ACM*, Vol. 21, No. 1, January, 1978, pp. 44-63.
- Blaauw, G. A., and F. P. Brooks, Jr., "The Structure of System/360, Part 1: Outline of the Logical Structure," *IBM Systems Journal*, Vol. 3, No. 2, 1964, pp. 119-135 (reprinted in Reference 7, Chapt. 43, pp. 588-601).
- Model 8000 Multiprocessor System Theory of Operations and Technical Summary*, preliminary, BTI Computer Systems, Sunnyvale, California, September, 1978.
- Model 8000 Multiprocessor System CPU Instruction Reference Manual*, preliminary, BTI Computer Systems, Sunnyvale, California, November, 1978.
- Model 8000 Multiprocessor System Virtual Machine Reference Manual*, preliminary, BTI Computer Systems, Sunnyvale, California, 1979.
- Model 8000 Multiprocessor System Control Mode Reference Manual*, preliminary, BTI Computer Systems, Sunnyvale, California, 1979.
- Case, Richard P., and Andris Padegs, "Architecture of the IBM System/370," Special Issue on Computer Architecture, *Communications of the ACM*, Vol. 21, No. 1, January, 1978, pp. 73-96.
- Chang, Donald Yi-Chung, *Further Results Regarding Multiprocessor Systems*, Ph.D. thesis, Report UIUCDCS-R-77-908, Department of Computer Science, University of Illinois, Urbana, Illinois, October 1977, p. 18.
- Cohen, Ellis, "Symmetric Multi-Miniprocessors: A Better Way to Go?," *Computer Decisions*, Vol. 5, No. 1, January, 1973, pp. 16-20.
- Enslow, Philip H., Jr., "Multiprocessor Organization: A Survey," *PARALLEL PROCESSORS AND PROCESSING*, *Computing Surveys*, Vol. 9, No. 1, March, 1977, pp. 103-129.
- Flynn, Michael J., "Very High Speed Computing Systems," Special Issue on Computers, *Proceedings of the IEEE*, Vol. 54, No. 12, December, 1966, pp. 1901-1909.
- Heart, F. E., S. M. Ornstein, W. R. Crowther and W. B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network," 1973 National Computer Conference and Exposition, *AFIPS Conference Proceedings*, Vol. 42, AFIPS Press, Montvale, NJ, June 1973, pp. 529-537. Reprinted in *Advances in Computer Communications*, Wesley W. Chu (ed.), revised edition, Artech House, Delham, MA, 1977, pp. 397-405; *Computer Communication Networks*, R. L. Grimsdale and F. F. Kuo (eds.), Noordhoff International Publishing, Leyden, The Netherlands, 1975, pp. 159-180; *Computer Communications*, Paul E. Green, Jr., and Robert W. Lucky (eds.), IEEE Press, New York, NY, 1975, pp. 366-374.
- Katzman, James A., "A Fault-Tolerant Computing System," in "Selected Papers in Mini and Micro Computer Systems," *Proceedings of the Eleventh Hawaii International Conference on System Sciences*, Bruce Shriver, Richard Eckhouse and Ralph H. Sprague, Jr. (eds.), Vol. 3, Western Periodicals Co., Honolulu, January 1978, pp. 85-102.
- Knuth, Donald E., "Fundamental Algorithms," *The Art of Computer Programming*, Vol. 1, Second Edition, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1973, pp. 277, 550.
- MacKinnon, R. A., "Advanced Function Extended with Tightly Coupled Multiprocessing," *IBM Systems Journal*, Vol. 13, No. 1, 1974, pp. 32-59.
- Newell, Allen, and George Robertson, "C.mmp: A Progress Report on Synergistic Research," in *Perspectives on Computer Science*, Anita K. Jones (ed.), Academic Press, Inc., New York, 1977, pp. 147-182.
- Olson, J. W., "Architecture of the Central Logic and Control, Safeguard Data Processing System," *Bell System Technical Journal*, special supplement, 1975, pp. S41-S61.
- Ornstein, S. M., W. R. Crowther, M. F. Kraley, R. D. Bressler, A. Michel and F. E. Heart, "PLURIBUS: A Reliable Multiprocessor," 1975 National Computer Conference, *AFIPS Conference Proceedings*, Vol. 44, AFIPS Press, Montvale, New Jersey, May, 1975, pp. 551-559.
- Stanga, D. C., "Univac 1108 Multiprocessor System," 1967 Spring Joint Computer Conference, *AFIPS Conference Proceedings*, Vol. 30, Thompson Books, Washington, D.C., April, 1967, pp. 67-74.
- Stevens, W. Y., "The Structure of System/360, Part 2: System Implementations," *IBM Systems Journal*, Vol. 3, No. 2, 1964, pp. 136-143 (reprinted in Reference 7, Chapt. 44, pp. 602-606).
- Thurber, Kenneth J., E. Douglas Jensen, Larry A. Jack, Larry L. Kinney, Peter C. Patton and Lynn C. Anderson, "A Systematic Approach to the Design of Digital Bussing Structures," 1972 Fall Joint Computer Conference, *AFIPS Conference Proceedings*, Vol. 41, Part 2, AFIPS Press, Montvale, New Jersey, December, 1972, pp. 719-740.
- Wittmayer, Woodrow R., "Array Processor Provides High Throughput Rates," *Computer Design*, Vol. 17, No. 3, March, 1978, pp. 93-100.
- Wulf, William A., and C. G. Bell, "C.mmp: A Multi-Miniprocessor," 1972 Fall Joint Computer Conference, *AFIPS Conference Proceedings*, Vol. 41, Part 2, AFIPS Press, Montvale, New Jersey, December, 1972, pp. 765-777.

A survey of interconnection methods for reconfigurable parallel processing systems*

by HOWARD JAY SIEGEL, ROBERT J. MCMILLEN and PHILIP T. MUELLER, JR.

Purdue University
West Lafayette, Indiana

INTRODUCTION

This is a survey of a variety of interconnection networks for reconfigurable parallel processing systems that have appeared in the literature. A system is *reconfigurable* if it may assume several architectural configurations, each of which is characterized by its own topology of activated interconnections between modules.¹⁸ The systems whose networks will be examined include multiple-SIMD and MIMD systems, as well as both fixed and dynamic word size systems. This paper is restricted to networks for geographically-localized parallel processing systems using 12 or more processors in a reconfigurable manner. Related survey papers include References 1, 3, 10, 19, 20, 45-47.

The next section defines parameters that will be used to describe and evaluate networks. The later sections will discuss the interconnection networks, grouped by their overall structure—multistage switching networks, dedicated path networks, and shared path networks.

PARAMETERS

A variety of parameters which can be used to describe interconnection networks are briefly presented. Their purpose is to provide a common set of terms to use as a basis for the examination of the different networks; however, all parameters will not be applicable to all networks. It is assumed that a system has N processors and, if N is a power of two, $n = \log_2 N$.

Anderson and Jensen¹ define a *path* as "the medium by which a message is transferred between the other system elements" (e.g., wires or buses), and a *switching element* as "an entity which may be thought of as an 'intervening intelligence' between the sender and receiver of a message." Networks may be described by the type of switching elements used and the paths between switching elements. One classification is by the way the switching elements are

physically located with respect to the system processors. Two types are *distributed* and *centralized*. Networks may be used to connect processors and memories (*processor-to-memory*) or to connect processing elements (*PE*s) to other processing elements (*PE-to-PE*), where a *PE* is a processor-memory pair.

Reconfiguration method is the method used to reconfigure the network, i.e., to change the way in which submachines are organized. The *communications setup method* is the method used to establish an interprocessor communications path within an already existing submachine. *Delay* is the time it takes a network to transfer one data item from a source to the desired destination. The *ease of use* of a network is the degree to which connections are automatically established. The *cost* of a network is the asymptotic complexity of its implementation.

The *partitionability* of a network is its ability to divide the system into independent subsystems of different sizes. Partitionable systems may be characterized by any limitation on the subset of processors which may belong to a partition. Furthermore, a system may be *logically partitioned* using software techniques or *physically partitioned* using hardware switches within the network control structure. A network is *homogeneous* if it treats all processors similarly. *Modularity* is the ability of a network to be constructed from a small set of basic modules. *LSI compatibility* is the suitability of a module to be implemented as an LSI chip, i.e., high-circuit complexity and low external connection requirements. The *extensibility* of a network is its ability to be extended to a larger size, i.e., the amount of modification needed to make the network function for a larger number of inputs/outputs. *Fault tolerance* will be discussed in terms of a system's features which would allow the system to remain operational with faulty components (with possible degradation).

Let m be the number of processors which can transfer data simultaneously using the interconnection network. Then the degree of *simultaneity* supported by the interconnection network is $S = m/N$, $1 \leq m \leq N$. Permutations are one-to-one connections in which all processors participate. For networks with N inputs, N outputs, and $S = 1$, let r be the number of permutations possible in a single pass through an interconnection network. Then the *connectivity*

* This work was supported in part by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant No. AFOSR-78-3581. The United States Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

of the network is $C=r/(N!)$, $1 \leq r \leq N!$. The ability of a processor attached to the network to broadcast a single data item to all other processors can be measured by the *broadcast scope*. Let b be the maximum number of other processors which can receive data simultaneously from a given processor after one pass through the interconnection network. Then the broadcast scope is $B=b/(N-1)$. The *broadcast delay* is the number of transfers required for a complete broadcast. The *range* of a network can be measured by $R=x/(N-1)$, where x is the order of the set of processors (i.e., the number of processors) from which a single processor can choose to send data to in one pass through the network. The range can be further characterized by specifying the set of processors which can be sent data. Similarly the *domain* of a network can be measured by $D=x/(N-1)$, where x is the order of the set of processors a single processor can receive data from in one pass through the network, and can be further characterized by specifying the set of processors which can send the data. The networks discussed support SIMD, MSIMD, MIMD, or PSM parallelism. Furthermore, some support dynamic word sizes.

An *SIMD (single instruction stream—multiple data stream) machine*¹⁴ typically consists of a set of N processors, N memories, an interconnection network, and a control unit (e.g., Illiac IV⁸). The control unit broadcasts instructions to the processors and all active ("turned on") processors execute the same instruction at the same time. Each processor executes instructions using data taken from a memory to which only it is connected. The interconnection network allows interprocessor communication. An *MSIMD (multiple-SIMD) system* is a parallel processing system which can be structured as two or more independent SIMD machines (e.g. MAP²³). An *MIMD (multiple instruction stream—multiple data stream) machine*¹⁴ typically consists of N processors and N memories, where each processor may follow an independent instruction stream (e.g. C.mmp⁵⁰). As with SIMD architectures, there is a multiple data stream and an interconnection network. A *PSM (partitionable SIMD/MIMD) system*³² is a parallel processing system which can be structured as two or more independent SIMD and/or MIMD machines (e.g., PASM³⁵⁻³⁸).

There are two methods of achieving variable word sizes. The first method, *intraprocessor dynamic word size*, uses processors with long data words which can be split up to form several independent smaller data words (e.g. Illiac IV⁸). The second method, *interprocessor dynamic word size*, combines two or more processors with small data words to form a single processor with a long data word (e.g. Dynamic Computer¹⁸).

MULTISTAGE SWITCHING NETWORKS

Introduction

A *Multistage Switching Network (MSN)* is an interconnection network consisting of many (usually n) stages of

switches. Each stage is connected to the next by at least N paths. Each switch can choose from two or more input paths to connect to an output path. The multistage networks discussed in this section are all physically centralized and have simultaneity, range and domain $S=R=D=1$. All have a cost of $O(nN)$ and a transfer delay proportional to their number of stages. The switch elements are modular, but not complex enough for LSI. These MSNs are capable of exploiting pipelining to pass data through the network. For example, stage i could contain N w -bit registers, where w is the width of the network, and act as the i -th stage of the pipe^{38,39}. If a switch element fails, the network cannot perform completely without significant revision of data routing strategies and algorithms. Three parameters which are used to describe different multistage switching networks are topology, switch and control structure.³⁴

The *topology* of a multistage network is the actual interconnection patterns that are used to connect the stages of the network.³⁴ Interconnection functions specify these patterns. An *interconnection function*³⁰ is a bijection (permutation) on the set of input/output addresses, which consists of the integers from 0 to $N-1$. The interconnection function f connects input i to output $f(i)$, $0 \leq i < N$. There are n *Cube*-type interconnection functions:

$$\text{Cube}_i(p_{n-1} \dots p_1 p_0) = p_{n-1} \dots p_{i+1} \bar{p}_i p_{i-1} \dots p_0$$

where $p_{n-1} \dots p_1 p_0$ is the binary representation of P , $0 \leq P < N$, and $0 \leq i < n$.²⁸ There are $2n$ *PM2I (Plus-Minus 2)*-type interconnection functions:

$$\text{PM2}_{+i}(x) = x + 2^i \text{ modulo } N, \text{PM2}_{-i}(x) = x - 2^i \text{ modulo } N,$$

where $0 \leq x < N$ and $0 \leq i < n$.²⁸ The *Shuffle* interconnection function is:

$$\text{Shuffle}(p_{n-1} \dots p_1 p_0) = p_{n-2} \dots p_1 p_0 p_{n-1}$$

where $0 \leq P < N$.⁴⁰ The *Shuffle* is usually used with Cube_0 , also called the *Exchange*. Various properties of these interconnections are discussed in References 28-31, 33, 39, 40.

Cube-type networks

Cube-type multistage networks are composed of stages of $N/2$ switch elements, where each switch element may be viewed as an *interchange box*, a two-input, two-output device. Let the upper input and output lines be labeled i and the lower input and output lines be labeled j . The four legitimate states of an interchange box are: (1) *straight*—input i to output i , input j to output j ; (2) *exchange*—input i to output j , input j to output i ; (3) *lower broadcast*—input j to outputs i and j ; and (4) *upper broadcast*—input i to outputs i and j .²¹

The *control structure* of a network sets the states of the interchange boxes. *Individual stage control* uses the same control signal to set the state of all the interchange boxes in a stage, i.e., all the boxes in a given stage must be in the same state. *Individual box control* uses a separate control signal to set the state of each interchange box. *Partial stage control* uses $i+1$ control signals to control stage i , $0 \leq i < n$.³⁴

The STARAN flip network⁶ with flip control, shown in Figures 1a and 1b, consist of n stages and employs individual stage control. It is used in the STARAN SIMD machine^{5,9,12} for processor-to-memory and processor-to-processor connections. This network may also be operated using shift control, a partial stage control scheme as shown in Figure 1c. The interchange boxes may be set to either exchange or straight. In stage i, boxes set to exchange are

performing the $Cube_i$ interconnection function on their inputs, $0 \leq i < n$. With flip control, the network has connectivity $C=2^{n/2}N!$ (each stage is either "straight" or "exchange") and, with shift control $C=2^n N!$, where $s=1+2+\dots+n=n(n+1)/2$ (each control signal is either "straight" or "exchange"). The network is homogeneous under flip control, where all inputs are treated equally. Under shift control, it is not homogeneous. The network broadcast scope $B=1/(N-1)$, and a broadcast requires n passes. Due to the limited control schemes, it is not capable of being partitioned. The network can be expanded by factors of two. To double the size, from N to 2N, connect two size N networks with a "Cube_n" n-th stage.

This network, using only flip control, supports STARAN's multidimensional access (MDA) memory,⁷ which makes STARAN an interprocessor dynamic word size system. The MDA memory consists of 256 physical words, each of which is 256 bits long. Each of the system's 256 processing elements operates on one bit. This allows the system to operate on 2^i consecutive bit logical words, where the logical words are from physical words 2^i apart, $0 \leq i \leq n$. The two extremes are a word-slice (an entire physical word) and a bit-slice (the jth bit of all physical words). A total of N^2 different formations are possible.

The indirect binary n-cube network²⁶ is basically identical to the flip network (Figure 1b), except that individual box control is used. It is a PE-to-PE SIMD network. The connectivity $C=2^{n/2}N!$, since each box may be either "straight" or "exchange." As with the flip network, $B=1/(N-1)$, broadcast delay is n passes, and the network can be expanded. Due to the individual box control, all inputs may be treated in the same way, so the network is homogeneous.

It was proposed²⁶ that the individual box control be implemented by a single microprocessor for each stage. If this implementation is employed, the use of this network in MSIMD mode is limited. However, the network topology, interchange box and control structure do support partitioning for MSIMD mode, where the only restrictions are that the number of processors in each submachine is a power of two and the addresses of the processors agree in certain bit positions, as specified in References 34 and 39. For example, the system may be divided into independent partitions, the odd numbered PEs and the even, by physically setting all of stage 0 to straight.

The Omega network²¹ shown in Figure 2a, is a processor-to-memory network for SIMD machines, which can be adapted for MSIMD systems. The interchange boxes may assume any of the four legitimate states. Without the broadcast states, each stage of the network is a Shuffle interconnection function, followed by possible Exchange function. The network may be redrawn, as in Figure 2b, where the i-th stage corresponds to a possible $Cube_i$ interconnection function. Thus, the Omega is identical to indirect binary n-cube and has the same parameter values, except the stages are in reversed order and the Omega has the capability to broadcast from any input to all outputs in one pass ($B=1$).³⁴ Its use of n-bit destination tags to control the straight/exchange states make it easy to use and well suited to

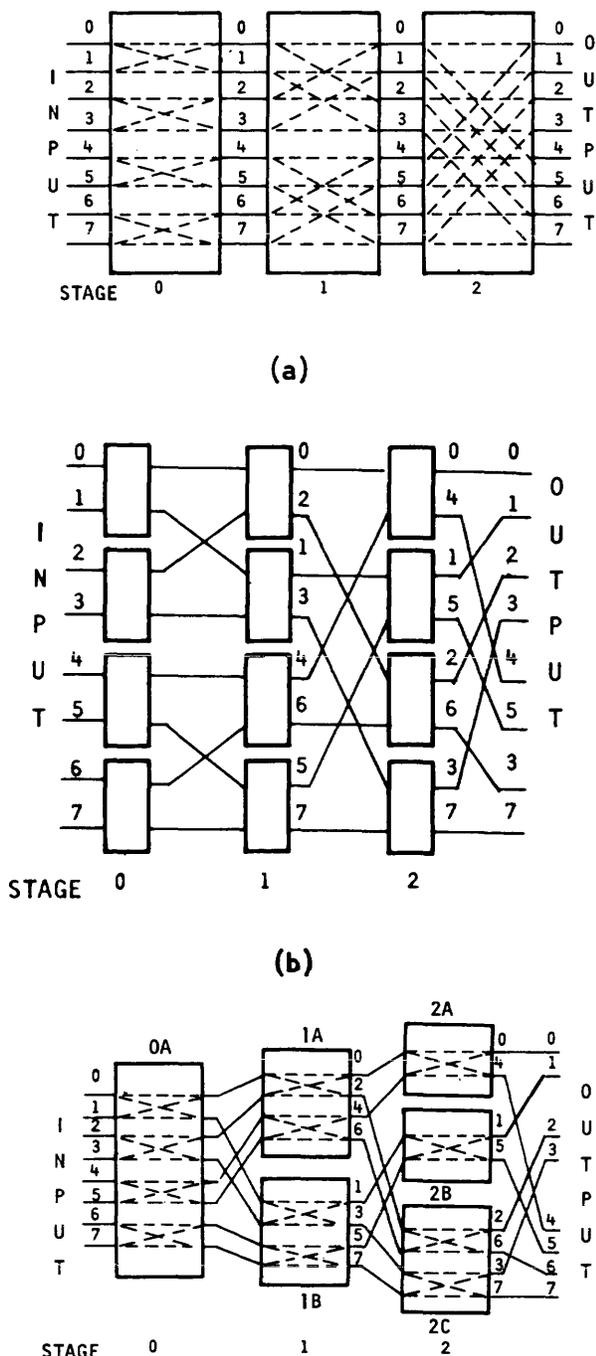
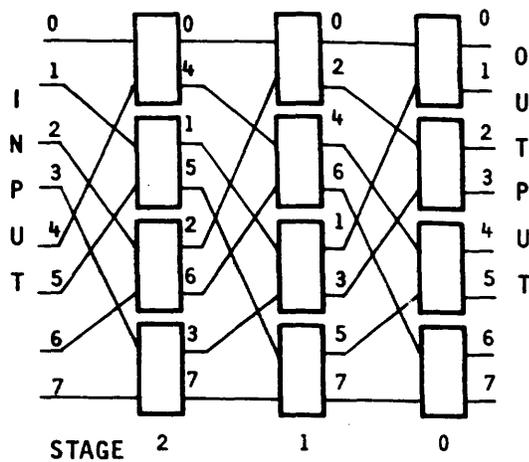
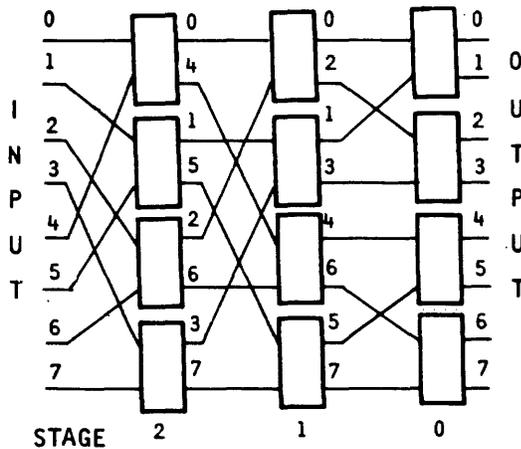


Figure 1—STARAN network for N=8. (a) with flip control, (b) "a" redrawn, and (c) with shift control (0A, 1A, etc. are the control signals).³⁴



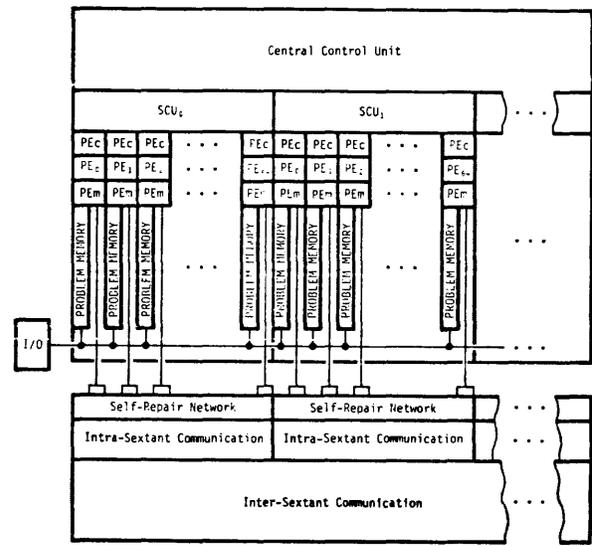
(a)



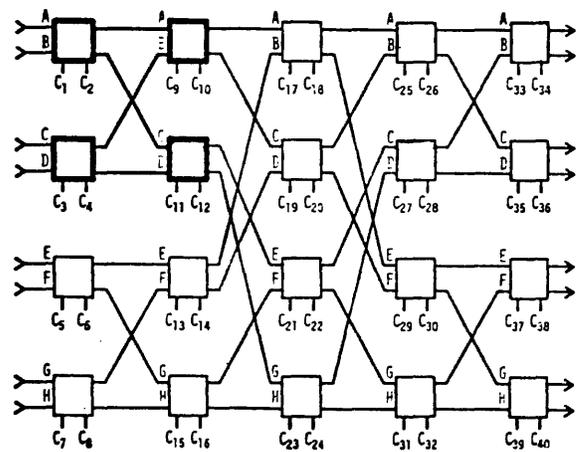
(b)

Figure 2—(a) Omega network for N=8 and (b) "a" redrawn.³⁹

stream which is broadcast by the central control unit. Thus, the system is not MSIMD, since the SIMD submachines are not independent. The network size is halved by setting the middle stage and the one before it (or after it) to all straight. The network can perform any permutation in one pass ($C=1$) and any processor can broadcast data to all other processors in one pass ($B=1$). The network can only be extended by doubling the number of processors (from N to $2N$) and adding two stages (a new $Cube_{n-1}$ and a $Cube_n$) immediately prior to the old $Cube_{n-1}$ stage. Each sextant of the Phoenix has a spare processor. This spare processor can be swapped in by the "self-repair network" when a faulty processor is detected. This allows the Phoenix to run unaffected even after a processor has failed. Sextants which have failed can be ignored.



(a)



(b)

Figure 3—Phoenix. (a) Block diagram overview. (b) Programmable switching network.¹¹

MSIMD mode, with the same restrictions as the n-cube.^{34,39}

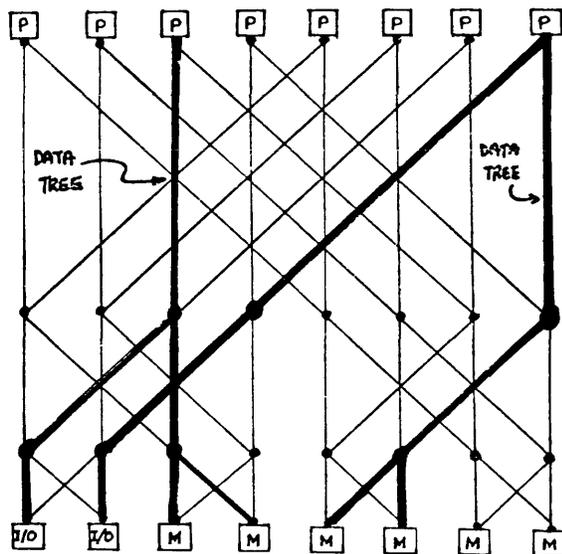
The *Phoenix Project*¹¹ is an SIMD system consisting of 1024 PEs with hierarchical control. The PEs are divided up into 16 sextants of 64 PEs each. Each sextant has a control unit, and there is a central control unit which broadcasts instructions to the sextant control units. A block diagram is shown in Figure 3a. Communication between PEs is handled by a $2n-1=19$ stage network, where stage i performs the $Cube_i$ function for $0 \leq i \leq 9$ and stage i performs the $Cube_{18-i}$ function for $10 \leq i \leq 18$. The network, whose interchange boxes can assume any of the four legitimate states under individual box control, is shown in Figure 3b. The algorithms used to calculate the switch settings are inherently serial and require $O(nN)$ time. Switch settings can be precomputed for commonly used connections.

The network is homogeneous and may be partitioned. The array can only be partitioned into subarrays of equal size (powers of two) all following the same instruction

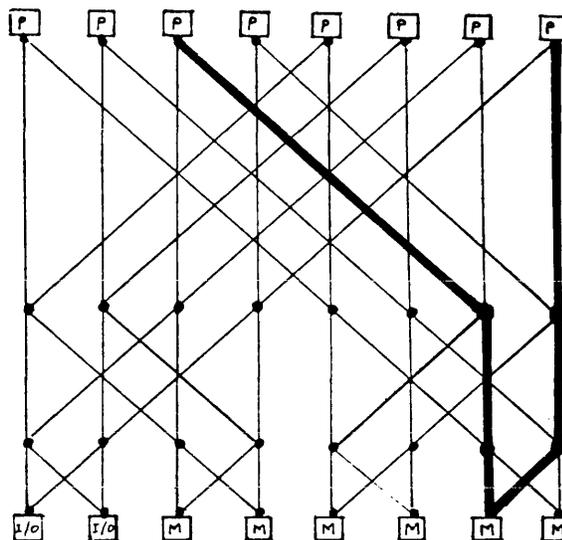
Banyan networks

The class of banyan networks is formally defined in 16. In particular, the SW-banyan with " $f=s=2$ " is used in the PSM *Reconfigurable Varistructured Array Processor (RVAP)*,²² as shown in Figure 4. This SW-banyan is topologically equivalent to the Omega, STARAN, and n-cube networks. It differs from these Cube networks in its switch elements and the way in which it is used.

The RVAP network is homogeneous and connects processors with memories or processors. The network includes lines for carry propagate and generate signals to support



(a)



(b)

Figure 4—SW-banyan network, (a) data trees and (b) instruction tree.²²

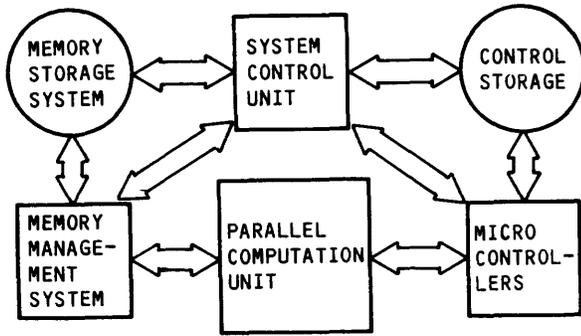
interprocessor dynamic word size. Figure 4a shows how processors can be linked to separate memories and I/O devices for data transfers and Figure 4b shows how processors can be linked to a single memory to receive instructions in SIMD mode. A data or instruction tree can be established by having each leaf send a signal to all potential roots. Exactly one of the potential roots which receives a signal from all of the leaves is selected to be the root by a priority circuit. The root broadcasts a signal downward and each leaf broadcasts a signal upwards. All paths receiving both upward and downward signals are connected to form the tree. Thus, it takes $p+2$ steps to, for example, connect p processors to a single memory. It is stated that this method may be converted into a one step algorithm.²²

Measurements in 15 indicate that on the order of 10 percent of the possible interconnection structures are not connectable, therefore there are many ways to partition the system. In the area of fault tolerance, this system differs from the other networks in this section in that a faulty switch is treated as a busy switch and is not a special case. Thus, depending on how many and which switches are "busy" the network may be successively reconfigured, interrupting, but not modifying, the algorithms being executed. It is possible to broadcast data by using a dummy memory node, thus the broadcast scope is $B=1$. The broadcast delay is two passes through the network. This network can be extended as in the STARAN network. A prototype system using a more complex SW-banyan (" $f=3, s=2$ ") is currently being constructed.

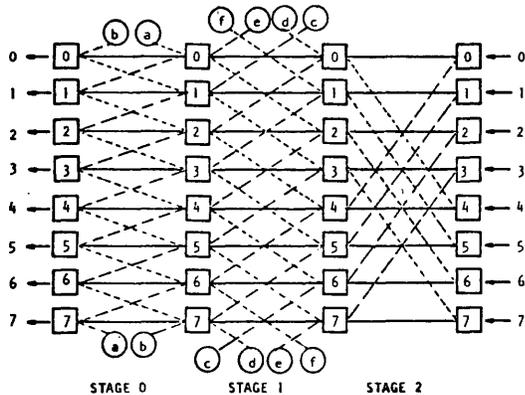
PM2I-type networks

The *Augmented Data Manipulator (ADM)* network^{34,39} is used in the PSM machine called *PASM (PARTitionable SIMD/MIMD)*.^{32,35-38} Figure 5a is a block diagram of PASM, a system being designed for image processing and pattern recognition at Purdue University. The heart of PASM is the *Parallel Computation Unit (PCU)*, which contains N processors, N memory modules, and the interconnection network. The *PCU processors* are microprocessors that perform the actual SIMD and MIMD computations. The *PCU memory modules* are used by the PCU processors for data storage in SIMD mode and both data and instruction storage in MIMD mode. The PE-to-PE ADM network, shown in Figure 5b, is composed of n stages, where each stage consists of N independently-controlled switch elements (cells). It is based on Feng's data manipulator, which uses a less flexible control scheme.¹³ At stage x , cell j can be connected to any or all of the following cells at stage $x-1$: $j, j+2^x$ modulo N , and $j-2^x$ modulo N , where $0 \leq x < n$. Thus, stage x is analogous to $PM2_{+x}$ and $PM2_{-x}$. Various schemes for controlling the network, including destination tags, are being studied.

The network may be partitioned into independent subnetworks of varying sizes, with the constraints that the size (number of inputs/outputs) is a power of two and that the input addresses of a subnetwork all agree in their low order bit positions. For example, the system may be partitioned



(a)



(b)

Figure 5—PASM. (a) Block diagram overview. (b) Augmented data manipulator, for $N=8$.³⁸

into two independent groups, the odd numbered PEs and the even, by setting all of stage 0 to straight (j to j , $0 \leq j < N$). The network can be expanded by factors of two. A size $2N$ network may be constructed by "interleaving" the cells of two size N networks, combining the two old stage x s into the new stage $x+1$, and adding a new $2N$ size stage 0 (the $PM2_{+0}$, $PM2_{-0}$, and straight connections). This is the "inverse" of the partitioning method.

The *Micro Controllers (MCs)* are a set of microprocessors which act as the control unit for the PCU processors in a virtual SIMD machine and orchestrate the activities of the PCU processors in a virtual MIMD machine. There are $Q=2^q$ MCs, *physically addressed (numbered)* from 0 to $Q-1$. Each MC controls N/Q PCU processors, in particular, MC i controls the N/Q processors whose low-order q address bits equal i . There is an MC memory module for each MC. A virtual SIMD machine of size MN/Q , where $M=2^m$ and $0 \leq m \leq q$, is obtained by loading M MC memory modules with the same instructions simultaneously. Similarly, a virtual MIMD machine of size MN/Q is obtained by combining the effort of the PCU processors of M MCs. In both cases, the physical addresses of these M MCs must

have the same low-order $q-m$ bits since the physical addresses of all PCU processors in a partition must agree in their low-order bits. Thus, each partition contains at least N/Q PEs and the number of partitions is at most Q .

It can be shown that the interconnection capabilities of the ADM are a superset of those of the Omega network,³⁴ i.e., it can do all the connections the Omega can and some the Omega can not (e.g. Shuffle). Thus, the ADM can perform more than the Omega's $2^{N/2}$ distinct permutations, however, it can not perform all $N!$ permutations (e.g. inverse Shuffle). It can broadcast data from any PE in one pass through the network ($B=1$). The cost of the ADM network is $O(nN)$ and it treats all PEs equally so it is homogeneous.

DEDICATED PATH NETWORKS

Introduction

Dedicated Path Networks (DPNs) are characterized by communication links (usually bidirectional) which connect exactly two processors. All of the DPNs in the following two subsections are physically distributed. DPNs are divided into two classes by their control structures: centralized or distributed.

DPNs with centralized control

The Illiac IV discussed here is the original design rather than the machine which was actually built. The *Illiack IV*⁴ is an MSIMD system composed of four quadrants, each of which has 64 PEs and a control unit. Each PE consists of a 64-bit processor and a memory. Each quadrant can operate independently, or in conjunction with other quadrants by having either two or four control units broadcast the same instruction stream. Thus, the partitioning of Illiac is limited to three states: four partitions each with 64 processors, two partitions each with 128 processors, or one partition with 256 processors. The PEs in a partition are numbered (addressed) from 0 to $M-1$. The network connects PE i with PE $i+1$, $i-1$, $i+8$, and $i-8$, modulo M . The processor to network interface is part of each PE and the cost of the network is $O(N)$.

Each Illiac processor supports intraprocessor dynamic word size and can be partitioned to operate on one 64-bit word, two 32-bit words, or eight 8-bit words. The system is restricted such that each processor must perform the same operation on all the fields of its partitioned word. If a PE fails, its quadrant can be ignored.

The degree of simultaneity of the Illiac network is $S=1$. The Illiac network has no special provisions for broadcasting a data item from one PE to the other PEs ($B=1/(N-1)$). However, the Control Unit may broadcast a data item from one PE to all PEs.⁴¹ The range and domain are $R=D=4/M$, where M is the number of processors in the partition. The worst delay using 256 PEs is 19 transfers (e.g. PE 0 to PE 124). The network is homogeneous and its control is cen-

tralized. The network can be extended, but the worst case delay would make significant increases in the number of PEs undesirable.

Using permutation group theory,³⁰ it can be shown that for an M PE system the number of permutations possible in one pass is $3+2*2^8=3+2^9$. There are eight cycles which can be in either the $+8$ state or the inactive state (the 2^8 factor). The same eight cycles could also be in either the -8 state or the inactive state (the 2^* factor). In addition there is one cycle which can be in either the $+1$ state, the -1 state, or the inactive state (the $3+$ term). Hence the connectivity is $C=(3+2^9)/M!$.

The overall architecture of *A Novel Multiprocessor Array (ANMA)*²⁴ is SIMD. ANMA has a host computer, a control unit, and 16 eight-bit processors. The PE-to-PE DPN of ANMA is made up of two parts: the *Variable Word Length (VWL)* structure and the *Logarithmically Structured Transfer (LST)*.

The VWL is used to connect a processor with its left or right neighbor to form longer data words, i.e., the VWL supports interprocessor dynamic word size. The VWL is controlled by an $N-1$ bit register where N is the number of processors in the system. Bit number i determines whether or not processor i is connected to processor $i+1$, $0 \leq i \leq N-2$. The VWL structure is made up of the carry, shift, VWL shift rotate and Partially Selectable Microinstruction control lines. These lines allow the communication necessary to link processors together to form longer data words. The cost is $O(N)$.

The LST is a set of bidirectional serial links between processors, as shown in Figure 6. If the processors are numbered (addressed) from 0 to 15, then the links are between processor i and processor $i+2^j$, for non-negative integers i and j , such that $i+2^j \leq 15$. This network is a subset of the PM2I containing connections i to $i+2^j$, where $i+2^j \leq 15$, and i to $i-2^j$, where $i-2^j \geq 0$, $0 \leq i \leq 15$, $0 \leq j \leq 3$. Thus, the network may be easily extended by factors of two. The cost is $O(nN)$.

The degree of simultaneity for the VWL structure is $S=1$. Assuming the LST can be used in only one direction at a time, then $S=1/2$. Note that the VWL structure only supports combining left or right neighbors to form longer data words. Faulty processors would create word partitions.

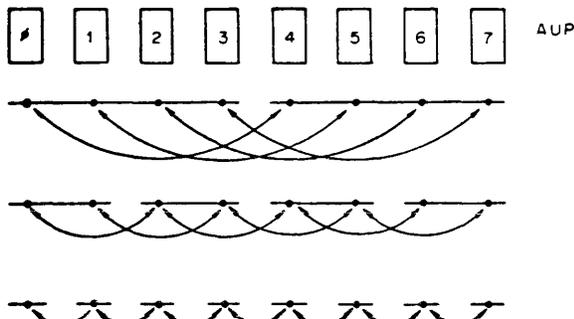


Figure 6—The LST network for ANMA, $N=8$.²⁴

For the LST, the broadcast scope $B=1/(N-1)$, but a processor can broadcast a value to all processors through the Control Unit. The delay of the LST is n transfers. The range and domain of the LST for "middle" processors (i.e., processors 7 and 8) is $R=D=(2n-1)/(N-1)=7/15$. For "end" processors (i.e., processors 0 and 15), $R=D=n/(N-1)=4/15$. However, for the VWL structure the range and domain for middle processors are $R=D=2/15$, and for end processors $R=D=1/15$. For both networks, R and D vary for different processors, so neither is homogeneous. Furthermore, the network interfaces can not be made of identical modules, unless some outputs are not used.

DPNs with distributed control

The *Columbia Homogeneous Parallel Processor (CHoPP)* MIMD system⁴² is made up of modular nodes, each of which consists of an application processor, a communications processor, and a memory. The CHoPP processor-to-memory network is based on the Cube interconnection functions described earlier. Assuming the nodes are numbered (addressed) from 0 to $N-1$, each node can communicate directly with any node whose address differs with its address in only one bit position. To communicate with a node which has an address which differs in more than one bit, the information must travel through intermediate nodes. Thus, there may be a delay of n transfers. As with the indirect binary n -cube, CHoPP may logically be partitioned into subsystems. Arbitrary subsystems may be constructed, but this may involve the use of communications processors not in the subsystem. Faulty nodes may be ignored.

The network control is distributed, based on destination tags. Each communication processor examines the tag and determines where to send the message. The network is homogeneous and the degree of simultaneity of CHoPP is $S=1$. The range and domain are $R=D=n/(N-1)$. The broadcast scope is $B=n/(N-1)$, and a complete broadcast requires n transfers, i.e., the broadcast delay is n . The cost of the network is $O(nN)$. The system can be extended by factors of two.

The *variable topology multicomputer (VTM)*²⁵ is an MIMD system composed of nodes similar to those of the CHoPP system. Each node of the VTM contains a processor, an inter-computer message handler, and a memory. The VTM network is made up of unidirectional serial communication lines. The topology of the network may be changed by adding or deleting lines by physically rewiring the network, e.g., via a patch panel. However, once a topology is chosen, communication links are formed under software control.

The message routing is done by two methods: message switching and circuit switching. When operating in the message switching mode, the inter-computer message handler examines the message to determine its destination and sends the message out on the proper link. In the circuit switching mode of operation, the inter-computer message

handler effectively shorts an input link to an output link, i.e., the node is bypassed. When a link is used for circuit switching, it can no longer be used by the node being bypassed.

Because this system allows rewiring, it is difficult to apply the parameters that depend on the topology. The degree of simultaneity is $S=1$, provided every node has at least one output link. By using the circuit switching capabilities a direct link between any pair of processors can be set up for partitioning the system, provided there are a sufficient number of links. The network is physically distributed and new nodes may be added to increase the system size.

SHARED PATH NETWORKS

Introduction

A *shared path network* is defined recursively as a communications path which may be used by more than two processors, or a set of shared path networks which may be interconnected by switch elements that can either make or break a connection between these paths. A shared path network can be distinguished from the multistage switching networks discussed previously based on the complexity of the switch element and the number of path segments. In a multistage switching network, there are at least N paths connecting one stage to the next, and there are at least n stages. Furthermore, each stage contains switch elements with the capability of choosing which of two or more input and output lines to connect, if any. A shared path network connects paths using simple connect/disconnect nodes or processors. In addition, there are typically fewer than $O(nN)$ bus segments.

Two buses are said to be *similar* if they (1) have the same width; (2) have the same delay (on the average), including that incurred by passing through controllers or arbiters; (3) are connected to the same class of sender/receivers; and (4) the priority of the data they carry, as assigned by the respective sender/receivers, is independent of the bus on which they arrive. If a shared path network contains only one bus, or contains several buses which are similar, it can be classified as *linear*. If there are multiple buses which are dissimilar, it is classified as *hierarchical*.

Linear shared path networks

The *Minerva Multi-Microprocessor*⁴⁸ is an MIMD system which connects eight 8080 and four 3000 Intel processors to a single demand-multiplexed bus called *IDBUS* (Inter-Device Bus) and to the *IDBUS ARBITER*. Use of the bus is controlled by the *IDBUS ARBITER*. The bus is used to gain access to public memory which is the means processors use to communicate with each other. Thus the *IDBUS* is used as a processor-to-memory structure. Processors use the *IDBUS* one at a time, so $S=1/N$. Any arbitrary group of processors can communicate with each other, and the

subset of processors that constitute a group can vary arbitrarily with time. Thus, the *IDBUS* fully supports system reconfiguration and submachines containing up to eight 8080 and four 3000 microprocessors may be formed to work toward a common goal.

The single shared bus does not permit one-to-many communication, so the broadcast scope $B=1/(N-1)$. To send the same data item to all other processors takes $O(N)$ time. The *IDBUS* interfaces are physically distributed and the *ARBITER* is physically centralized. Each *IDBUS* interface is part of a processor module and is LSI compatible.

The *IDBUS* is readily expandable at the cost of a slight increase in arbitration time, however, expansion is limited by contention problems that grow worse with each additional processor, since there is only one bus. While the *IDBUS* and *ARBITER* are not fault tolerant, it is possible for them to support the fault tolerance of the system as a whole, i.e., a defective processor may be ignored. Since the *ARBITER* handles all bus use automatically, control of the network from the user's point of view is simple. The number of elements used to construct the network is $O(N)$, which leads to a low cost, but the delay incurred while waiting for use of the bus is significant. The range and domain of each processor are $R=D=1$, since processor communications are through memory and any processor can read from memory. Finally, all processors view the bus in the same way, so it is homogeneous.

The *Dynamic Computer (DC)* described by Reference 18 is a multicomputer system which contains a reconfigurable bus and "carry" links, as shown in Figure 7. Software controllable switches (*MSs*) placed between each adjacent pair of processors allow 2^{N-1} different system configurations to be formed, via the reconfigurable bus, by linking subsets of the *computer elements (CEs)* together. This bus is 16 bits wide and is used only for broadcasting instructions to other CEs within a given group. The bus is used in the PE-to-PE interconnection mode. The "carry" links between adjacent CEs allow "carry" information to be passed along efficiently, with $S=1$.

The *V monitor* is responsible for initiating changes in the bus configuration and arbitrating conflicting requests for a change (made by concurrent programs). Changing any configuration includes writing new control words to each processor and sending new switch settings to the *MS* units. The reconfigurability of the bus and the interprocessor

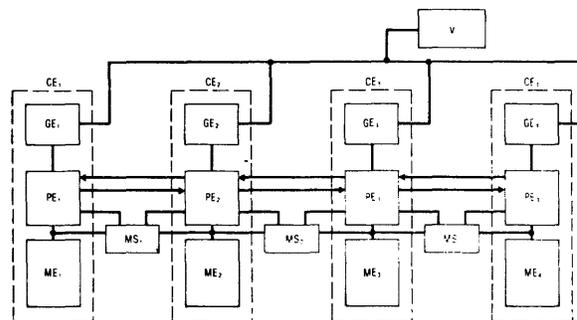


Figure 7—Dynamic computer group of size four.¹⁸

carry connections allow the system to be partitioned into independent computers whose word size may vary. The processors in a partition are called a DC group. The only constraint that determines which CEs may be in a group is that they must be adjacent. Because the bus structure each CE 'sees' is dependent upon where that CE is physically located (i.e., each CE has a different number of CEs on its left and right than the others), the network is not homogeneous.

The MS units are physically distributed, modular and complex (the latter being the result of their construction from microprocessors). These characteristics allow the network to exploit LSI technology. The cost of this network and the carry links is $O(N)$. Expansion is incremental and is limited only by the V monitor. The bus is fault tolerant, however, for each MS unit that becomes defective, the number of possible bus configurations is halved. The bus supports the fault tolerance of the system as a whole, permitting continued operation with reduced overall performance. Control of the bus configuration is completely automatic and thus presents no hindrance to the user. Data words may be passed between processors by shifting them serially through the carry links. All processors have a range and domain of $2/(N-1)$ except those on the ends, whose range and domain are $1/(N-1)$.

The DC may be viewed as an MIMD multicomputer system, where carry links are used for intercomputer data transfers instead of "carry" information. Thus, the broadcast scope $B=1/(N-1)$. Since data transfers are serial, the broadcast delay is $O(Pw)$, where P is the number of DC groups and w is the number of bits in the word to be broadcast. It may also be viewed as an MSIMD system, where each DC group operating as an SIMD machine establishes a multi-array mode in which a number of computers execute the same instruction stream broadcast by a computer supervisor. One way this may be accomplished is to enable all MSs between the processors that form the array for instruction broadcasting, but disable the carries at word boundaries. Since different parts of the system may operate in MIMD and MSIMD mode simultaneously, the Dynamic Computer may be categorized as a PSM computer with the ability to vary the word width at each "processing element."

The *Restructurable Computer System (RCS)*²⁷ is composed of 64 *Functional Units (FUs)*, and 24 *Bus Units (BUs)*, as shown in Figure 8. Sixteen BUs are responsible for processing the communications needs of at most four functional units each, while six are reserved for operand streams from memory and two for streams into memory. Thus, the buses are used both as a processor-to-memory and a processor-to-processor connection structure. The data words are 48 bits wide and each bus is ten words wide.

By properly loading registers in the BUs, any of the BUs can be assigned to any of the FUs, making the system restructurable. Arbitrary pipelined structures as well as SIMD array structures may be formed. Thus, the RCS is partitionable with each partition consisting of the FUs that form a single pipe or array. There are no restrictions on the

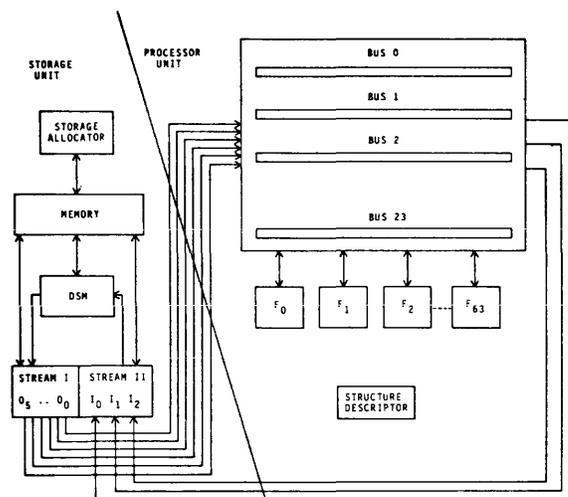


Figure 8—Restructurable computer system.²⁷

subset of FUs that may belong to a partition. Software associated with the system provides for conversion from a logical description of a desired configuration to the specific register values to be loaded.

The BUs are physically distributed, identical and thus modular units. In spite of their modularity and high complexity, the pin requirements prohibit implementation as LSI chips using current technology. The cost of the network is $O(N)$, but the coefficient of N may be large due to the very high bandwidth of the buses and the high complexity of the BUs. The worst case delay for an interprocessor data transfer is four bus cycles, however, in time, critical applications more than one BU can be assigned to a FU.

Each FU contains a bus unit scanning station (*BUSS*) that can look for input from any of four BUs, thus the domain is $D=4/(N-1)$. Similarly the BUs can send data to only four other BUs so the range is $R=4/(N-1)$. Structurally, the network of BUs looks the same to all FUs so the network is homogeneous. Since there are 16 independent buses which may be transferring data simultaneously, the degree of simultaneity $S=16/64$. One FU can send data to only four other FUs at the same time, so the broadcast scope $B=4/63$.

Reddi and Feustel point out that fault tolerance may be acquired by forming three identical pipes and routing their output to a FU that performs a voting function. Any bad units could be tagged and avoided by the operating system. The network itself is fault tolerant in the sense that multiple, identical units are used for communications, and the system can operate without all BUs functioning. To expand the RCS it would be necessary to increase the size of several registers in each BU that identify the FUs assigned to it. As the number of FUs increases, the ratio of BUs to FUs decreases, degrading the total throughput of the system. Thus, it is desirable to add BUs to maintain system performance, which would require increasing the size of certain identification registers in each *BUSS*.

Hierarchical shared path networks

The Cm^* ^{17,43,44} shown in Figure 9, is an MIMD system composed of a large number of computer modules that are grouped into clusters and connected by a hierarchy of buses. Communications are processor-to-memory. All buses are physically 16 bits wide, however local buses use an address space of 16 bits, *Map Buses* (intra-cluster buses) use an address space of 22 bits, and *Inter-cluster (IC) Buses* use an address space of 28 bits. Address translation and data routing are performed by local switches called *S.locals* and mapping controllers called *K.maps*, both considered switching elements. At most 14 computer modules may belong to the same cluster. If all clusters are filled, in the best case the degree of simultaneity $S=1/14$, where it is assumed all communications are intra-cluster, i.e., they take place on *Map Buses*. If all communication requires use of IC buses, neither S nor the delay can be determined since the topology of IC buses is not specified. There are no facilities for broadcasting data, so $B=1/(N-1)$. The broadcast delay is $O(n)$, since a recursive doubling algorithm could be used to transfer a given data item to all processors.

There is no hardware in Cm^* that is designed to partition the computer modules. There is, however, firmware (i.e. microcode) built into the address translation units that supports logical partitioning. Capability lists are associated with each computer module that specify those areas of memory to which it may have access. The network is homogeneous, since it treats all processors equally. The switching elements (*S.locals* and *K.maps*) are physically distributed and modular. It appears that they could be implemented by a small set of LSI chips, due to their complexity and reasonable pin requirements. The cost of the network is $O(N+N/14)=O(N)$.

Cm^* is readily expanded by attaching more switching elements to the appropriate buses, with the limiting factor being the maximum amount of memory that can be addressed (2^{28} words). Each computer module can access any word in memory and so can communicate directly with any other computer module in its cluster, making the range and domain $R=D=13/(N-1)$. Intercluster communications can

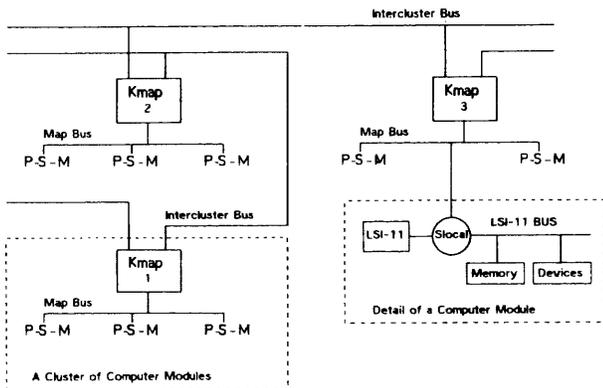


Figure 9—A simple three cluster Cm^* system.⁴³

expand the range and domain, but this is dependent on the unspecified IC bus system topology. All address translation is transparent to the user, so the network is very easy to use. Two-way communication is employed and status of the progress of a message is kept, making it possible to identify a bad computer module and avoid using it. There is, however, no specific fault tolerance built into the network.

A network for message routing in a *Mega-Micro-Computer (MMC)*⁴⁹ is shown in Figure 10. Buses connecting computers logically subdivide the toroidally continuous physical space into nested square groups of $16, 16^2, 16^3, \dots, 16^{(j+1)}$ computers. Spanning, but not leaving, each $16^{(k+1)}$ group are level- i buses (*i-buses*), each connected once in each inner 16^i group, $0 \leq i \leq k$. Each computer is connected to one 0-bus and one higher-level bus, thus, the network is not homogeneous since all computers are not connected to buses of equal "distance" capability. The computers in Figure 10 are signified by numbers indicating the higher-level bus to which they are attached. The "X" is for buses of level > 4 . The computers shown are connected to 24 different 0-buses, 16 1-buses, 64 2-buses, 48 3-buses, and 34 4-buses. However, only one 1-bus, one 0-bus and parts of a 2-bus and another 1-bus are explicitly shown. Each bus is connected to only 16 computers and is used in the PE-to-PE configuration.

Since each processor is connected to two buses and there are 16 processors per bus, in one bus cycle a given processor can communicate with any of 30 other processors, thus $R=D=30/(N-1)$. No particular hardware features are described that could be used to partition the system. It is also not designed to broadcast data, so $B=1/(N-1)$. Using a recursive doubling algorithm, data can be passed to all processors with broadcast delay $O(n)$.

The mechanisms that move data from bus to bus are built into the computers, thus the network is physically distributed as well as modular. If bus widths are kept reasonable, it appears the system is LSI compatible. The cost is $O(N)$. Routing in the network is automatically performed by com-

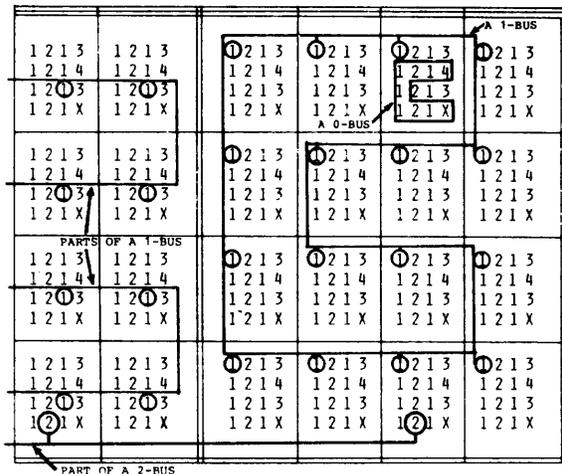


Figure 10—Logical groups of computers showing bus sharing in a Mega-Micro-Computer network.⁴⁹

paring a destination tag to a local address, making it necessary only for a user to specify the desired destination. Since there are a large number of routing choices, it is a simple matter to bypass faulty computers. If A is the absolute difference between the addresses of the sender and receiver, the delay is $O(\log_2 A)$. If the maximum level is i and there are 16^{i+1} computers, then the number of buses in the system is $b = \sum_{j=0}^i 16^j/2^j$. Since all buses may be in use simultaneously, $S = b/16^{i+1}$. For example, for $i=4$, $S=0.12$.

The *Hierarchical Restructurable Multi-Microprocessor (HRMM)*² architecture employs multiple control buses called *Control Groups (CGs)* and a circulating data bus. Switching elements, called *Block Short Modules (BSMs)*, segment the control group buses between adjacent pairs of processors, as shown in Figure 11. A CG consists of three buses—(1) the CMD bus that carries commands to processors; (2) the ACK/NAK bus with which a processor recognizing a command can acknowledge its acceptance or rejection (due to a full queue); and (3) the DONE bus where a command processor can acknowledge the completion of a required task. When a command is received by a processor, it is placed into a queue and cannot be executed until the data associated with it arrives on the data bus, consequently all processors execute instructions independently as an MIMD machine. Each CG is given a fixed priority, thus enabling establishment of a hierarchy for communication.

While the buses that form a CG are of the conventional type, the data bus is of the circulating loop (or Pierce Loop) type where data packets are moved a fixed distance and direction in each unit of time. Both buses are used in the PE-to-PE configuration. The simultaneity of the data bus is $S=1$, since the bus may be viewed as a parallel shift register and each processor may place one data packet on the bus at one time. "Carries" and synchronization information are provided between processors by the sync/carry loop (see Figure 11). Thus, there is interprocessor dynamic word size. Changing the settings of the BSMs changes the structure of HRMM and is accomplished by issuing commands on the Master CG CMD bus. The structure of the system may be viewed as a tree. The broadcast scope $B=1/$

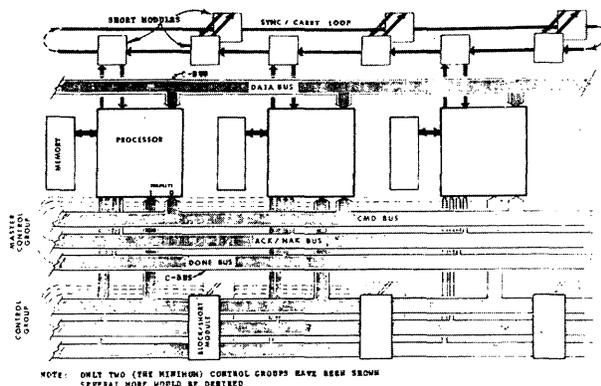


Figure 11—Hierarchical, restructurable multi-microprocessor architecture.²

$(N-1)$, but broadcasting is possible on the data bus in $N-1$ bus cycles by placing a "don't care" in the destination address.

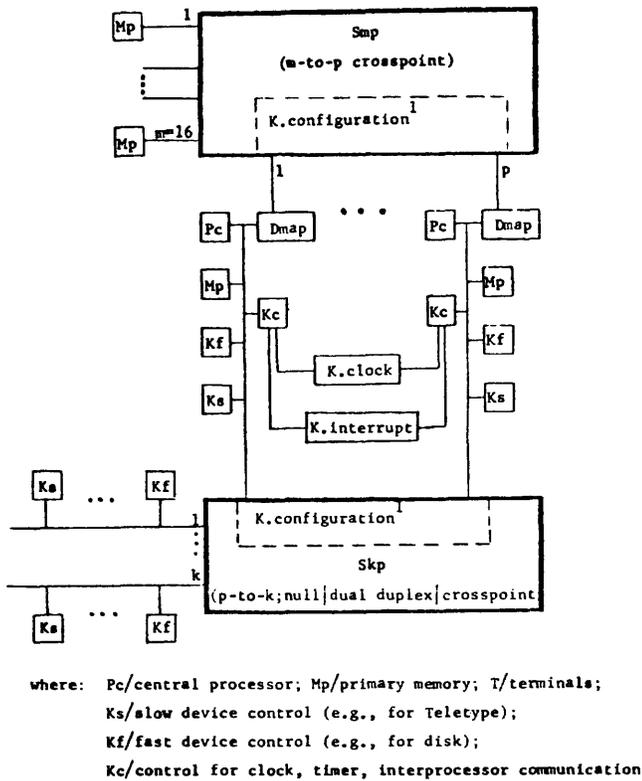
The data bus and CG buses are physically distributed and modular. CG and data bus widths and complexity are not completely specified so LSI compatibility cannot be established. The cost of the CGs is $O(mN)$ and the cost of the data bus is $O(N)$, where m is the number of CGs. The delay through the CG buses is one control bus cycle. The worst case delay through the data bus is $N-1$ data bus cycles, where a bus cycle is on the order of 50 ns.⁴⁹ The range and domain of the data bus are $R=D=1/(N-1)$ because of its unconventional structure. Since data packets move from one processor to the next in a fixed amount of time (i.e., one bus cycle), one processor can only send data to or receive data from one other processor each bus cycle. The range and domain of the CG buses are a function of the BSM settings with a best case of $R=D=1$. The HRMM is readily expandable by adding more processors to either end. All processors are treated equally by the data bus, thus it is homogeneous.

The data bus is easy to use, whereas the CG buses are more complex due to their flexibility. The CGs are fault tolerant; however, the flexibility to configure them is reduced by varying degrees, depending on which BSMs fail. The data bus is not fault tolerant since a break in the loop makes it virtually unusable.

Crossbar switches

Crossbar switch (CBS) networks are shared bus networks in which p nodes can be connected to q nodes. Such a CBS is called a $p \times q$ CBS. The cost of a $p \times q$ CBS is $O(pq)$, and the delay of a CBS is constant. CBSs can be extended incrementally, the difficulty of which is implementation dependent. CBSs are modular in design, and may be appropriate for LSI, depending on the complexity of the crosspoints (e.g. queues).

*C.mmp*⁵⁰ is an MIMD system consisting of p processors, one large shared memory with m memory modules, and k I/O buses, as shown in Figure 12. The system contains two CBSs. One, the *Skp*, is a $k \times p$ CBS which connects the k I/O buses with the p processors. The other, the *Smp*, connects the m memory modules with the p processors and is a $m \times p$ CBS. Each processor is a self-contained PDP-11. The simultaneity is $S = \min(k,p)/p$ for the *Skp* and $S = \min(m,p)/p$ for the *Smp*. Both the CBSs are homogeneous and physically centralized. However, the (software) control is distributed. A memory address is translated by hardware into a setting for the CBS. This makes the CBS transparent to the user. There are hardware switches which can force the CBSs into a given state. The connectivity, range and domain are $C=R=D=1$, since any processor can be connected directly to any memory; however, processor-to-processor communications are limited to using a memory as an intermediate node. The broadcast scope is $B=1/(N-1)$ and the broadcast delay is $O(n)$, using a recursive doubling scheme.



where: Pc/central processor; Mp/primary memory; T/terminals;
 Ks/slow device control (e.g., for Teletype);
 Kf/fast device control (e.g., for disk);
 Kc/control for clock, timer, interprocessor communication

¹ Both switches have static configuration control by manual and program control

Figure 12—Block diagram of C.mmp which shows both CBSs.⁵⁰

The C.mmp is completely partitionable, i.e., it can be reconfigured so that any set of processors can work together by sharing a memory. However, there may be some interference. As for fault tolerance, if a memory module or I/O bus fails, the hardware switches can be set to isolate the bad hardware from the rest of the system. Similarly, if a processor fails the switches can be set so that it cannot access any memory modules or I/O buses. However, a failure in either of the CBSs could force the entire system to stop.

The *Multi Associative Processor (MAP)*²³ is an MSIMD system consisting of eight control units and 1024 PEs. The PEs are grouped together into 16 sectors of 64 PEs each. All of the PEs in a sector are connected via a bus. The bus of each sector can be connected to any one of the eight control units via a 16×8 CBS, as shown in Figure 13. Having eight control units allows up to eight independent SIMD programs to be executed simultaneously. Any number of processors can be dynamically allocated to any one of the eight control units; however, the most efficient partitions will be those which put all of the processors of a given sector into the same partition.

The CBS of MAP is physically centralized. Each control unit makes its own requests, and there is a control unit supervisor which arbitrates conflicts.

The simultaneity is $S=16/1024$ for intra-sector PE-to-PE communications and $8/1024$ for inter-sector communica-

tions. Since each processor is treated equally, the network is homogeneous. Any processor can broadcast a data item to all of the processors via the CBS, and can send data to, or receive data from, any other processor, so $B=R=D=1$.

If a processor fails, it can be removed from the list of available processors. This means it can never be assigned to a program. If a sector fails, all of the processors in the sector can be removed from the list of available processors. A control unit failure means one less SIMD program can be run simultaneously; however the rest of the system can run unaffected.

CONCLUSIONS

Summaries of a wide variety of methods for providing interprocessor communications in reconfigurable large-scale parallel processing systems have been presented. A set of parameters for describing the features of these networks were defined. These parameters were used in the descriptions of the networks to provide a common basis for comparison. The rest of this section discusses future research directions in network design for a reconfigurable system.

Reconfigurable large-scale parallel processing systems are becoming more prevalent as hardware costs decrease and the knowledge about exploiting parallelism in tasks increases. The interconnection networks for these systems should be restructurable under software control. In applications where it is possible, parallel programs, including interprocessor communications, should be generated automatically, with the explicit parallelism hidden from the user. In applications where this is not possible, or for those who wish to have direct command over the parallel system

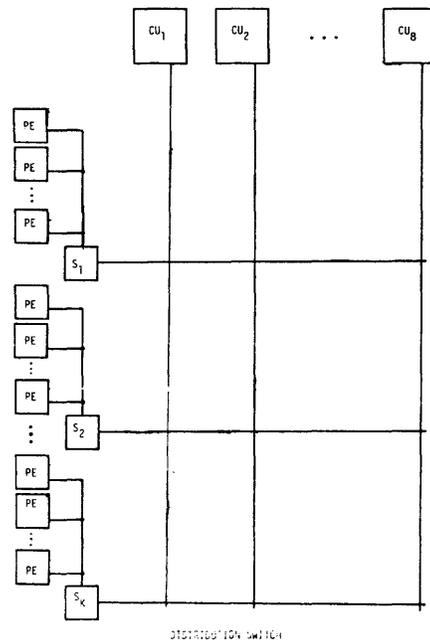


Figure 13—Diagram of the CBS for the MAP system.²³

for efficiency or research, the network control must be accessible. For ease of use, each programmer dealing with a partition or submachine of size M should write network commands based on a logical numbering of the processors, from 0 to $M-1$. Furthermore, the user should specify communication paths in terms of destination tags, and let the processors or the network hardware itself compute the data paths, such as is done with the Omega network.²¹ There should also be machine instructions for specifying particular network connections, e.g., a method to specify "+1 modulo M " on the Illiac.⁴ Since these systems and their networks will have a large number of components, fault tolerance is very important. Networks should have the ability to work around faulty switch elements, as in the SW-banyan network.²² Efficient methods for detecting faults in a network must be devised.

As the size of systems increases, the interest in SIMD machines will shift to MSIMD architectures, such as MAP.²³ To increase the range of applications these systems can handle, networks should be able to (1) allow all processors to transfer data simultaneously, (2) prevent independent users' submachines from communicating with each other, (3) allow a single user's submachines to communicate when desired, (4) allow submachines to be of different sizes, and (5) allow each submachine to control its network independently (as the ADM network can^{34,35}). MSIMD systems with such communication abilities will have some fault tolerance in that a faulty component need only shut-down the smallest size partition (e.g. an MC group in PASM³⁵). Furthermore, in SIMD applications that require high reliability, the task may be run simultaneously in several different partitions. Then the partitions can communicate with each other to confirm the validity of their results. (This assumes a suitable backup scheme for operating without or replacing the "master" controller in case of its failure.)

MIMD system networks should be able to support a high degree of simultaneity so that independent subsystems can communicate with minimum interference. Crossbar switches are too costly for large systems. Multiple bus systems such as those in CHO⁴² and MMC⁴⁹ appear to be a promising approach. Ways to implement these networks, the use of packet switching, and techniques for evaluating MIMD networks need to be examined.

PSM systems have all of the advantages of MSIMD and MIMD systems. Furthermore, they allow a single system to be built, and then have its processors operate in any combination of either mode (MSIMD or MIMD), depending on the users' needs. Thus, for example, a system may simultaneously behave as four independent SIMD machines and two independent MIMD machines, all of different sizes. In addition, a group of PEs may, for example, do preprocessing for a pattern recognition task in SIMD mode and then the same set of PEs may continue the task in MIMD mode. How well proposed PSM networks such as the ADM and SW-banyan can support such activities must be evaluated, and new approaches must be explored.

Systems capable of varying their word size will also be more prevalent. Machines such as RVAP²² and DC¹⁸ will

provide flexible systems, capable of functioning as PSM computers, where each "composite processor" operates on a user designated word size. The "carry" lines portion of the network should be easily reconfigured, as with the "carry" network in DC. The inter-"composite processor" network should have the range and domain of a network like the SW-banyan, if operating on tasks which use a large number of "composite processors" that must communicate often. As in the PSM network area, network evaluation techniques and new schemes must be investigated.

Ways in which networks can exploit LSI technology must be studied. Physically distributed networks, such as those in DC¹⁸ and CHO⁴², can be incorporated with other system components and take advantage of LSI. Most proposed and existing physically centralized networks do not take advantage of LSI, due to low complexity/pin count ratio. Future networks may make use of LSI by becoming more "intelligent." For example, architects could design switch elements capable of supporting features such as pipelining, conflict (switch contention) resolution, fault detection, fault tolerance, and destination tag-based routing.

Finally, network designers must not forget the user. Architects must remember that the networks they design must function efficiently for user problems. Therefore, networks should not be designed without considering the intended applications of the system the network is supporting. Work needs to be done in defining descriptive parameters for both networks and the communication needs of users' problems. By establishing a relation between these two sets of parameters, a problem could be analyzed to find its "communication needs parameters," and then the appropriate "network parameters" necessary to solve the problem efficiently could be determined.

REFERENCES

1. Anderson, G. A., and E. D. Jensen, "Computer interconnection structures: taxonomy, characteristics, and examples," *ACM Computing Surveys*, Vol. 7, No. 4, Dec. 1975, pp. 197-213.
2. Arnold, R. G., and E. W. Page, "A hierarchical, restructurable multi-microprocessor architecture," *3rd Annual Symposium on Computer Architecture*, Jan. 1976, pp. 40-45.
3. Baer, J. L., "Multiprocessing systems," *IEEE Trans. on Comput.*, Vol. C-25, No. 12, Dec. 1976, pp. 1271-1277.
4. Barnes, G., et. al., "The Illiac IV computer," *IEEE Trans. Comput.*, Vol. C-17, No. 8, Aug. 1968, pp. 746-757.
5. Batcher, K. E., "STARAN parallel processor system hardware," *Nat'l. Computer Conf.*, May 1974, pp. 405-410.
6. Batcher, K. E., "The flip network in STARAN," *1976 Int'l. Conf. on Parallel Processing*, Aug. 1976, pp. 65-71.
7. Batcher, K. E., "The multi-dimensional access memory in STARAN," *IEEE Trans. on Comput.*, Vol. C-26, No. 2, Feb. 1977, pp. 174-177.
8. Bouknight, W. J., et. al., "The Illiac IV system," *Proc. IEEE*, Vol. 60, Apr. 1972, pp. 369-388.
9. Davis, E. W., "STARAN parallel processor system software," *Nat'l. Computer Conf.*, May 1974, pp. 17-22.
10. Enslow, P. H., Jr., "Multiprocessor organization—a survey," *ACM Computing Surveys*, Vol. 9, Mar. 1977, pp. 103-129.
11. Feierbach, G., and D. Stevenson, *A Feasibility Study of Programmable Switching Networks for Data Routing*, Institute for Advanced Computation Phoenix Project Memorandum No. 003, May 1977.
12. Feldman, J. D., and L. C. Fulmer, "RADCAP—an operational parallel processing facility," *Nat'l. Computer Conf.*, May 1974, pp. 7-15.

13. Feng, T., "Data manipulating functions in parallel processors and their implementations," *IEEE Trans. Comput.*, Vol. C-23, No. 3, Mar. 1974, pp. 309-318.
14. Flynn, M. J., "Very high-speed computer systems," *Proc. IEEE*, Vol. 54, Dec. 1966, pp. 1901-1909.
15. Goke, L. R., "Connecting networks for partitioning polymorphic systems," Doctoral dissertation, Dept. of Electrical Engineering, University of Florida, 1976.
16. Goke, L. R., and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," *1st Annual Symposium on Computer Architecture*, Dec. 1973, pp. 21-28.
17. Jones, A. K., et. al., "Software management of Cm*—a distributed multiprocessor," *Nat'l. Computer Conf.*, June 1977, pp. 657-663.
18. Kartashev, S. I., and S. P. Kartashev, "Dynamic architectures: problems and solutions," *Computer*, Vol. 11, July 1978, pp. 26-41.
19. Kartashev, S. I., and S. P. Kartashev, guest eds., "Modular computers and networks," *Computer*, Vol. 11, July 1978, whole issue.
20. Kuck, D. J., "A survey of parallel machine organization and programming," *ACM Computing Surveys*, Vol. 9, Mar. 1977, pp. 29-59.
21. Lawrie, D., "Access and alignment of data in an array processor," *IEEE Trans. on Comput.*, Vol. C-24, No. 12, Dec. 1975, pp. 1145-1155.
22. Lipovski, G. J., and A. Tripathi, "A reconfigurable varistruce array processor," *1977 Int'l. Conf. on Parallel Processing*, Aug. 1977, pp. 165-174.
23. Nutt, G. J., "Microprocessor implementation of a parallel processor," *4th Annual Symposium on Computer Architecture*, Mar. 1977, pp. 147-152.
24. Okada, Y., H. Tajima and R. Mori, "A novel multiprocessor array," *2nd Symposium on Micro Architecture*, 1976, pp. 83-90.
25. Paker, Y., and M. Bozyigit, "Variable topology multicomputer," *2nd Symposium on Micro Architecture*, 1976, pp. 141-151.
26. Pease, M. C., "The indirect binary n-cube microprocessor array," *IEEE Trans. Comput.*, Vol. C-26, No. 5, May 1977, pp. 458-473.
27. Reddi, S. S., and E. A. Feustel, "A restructurable computer system," *IEEE Trans. Comput.*, Vol. C-27, No. 1, Jan. 1978, pp. 1-20.
28. Siegel, H. J., "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," *1975 Sagamore Computer Conf. on Parallel Processing*, Aug. 1975, pp. 106-109.
29. Siegel, H. J., "Single instruction stream-multiple data stream machine interconnection network design," *1976 Int'l. Conf. on Parallel Processing*, Aug. 1976, pp. 272-282.
30. Siegel, H. J., "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," *IEEE Trans. Comput.*, Vol. C-26, No. 2, Feb. 1977, pp. 153-161.
31. Siegel, H. J., "The universality of various types of SIMD machine interconnection networks," *4th Annual Symposium on Computer Architecture*, Mar. 1977, pp. 70-79.
32. Siegel, H. J., "Preliminary design of a versatile parallel image processing system," *Third Biennial Conf. on Computing in Indiana*, April 1978, pp. 11-25.
33. Siegel, H. J., "Partitionable SIMD computer system interconnection network universality," *16th Annual Allerton Conf. on Communication, Control, and Computing*, Oct. 1978.
34. Siegel, H. J., and S. D. Smith, "Study of multistage SIMD interconnection networks," *5th Annual Symposium on Computer Architecture*, Apr. 1978, pp. 223-229.
35. Siegel, H. J., P. T. Mueller, Jr., and H. E. Smalley, Jr., *Preliminary Design Alternatives for a Versatile Parallel Image Processor*, School of Electrical Engineering, Purdue University, Technical Report TR-EE 78-32, June 1978.
36. Siegel, H. J., P. T. Mueller, Jr., and H. E. Smalley, Jr., "Control of a partitionable multimicroprocessor system," *1978 Int'l. Conf. Parallel Processing*, Aug. 1978, pp. 9-17.
37. Siegel, H. J., and P. T. Mueller, Jr., "The organization and language design of microprocessors for an SIMD/MIMD system," *Second Rocky Mt. Symp. on Microcomputers*, Aug. 1978, pp. 311-340.
38. Siegel, H. J., R. J. McMillen, P. T. Mueller, Jr., and S. D. Smith, *A Versatile Parallel Image Processor: Some Hardware and Software Problems*, School of Electrical Engineering, Purdue University, Technical Report TR-EE 78-43, Oct. 1978.
39. Smith, S. D., and H. J. Siegel, "Recirculating, Pipelined, and Multistage SIMD interconnection networks," *1978 Int'l. Conf. Parallel Processing*, Aug. 1978, pp. 206-214.
40. Stone, H. S., "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, Vol. C-20, No. 2, Feb. 1971, pp. 153-161.
41. Stone, H. S., "Parallel Computers," in *Introduction to Computer Architecture*, H. S. Stone, ed., S.R.A., 1975.
42. Sullivan, H., T. R. Bashkow, and K. Klappholz, "A large scale homogeneous, fully distributed parallel machine," *Fourth Annual Symposium on Computer Architecture*, Mar. 1977, pp. 105-124.
43. Swan, R. J., S. H. Fuller, and D. P. Siewiorek, "Cm*: a modular, multi-microprocessor," *Nat'l. Computer Conf.*, June 1977, pp. 637-644.
44. Swan, R. J., et. al., "The implementation of the Cm* multi-microprocessor," *Nat'l. Computer Conf.*, June 1977, pp. 645-655.
45. Thurber, K. J., "Interconnection networks—a survey and assessment," *Nat'l. Computer Conf.*, May 1974, pp. 909-919.
46. Thurber, K. J., "Circuit switching technology: a state-of-the-art survey," *Comcon 78*, Sept. 1978.
47. Thurber, K. J., and L. D. Wald, "Associative and parallel processors," *ACM Computing Surveys*, Vol. 7, No. 4, Dec. 1975, pp. 215-255.
48. Widdoes, L. C., Jr., "The Minerva Multi-Microprocessor," *3rd Annual Symposium on Computer Architecture*, Jan. 1976, pp. 34-39.
49. Wittie, L. D., "Efficient message routing in mega-micro-computer networks," *3rd Annual Symposium on Computer Architecture*, Jan. 1976, pp. 136-140.
50. Wulf, W. A., and C. G. Bell, "C.mmp—a multi-mini-processor," *FJCC*, Dec. 1972, pp. 765-777.

Adaptation properties for dynamic architectures

by STEVEN I. KARTASHEV

Dynamic Computer Architecture, Inc.
Lincoln, Nebraska

and

SVETLANA P. KARTASHEV

University of Nebraska-Lincoln
Lincoln, Nebraska

and

C. V. RAMAMOORTHY

University of California, Berkeley
Berkeley, California

INTRODUCTION

The main characteristic of most powerful parallel systems is that their architecture is oriented towards specific types of problems to be computed. Such an orientation limits the range of tasks that can be effectively solved. As a result, systems are produced in small quantities which must bear all the costs of software and hardware development, and which are, as a result, very expensive. This slows down the computerization of many other complex problems just because they require that new high cost parallel systems be manufactured for their computation.

That is why it is timely to develop parallel systems that can change their architecture via software and thus can be oriented towards whatever problem is currently being computed. Such systems provide a programmer with an essentially new option—during the run-time of a single task he can perform multiple variations in the system architecture, each time selecting those system structures that speed up computation.

In general the architectures which change via software their structure to adapt to that of a program are sometimes called *adaptable* architectures. Now we may distinguish three classes of adaptable architectures—*microprogrammable*, *reconfigurable* and *dynamic*.

Consider historic evolution of adaptable architectures. The first computers had a static architecture which completely precluded any variation in the interconnections between their devices or functional units. With the advent of microprogrammable computers, a programmer was allowed to reconfigure interconnections between different devices such as registers, adders, counters, etc. This resulted in software-controlled variation of an instruction's microprogram. Consequently, computation was speeded up by the expedience of selecting microoperations which were more

task-oriented. Therefore, first adaptable architectures appeared in microprogrammable computers.

The next stage of adaptation had been achieved by introducing reconfigurable interconnections between various functional units, such as processors, memories, I/Os. Such architectures began to be called reconfigurable architectures. One of the first systems of this type was described by Estrin.^{1,2} His restructurable system could change into a variety of problem-oriented special purpose configurations which speeded up computations for several classes of applications.

In the beginning of the 60s there appeared reconfigurable array parallel systems in which there was the option to change the number of and interconnections between processors working in the array.^{3,4} Further development of array processors proceeded in the direction of establishing new patterns of configurations between various processors and in changing the processors' sizes. Subsequently, reconfiguration was used in other types of systems—pipeline, multiprocessing, multicomputer, etc.

In all, reconfigurable architectures provide the following performance gains:

1. In array systems an augmented vector parallelism is achieved by enlarging the dimension of a data vector processed with a single instruction.⁵⁻¹¹
2. In pipelined systems the number of dummy-time intervals in the pipeline is minimized, caused by the disparity between program and pipeline structures.¹²⁻¹⁷
3. In multiprocessing systems an additional program parallelism is created by minimizing idle time of processor resource.¹⁸⁻²⁶
4. In multicomputer systems computations are speeded up by allowing a closer match between the multicom-

puter network topology and that of a complex computed task.²⁷⁻²⁹

5. System reliability may be improved by deactivating faulty modules which may then be replaced with spares.³⁰⁻³⁸

Further evolution in adaptable architectures was marked by appearance of LSI modules with high throughput. Consequently, it became possible to achieve a higher degree of reconfiguration by activating not only interunit but also intermodule connections. As a result, the available hardware resources could be partitioned via software into a variable number of computers with variable sizes. This allowed one to accomplish the architectural adaptation to the number of available program streams by switching the architecture into the matching number of independent computers.³⁹⁻⁵² Such architectures were named dynamic.⁵³

Although the basic feature of dynamic architectures which distinguishes them from other adaptable architectures is the adaptation to the number of parallel program streams, they may accomplish other adaptations, such as:

- a. The instruction set adaptation when the control unit may activate selectively not a single but tens of different instruction sets, each of which is oriented towards a separate class of dedicated applications. For instance, one set is dedicated to handling trigonometrical functions, another one to array processing, a third one handles FFT occurring in signal processing, etc.
- b. Pipeline adaptation means adaptation to parallel program streams when the architecture assumes different states characterized by the number of concurrent pipelines and the number of stages in each pipeline: or adaptation of the number of pipeline stages when the instruction activates the number of consecutive stages in the pipeline which matches the number of operations it realizes; or adaptation to operation time in each stage when each stage of a pipeline changes the time of operation; or adaptation on conditional branch, etc.
- c. Array adaptation means software formation of an architecture suitable for array processing. Furthermore, since different size computers are possible for a single state, array processing may be performed over different sizes of data words. When a dynamic architecture is switched to a state characterized by the array mode of operation, one of the computers assumes the function of a computer supervisor and broadcasts instructions and data addresses to all the other computers of the array, etc.

Finally, dynamic architectures require that each program be preprocessed by a so-called adaptation system in order to find the best architectural states to be assumed by an architecture in execution of a given set of concurrent programs.

In all, modern modular architectures may accomplish all three classes of adaptations mentioned previously—microprogrammable, reconfigurable and dynamic. For instance, the architecture may reconfigure interconnections on a mi-

crolevel (registers, adders, conditional flip-flops) and thus accomplish a microprogrammable adaptation. Or, it may reconfigure on the level of separate functional units and perform a reconfigurable adaptation. Or, it may reconfigure on the level of separate modules and perform a dynamic adaptation. Thus one obtains microprogrammable, reconfigurable and dynamic properties implemented in a single modular architecture.

Given paper is focused on adaptations to executed programs performed by dynamic architectures. Degrees of such adaptations will be characterized by so-called adaptation parameters the paper introduces. These are easily computed objective measures of evaluations expressed either in terms of time and/or in terms of hardware complexities. The paper also outlines general ideas of Dynamic Pipeline organization.

ADAPTATION PARAMETERS FOR DYNAMIC ARCHITECTURES

Introduction of adaptation parameters is caused by the following reasons:

- a. They allow evaluation of different dynamic architectures from the viewpoint of how well they take into account computational specificities of programs.
- b. They may improve the work of an operating system. Indeed, an operating system must preprocess each program in order to find the best architectural states to be assumed by the dynamic architecture in its execution. Since several alternatives may be obtained, each of them must be evaluated using the adaptation parameters.

Let us show what adaptation parameters may be used to characterize major adaptation properties of dynamic architectures:

Bit size adaptation

Dynamic architecture must be capable of forming variable size computers from the available hardware. Each new partitioning of the resource into a new set of concurrent computers is performed during the architectural switch from one state to another.^{41,42} The time of such a switch will be called *the speed of bit size adaptation* (SBA). The SBA may vary depending on the concrete hardware organization. Since during this time execution of some programs may be halted, the SBA should be minimized.

Example. As shown in Reference 41, for a DC group, an architectural transition from one state to another is performed by a special architectural switch instruction whose duration is $2mt_0 + 12t_0$, where $m \cdot t_0$ specifies memory speed, and t_0 is the clock period equal to the time of eight-bit addition. For $t_0 = 200$ nsec and $m = 1$, $SBA = 400 + 2400 = 2.8$ μ sec. •

The correct selection of computer size is another factor in executional speed-up. Since not only different programs but also different portions of the same program may handle

variable size data words, one program may be computed by a sequence of different size computers so that each task of this program is computed by a minimal size computer. This achieves executional speedup for all processor-dependent operations. This speedup depends on how accurately a computer's size matches that of the data words. Indeed, as an ideal computer, one may consider a computer which for each instruction assumes the maximal size of the data word used in its execution. However, this may lead to a rate of variation in computer sizes which coincides with the instruction rate, i.e., for a program made of N instructions, the overhead caused by reconfiguration will be $SBA \times N$.

Therefore, programs should be partitioned into tasks, so that each task is assigned an individual computer size. Each task may contain hundreds of instructions. For this case there will be some processor operations in each task which handle data words of smaller size than that of the computer selection. Execution of each such operation will be delayed by the time Δt , and the delay in execution of the entire task will be characterized by its *precision of bit size adaptation* (PBA). Clearly, $PBA = \sum_{i=1}^f \Delta t_i$, where f is the number of processor-dependent operations requiring a smaller computer size.

There is another factor which leads toward an increase in the PBA. Since the processor is assembled from h -bit modules, computer size is obtained in h -bit increments. Presently $h=4$ or 8. However, technological advances leading toward a growth in the chip size will also lead toward an increase in h , but this will increase the PBA and therefore slow down computer processing.

Example. Let the program containing 800 instructions be partitioned into four tasks (Figure 1). For the first task, computed by a 48 bit computer, one constructs a diagram of bit sizes required by each processor operation in this task.⁵⁴ For the i th operation, delay in execution is defined as $\Delta t_i = T(48) - T_i$, where $T(48)$ is the time of 48-bit addition. The time $T(48)$ depends on the organization of carry propagation in the processor. Let the adder be equipped with four-bit group carry propagation, and suppose that the CLA circuit introduces a $2t_d$ delay, where t_d is the time to switch

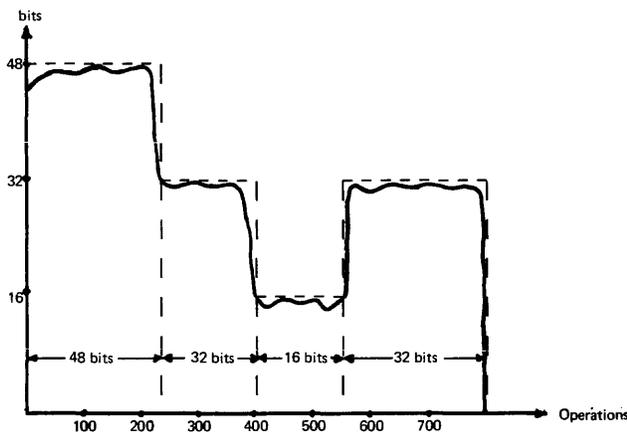


Figure 1—Partitioning of the program into tasks computed respectively by 48-, 32-, 16-, 32-bit computers.

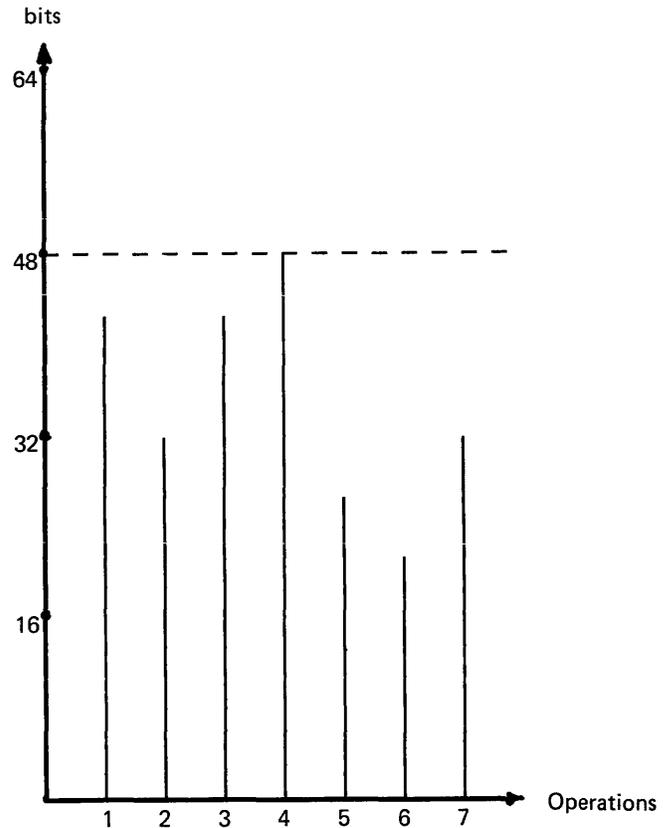


Figure 2—Diagram of bit sizes for processor dependent operations in 48-bit task.

one gate. Since 48-bit addition requires 12 CLA groups, $T(48) = 12 \cdot 2t_d = 24t_d$. In this task, the first operation handles 40-bit words (Figure 2) and its time $T_1 = 2t_d \cdot 10 = 20t_d$. Therefore $\Delta t_1 = T(48) - T_1 = 4t_d$. The second operation handles 32-bit words and $T_2 = 16t_d$ and $\Delta t_2 = 8t_d$, etc. By having found times Δt_i for all processor dependent operations and assuming that $t_d = 10$ nsec, we find that $PBA = 827t_d = 8.27 \mu\text{sec}$. •

Adaptation to program streams

Dynamic architectures allow one to achieve a trade-off between computer sizes and parallelism, i.e., by reducing bit sizes one may augment the number of concurrent computers formed from the same equipment. Such an architectural feature allows one to maximize the number of program streams computed by the same equipment. As shown in Reference 54, the assignment of independent programs to computers working in different architectural states is performed by the operating system. The effectiveness of this assignment may be evaluated with a so called *resource utilization factor* (RUF) found for each architectural state, N_a , as follows. Assume that for a state, N_a , a complete utilization of the resource is achieved if all its concurrent computers work continuously during the entire time, T_a , the state N_a exists. However, for the real case, some of the computers of state N_a may work only part of the time, so

that if N_a is characterized by d computers with sizes $h \cdot k_1, h \cdot k_2, \dots, h \cdot k_d$, working during times T_1, T_2, \dots, T_d respectively, then the RUF for state N_a is found as follows:

$$\text{RUF}(N_a) = \frac{h \cdot k_1 \cdot T_1 + h \cdot k_2 \cdot T_2 + \dots + h \cdot k_d \cdot T_d}{h \cdot T_a \cdot (k_1 + k_2 + \dots + k_d)} \cdot 100\%$$

Since $k_1 + k_2 + \dots + k_d = n$, where n is the number of computer elements,⁵³ then

$$\text{RUF}(N_a) = \frac{k_1 \cdot T_1 + k_2 \cdot T_2 + \dots + k_d \cdot T_d}{n \cdot T_a} \cdot 100\%$$

Note, that if an $h \cdot k_1$ -computer does not work in the state N_a , then the time $T_i = 0$.

Example. For state N_6 , let a DC group with $n=4$ computer elements and $h=16$ form two 16-bit computers $C_1(1)$ and $C_2(1)$ and one 32-bit computer $C_3(2)$. Let this state be established for 32 min, but $C_1(1)$ works during $T_1=24$ min, $C_2(1)$ works during $T_2=27$ min and $C_3(2)$ works for $T_3=32$ min. For $C_1(1)$, $k_1=1$; for $C_2(1)$, $k_2=1$, for $C_3(2)$, $k_3=2$. Thus

$$\text{RUF} = \frac{1 \cdot 24 + 1 \cdot 27 + 2 \cdot 32}{4 \cdot 32} \cdot 100\% = 89.8\% \bullet$$

Adaptation to program structures

A significant executional speed-up may be achieved by introducing special instructions which take into account specifics of the computed programs. Such instructions may activate execution of complex sequences of operations, such as $\frac{A^2 - B^2}{C} \cdot D$ or $(A + B - C^2) \cdot (A - C)$, etc. Each such instruction is equivalent to several conventional instructions, so its use results in a minimization of the number of memory access operations. However, introduction into an instruction set of a large number of dedicated instructions narrows the area of application for a parallel system. To offset this problem one has to augment the size of the instruction set. However, such an augmentation is accompanied by an increase in the op-code size. It then follows that the word size, h , of a memory element, ME , also increases. However, as was shown above, it is unreasonable to increase h because it leads to a corresponding increase in the size of each computer element. For instance if one increases h from 16 to 24 bits, then the dynamic architecture may form computer sizes in 24-bit increments, i.e., it forms 24-, 48-, 72-bit computers, etc. Therefore, architectural adaptation to program streams and bit sizes will decrease, leading to a worsening of the PBA and RUF factors introduced above. As a result, the architecture reduces the number of parallel program streams which may be formed on the existing equipment. It will likewise increase delays in execution of processor dependent operations. That is why the expansion of an instruction set must be accompanied by no consequential increase in the size of the op-code.

Let us introduce one technique named *dynamic adaptation of an instruction set* which accomplishes this objective. The idea of this technique consists of the following. The

control unit may activate selectively not a single but tens of different instruction sets each of which is oriented towards a separate class of dedicated applications. For instance, one set is dedicated to handling trigonometrical functions, another one to array processing, a third one handles FFTs occurring in signal processing, etc. Each set contains for instance 256 instructions partitioned into two categories—dedicated and general-purpose. The difference between sets is in the dedicated instructions. A computed program activates the instruction set which most closely matches its computational algorithm. Such selective activation is made via software by writing a special control code γ_i .

Consider now how one may organize the dynamic adaptation of an instruction set. The control unit contains a special unit which implements the Dynamic Instruction Set, DIS unit. This unit contains p states, IS_1, IS_2, \dots, IS_p , each of which is identified with one instruction set. Activation of set IS_i is performed by the writing of a special code γ_i . Transition of the DIS unit from one instruction set, IS_i , to the next instruction set, IS_j , is performed by writing another code γ_j corresponding to IS_j . When the Dynamic Instruction Set unit establishes state IS_i , the computer executes instructions from instruction set IS_i . This is recognized by the op-code D so that if w is the bit size of op-code D , each IS_i contains 2^w different instructions. Therefore, the same op-code corresponds to p different instructions belonging to IS_1, IS_2, \dots, IS_p respectively. It follows that D and γ_i together recognize a unique instruction from the instruction set IS_i . Thus this organization allows one to implement a Dynamic Instruction Set without any consequential increase in the size of code D . Before the computation of a program or task one has to write to the control unit only one code, γ_i , and the computer begins to activate instructions belonging to the instruction set IS_i .

Selection of the instruction set IS_i which most closely matches the structure of a given program is performed by the assignment subsystem of the operating system that also defines the computer sizes for consecutive tasks of a computed program.⁵⁴ To this end, the same program graph is used for analysis that was constructed for finding computer sizes. For each graph node, the operating system specifies all the sequences of consecutive operations that this node executes. For instance, if node b executes formula $[(A^2 - B^2) \times C] / D$, then the sequence of operations which is found is $(\times, \times, -, \times, \div)$. Next, for each sequence of operations constructed in the node, the operating system finds the frequency, N , of its appearance in the program. The number N will be used in the future for determining the executional speed-up gained from using this sequence as a dedicated instruction. Consider now how the operating system may select the most appropriate instruction set. This is done by finding, for each instruction set IS_i , the executional speed-up SPA_i (speed-up on program adaptation) prompted by the use of IS_i in the computation of a given program. Indeed, an operation sequence composed of d consecutive operations is executed by a conventional microprogrammable computer as a subroutine made of d instructions where each instruction executes one operation. (We exclude from consideration all instructions fetching data for the d operations,

because it is assumed that the number of such instructions is the same for a complex dedicated instruction and for a subroutine, and thus depends solely on the way storage of the data words required for the operation is organized.) Each instruction in such a subroutine takes time t_m for instruction fetch and the time t_{op} for operation execution. It then follows that if one operation sequence containing d operations is organized as one dedicated instruction, the resulting speed-up, SIA_j , prompted by instruction adaptation is $SIA_j = t_m \cdot (d-1)$.

Therefore, the procedure for finding the speed-up by program adaptation, SPA_i , for instruction set IS_i , consists of the following: For each sequence of operations formed in the program, the operating system finds whether or not such a sequence may be implemented as a single dedicated instruction, I_j , from instruction set, IS_i . If it does, then the operating sequence finds the overall speed-up $SIA_j \cdot N_j = t_m(d_j-1) \cdot N_j$ where N_j is the frequency with which this sequence appears in the program. If it does not, then $SIA_j \cdot N_j = 0$ and the next sequence is analyzed. Having found all $SIA_j \cdot N_j$ speed-ups, the operating system finds the total SPA_i for the instruction set IS_i :

$$SPA_i = \sum_{j=1}^t SIA_j \cdot N_j$$

where t is the number of operation sequences constructed for the program. Of the p times, $SPA_1, SPA_2, \dots, SPA_p$ found for the p instruction sets IS_1, IS_2, \dots, IS_p respectively, the maximum, SPA^* , is selected. The respective IS^* provides the best adaptation to the program structure.

Example. Let t_m be 200 nsec and the operation sequence $A \cdot B - (C + A)$ be repeated in the program 43 times ($N_1 = 23$). Since this sequence has three operations, $SIA_1 = 200 \cdot (3-1) = 400$ nsec. A second operation sequence $(A-B) \cdot C$ has $N_2 = 35$ and $SIA_2 = 200$ nsec. The third sequence $(A-B+C)/C-D+C$ has $N_3 = 72$ and $SIA_3 = 200 \cdot 4 = 800$ nsec; the fourth sequence

$(A^2 - B^2) \times (C^2 - D^2) / A + B$ has $N_4 = 46$ and $SIA_4 = 200 \cdot 8 = 1600$ nsec. Let instruction set IS_1 have dedicated instructions 1, 2, 4 and IS_2 have dedicated instructions 1, 2, 3. Find SPA_1 and SPA_2 :

$$SPA_1 = 400 \cdot 43 + 200 \cdot 35 + 1600 \cdot 46 = 97.8 \mu\text{sec.}$$

$$SPA_2 = 400 \cdot 43 + 200 \cdot 35 + 800 \cdot 72 = 71.8 \mu\text{sec.}$$

Therefore IS_1 achieves a better adaptation to the program structure. •

Array adaptation

Dynamic architecture may form an architecture suitable for array computations. Furthermore, since different size computers are possible for a single state, array processing may be performed over different sizes of data words. When dynamic architecture is switched to a state characterized by the array mode of operation, one of the computers assumes the function of a computer supervisor and broadcasts instructions and data addresses to all the other computers of the array. This transfer is done not through the I/O devices but by using the memory-processor interconnection bus which connects memory modules to processor modules. Note that introduction of different size computers requires that the computer supervisor broadcast instructions at the processing rate of the largest computer. Such synchronization in array processing may be accomplished if the largest computer sends a completion signal the moment it completes execution of the present instruction. This signal is sent through a special line of dynamic signals⁴¹ which connects all PEs.

Example. Let the hardware resource containing $n=6$ computer elements (Figure 3) be switched into an array structure containing three computers $C_1(1), C_2(2), C_4(3)$. Assume that $C_1(1)$ is the computer supervisor, so its ME_1 stores all instructions which are broadcasted to all PEs. The same data

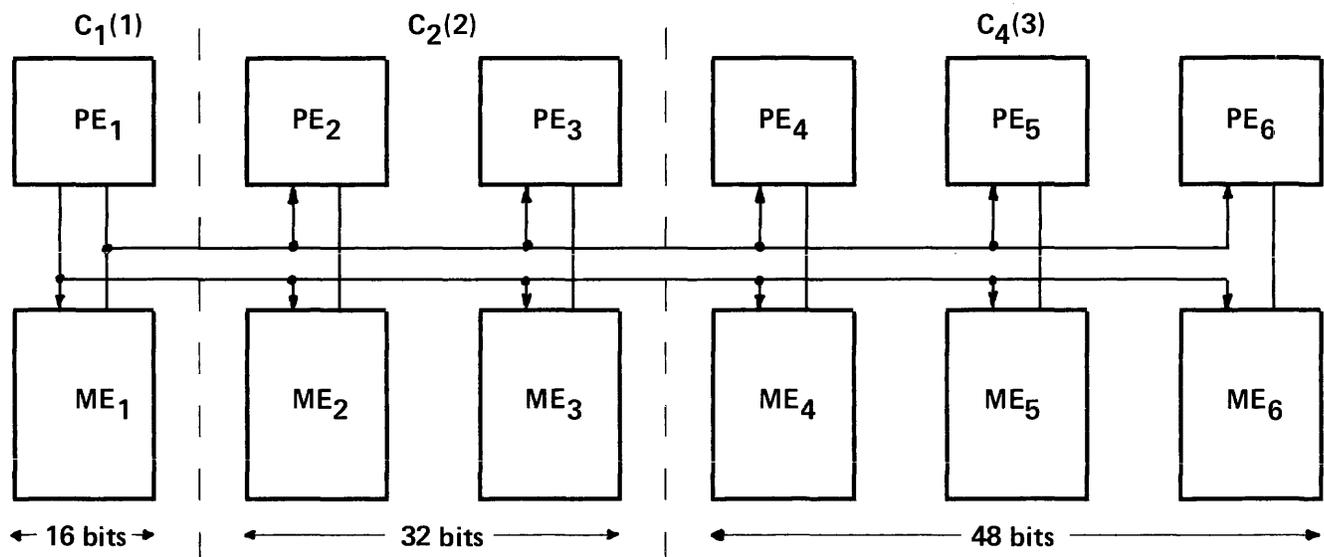


Figure 3—Array adaptation of dynamic architectures.

address generated by PE_1 is sent to all six MEs , causing concurrent fetch of a data vector composed of a 32-bit and a 48-bit operand handled by $C_2(2)$ and $C_4(3)$ respectively. The rate of instruction broadcast is limited by the speed of the 48-bit computer. •

Note: The option of forming various size computers for array computations may lead toward an increase in the dimension of a data vector computed with a single instruction, i.e., the number of operands computed in parallel may increase. This may be accomplished by selecting the minimal computer size for each operand.

Therefore, the array mode of operation may be characterized by the *array adaptation of equipment* (AAE) which shows the percentage of redundant equipment left in each computer by selecting a computer size exceeding that of an operand. Namely, in the array made of r data streams, for an i th data stream ($i=1, \dots, r$), one finds the maximal data word size, WS_i . If the i th stream is computed by a computer of size CS_i , then for the i th stream, $AAE_i = (1 - WS_i/CS_i) \cdot 100\%$ and the overall AAE shows the percent of unused equipment in the entire array:

$$AAE = \left(1 - \frac{WS_1 + WS_2 + \dots + WS_r}{CS_1 + CS_2 + \dots + CS_r}\right) \cdot 100\%$$

Example. Let a program requiring array computations have four data streams with $WS_1=20$ bits, $WS_2=28$ bits, $WS_3=22$ bits and $WS_4=32$ bits. Let this program be computed by two dynamic architectures. The first one forms computer sizes in eight-bit increments, i.e., $h=8$, and the second one has $h=16$. Find the AAE for both architectures. The first architecture forms the following computer sizes in the array: $CS_1=24$ bits, $CS_2=32$ bits, $CS_3=24$ bits and $CS_4=32$ bits. Therefore,

$$AAE(I) = \left(1 - \frac{20+28+22+32}{24+32+24+32}\right) \cdot 100\% \\ = (1-0.91) \cdot 100\% = 9\%$$

The second one has $CS_1=32$, $CS_2=32$, $CS_3=32$, $CS_4=32$.

$$AAE(II) = \left(1 - \frac{20+28+22+32}{32 \cdot 4}\right) \cdot 100\% \\ = 20.3\% \bullet$$

Pipeline adaptation

Dynamic architectures must be capable of switching into configurations specific for pipeline computations. They will then be called dynamic pipeline architectures (DPA). Consider the properties a DPA should possess in order to achieve adaptation to an executed program.

Adaptation to parallel streams

Since a computed program may be decomposed into several pipelined streams, DPA must be capable of assuming different architectural states where each state is specified by the number of concurrent pipelines and the number of

stages in each pipeline. A transition from one state to another should be performed via software, during the time of program computation. This allows the performing of dynamic redistribution of the resource by selecting pipelines which take into account the specificity of the computed programs.

Example. Let a DPA be equipped with S architectural states, N_0, N_1, \dots, N_{S-1} . Then for state N_0 , the entire resource goes into the formation of, say, three pipelines each containing five stages, i.e., $N_0=(3 \times 5)$. For another state, N_1 , it is formed into two pipelines with four stages each and two pipelines with three stages each, $N_1=(2 \times 4, 2 \times 3)$. For the third state N_2 , it forms three pipelines with six, five, and four stages respectively, i.e., $N_2=(1 \times 6, 1 \times 5, 1 \times 4)$. •

Adaptation on operation sequences

As was just shown, effective adaptation to a program may be achieved if the computer architecture is equipped with dedicated instructions each of which realizes a complex sequence of operations. When such an instruction is executed by the pipeline, each state of the pipeline must be assigned to one operation in the sequence. It then follows that each stage must be capable of executing any operation. This may be achieved if each pipeline stage is implemented as an h - k -bit computer capable of executing any operation provided by an instruction.

Example. A single pipeline must be capable of implementing different sequences of operations such as $(-, +, \times, \div)$ in arithmetic expression $[(A-B) \times (C+D)]/E$, or $(\times, -, \div, +)$ in $(A \times B - C)/D + E$, etc. •

Adaptation on the number of pipeline stages

One of the basic problems of modern pipelined systems is their inability to quickly vary the number of stages contained in a single pipeline. Indeed, existing techniques⁵⁵⁻⁵⁷ provide that instructions bypass the unneeded stages. But this may involve conflicts when one instruction, as a result of such bypassing, jumps to the operands prepared for preceding instructions. On the other hand, if no such bypassing is implemented, the pipeline contains a permanent number of stages specified by the instruction containing the maximal sequence of operations. All other instructions which implement shorter sequences are slowed down because they have to pass through all stages. To alleviate this problem, each instruction must activate the number of consecutive stages in the pipeline which matches the number of consecutive operations it realizes. Consequently, the duration of each instruction will depend only on the number of its consecutive operations.

Example. In a pipeline made of five stages the instruction which implements the formula $(A+B) \times C - D$ containing three operations $(+, \times)$ must be executed only in the first three stages, i.e., the computational result should be output

by the third stage and be prevented from going to the fourth and fifth stages. •

Adaptation to operation time in each stage

Each stage of a pipeline may speed up execution if it changes the time of operation. As was shown in References 41 and 42, such a variation may be accomplished if each $h \cdot k$ -bit computer, implemented as a stage, is equipped with modular control organization, which can change the time of operation with a special control code p , where p is the number of minor clock periods, t_0 , a processor-dependent operation requires to execute in a given size computer. As was shown in Reference 42, this code p may also reconfigure the processor into a minimal size. Therefore, if each pipeline instruction stores its own code p , then each pipeline stage will be able to reconfigure into the minimal size computer and generate the minimal time interval for a processor dependent operation. As for processor-independent operations (Boolean, shifts, conditional branches on $=$, \neq , tests of flip-flops, etc.), the modular control organization may speed up their executions as well. The reason for this is that for any size computer the same clock period, t_0 , the time of eight-bit addition, is used. That is, a processor-independent operation may be executed during one t_0 rather than during the longer time of a processor-dependent operation.

Therefore, if each stage is realized as an $h \cdot k$ -bit computer, then the time of operation in this stage may range from one t_0 to $t_0 \cdot k$. For instance, in a 64-bit computer with no CLA circuits, $h=8$, $k=8$ and the time of operation may range from one t_0 to $8t_0$. If CLA circuits are used, a $p < 8$ is selected. It then follows that if the time of an operation executed in the i th stage is also adaptable, one obtains additional executional speed-up in the pipeline.

However, the problem which must be overcome is the problem of so-called *pipeline races*. Indeed, variation of operation times in pipeline stages may lead to a situation in which an instruction containing a shorter operation executed in the i th stage may beat the preceding instruction which executes a longer operation in the $(i+1)$ th stage. Thus, the result of the i th stage will be routed to the $(i+1)$ stage before the $(i+1)$ stage finishes its operation. As will be shown in the fourth section pipeline races can be easily solved in dynamic pipeline architectures by introducing synchronous movement of instructions from stage to stage. For this case, the pipeline rate is governed by the same traffic rules which are applicable to a highway traffic when a police car appears. Assume that from the viewpoint of a pipeline, a police car has a long operation time. Thus all cars (instructions) which follow a police car within the county limits (pipeline) move with the speed of this car. On the other hand, preceding cars may move with higher speed, i.e., for preceding instructions with shorter operations, the pipeline rate may increase. Therefore, the rate of both a highway and a pipeline may increase if the highway has no police car and the pipeline contains no instructions with long operations.

Example. Let us compare two types of pipeline: P_1 with permanent rate and P_2 with changeable rate. Assume

that P_1 and P_2 have pipeline stages implemented as 64-bit computers so that P_1 's rate is that of a 64-bit addition whereas P_2 's rate depends on the sizes of data words and operation types. Let the time of 64-bit addition, $T(64)=400$ nsec, and the time of 32-bit addition be $T(32)=200$ nsec. Find the times, $T(P_1)$ and $T(P_2)$, required by both pipelines for execution of the same operation sequence $((A \wedge B) + C - F) > K$ corresponding to a conditional branch instruction where A, B, C, F, K are 32-bit operands. Since the first pipeline has the same time for each stage and the operation sequence $(\wedge, +, -, >)$ takes four operations, $T(P_1)=400+400+400+400=1600$ nsec. As for the second pipeline, it executes 32-bit addition for $t(32)=200$ nsec and logical multiplication for $t_0=100$ nsec. Therefore it executes instruction $((A \wedge B) + C - F) > K$ during the time $T(P_2)=100+200+200+200=700$ nsec. Thus, implementation of changeable rates in pipelines is a source of additional speed-up. •

Adaptation on conditional branch

As was shown in Reference 55, a general weakness of pipeline architectures is pipeline drains due to conditional branching. Indeed, a pipeline architecture may have dummy-time intervals when no processing is performed if, as a result of a conditional test, the program switches to another instruction sequence which was not already being processed by the pipeline stages. The problem of conditional branch may be solved if the architecture switches into architectural states containing several independent pipelines, of which one is selected as the main and others are subsidiary. For this case true (incremental) and false (specified with jump address) program sequences may be computed respectively by two independent pipelines, main and subsidiary, where a subsidiary pipeline is switched into operation only during the respective conditional branch instruction, i.e., its instruction memory replicates the entire program. If the conditional branch is made to the instruction sequence computed in the subsidiary pipeline, then it transfers all computational results necessary for further computations to the main pipeline and stops computation.

Note, if several subsidiary pipelines are available, the architecture may effectively organize multiport branching, minimizing the number of dummy-time intervals. Since DPA may form several concurrent pipelines for a single architectural state, it may effectively solve the problem of adaptation on conditional branch.

Example. Let the conditional branch instruction I_1 requiring execution of the conditional test $(A - B)^2 / K + C > W$ store jump address A_F . Since decision on selection of either true or false program sequences is made only at the fifth stage of the basic pipeline, it begins execution of instructions I_{2T} and I_{3T} , which immediately succeed I_1 in the true sequence, before it completes the conditional test. However, before entering the basic pipeline, instruction I_1 transfers the jump address A_F to the subsidiary pipeline, which begins execution of the false instruction sequence I_{2F}, I_{3F} , etc. If the fifth stage of the basic pipeline transfers control to the

false sequence executed in the subsidiary pipeline, then each i th stage of the subsidiary pipeline ($i=1, \dots, 4$) broadcasts its computational result to the $i+1$ stage of the basic pipeline. Therefore, the basic pipeline continues computation as if it had contained the false sequence. •

Adaptation to program compatibility

As was noted above, a dynamic architecture may organize execution of a program with a sequence of computers which are different in size. Such organization speeds up computations and augments program parallelism using the same resource. However, a real performance improvement may be realized only if during a transition from one computer size to another, the program undergoes only insignificant changes in the instruction fields. Therefore an important characteristic of dynamic architecture is how well it adapts to execution of a program by a sequence of computers with variable sizes. Ideal was the case when complete universality of executed programs was achieved.^{41,42}

This was the case requiring no modifications in instruction fields when a program was computed by different sizes computers. The following architectural solutions implemented such program universality:

1. Modular control organization,^{42,58} which eliminated from instructions all codes which changed their meanings when computer size changed. Instead, it provided that all such codes be stored in every LSI module of an $h \cdot k$ -bit computer.
2. A new memory allocation technique, called parallel-serial exchange, which provided (a) a unique instruction size (h -bits); (b) storage of instructions consecutively in a single memory element having the same width (h -bits); and (c) storage of an $h \cdot k$ -bit data word in a single parallel cell of k memory elements, all specified by the same relative address. Such a storage technique maintained the sizes of data and program arrays when they were moved from one computer to another. As a result, no modifications of addresses (operand or jump) stored in the instruction field were required. However such universality was achieved only for DC group computers.^{41,42}

Therefore, any dynamic architecture must be characterized by the time a computed program adapts, before it may be executed. This time is called the Time of Program Adaptation (TPA),

$$TPA = \sum_{k=1}^N t_k$$

where t_k is the time to modify a k th program instruction, N is the overall number of instructions which should be modified.

The use of adaptation parameters in operating systems

The adaptation parameters introduced above should be used by the operating system in order to specify what states a dynamic architecture should assume in order to enhance performance. A program written in a high-level language is preprocessed by the operating system, which consists of three subsystems—adaptation, assignment and monitor.

The adaptation system finds a dedicated instruction set IS_i to be activated by each user program in Dynamic Instruction Set unit.

The assignment system assigns the available hardware resources between user programs and constructs a flow chart of architectural states assumed by a dynamic architecture in executing a given set of concurrent programs.

The monitor system supervises correct execution of the flow chart constructed by the assignment system.⁵⁴

Let us briefly describe each of the subsystems.

The adaptation system analyzes a program written in a high-level language in order to define a set of complex dedicated instructions which speeds up computation of this program. Since during such analysis one may obtain several sets of dedicated instructions, the adaptation system has to evaluate each such set from the viewpoint of the executional speed-up it gives. To this end it finds the speed-up on program adaptation, SPA_i , for each dedicated instruction set and selects that instruction set which gives a maximal SPA.

In case that program should be partitioned into tasks each requiring its own dedicated instruction set, IS_i , the adaptation system assigns to each task the most appropriate instruction set and specifies the sequence of different instruction sets required to execute this program. Transition from instruction set IS_i to IS_j is performed by a special program instruction which stores code γ_j .

Note that the adaptation subsystem should also preprocess programs which are computed by a dynamic pipeline architecture. For this case a program is partitioned into a sequence of tasks, TA_1, TA_2, \dots, TA_k , in which TA_i is characterized by the maximal length F_i of the pipeline required. Each instruction belonging to task TA_i activates only w consecutive stages in this pipeline where $w (w \leq F_i)$ is the number of its operations, each of which is executed by one pipeline stage. The number F_i is identified by the assignment subsystem considered below.

The assignment system may have two modifications dependent on the type of computations performed by dynamic architecture—conventional and pipelined. Basic features of the conventional assignment system are considered in Reference 54. As for a pipelined system, their development is one of the objectives of current research performed by these authors.

For conventional computations, the assignment subsystem specifies a sequence of minimal size computers which may execute every user program. Reference 54 describes techniques for partitioning a program into tasks where each task is computed by a permanent size computer. Since there

are different alternatives in this partitioning, each has to be evaluated using the precision of bit size adaptation criteria, PBA, considered above. Since each PBA characterizes the delay in execution of a single task introduced by selecting an inaccurate computer size, then the partitioning should be performed which gives the minimal delay in execution of all tasks.

Next, the available hardware resources have to be assigned among concurrent user programs, each of which is executed by a sequence of the minimal size computers found earlier. Since the assignment system may find several alternatives for assigning programs to computers, for each alternative it has to compute the resource utilization factor, RUF, which shows the percentage of the equipment use achieved with this alternative. Thus, it selects the option characterized by the highest RUF.

For pipeline computations, each program is first preprocessed by the assignment subsystem and then by the adaptation system. The reason for this is as follows. In the execution of a set of concurrent problems, a dynamic pipeline architecture executes a system flow chart. Each state of this flowchart is specified by the number and lengths of concurrent pipelines. It then follows that construction of a flowchart establishes the length (number of stages) of each pipeline in each architectural state. Thus a sequence of operations assigned to one pipeline cannot exceed its length. If one sequence exceeds the length of the pipeline where it is computed, then it has to be split into several sequences, each of which has to be assigned to a separate pipeline instruction. Therefore, for each program one may find a dedicated instruction set only after construction of the system flow chart, because each architectural state of the flowchart establishes the limit on the number w of consecutive operations to be assigned to a single dedicated instruction. As was shown, $w \leq F$, if this instruction is assigned to a pipeline with F stages. The basic features of the monitor system for conventional computations were described in References 41 and 42.

For pipelined computations they are yet to be developed.

DYNAMIC PIPELINE ARCHITECTURE

Earlier we established the properties needed by a dynamic pipeline architecture (DPA) in order to be effectively adapted to executed programs. Let us show that all these properties may be implemented in a DPA assembled from DC groups described in Reference 42.

Consider now how one may organize one pipeline. Its hardware resource may be formed into a single computer supervisor, C_0 , and several $h \cdot k$ -bit computers, C_1, C_2, \dots, C_F , forming consecutive pipeline stages. Computer C_0 stores instructions in memory M_0 and fetches them to processor P_0 (Figure 4). The C_0 computer's size matches that of one instruction. Each pipeline computer C_i has memory M_i for storing data, processor P_i , and general register set m_i , which stores temporary results required by the P_i pro-

cessor. These are either computed by P_i or by other processors.

Two consecutive pipeline stages C_i and C_{i+1} are separated by the connecting element MSE_i , which may assume two modes of operation:

- a. Right transfer—it receives the pipelined instruction and sends it to the next stage C_{i+1} with a delay of one interval.
- b. No transfer—the instruction it receives is not transferred to the next C_{i+1} .

Connecting elements ASE_1, \dots, ASE_F , are connected into a shifting sequence, and transfer several addresses. Each time a pipeline stage C_i receives an operand and executes the operation assigned to it by the program instruction i , receives a word shifted from ASE_{i-1} . It then follows that ASE_i is synchronized by state C_i and stores a word received from ASE_{i-1} during the time C_i executes operation. If the result of this operation should be written to some general register set m_j , the ASE_i sends the address for that set to state C_i . Thus the number of MSE and ASE connecting elements is the same and matches F , the number of pipeline stages. Both an MSE and ASE may be implemented on a universal LSI module equipped with the modular control organization.^{42,58}

Each instruction fetched from M_0 to P_0 includes two portions—the pipeline portion, PI , and the address portion, AI . By passing through the bus made of MSE connecting elements, the pipeline portion, PI , propagates through consecutive pipeline stages with a delay of one interval, causing execution of an operation assigned to each stage. Concurrently, the address portion, AI , of the instruction propagates through the bus made of ASE connecting elements and specifies a pipeline stage, C_i , which should output the result, and a general register set m_j , which should receive this result.

We introduce now a format for the portions PI and AI of the instruction.

PI instruction

In order to adapt to the operation, it is necessary that each PI store its own op-code D . However, when the same D propagates through the pipeline stages, it will activate the same operation in each stage. In order that each stage, C_i , execute an individual operation, C_i should store a position code, i , which shows its position within the pipeline. These two codes, D and i , achieve the selective activation of an operation assigned to stage C_i by instruction PI . The number of bits in code D is $\log_2 \#(IS)$, where $\#(IS)$ is the size of an instruction set.

Adaptation to the length of the pipeline is performed with another code, w , which shows the number of consecutive pipeline stages which execute instruction PI . Clearly w matches the number of consecutive operations which are

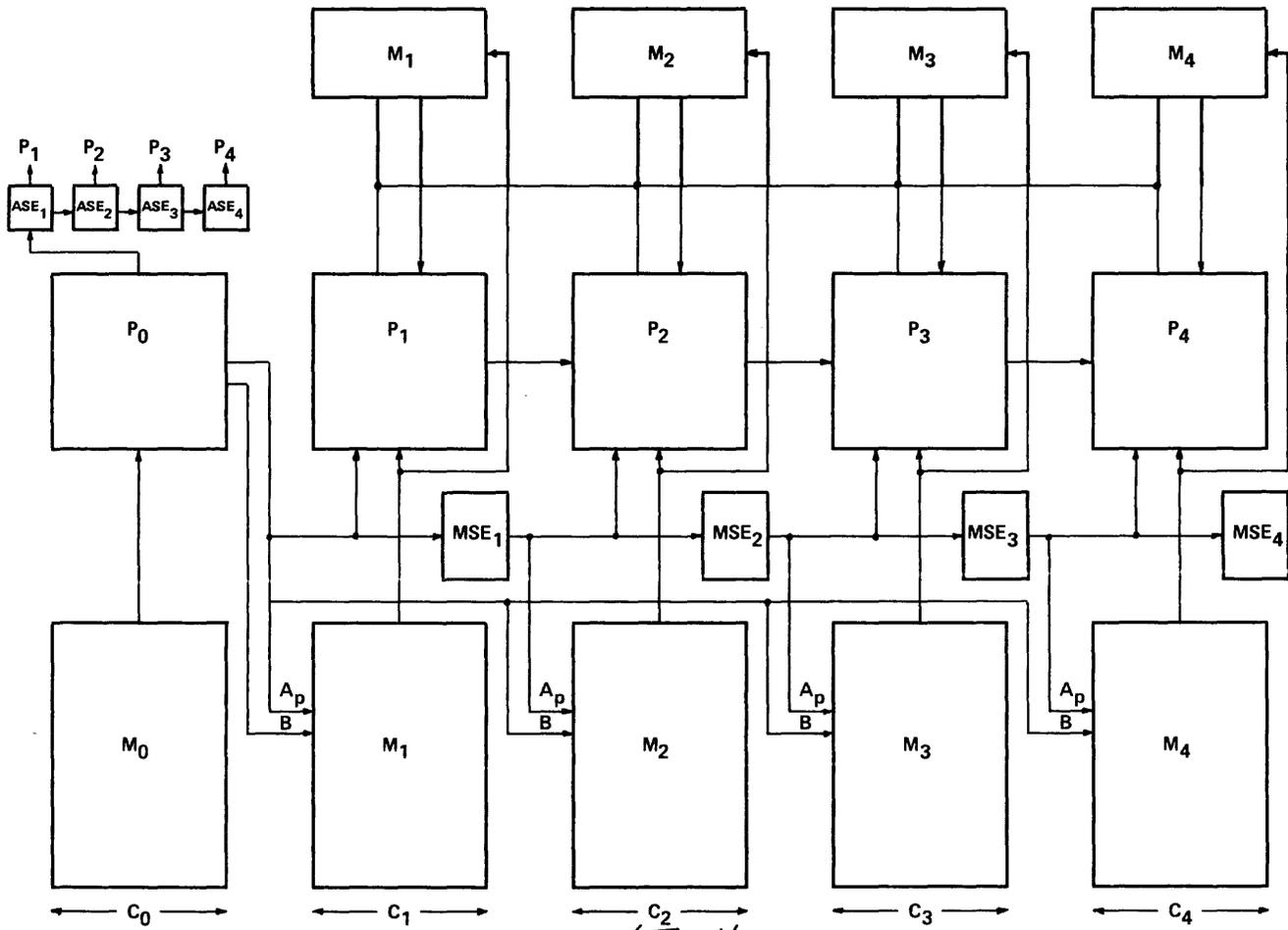


Figure 4—Dynamic pipeline architecture.

realized in PI . By propagating through each connecting element, MSE_i , containing position code i , w is compared with i . If $i < w$, MSE_i propagates this PI instruction to the next stage with delay of one time interval. If $i \geq w$, MSE_i blocks further instruction propagation, ending the execution of instruction PI . Therefore, by writing a code w into the instruction field, one may select the number of pipeline stages required by a single instruction.

This technique allows one to perform a simple adaptation of the number of pipeline stages that requires no bypassing of unneeded stages and no conflict resolution associated with such bypassing. The entire problem is solved by the code w with size $w = \log F$ bits, where F is the maximal length of the pipeline, which can be formed by Dynamic Pipeline Architecture.

Adaptation of each stage to operation time is performed with another code, p , stored in the PI instruction. Since each LSI module of processor P_i is equipped with the same modular control organization,^{42,58} this allows one to organize variable time intervals for any processor dependent operation. To this end a special sequencer, $CAD-M$, is activated. It passes through a loop containing p states where each state lasts one clock period, t_0 . For $p=2$, $CAD-M$

passes through a two-state loop, giving $T_{op} = 2t_0$; for $p=3$, $t_{op} = 3t_0$, etc.

For pipelines however, the modular control organization is slightly modified as compared to conventional DC groups. Indeed, in a DC-group a new meaning of code p is written to each LSI module during each architectural transition. Thus all the LSI modules contained in a computer store the same p during the time this computer exists. For pipelines, the time of operation required by stage C_i depends on the PI instruction. A new meaning of code p must therefore be brought to C_i with each PI instruction. The size of p is $\log_2 n$ where n is the maximal computer size which may be formed by DPA.

Therefore three codes, D , w , and p , effect a pipeline's adaptation to the operation, to the length of the pipeline, and to the time of operation executed by each pipeline stage.

In addition to adaptation codes, each PI instruction stores the relative address, A_p , of M_i memories where each memory M_i stores a word required by the C_i stage. The cell accessed by address A_p may store either an operand used for execution in the stage C_i , or an r_p -address of the general register set m_i . Using this address, a second operand is fetched to C_i from m_i . A special tag bit, e , in the cell

accessed by A_p address recognizes whether the second operand is stored in M_i or m_i .

In order not to have any limitations on the height of memory M_i and not to increase the bit size of PI , it is assumed that A_p is a relative eight-bit address. The effective address E (24 bits) is formed from a concatenation of the base address B (16 bits) and address A_p (eight bits). Base address B is fed continuously to all M_i memories during the execution of one task. It is changed when data words have to be fetched from a new page. Such organization provides that each M_i memory may contain 2^{24} words. The overall size of instruction PI is:

$$PI = \log_2 \#(IS) + \log_2 F + \log_2 n + 8$$

For a DPA with $\#(IS)=255$, $F=16$, $n=8$, $h=8$

$$PI = 8 + 4 + 3 + 8 = 23 \text{ bits.}$$

(This means that this DPA has 255 instructions, it forms pipelines not exceeding 16 stages, each stage has no more than 64 bits.)

Therefore, in spite of its insignificant bit size, instruction PI not only brings to each pipeline stage all the necessary information about operands and operations, but it also performs effective on-line adaptation of the pipeline in order to match the sequence of operations activated by this PI . In addition, the relatively small size of PI allows one to implement each MSE connecting unit using a single universal LSI module with 64 pins.

AI instruction

This instruction stores the following codes: position code j of the C_j stage which should fan-out its result to a general register set m_b , position code b of the register set m_b which should receive this result, and the address r_p of the destination register in m_b where the result should be written into. Thus the AI bit size is:

$$AI = 2 \log_2 F + \log_2 K;$$

where F is the maximal length of a pipeline and K is the number of registers in each m_j . Therefore, for $F=16$, $K=256$, $AI=2 \cdot 4 + 8 = 16$ bits.

Thus the AI instruction allows each pipeline stage (not just the last one) to send its computational result and permits each general register set to receive this result. Such an on-line organization of providing the stages with temporary results they might need for future computations eliminates the alternative waiting time associated with sorting and sending temporary results to stages which require them.

Variable time intervals

Each pipeline computer C_i is equipped with the control organization required by a dynamic instruction set. However, in order to organize a pipelined mode of operation this control organization has to be modified.

In computers with a dynamic instruction set, the DIS unit of one computer contains two sequencers, $CAD-I$ and $CAD-M$. The $CAD-I$ sequencer activates a sequence of operations which corresponds to one instruction, whereas the $CAD-M$ sequencer specifies the time of each operation. For instance, if an instruction activates the sequence $((A+B)-C) > K$, then the first state of $CAD-I$ fetches the instruction, the second state executes $A+B$, the third state performs $(A+B)-C$, and the fourth state executes the comparison $(A+B-C) > K$.

In a pipeline, however, all these operations are distributed among separate computers, so that computer C_0 fetches the instruction, computer C_1 executes $A+B$, computer C_2 executes $(A+B)-C$, and computer C_3 executes $((A+B)-C) > K$. Therefore, each pipeline computer executes only one operation out of the whole sequence. Furthermore, two consecutive pipeline computers, C_i and C_{i+1} , may simultaneously execute operations activated by two instructions, PI_j and PI_{j+1} . Therefore, for each pipeline computer C_i , its $CAD-I$ sequencer has to establish its state only for the time interval it keeps instruction PI ; at the next time interval, when the new instruction PI_{j+1} is written, $CAD-I$ has to establish another state which corresponds to that instruction.

It then follows that for non-iterative operations (addition, subtraction, Boolean) the $CAD-I$ functions as a decoder, and for iterative operations (multiplication, division, etc.) it works as a sequencer. The $CAD-I$ functioning is controlled with the following codes: the op-code D it receives with the PI instruction, position signal i produced locally by the position code i , and code γ_j which distinguishes instruction set IS_j .

Consider now how one may organize a variable time of operation, T , executed in stage C_i of the pipeline: T is variable, i.e., $T = t_0 \cdot b$, where t_0 is the time of h -bit addition in one LSI module, and b depends on codes p and D stored in instruction PI . For a processor dependent operation (addition, subtraction), that has to last $T = p \cdot t_0$, the output of decoder $CAD-I$ initiates the $CAD-M$ sequencer which executes a loop having p states. During this time $CAD-I$ maintains output continuously, and activates operation in the processor. When $CAD-M$ completes its loop, this terminates the $CAD-I$ output. If the operation is independent of the processor size (Boolean, shift, etc.), then it is activated by the $CAD-I$ decoder only. Namely, $CAD-M$ is not initiated and the operation takes t_0 , the time of one minor clock-period. If stage C_i executes an iterative operation, $CAD-I$ executes a sequence containing several states. If a state in this sequence has to last time $p \cdot t_0$, $CAD-I$ initiates $CAD-M$ and performs a transition to the next state only after the completion signal issued by $CAD-M$.

Routing algorithm for PI and AI instructions

Consider movement of instructions PI and AI within the pipeline. Let the computer supervisor, C_0 , fetch instruction I^* from M_0 to P_0 . At the first interval, its PI^* portion is written to stage C_1 and connecting element MSE_1 . Since

PI^* stores relative address A_p and the base address B is continuously fed by P_0 to all M_i memories, memory M_1 , receiving effective address $E=B+A_p$, retrieves either an operand for stage C_1 (bit $e=0$) or address r_p of the register set m_1 ($e=1$). This address connects the respective register contained in the m_1 register set to the adder input in processor P_1 . At the same time processor P_0 sends instruction AI^* to ASE_1 and fetches the next instruction, which follows I^* , from the memory M_0 .

For the next interval the following actions are executed in the pipeline in parallel: (Assume that for one I^* instruction, its PI^* portion is stored in P_i and MSE_i , and its AI^* portion is stored in ASE_i .)

- a. The p code stored in instruction PI^* activates a new processor size in P_i and new durations of processor-dependent operations.
- b. The P_i processor ($i=1, \dots, w$) which received PI^* during previous interval executes the operation provided by the PI^* instruction for the i th stage, and receives a new PI which immediately succeeds PI^* .
- c. The MSE_i connecting element compares the code w stored in the PI^* instruction with its own position code i .
 - If $i < w$, it sends the PI^* instruction to the next stage C_{i+1} and connecting element MSE_{i+1} , respectively.
 - If $i \geq w$, then PI^* instruction ends its execution.
 - If instruction PI^* passes to the next stage C_{i+1} , a word is fetched from memory M_{i+1} . Its effective address is $E=B+A_p$, where A_p is stored in instruction PI^* .
- d. The connecting element ASE_i compares its own position code i with position code b stored in instruction AI^* which shows that pipeline stage C_b should send its result to a general register set.
 - If $i=b$, the AI^* is sent to processor P_i . At the next interval, P_i will send the result of the computation it executes during the present interval to a destination address stored in AI^* .
 - If $i \neq b$, then AI^* is sent to the next ASE_{i+1} .
- e. P_0 sends new instruction PI to stage C_1 and connecting element MSE_1 , and it sends new instruction AI to connecting element ASE_1 .
- f. The next instruction is fetched from M_0 to P_0 .

Consider now the organization of vector computation when the same instruction, I^* , handles data arrays each having z words. Then the PI^* and AI^* portions propagate through the pipeline z times, and C_0 stops fetching of other instructions from M_0 until the vector computation ends. After each interval, the P_0 processor increments the current address A_p stored in PI^* with some constant, a , stored in I^* , so that the new address equal to A_p+a , is written back to PI^* . Thus, during each interval, the C_i stage receives PI^* with a modified relative address. In addition, if the I^* instruction provides that address r_p , stored in instruction AI^* , also be changed, then during each time interval, P_0 computes a new address, r_p+b , in a destination register set and writes it back to AI^* so that connecting element ASE_1 also receives instruction AI^* with a modified address.

It then follows that for vector computations, in addition to the AI^* and PI^* portions, each I^* instruction should also store the constants a and b for modification of addresses A_p and r_p and a constant z showing the dimension of the data arrays in memories M_i .

Advantages of dynamic pipeline architectures

Dynamic pipeline architecture (DPA) eliminates most of those drawbacks associated with disparity between program and pipeline structures. Indeed:

- a. For existing pipelines, if a sequence of operations met in the program does not correspond to a sequence of operational units connected into the pipeline, the pipeline is switched into a new configuration in order to form a new sequence that matches the one encountered in the program. This introduces an additional delay associated with such a reconfiguration.

In the DPA no such delay is introduced because each pipeline stage is capable of executing any operation. Thus, a pipeline with F stages may execute any sequence of w operations if $w \leq F$.

- b. In conventional pipelines a disparity between the number of consecutive operations in the instruction and the number of pipeline stages in the pipeline which processes this instruction creates additional delays (dummy-time intervals) associated either with instruction propagation through unneeded stages or with conflict resolution when the instruction bypasses the unneeded stages and encounters operands prepared for some of its predecessors.

In the DPA no such delays occur because the number of stages the instruction propagates through is defined by the w code it stores. Consequently, on passing through w stages the instruction completes its execution.

- c. In existing pipelines the time for non-iterative processor dependent operations (addition, subtraction, $>$, etc.) is permanent and does not depend on operand sizes. However, selection of a permanent operation time in each stage requires that it be selected as the time of the longest operation (addition handling maximal word sizes). It then follows that all faster operations (processor dependent operations handling smaller word sizes or Boolean operations, shift operations, etc.) are slowed down because they are executed during the time of the longest processor dependent operation.

In DPA, however, no such slowdown occurs because each stage is provided with a variable time interval. It then follows that each stage is capable of generating a minimal operation time. Therefore, in DPA a pipeline is capable of working at a variable rate. If it is filled with short operations,

it fans out results much faster, at the maximal rate of the short operation.

- d. In existing pipelines, additional waiting time occurs when the computational result obtained for one instruction in one stage is required as an operand for another instruction in another stage. For instance, if the instruction executes $(A-B)^2 + D \cdot K$, the temporary result $(A-B)^2$ cannot be used on-line as an operand of some other pipeline stage.

For DPA, however, a proposed addressing procedure allows each stage C_j to receive on-line a temporary result produced by another stage C_i in its general register set m_j . Thus the waiting time associated with feeding temporary results to the pipeline stages that need them may be eliminated.

- e. All existing pipeline systems cannot partition the available resources into a variable number of parallel pipelines, each of which is provided with a changeable number of stages. As a result, the number of parallel program streams is permanent and cannot be changed. Likewise, selection of pipelines with a permanent number of stages delays execution of instructions requiring a smaller number of stages.

On the other hand, DPA may perform multiple on-line switching of the resource into different states. This maximizes the number of program streams computed by the same equipment.

- f. It has been noted previously that the ability of DPA to form different states is convenient for handling conditional branching, for then no time is lost due to selection of the program sequence which currently does not fill in the pipeline. Also, switching a DPA into states with multiple pipelines allows one to organize multipoint branching.

CONCLUSIONS

A modern dynamic architecture allows redistribution of the hardware resource forming a variable number of concurrently operating computers. This means that a parallel system may increase the number of parallel program streams executed on the same equipment by changing the number of computers it has. However, a dynamic architecture may provide the system with other powerful sources of throughput increase:

- a. It may form its resource into different types of architectures. This means that the system may turn from a multicomputer system to, say a pipeline or array system, and vice versa. Or the system may have all three types of architecture co-resident at the same time—namely, part of the resource functions as a multicomputer system. Another part behaves as a pipeline and/or array subsystems. This feature will allow to perform

multicomputer, multiprocessing, pipeline and array computations using the same hardware instead of implementation of separate dedicated subsystems.

- b. The system will be able to change via software an activated instruction set. This may be done either within a single program or on the level of several programs forming a queue.

Therefore, a merger of dynamic, reconfigurable and microprogrammable adaptations into a single adaptable architecture will allow one to create complex parallel systems with high throughput.

REFERENCES

1. Estrin, G., "Organization of Computer Systems: The Fixed Plus Variable Structure Computer," *Proc. Western Joint Computer Conf.*, pp. 33-40, 1960.
2. Estrin, G., "Parallel Processing in a Restructurable Computer System," *IEEE Transactions on Electronic Computers*, Vol. EC-12, 1963, pp. 747-755.
3. Slotnick, D., W. Borck and R. McReynolds, "The SOLOMON Computer," *AFIPS Proc. FJCC*, Vol. 22, 1962, pp. 97-107.
4. Gregory, J., and R. McReynolds, "The SOLOMON Computer," *IEEE Transactions Electronic Computers*, Vol. EC-12, No. 6, Dec. 1963, pp. 774-781.
5. Barnes, G., R. Brown, Maso Kato, D. Kuck, D. Slotnick and R. Stokes, "The Illiac IV Computer," *IEEE Transactions on Computers*, Vol. C-17, August, 1968, pp. 746-757.
6. Bouknight, W. J. et al., "The Illiac IV System," *Proc. IEEE*, Vol. 60, April 1972, pp. 369-388.
7. Batchner, K., "STARAN Parallel Processor System Hardware," *AFIPS Conf. Proc.*, Vol. 43, pp. 405-410.
8. Batchner, K., "The Multidimensional Access Memory in STARAN," *IEEE Trans. on Computers*, Vol. C-26, February 1977, pp. 174-177.
9. Reddaway, S., "DAP-A Distributed Array Processor," *Proc. of the 1st Symposium on Computer Architecture*, 1973, pp. 61-72.
10. Okaga, Y., H. Tajima and R. Mori, "A Novel Multiprocessor Array," *Proc. 2nd Euromicro Symposium on Microprocessing and Microprogramming*, Venice, 1976, pp. 83-90.
11. Yau, S., and H. Fung, "Associative Processor Architecture—A Survey," *ACM Computing Surveys*, Vol. 9, No. 1, March 1977, pp. 3-27.
12. Anderson, D. W., F. J. Sparacio and R. M. Tomasulo, "IBM System 360 Model 91, Machine Philosophy and Instruction Handling," *IBM Journal of Research and Development*, pp. 8-24, January 1967.
13. Anderson, S. F., J. G. Earle, R. E. Goldschmidt and D. M. Powers, "The IBM System 360, Model 91: Floating-point Execution Unit," *IBM Journal of Research and Development*, pp. 34-53, January 1967.
14. Hintz, R. G., and D. P. Tate, "Control Data STAR-100 Processor Design," *COMPCON 72. IEEE*, N.Y., 1972, pp. 1-4.
15. Control Data STAR-100 Computer System, *Hardware Reference Manual*, Publication 60256000, CDC, Arden Hills, MN, 1973.
16. Watson, W. J., "The TI ASC—A Highly Modular and Flexible Super Computer Architecture," in *AFIPS 1972 Fall Jt. Computer Conf.*, AFIPS Press, Montvale, N.J. 1972, pp. 221-228.
17. CRAY-1 Computer System, *Reference Manual*, Publication 2240004, Cray Research, Inc., Bloomington, MN, 1976.
18. Wulf, W. A., and Bell, C. G., "C.mmp—A Multi-Mini-Processor," *AFIPS Conference Proceeding*, Vol. 41, 1972, pp. 765-777.
19. Davis, R. G., and Zucker, S., "Structure of a Multiprocessor Using Microprogrammable Building Blocks," *Proc. of National Aerospace Electronics Conference*, Dayton, Ohio, IEEE Press, 1971, pp. 186-200.
20. Davis, R. L., S. Zucker and C. M. Campbell, "The Building Block Approach to Multiprocessing," in *AFIPS 1972 Spring Jt. Computer Conf.*, AFIPS Press, 1972, pp. 685-703.
21. Reigel, E. W., D. A. Fisher, and V. Faber, "The Interpreter—A Microprogrammable Processor," *AFIPS Conference Proceedings*, AFIPS Press, 1972, Vol. 40, pp. 705-723.

22. Enslow, P. H., Jr., "Multiprocessor, Organization—A Survey," *ACM Computing Surveys*, Vol. 9, No. 1, March 1977, pp. 103-129.
23. Enslow, P. H., Jr. (ed.), *Multiprocessors and Parallel Processing*. John Wiley and Sons, Inc., New York, 1974.
24. Swan, R. J., S. H. Fuller, and D. P. Siewiorek, "Cm*—A Modular, MultiMicroprocessor," *AFIPS Conference Proceedings*, AFIPS Press, Vol. 46, pp. 637-643.
25. Swan, R. J., A. Bechtolsheim, K. Lai and J. Ousterhout, "The Implementation of the CM* Multi-Microprocessor," *AFIPS Conference Proceedings*, AFIPS Press, Vol. 46, 1977, pp. 645-655.
26. Baer, J. L., "Multiprocessing Systems," *IEEE Transactions on Computers*, Vol. C-25, December, 1976, pp. 1271-1277.
27. Paxer, Y., and M. Bozyigit, "Variable Topology Multicomputer," *Proc. of 2nd Euromicro Symposium on Microprocessing and Microprogramming*, Venice, 1976, pp. 141-149.
28. Anderson, G. A., and E. D. Jensen, "Computer Interconnection Structures, Taxonomy, Characteristics, and Examples," *ACM Computing Surveys*, Vol. 7, No. 4, December 1975, pp. 197-213.
29. Siegel, H. J., R. J. McMillen and P. T. Mueller, Jr., "A Survey of Interconnection Methods for Reconfigurable Parallel Processing Systems," NCC '79, *AFIPS Conference Proceedings*, AFIPS Press, Vol. 48, 1979.
30. Friedman, A. D., and F. Saheban, "A Survey and Methodology of Reconfigurable Multi-Module Systems," *Proceedings Computer Software and Applications Conference (CompSoc)*, 1978.
31. Pariser, J. J., and H. E. Maurer, "Implementations of the NASA Modular Computer with LSI Functional Characters," *AFIPS Conference Proceedings*, Vol. 35, 1969, AFIPS Press, pp. 231-245.
32. Avizienis, A., G. C. Gilley, F. P. Mathur, D. A. Rennels, J. S. Rohr and D. K. Rubin, "The STAR (Self-Testing-And-Repairing) Computer: An Investigation of the Theory and Practice of Fault-tolerant Computer Design," *IEEE Trans. on Computers*, Vol. C-20, No. 11, November 1971, pp. 1312-1321.
33. Stiffler, J., "The SERF Fault-Tolerant Computer, Part I: Conceptual Design," *Digest of the 1973 International Symposium on Fault-Tolerant Computing*. Palo Alto, California, IEEE Computer Society, 1973, pp. 23-26.
34. Parke, J. V., N. G. and P. C. Barr, "The SERF Fault-Tolerant Computer, Part II: Implementation and Reliability Analysis," *Digest of the 1973 International Symposium on Fault-Tolerant Computing*. Palo Alto, California, IEEE Computer Society, 1973, pp. 27-31.
35. Conn, R. B., N. A. Alexandridis and A. Avizienis, "Design of Fault-Tolerant Modular Computer with Dynamic Redundancy," *AFIPS Conference Proceedings*, Vol. 41, Fall JCC 1972, pp. 1057-1067.
36. Negrini, R., and M. G. Sami, "The 'Assembly-line' Multimicroprocessor Structure," *Proc. 3rd Euromicro Symposium on Microprocessing and Microprogramming*. Amsterdam, 1977, pp. 266-275.
37. Borgerson, B. R., and R. F. Freitas, "A Reliability Model for Gracefully Degrading and Standby-sparing Systems," *IEEE Trans. on Computers*, Vol. C-24, pp. 517-525, May 1975.
38. Borgerson, B. R., and R. F. Freitas, "An Analysis of 'Price' Using a New Reliability Model," *Digest of the Fourth Annual International Symposium on Fault-Tolerant Computing*, Urbana, Illinois, IEEE Computer Society, June 1974, pp. 2-26 to 2-31.
39. Kartashev, S. I., and S. P. Kartashev, "Designing LSI Metacomputer System with Dynamic Architecture," DCA Association, Lincoln, Nebraska, 1974.
40. Kartashev, S. I., and S. P. Kartashev, "Designing of LSI Metacomputer System with Dynamic Architecture Made of Microcomputers," *Proc. 3rd Annual International Symposium on Mini- and Microcomputers and their Applications*. Zurich, 1977, pp. 88-93.
41. Kartashev, S. I., and S. P. Kartashev, "A Multicomputer System with Software Reconfiguration of the Architecture," *Proceeding of the Eighth International Conference on Computer Performance, SIGMETRICS*, Washington, D.C., 1977, pp. 271-286.
42. Kartashev, S. I., and S. P. Kartashev, "Dynamic Architectures: Problems and Solutions," *Computer*, Vol. 11, July 1978, pp. 26-40.
43. Kartashev, S. I., and S. P. Kartashev, "Selection of the Control Organization for a Multicomputer System with Dynamic Architecture," *Proc. 4th Euromicro Symposium on Microprocessing and Microprogramming*, Munich, 1978.
44. Lipovski, G. J., and A. Tripathi, "A Reconfigurable Varistructured Array Processor," *Proc. International Conference of Parallel Processing*, 1977, pp. 165-174.
45. Goke, L. R., "Connecting Networks for Partitioning Polymorphic Systems," Doctoral dissertation, Dept. of Electrical Engineering, University of Florida, 1976.
46. Miller, R. E., and J. Cocke, "Configurable Computers: A New Class of General Purpose Machines," in *Int. Symp. Theoretical Programming, Lecture Notes in Computer Science*, Vol. 5, Springer-Verlag, Berlin, Germany, 1974.
47. Reddi, S. S., and E. A. Feustel, "A Restructurable Computer System," *IEEE Transactions on Computers*. Vol. C-27, No. 1, January 1978, pp. 1-20.
48. Reddi, S. S., and Feustel, "An Approach to Restructurable Computer Systems," in *Parallel Processing (Lecture Notes in Computer Science)* Vol. 24, Berlin, Germany, Springer-Verlag, 1975, pp. 319-337.
49. Feuster, E. A., "On the Advantages of Tagged Architecture," *IEEE Transactions on Computers*, Vol. C-22, July 1973, pp. 644-656.
50. Reddi, S. S., "A Modular Computer with Petri Net Array Control," *Proc. ACM '78*, Washington, D.C., 1978.
51. Siegel, H. J., P. T. Mueller, Jr. and H. E. Smalley, Jr., "Control of a Partitionable Multimicroprocessor System," *Proc. 1978 International Conference on Parallel Processing*, pp. 9-17.
52. Siegel, H. J., and P. T. Mueller, Jr., "The Organization and Language Design of Microprocessors for an SIMD/MIMD System," *Second Rocky Mountain Symposium on Microcomputers*, 1978, pp. 311-340.
53. Kartashev, S. I., and S. P. Kartashev, "LSI Modular Computers, Systems and Networks," *Computer*, Vol. 11, July 1978, pp. 7-15.
54. Kartashev, S. I., and S. P. Kartashev, "Software Problems for Dynamic Architecture: Adaptive Assignment of Hardware Resources," *Proceedings Computer Software and Applications Conference (CompSoc)*, 1978, pp. 775-780.
55. Ramamoorthy, C. V., and H. F. Li, "Pipeline Architecture," *ACM Computing Surveys*, Vol. 9, No. 1, March 1977, pp. 61-102.
56. Davidson, E. S., "The Design and Control of Pipeline Function Generators," Report Stanford University, 1972.
57. Ibbett, R. N., and P. C. Capon, "The Development of the MU5 Computer System," *Communications of the ACM*, Vol. 21, No. 1, January 1978, pp. 13-24.
58. Kartashev, S. I., and S. P. Kartashev, "A Multiprocessor with Modular Control as a Universal Building Block for Complex Computers," *Proc. 3rd Euromicro Symposium on Microprocessing and Microprogramming*, Amsterdam, 1977, pp. 210-216.

Architectural considerations of the NEC mass data file subsystem

by AKIRA SEKINO and TAKUO KITAMURA

Nippon Electric Co., Ltd.
Tokyo, Japan

INTRODUCTION

During the past decade the use of computers in the on-line environment has steadily increased. In particular, there has been a drastic growth of on-line systems for business-oriented applications. We see a shift of interest in computers from their mere computational power to their data management capability, especially in these applications. User demand for the growth of on-line storage capacity in this environment is said to be much larger than the demand for the growth of computational power. We expect the business-oriented computer system to become an information utility in the future.

We have been generally using disks as our on-line storage, but the ever-growing demand for file storage capacity forced us to use more inexpensive magnetic tapes, though they are more difficult to handle in the on-line environment. Now it is quite common to see libraries of several thousand tape reels storing tens of billions of bytes of data at large computer installations. Some government data processing installations actually possess nearly 100,000 tape reels in their tape libraries. The handling of these tape reels require many operators to mount these reels manually on tape units as well as a large floor space for the tape units and the tape library. Furthermore, the traditional tape handling is by nature prone to various human errors.

Several mass storage systems (MSS) were developed to solve these problems of traditional magnetic tapes by automating the tape handling process.¹⁻⁴ These systems have not yet spread widely because of their high cost, but they seem to be steadily gaining popularity. We believe that these systems will gradually replace most traditional tape libraries and become essential to large information utilities in the years to come.

NEC has been engaged in the product planning and development of an MSS. We have studied the existing MSSs in light of the essential requirements of these systems and derived an MSS architecture which is believed to relax the limitations of the existing systems. As the name of mass storage systems has been historically applied to disks in NEC, we decided to call our newly-developed mass storage system the NEC mass data file subsystem or the MDF subsystem. The present paper describes the major architec-

tural considerations of the MDF subsystem in relation to the requirements of mass storage systems in general.

REQUIREMENTS OF MASS STORAGE SYSTEMS

Before describing the architectural considerations of the NEC MDF subsystem, we must explain the background of our architectural decisions, i.e., the judgment criteria with which we designed our MSS architecture. The following gives the MSS requirements, arranged in (more or less) descending order of importance.

Large Storage Capacity—An MSS must have a storage capacity of up to several hundred billion bytes, equivalent of more than one hundred thousand traditional half-inch tape reels.

Inexpensive Storage—The cost of an MSS must be comparable to that of a traditional tape library on a cost-per-unit-capacity basis.

Automatic Management of Storage Media—Access to storage media should not require human interventions such as tape selection and tape mounting by human operators.

Smaller Entry Cost—The increase of the total system cost for migrating to one of the entry MSS models should be as small as possible.

Flexibility of MSS Configuration—Various MSS configurations must be possible to meet the needs of many installations and must allow easy future expansion.

Continued System Operation Through Graceful Degradation—An MSS must be able to automatically reconfigure itself to allow continued system operation in the event of component failure, perhaps with degraded performance, and must allow concurrent repair of the failed component.

Smaller Floor Space—The floor space needed by an MSS must be considerably smaller than that of the conventional tape library.

Easy Conversion to an MSS—Changes required in JCL statements and application programs must be none or minimal. Conversion of existing files on magnetic tapes and disks must be as simple as possible.

Absorption of Tape-Oriented Applications—An MSS must be capable of accommodating the existing tape applications, and hopefully make new applications possible.⁵

Support of Various File Organizations—All important file organizations currently available on magnetic tapes and disks should be available for MSS files.

Multiple Access Modes—It is desirable that multiple access modes are available to application programs in accessing MSS files. These should include the staging mode and the direct access mode.

High Performance—The performance of an MSS must be sufficiently high so that it is usable as on-line storage.

Efficient Utilization of Storage Capacity—The effective capacity of the MSS storage media as well as that of disk storage used for staging must be efficiently usable.

Use of an MSS in the Multidimensional Processing Environment—An MSS must allow simultaneous use by coexisting local/remote batch processing jobs, time-sharing terminal-oriented jobs, and on-line database jobs.

Shared Use of an MSS by Multiple Computer Systems—An MSS must allow the shared use by a set of loosely-coupled multiple computers as well as by multiple computers connected via a communication network.

RASIS Considerations—An MSS must have a high degree of reliability, availability, serviceability, integrity, and security.

High Potential for the Future—An MSS must have a high potential for supporting the very large data utility of the future.

In general, it is not easy to satisfy all these requirements fully, but we feel that none of these requirements can be sacrificed in favor of the others.

MAJOR ARCHITECTURAL FEATURES

Several MSSs developed so far, notably the IBM 3850, the CDC 38500 and the Ampex Terabit Memory, represent serious attempts to satisfy the above requirements of a large file storage.¹⁻⁴ In the early stages of our MSS design, we examined the architectural features of these systems.⁵ We felt that each of these systems is satisfactory in several requirements but not in all. We spent a substantial amount of time in deciding the kind of architectural features which could meet the requirements under the constraints of our development resources.

At the end of the conceptual design stage, we came to a conclusion that our MSS should support the following three major architectural features on an IBM-3850-like storage device:²

1. Use of the virtual file concept.
2. Device independence for staging disks.
3. Support of the direct access mode as well as the staging mode.

The CDC 38500 has similar features,⁴ but a combination of the above features and our IBM-3850-like storage device would create a system with characteristics very different from either system. We decided to develop our storage device with specifications similar to those of IBM 3851 mass

storage facility, a set of new operating system components and service utility programs to support MSS functions, and a storage device controller capable of interfacing between these host software components and the storage device. We believe that this design would make up for certain limitations of the IBM 3850 architecture, as will be described later.

The virtual file concept allows users to access all the MSS files uniformly by the associated cataloged names in the staging access mode without having to know whether these files are still on the mass storage device or have been already staged to the staging disks. When an MSS file that is still on the mass storage device is accessed the entire file is staged to one of the staging disks. As a matter of fact, a user can access all MSS files as if they were ordinary disk files. This virtual file concept is our counterpart of the IBM 3850's virtual disk concept in implementing virtual file storage. The virtual file and virtual disk concepts correspond respectively to the segmentation and paging concepts of virtual memory implementation for executable programs. It follows that many comments on segmentation and paging in virtual memory apply equally to the virtual file and virtual disk concepts of virtual file storage.⁶ However, we have observed here that an implementation of virtual file storage by the virtual file approach tends to provide more efficient operation than that by the virtual disk approach because of the relatively slow operating speed of MSSs. This point will be discussed in detail later.

Now let us describe how the NEC MDF subsystem generally works using Figure 1 to explain the architecture of our system. As this figure shows, the MDF subsystem with the mass storage device and controllers is configured to be independent of the disk subsystem which is used for storing

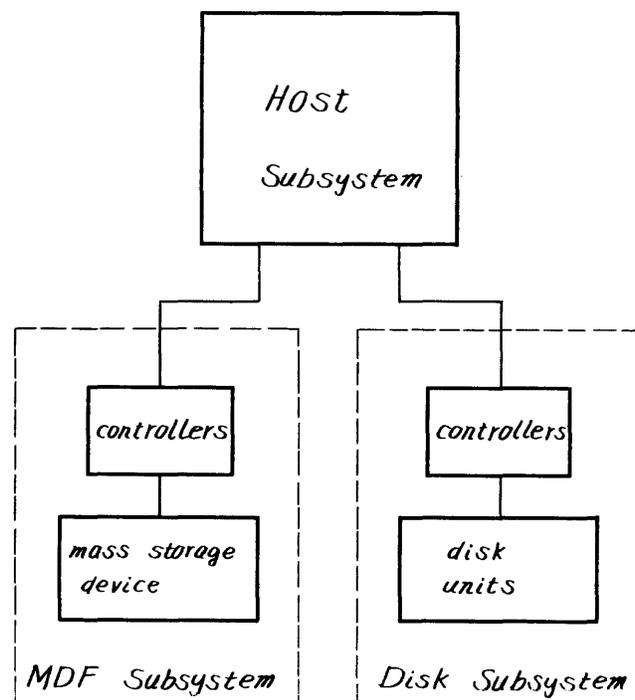


Figure 1—Structure of the NEC mass data file system.

staged MSS files as well as ordinary disk files. When an MSS file is accessed by a user job in the staging mode of operation, the operating system components residing on the host subsystem (CPU, I/O processor, memory, etc.) reserve an appropriate amount of space for the file in the disk subsystem and requests the file from the MDF subsystem. As the file is received part-by-part in the host buffer, it is piecemeal staged into the reserved space of the disk subsystem. Thus, these two subsystems do not have any direct data transfer path like the one seen in the IBM 3850 architecture.^{2,3} Tight coupling of a mass storage device and a disk controller by a direct path in this architecture requires matching of the mass storage device and particular disks controlled by the disk controller at a complicated physical level. This would not only make this disk subsystem significantly different from the ordinary disk subsystem, but also tends to limit the types of disk units usable for staging. Our independent subsystem architecture is, however, flexible enough to allow various types of disk units for staging, e.g., 100, 200, and 317 megabyte disk units and our future larger capacity units, in an arbitrary combination.

The third architectural feature of our system is support of two access modes: the staging mode and the direct access mode for reading and writing an MSS file. The staging mode can be widely used for accessing an MSS file with arbitrary file organization, but this mode of operation becomes inefficient if the file to be staged is very large like some sequential files. Then, it is recommended to use the direct-access mode which allows direct access to sequential files without staging like a traditional magnetic tape unit. The important difference, however, is the fact that a data cartridge used as the mass storage media allows a partial data modification to any of many MSS files stored on its cartridge tape, but the traditional half-inch tape does not. Our system is designed to support the compatibility of the above two access modes for sequential files; a user may dynamically specify his choice of access mode in reading or writing a sequential file by a JCL parameter. Support of these complementary access modes is expected to provide wider usage of MSS in actual computer applications.

Finally, some comments are in order about the influence of these architectural features upon satisfaction of the above MSS requirements. First, the choice of an IBM-3850-like device with its advanced recording technology and 50 megabyte data cartridges was made especially to satisfy the first, second, and seventh requirements. It is generally advantageous to use larger capacity cartridges in satisfying these requirements. The adoption of the virtual file concept is mainly based on the eighth and twelfth requirements and other considerations such as security, recovery, and accounting. Our decision to support the device independence for staging disks using ordinary disk units aims to satisfy the fourth, fifth, seventh, and thirteenth requirements. Besides, this decision allowed us to avoid the development of a disk subsystem dedicated to MSS. Lastly, support of two complementary access modes is intended for the ninth, eleventh and twelfth requirements. We will discuss this aspect of the MDF subsystem further from the viewpoint of system performance.

A PERFORMANCE COMPARISON OF THE VIRTUAL DISK AND VIRTUAL FILE APPROACHES

It was stated earlier in this paper that the virtual file approach tends to be more efficient than the virtual disk approach. This section proceeds to evaluate the performance characteristics of these two approaches. We begin by noting that decisions on file allocation to various types of file storage devices are usually based on the following file attributes:

- File size
- Access frequency
- File organization

The average size of files allocated to each type of file storage devices after the migration of selected tape and disk files to an MSS is typically found to be in the following range:

- Disk files 0.2-1 MB
- MSS files 1 -3 MB
- Tape files 4 -6 MB

We will mainly evaluate the performance of the staging mode operation of these two approaches where MSS files are staged in their entirety. It will be shown that the performance of the virtual file approach is relatively high in the file size range of 0-3 megabytes. We will also briefly comment on the performance of the staging mode operation of the virtual disk approach allowing cylinder faults, and the performance of the direct access mode operation of the virtual file approach. The analysis given below aims to clarify the performance characteristics inherent in the above two approaches, and is not intended to describe the actual performance of a product such as the IBM 3850 MSS or the NEC MDF subsystem. However, the analysis will help the reader gain a good perspective of the performance of MSS in general.

System operation

The virtual disk approach allows a job step to access an MSS file in the following manner.² First, a data cartridge is selected from the cell-structured cartridge library by an accessor and physically moved to an appropriate data recording device (DRD) at the beginning of the job step, for the purpose of mounting the MSS volume containing the MSS file on a virtual disk unit. The cartridge is loaded into the DRD and the VTOC (volume table of contents) information recorded at the beginning of the cartridge tape is staged to a staging disk. This staging operation completes mounting of the MSS volume; the operating system can now locate the MSS file within the cartridge by reading the staged VTOC information from the disk just like locating an ordinary disk file. The cartridge is then unloaded from the DRD and returned to the cartridge library by an accessor. The duration of time starting from the accessor movement to select a cartridge and ending with the accessor movement

to return the cartridge roughly corresponds to the period of time during which a particular DRD is exclusively allocated to this cartridge. For this reason, we call this duration of time a DRD cycle in this paper. When the MSS file is OPENed during the execution of the job step, the second DRD cycle is required. The cartridge containing the MSS file is selected by an accessor to load it into an appropriate DRD. After the cartridge loading is complete, the DRD locates the MSS file and stages the entire file into an appropriate space of a staging disk. The cartridge is then unloaded and returned to the library by an accessor again. When the job step terminates eventually, the third DRD cycle is required to destage the MSS file to the original data cartridge in demounting the MSS volume. Only the modified cylinders of the MSS file are actually destaged this time.

On the other hand, the virtual file approach requires a total of only two DRD cycles. The first DRD cycle is required at the beginning of a job step. The data cartridge containing a desired MSS file is selected and loaded into an appropriate DRD. The DRD then reads the file label information, the counterpart of the VTOC information, in the direct access mode, in order to find the location of the MSS file. The located file is staged to a staging disk and the cartridge is unloaded and returned to the library. Thus, it is clear that the first DRD cycle of this approach actually corresponds to the first and second DRD cycles of the virtual disk approach. Finally, the second DRD cycle is required at the end of the job step to destage the entire MSS file.

Response time characteristics

Some more details of a DRD cycle must be explained in order to discuss the response time characteristics of the previous two approaches. By definition, a DRD cycle is a sequence of the following events.

- Accessor movement (library→DRD) t_a seconds
- Cartridge loading t_1 seconds
- Winding for data t_w seconds
- Data transfer (staging/destaging) nt_s seconds
- Rewinding for tape head t_r seconds
- Cartridge unloading t_u seconds
- Accessor movement (DRD→library) t_a seconds

The length of the DRD cycle is called a DRD cycle time in this paper. Some components of the DRD cycle time just shown probably need explanations. The length of time required by "winding for data" is the one needed by an DRD to wind the tape from the tape head position to the data position where recording of particular data starts. Similarly, the length of time required by "rewinding for tape head" is the one needed by the DRD to rewind the tape from the data position to the tape head position. The length of data transfer time needed by staging or destaging is proportional to the amount of data. Letting t_s be the length of time required to transfer a 250 kilobyte cylinder of data, the "data transfer"

time of n cylinder data is nt_s seconds. A typical numerical example is shown below.

$$t_a=4, \quad t_1=t_u=5, \quad t_s=1$$

$$t_w=t_r = \begin{cases} \sim 0 & (VTOC) \\ 4 & (MSS \text{ file}) \end{cases} \quad (\text{seconds})$$

The choice of different values of t_w or t_r for the VTOC information and an MSS file is based on the fact that the VTOC information is recorded at the beginning of the tape while an MSS file may be recorded anywhere on the tape. It should be noted that a DRD cycle time thus defined is the ideal one; an actual DRD cycle time measured in a real situation is prolonged by randomly inserted queueing delays.

Now we proceed to evaluate the task delay time incurred in making an MSS file ready for processing on disk in the virtual disk and virtual file approaches.³ This delay time is the MSS's counterpart of a manual mount time of the traditional half-inch tapes. The task delay time (T_{rd}) of the virtual disk approach is obtained as follows:

$$T_{rd} = (\text{the DRD cycle time for mounting a virtual disk})$$

$$+ (\text{the first half of the DRD cycle time for staging})$$

$$= (2t_a + 2t_1 + t_s) + (t_a + t_1 + t_w + nt_s)$$

$$= 3t_a + 3t_1 + t_w + (n+1)t_s$$

Similarly, the task delay time (T_{rf}) of the virtual file approach is

$$T_{rf} = (\text{the first half of the DRD cycle time for staging})$$

$$= t_a + t_1 + t_w + nt_s$$

Figure 2 shows how T_{rd} and T_{rf} compare as a function of the file size. Considering that the value of T_{rd} or T_{rf} can easily increase by a factor of two because of various internal queueing delays associated with allocations of an accessor, a DRD, and a data transfer path in a real situation, an actual task delay time of the virtual disk approach would be roughly comparable to a human tape mount time of traditional half-inch tapes while that of the virtual file approach would be about half of the human tape mount time.

We had an assumption until now that an MSS file was staged in a burst in its entirety. Next, let us briefly consider what happens if an MSS file is staged on demand cylinder by cylinder. The length of time required to process a cylinder fault (T_c) is calculated as

$$T_c = (\text{the first half of the DRD cycle time for cylinder staging})$$

$$= t_a + t_1 + t_w + t_s$$

Noting again that the value of T_c can easily increase by a factor of two because of internal queueing delays on an actual system, we observe from this result that a single cylinder fault would typically cost a task as much as 30 seconds of lost real time. Thus, if a job encounters many cylinder faults, the turnaround time of this job would be intolerably prolonged.

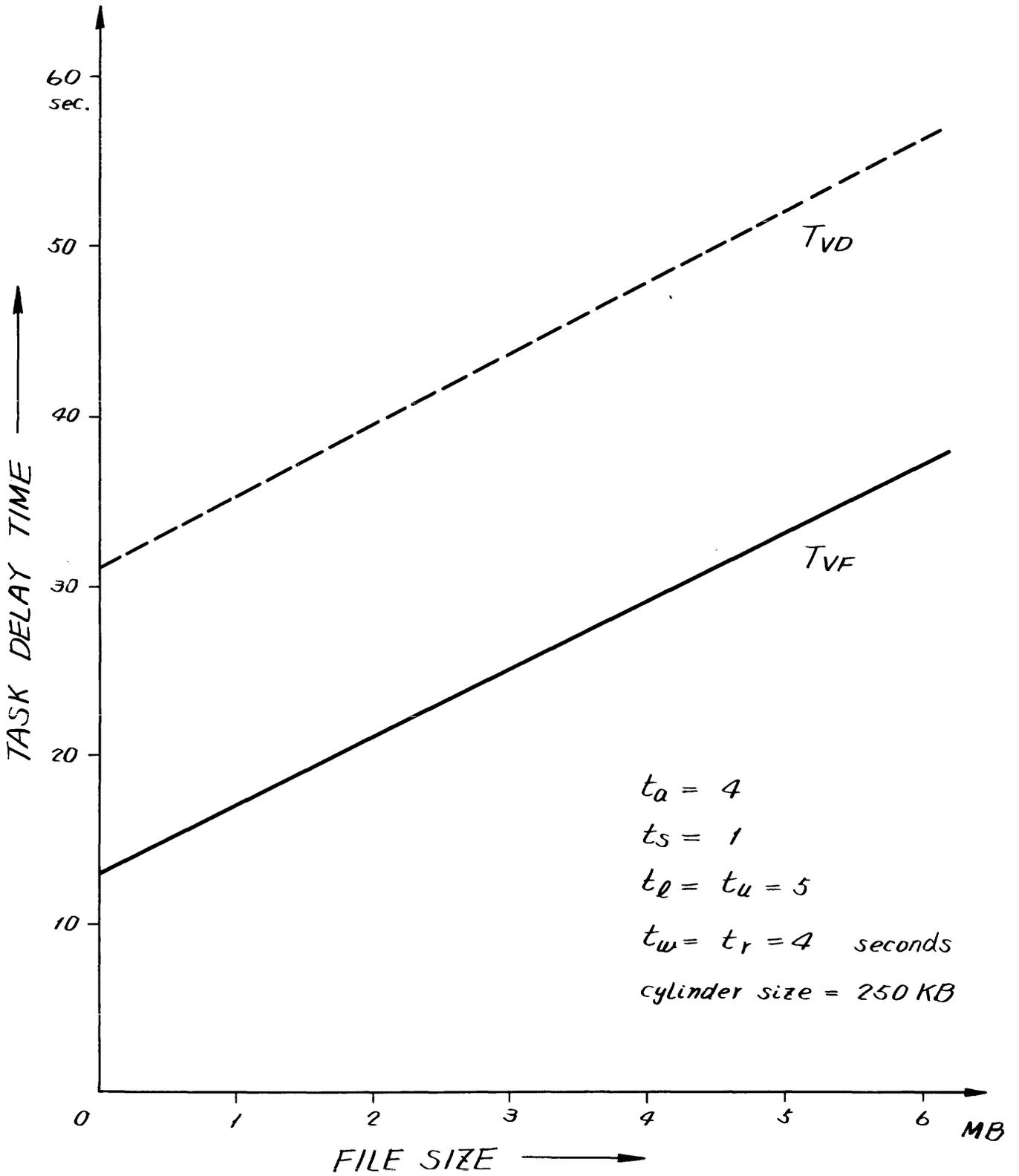


Figure 2—Comparison of task delay times of two approaches.

Throughput characteristics

We proceed to evaluate the throughput characteristics of the virtual disk and virtual file approaches. The following assumptions are made about user jobs to simplify the analysis.

1. Each user job requiring access to an MSS accesses only one MSS file in a job step. The type of processing for this file includes modifications to this file.
2. Virtual Disk Approach—The MSS volume is mounted on a virtual disk unit at the initiation time of a job step. The MSS file is staged in its entirety at the file OPEN time and only the modified cylinders of this MSS file are destaged at the termination time of the job step.
3. Virtual File Approach—The access mode to be used is the staging mode (not the direct access mode) where the entire file is staged at the job step initiation time and is destaged at the job step termination time.

An actual job environment is much more complicated, but the result of the following analysis can be modified to reflect such complications.

We now derive the total amount ($T_{i,vd}$) of MSS services required during a job step execution period of the virtual disk approach. Noting that a total of three DRD cycles are required by a job step, we have

$$\begin{aligned}
 T_{i,vd} &= (\text{the DRD cycle time for mounting a virtual disk}) \\
 &+ (\text{the DRD cycle time for staging an MSS file}) \\
 &+ (\text{the DRD cycle time for destaging the MSS file}) \\
 &= (2t_a + 2t_1 + t_s) + (2t_a + 2t_1 + 2t_w + nt_s) \\
 &+ (2t_a + 2t_1 + 2t_w + npt_s) \\
 &= 6t_a + 6t_1 + 4t_w + (n + np + 1)t_s
 \end{aligned}$$

where p is the probability that some portion of file data stored in a disk cylinder is modified on a staging disk. Similarly, the total amount ($T_{i,vf}$) of MSS services required by a job step of the virtual file approach is

$$\begin{aligned}
 T_{i,vf} &= (\text{the DRD cycle time for staging an MSS file}) \\
 &+ (\text{the DRD cycle time for destaging the MSS file}) \\
 &= (2t_a + 2t_1 + 2t_w + nt_s) + (2t_a + 2t_1 + 2t_w + nt_s) \\
 &= 4t_a + 4t_1 + 4t_w + 2nt_s
 \end{aligned}$$

Table I shows how the above total amount of MSS service requirement by a job step breaks down into the amount of particular services by accessors, DRDs and data transfer paths. Each of these values actually represents the total amount of particular service requirement by a single MSS file.

The throughput of an MSS can be measured by the number of MSS files staged per hour. We assume that we have four MSS models which differ in storage capacity and in the number of major MSS components, as shown in Table II. Then, the throughput of accessors, DRDs, or data transfer

TABLE I.—Service Requirement for Major MSS Components in the Total MSS Service Requirement by a Job Step

	Virtual Disk Approach	Virtual File Approach
(1) Accessors	$6t_a$	$4t_a$
(2) DRDs	$3t_a + 6t_1 + 4t_w + (n + np + 1)t_s$	$2t_a + 4t_1 + 4t_w + 2nt_s$
(3) Data transfer paths	$4t_w + (n + np + 1)t_s$	$4t_w + 2nt_s$
Total MSS service requirement	$6t_a + 6t_1 + 4t_w + (n + np + 1)t_s$	$4t_a + 4t_1 + 4t_w + 2nt_s$

paths is obtained by dividing the total available time by each particular service requirement of an MSS file shown in Table I. For example, the throughput of Model 1 DRDs of the virtual disk approach is

$$2 \times 3600 / [3t_a + 6t_1 + 4t_w + (n + np + 1)t_s] \text{ files/hour}$$

A tight upper bound for the throughput of each MSS model can then be given as the minimum of the throughput values calculated for accessors, DRDs, and data transfer paths. The final result thus obtained is depicted in Figure 3.

We have the following observations from this figure.

1. The virtual file approach attains considerably higher throughput in the important file size range of 0-3 megabytes.
2. The virtual disk approach attains somewhat higher throughput for the file size greater than 3 megabytes.

The first point is due to the throughput limitation of the virtual disk approach caused by the additional uses of accessors and DRDs in the DRD cycles for mounting virtual disks. This limitation of the virtual disk approach becomes serious if many small files must be staged on a larger MSS such as Model 3 or 4. On the other hand, the second point is based on the fact that unmodified cylinders of data need not be destaged in the virtual disk approach.

Comments on direct access to MSS

Finally, this section describes the performance characteristics of the direct-access mode which is supposed to be useful in processing large sequential files. Because of the limitation of space in this paper, we will only briefly touch upon the point. The idea of including the direct-access mode in our MSS architecture originated from the well known fact that traditional tape units are much more efficient in processing sequential files than disk units. In particular, the direct-access mode of the NEC MDF subsystem has the following performance advantages:

TABLE II.—Description of Four MSS Models

	Model 1	Model 2	Model 3	Model 4
No. of active accessors	1	2	2	2
No. of DRDs	2	4	6	8
No. of data transfer paths	1	2	3	4
Storage capacity (Gigabytes)	35	102	169	236

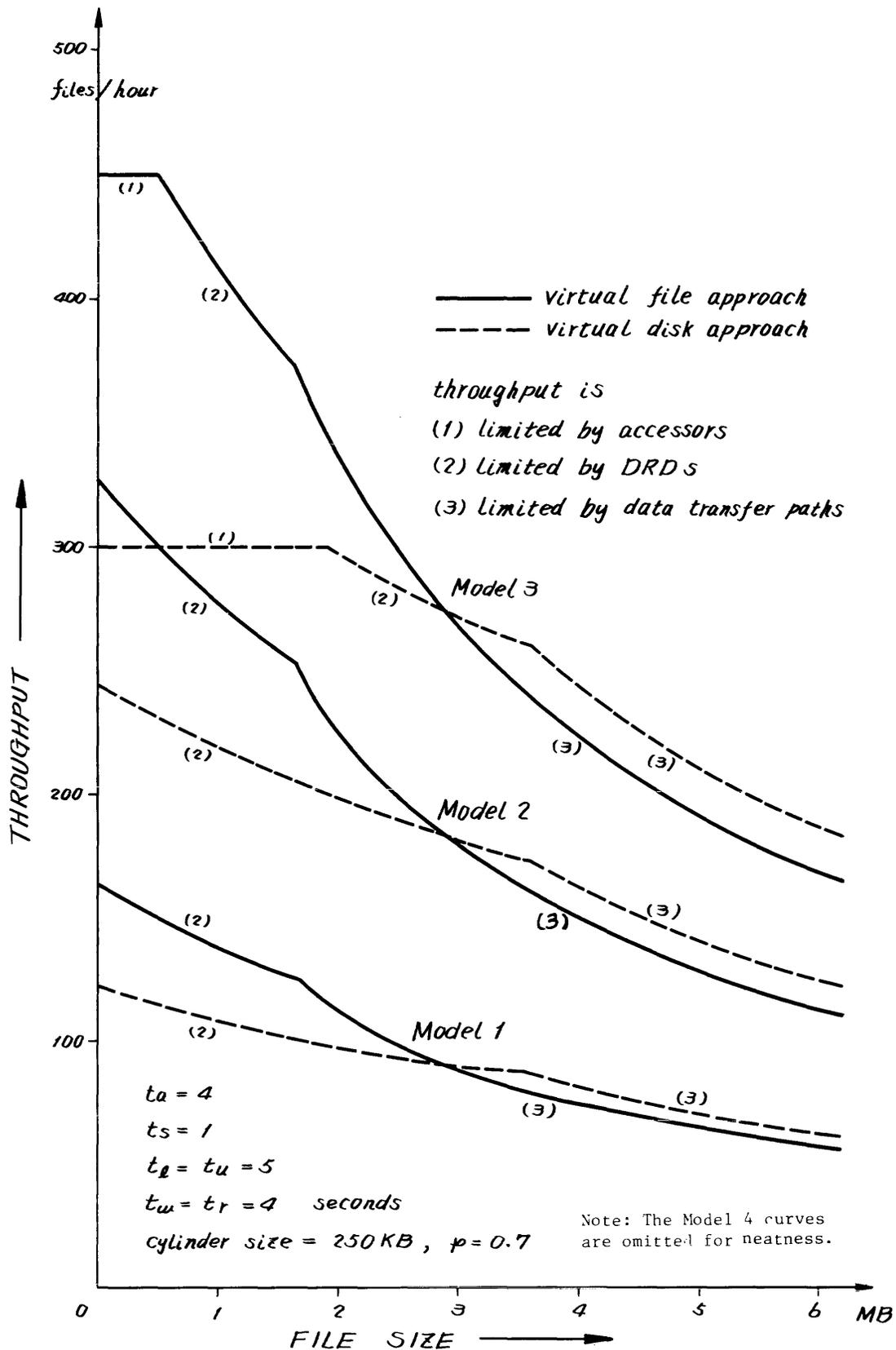


Figure 3—Throughput comparison of two approaches.

1. All the overhead time associated with file staging is avoided.
2. The access time of a data cartridge is somewhat shorter than that of a disk storage in sequential file processing.

An examination of file organization usage in actual user environments reveals that the overwhelming majority (e.g., more than 80-90 percent) of MSS files are sequentially organized. For this reason alone, sequential files might deserve a special attention by the direct access mode with the above performance advantages. It should, however, be noted that one or more DRDs is exclusively allocated to a job for the entire duration of its execution in this mode. The operating system must carefully allow jobs to use this access mode so as to avoid DRD request conflicts among jobs.

CONCLUSION

This paper has described the implications of major architectural features of the NEC mass data file subsystem. These architectural features include the use of the virtual file concept, device independence for staging disks, and support of two complementary access modes for MSS files, as well as the adoption of an IBM-3850-like mass storage device. This MSS architecture attempts to satisfy the MSS requirements stated in this paper to the utmost extent feasible in the current state of MSS technology. It was specifically found to relax the limitations of the existing MSSs on storage capacity, system cost as well as MSS entry cost, floor space, performance, and applications among other things. This paper examined the performance aspect of our MSS architecture particularly in detail. The analysis on response time and throughput characteristics of the staging mode operation of MSS shows that the virtual file approach of the NEC MDF subsystem has significant performance advantages over the virtual disk approach taken by the IBM 3850 MSS. Our observation is that the virtual file approach attains considerably higher throughput in staging small to medium MSS files than the virtual disk approach. On the other hand, the direct access mode of the MDF subsystem gives high performance in processing large sequential files. On-demand staging of the virtual disk approach was found to be inefficient in that a job's turnaround time is likely to be intolerably prolonged by a series of slow cylinder fault handlings.

As stated earlier, the MDF subsystem is independent of the disk subsystem. This implies that the data transfer operations for staging and destaging between these two sub-

systems require services by the host subsystem. The current implementation of MSS functions on the host subsystem ordinarily requires 0-5 percent of CPU time for the data transfer services. We plan to offload this service from CPUs to input/output processors in our distributed-function architecture of NEC ACOS 800 and 900 computers. In analyzing the performance characteristics of the virtual disk and virtual file approaches in this paper, we chose to use a set of parameter values common to these two approaches because of our interest in the impact of architectural differences upon the performance. Our recent experiences, however, indicate that the independent subsystem architecture actually incurs smaller amount of contentions in staging and destaging files than the tightly-coupled subsystem architecture. Hence, the actual performance of the MDF subsystem is generally higher than suggested in this paper using a common set of parameter values.

The MSS device technology is still young and has ample room for future improvement. In particular, we may expect considerable improvement in data recording density and in data transfer rate. So long as mass storage systems satisfy the MSS requirements, they will evolve into much more easy-to-use large on-line storage as the MSS technology matures. MSS is just beginning to play an essential role in the data utility type computers.

ACKNOWLEDGMENT

The MDF project involved a number of people over the past several years and is indebted to each of them. In particular, the authors wish to thank Y. Inada, T. Koterazawa, A. Tashiro and H. Mizutani for numerous stimulating discussions on MSS architectures.

REFERENCES

1. Wildmann, M., "Terabit Memory Systems: A Design History," *Proc. of the IEEE*, Vol. 63, No. 8, August 1975, pp. 1160-1165.
2. Harris, J. P., R. S. Rohde and N. K. Arter, "The IBM 3850 Mass Storage System: Design Aspects," *ibid.*, pp. 1171-1176.
3. Johnson, C., "IBM 3850—Mass Storage System," *AFIPS Conference Proceedings*, Vol. 44, 1975, pp. 509-514.
4. "Introduction to Control Data Mass Storage System for System 370," Minneapolis, Minnesota.
5. Howie, H. R., Jr., "More Practical Applications of Trillion-Bit Mass Storage Systems," *CompCon Spring Proceedings*, Feb. 1976, pp. 53-56.
6. Boyd, D. L., "Implementing Mass Storage Facilities in Operating Systems," *Computer*, Vol. 11, No. 2, February 1978, pp. 40-45.

Error-oriented architecture testing*

by LARRY KWOK-WOON LAI

Carnegie-Mellon University
Pittsburgh, Pennsylvania

ARCHITECTURE VALIDATION

Motivation

Architecture validation is becoming more and more important as diverging cost/performance criteria and competition cause the number of models within a computer family to proliferate. Some popular architectures are now being manufactured by many different companies and the chances of a company inexperienced with the architecture making mistakes is very high. Not only will errors in an implementation cause software incompatibility, the costs of fixing them are usually prohibitively high once there are a large number of defective machines in the field. Excellent evidence demonstrating the inadequacies of present testing techniques is implementation errors discovered in the field for many major computer families. This study was initiated in the hope that an error-oriented approach to architecture testing may provide a better detection of implementation errors.

In this paper, the term *architecture* refers to the time-independent functional appearance of a computer system to its users. An *implementation* of an architecture is an ensemble of hardware/firmware/software that provides all the functions as defined in the architecture.

Architecture validation

Architecture validation is the process of validating that a given machine indeed implements a specified architecture. There are three basic approaches:

1. *Verification*—prove the correctness of the design of an implementation using formal mathematical techniques.
2. *Simulation*—based on models of the physical building blocks and a description of the design, simulate the implementation to see if it behaves as expected.
3. *Testing*—establish a certain level of confidence in an

* This research was sponsored by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 3597, and monitored by the Air Force Avionics Laboratory under Contract F33615-78-C-1151. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. government.

implementation by running test programs on a prototype machine.

The first two approaches are most useful when the implementation is still being designed or when architecture specifications are still being formulated. Once a machine is built, however, the only way to find out whether it actually works is to run programs on it—i.e. through testing.

Before one can set out to validate any implementation, one needs to have a specification of the target architecture. There are two basic ways to specify/describe an architecture: (i) using a formal language, e.g. ISPS,¹ VDL/APL;² and (ii) using a natural language, e.g., Principles of Operation,¹⁵ Processor handbook.⁵ The latter is often the most important because many architectures do not have a formal specification while a natural language description is almost always available, is more readable, and hence is read by users and implementors alike.** For verification and simulation, a complete and consistent formal description is needed. For testing, a natural language description is usually adequate for the test programmer, although any ambiguities in the description must be resolved before tests can be derived for the parts affected.

Before we move on to architecture testing, we will briefly review the work that has been done in verification and simulation.

Verification

Verification seeks to confirm absolutely, on paper, that a given implementation does meet its specifications. Two implementation-specific ways of architecture verification are microprogram verification and hardware verification.

The microprogram verification approach^{16,18,42} can be summarized as follows: given the formal specifications of the target machine and the description of the underlying microengine, the formal verification system looks at a microprogram written for the microengine and attempts to prove that the microprogram running on the microengine would emulate (i.e. implement) the target machine. So far this approach has only been applied to very simple machines.

** For a detailed exposition on architecture specification, see Reference 17.

Hardware verification^{24,13} deals with methods to prove the correctness of hardware designs. In this approach, descriptions of low level components and a description of how they are interconnected are taken as input. The goal is to verify that the given interconnection satisfies some higher level specification.

Both microprogram verification and hardware verification can only verify paper designs. Some test programs are still needed to check out the actual hardware. They must also develop accurate models of low level components which are changing rapidly with technology.

Simulation

Simulation has the advantage that it is easier to do than verification. Simulating computer hardware using software, however, usually results in a speed penalty ranging from a thousand to one up to a million to one. Hence it is usually impractical to test a design using simulation beyond running some very short tests that check some internal workings and critical paths. Besides the severe speed penalty, simulation also faces the problem of developing good models for rapidly advancing components and technology.

Testing

Testing has the strong appeal that it deals directly with the physical implementation, which is what one really wants to validate, rather than some abstract description of the implementation. Testing is also much easier to do than verification or simulation. One can start writing a test program with nothing more than a natural language specification whereas verification and simulation both require modeling, formal description of the architecture and of the particular design, and a software system that can carry out the verification and simulation. These advantages make testing the most practical, though not totally satisfying, way to validate an implementation.

The drawback of testing is that it cannot give complete assurance—in practice it often gives less than satisfactory assurance. The former is a direct result of the affirmative nature of testing and of the complexity of a computer—because exhaustive testing, which is the only way to give complete assurance, is impractical for even a simple computer. The latter, however, is usually caused by the lack of good test methodologies and test programmers. It can be drastically improved if the object to be tested can be analyzed in detail before test programs using the analysis as guidelines are written.

The viability of testing as a validation tool lies in the empirical fact that implementations usually have regular structures and therefore the errors made in designing them are not totally random. To illustrate this, let us consider testing the ADD instruction of a computer. In almost all cases, only a tiny fraction of the $2^n \times 2^n$ (where n is the word length) possibilities would be tested and then one would declare with confidence that the adder *works*, and indeed it usually *does*! The reason for this is that the tests usually

cover most of the probable errors: experience shows that the chances of having errors that could not be discovered by the tests is pretty small. If errors are truly random, however, and one is asked to test a black box whose internal structure one does not know about, then one cannot hope to achieve any high level of confidence by testing only a tiny fraction of the input possibilities. Needless to say, testing all the possibilities is out of the question—e.g. $2^{32} \times 2^{32} = 2^{64} > 10^{19}$. The question then is: amongst a sea of possible errors, how do we pick out the most probable ones and test for them? The error-analysis techniques developed later in this paper attempt to answer this question.

ARCHITECTURE TESTING

Architecture testing defined

Architecture testing is functional testing aimed at validating implementations of an architecture. An *architecture testing program* is designed to be a tool for certifying different machines claimed to implement a specific architecture. “Functional testing” means that the tests primarily aim at finding *design and logical errors* rather than problems in realization (e.g. repeatability and bit dependencies) or hardware failures. The level of confidence an architecture testing program can provide depends on the probability of having errors undetected by its tests.

Considerations in writing an architecture testing program

The most crucial constraint for an architecture testing program is time. Any testing that can be done within a reasonable amount of time is but a tiny fraction of all possible tests. The critical issue in writing an architecture testing program is therefore how to select the most profitable tests and test data for a given time constraint.

Unlike diagnostics which often must locate faults rapidly in the field, architecture testing programs can have run time in the order of days. But that is still only a fraction of the test cycles one would like to have. Fortunately the tests in an architecture testing program do not depend on the results of each other and therefore different parts of the program can be run simultaneously on many prototypes in parallel to obtain more test cycles. Program size should not be a constraint since only one test needs to be in core at any one time (can simply use overlays). Because the total program is likely to be huge, however, a good testing methodology should allow automatic generation of test data and test programs to avoid the tedious task of test programming.

Ideally we would like an architecture testing program to be as *implementation-independent* as possible. However, as we have pointed out before, the only way to get high confidence in testing “black boxes” is exhaustive testing. Therefore any practical architecture testing program must necessarily make certain assumptions about the implementations it is going to be run on in order to cut down the test space. In other words, an architecture testing program must

be written with certain *classes of implementations**** in mind. Within the target classes, the program should be *non-implementation-specific* in that it should be designed to effectively test all implementations within the classes. This is a case of tradeoff between generality and efficiency—one wants to have a program that can effectively validate as many kinds of implementations as possible while there are practical limitations as to how much resources the program can consume.

Recent developments and related work

Two current developments have contributed to the recent interest in architecture testing. One is standardization efforts like the MCF project³ which need independent validations of prototypes submitted by various contract bidders. The second development is the spread of microprocessors and LSI components. Many microprocessor and LSI parts are now manufactured by a half-dozen companies representing almost as many different implementations. The need for assurance of compatibility, together with pin limitation, have generated considerable interest in functional testing.¹⁹

Some research in the field of program testing^{20,21,11} is of considerable interest to architecture testing. The work that is closest to architecture testing is that of compiler validation.^{12,14} An area of special interest is test data selection techniques⁷⁻⁹—one can represent a function in an architecture by a canonical procedure written in a hardware description language like ISPS and then use the selection techniques to choose test data. Architecture testing and program testing bear many similarities, and research done in one area is likely to benefit the other.

ERRORS IN IMPLEMENTING AN ARCHITECTURE

Why do people make errors? What errors do people and design systems make? If one knows why people make errors, one can try to prevent them in the first place, thereby getting at the root of the problem. If one knows what kind of errors are likely to occur in a particular environment, one can orient one's testing effort accordingly to maximize return on the effort. It is wasteful to test for errors that are almost certain not to occur while more likely errors are not tested for. The kind of errors that people make varies with their task environments. In implementing a computer architecture, the likelihood of different types of errors varies with technology, design tools used, experience and training of the design group, available history of previous implementation errors (designers are usually more aware of them), project management etc..

We began with the conjecture that although every imple-

mentation effort has its own error probability distribution, overall the errors are likely to fall into several general categories. Identifying these categories would give some insight into the nature of implementation errors as well as providing guidance for the writing of architecture testing programs. Instead of grouping known implementation errors into categories like what some people have done with errors in programming,^{10,7,25} the approach of first conjecturing error categories and then calibrating them was adopted. This approach was chosen because (i) we wanted to explore implementation errors from an implementor's viewpoint, and (ii) very few error histories were publicly available.†

Design of the experiment

There are four phases in our experiment: preliminary study, conjecture, case study and calibration of conjecture. Each phase is explained in detail below.

Preliminary Study—To find out why people make errors and what the sources of errors are, the following studies were conducted:

1. Architecture specifications (mainly those of the PDP-11§) were studied for error-prone spots.
2. Experienced programmers of the PDP-11 were interviewed. They were asked to recall "errors" they have made in learning to use the architecture. The "errors" include: wrong assumptions, unclear specifications, confusions caused by counter-intuitive features etc.
3. Existing classifications of errors in programming were used as food for thought.^{7,10,25}

Conjecture—Based on the information gathered in the preliminary study, the author proposed several categories as likely sources of errors.

Case Study—Using the proposed categories as a guide, a "likely error analysis" of the PDP-11 computer architecture was made. The analysis was aimed at revealing the error-prone spots in the written specifications and the architecture itself. The idea is to simulate an actual architecture test programmer using the categories as guidelines for testing. Based on the results of the analysis, specific tests are recommended so that the effectiveness of the methodology can be calibrated later on.

Calibration—The above three phases were completed without knowledge of the actual implementations errors that have actually occurred. To see how well the methodology had done, the specific tests were compared against lists of real implementation errors which were only made known to the author *after* the tests had been developed.

*** An example of an implementation class is the family of 16 bit adders implemented by n bit full adder slices ($0 < n < 16$) with carry look ahead. Another example is multiplication implemented by repeated additions, whether it is implemented in hardware, firmware, or software. Multiplication implemented by table lookup is in a different class because it requires a totally different test strategy.

† Complete error histories are rarely published. In fact, some manufacturers try their best to conceal errors they have made in the past.

§ The PDP-11 04/05/10/35/40/45 Processor Handbook published in 1975 by Digital Equipment Corporation is used throughout this study. For those who are unfamiliar with the PDP-11, see Appendix A for a brief description of the addressing modes and instructions.

Errors in implementing an architecture

The preliminary study revealed eight likely *sources* of errors. They are:

1. Incomplete and imprecise specification
2. Interdependent side-effects
3. Asymmetry/nonuniformity
4. Logical complexity
5. Boundary values
6. Counter-intuitive and unusual features
7. Inconsistencies in nomenclature and style
8. Missing functions

Each of these categories is explained in detail in the following subsections. Real-life examples, mostly from the PDP-11, are given whenever appropriate. Since the categories overlap with one another, some examples have been somewhat arbitrarily classified.

Incomplete and Imprecise Specification

Whether the incompleteness in an architecture specification is intentional or accidental, the hardware of any implementation does *something* for the unspecified operations. Users are often tempted to use those peculiar "features" in their programs. If later models do not have the same "features," there is a software incompatibility problem. An incomplete specification may cause implementors to use an incompatible scheme in implementing the unspecified operations.

A specification which appears precise to its writer may be imprecise or ambiguous for others because of nontrivial implicit assumptions made by the former. Following is an example from the PDP-11 handbook:

- The overflow and carry condition code settings for the subtract (SUB) and compare (CMP) instructions are described in the processor handbook as follows:
V: set if there was arithmetic overflow as a result of the operation, that is if operands were of opposite signs and the sign of the source was the same as the sign of the result; cleared otherwise.
C: cleared if there was a carry from the most significant bit of the result; set otherwise.
 One must have a good knowledge of two's complement arithmetic to be able to understand this. The setting of the carry condition code even presumes a particular way of implementing the subtract operation—complement and add. In fact, the borrow generated by a hardware subtractor would be the exact opposite of the carry generated by the presumed method. In any case, the operations should have been precisely defined to avoid any misunderstanding.

Interdependent side-effects

Instructions which have multiple side-effects are error-prone, especially if the outcome of the instruction depends on the order in which the side-effects are carried out. Sometimes ambiguities can occur at the interfaces of architectural features which are individually well-defined. An instruction consisting of multiple operations is inherently ambiguous if the order of the operations is not clearly specified and the effect of the instruction depends on this order. Most often this arises when there are multiple operations on the *same* register or memory location within an instruction and the order of operations is not explicitly stated in the specification.

Asymmetry/nonuniformity

Asymmetry/nonuniformity often causes additional complexity in programming and in implementation. Asymmetrical/nonuniform side effects, notably condition code settings, are usually counter-intuitive as well.

- The instruction MUL Rn, SRC will cause Rn to contain the low order part of the result if R is an *odd*-numbered register and cause Rn to contain the high order part of the result if Rn is an *even*-numbered register. This asymmetry would not have occurred if the multiply instruction is defined such that the low order part of the result, which is what is needed most of the time, is always stored into Rn, and not R_{n+1}.
- The automatic sign extension that occurs in moving a byte to a register with the MOV_B instruction often catches programmers off guard. One would expect a byte instruction to operate on a byte and yield a one byte result. This nonuniformity is actually caused by the more fundamental nonuniformity* that registers are not byte addressable while all memory locations are.

Logical complexity

Some instructions are error-prone due to their sheer complexity. The human mind does not efficiently handle complexities beyond a certain threshold. Complex interactions that change a lot of processor states (trace traps, interrupts etc.) are conceptually hard to understand as well as difficult to implement, especially if multiple activities can occur at the same time. Extra testing is required to ensure the correctness of complex instructions.

* We are not saying that this nonuniformity was a bad design decision; in fact it was probably a good one. We just want to point out that any nonuniformity is a likely source of errors.

Boundary values

Boundary values for an instruction are input values that are at the boundaries of different decision regions in the input domain of the instruction. This concept is analogous to decision branches inside a program which compare input or computed values against some test values in order to determine the execution paths that the program should follow. In fact, the inclusion of this category is inspired by the frequent occurrence of boundary value errors in programming. The most prevalent kinds of errors in this category are *missing boundaries* and *off-by-one* errors. Missing boundaries are situations in which one or more of the boundaries between decision regions are missing. Off-by-one errors are errors in which a boundary is off a distance of one (for some appropriate definition of distance) away from where it should be.

Counter-intuitive and unusual features

Features that deviate from or behave just opposite to what one would normally expect or find in other architectures are error-prone. They also considerably slow down programmers who have to deal with them. Similarly, inappropriate or non-mnemonic names for instructions invite errors. Without proper explanation and motivation, even a useful feature may create confusion.

- Some condition code settings in the PDP-11 are counter-intuitive. For example, the increment and decrement instructions do *not* affect the carry.²³

Inconsistencies in nomenclature and style

Inconsistencies and exceptions are often introduced due to carelessness or ignorance. It is often penny-wise and pound-foolish to foul up an otherwise uniform and clean style just to squeeze an extra bit of performance out of an architecture.

- In PDP-11 instruction nomenclature, instructions that end with a B are supposed to be byte instructions. The SWAB instruction, despite having a B as the last character of its name, is actually a word instruction—it takes a word operand and generates a word result. It would probably be better to call it SWAP.
- The multiply and divide instructions store the high-order word of their two-word results in R_n and the low order word in R_{n-1} , which is just opposite to the practice of storing a high byte at the *higher* address within a word. The same comment applies to the scheme of storing the high part of a floating number in the *lower* word.

Missing features

This is not really a source of error, but it is put here as a reminder that a good test program should at least test for the existence (not necessarily the complete correctness) of every feature. It is not uncommon that relatively simple features are left out of an implementation due to the oversight or lack of experience of its designers. To illustrate what I mean by features in an architecture, the major features of two typical instructions are presented below.

ADD:

- Add source operand to destination operand and store result in destination.
- If there is a carry out, set the carry bit, clear it otherwise.
- If the result is zero, set the zero indicator, clear it otherwise.
- If the result is negative, set the negative indicator, clear it otherwise.
- If the addition results in an overflow/underflow, set the corresponding indicator.

HALT:

- If in user mode, causes an “illegal user instruction” trap.
- If in supervisor mode, stop all operations:

1. All flags are left untouched,
2. The program counter points to the next instruction following the HALT.

A major feature can often be broken down further into several minor features, depending on the complexity of the instruction. To guard against leaving some major features untouched, a comprehensive checklist for features of each instruction should be used. Each entry on the checklist roughly corresponds to a leaf on the decision tree of the instruction.

More likely than not, most features would have been exercised by tests written for other categories, thus it will only be necessary to write special tests for the items that are left untouched by other tests. In order to reduce the test space to a practical size, often only the existence, and not the correct functioning, of features can be established by such tests.

A CASE STUDY OF THE PDP-11 ARCHITECTURE

The architecture testing philosophy advocated in this paper is rather straightforward: given the amount of resources one is willing to spend in testing, try to *minimize the probability* that an error is undetected. This implies that one should test for the most likely errors first. In fact, these are often the only errors that one can afford to test for. The crucial question here is: *what* are the most likely errors?

This section presents a "likely error analysis" of the basic PDP-11 architecture** as specified in PDP-11 processor handbooks⁵. The likely errors are identified and classified using the proposed criteria. It must be pointed out here that what "likely errors" are depends on when in the development of an implementation the tests are made. We have been assuming all along that we will test prototypes that have most instructions "working" and that we are looking for the *obscure* errors. The PDP-11 is selected for case study because (i) it is a major computer family having numerous implementations; (ii) the history of its implementation errors is readily accessible to allow evaluation of the proposed testing strategy; and (iii) the basic architecture is simple enough for a thorough study of this sort.

The analysis is intended to be *illustrative* rather than complete—someone who is willing to spend the energy needed to analyze an architecture for likely errors can probably turn up more potential bugs. A list of recommended tests is given at the end of each section. As a whole the recommended tests should be viewed as "hole-plugging" tests to be added on top of any testing scheme that covers basic and obvious functions.

Incomplete or imprecise specification

The handling of hardware error conditions is only briefly mentioned in the processor handbook. There is no well-defined priority scheme for handling multiple, simultaneous processor trap conditions. The handbook also does not specify clearly what happens if a trap occurs in the *middle* of an instruction.

Other problems in the specification:

- The specification of the overflow V bit setting is wrong for the subtract carry (SBC) instruction. It is stated as follows in the manual:
V: set if (dst) was 10000; cleared otherwise
It should instead be
V: set if (dst) was 10000 and C was 1; cleared otherwise
- The specification of the carry condition code settings in the SUB and CMP(B) instructions presume particular implementation schemes (see *Incomplete and Imprecise Specification*).

Tests recommended:

- Trap priority—special hardware is probably required to carry out this test.
- Handling of multiple trap conditions caused by a single instruction.
- Test the V bit setting of the SBC and SBCB instructions.
- Test the C bit setting of SUB and CMP(B) instructions.

** For the purpose of this study, FIS, FIS, memory management, floating point instructions are not considered part of the basic PDP-11 architecture.

Interdependent side-effects

In the PDP-11, many instructions having interdependent side-effects are also inherently ambiguous because the architecture specification often does not specify the orders of execution for multiple side-effects.

Multiple, explicit operations on the same register

A double operand instruction is inherently ambiguous if (i) its source addressing mode uses Rn and its destination addressing mode is one of (Rn)+, @(Rn)+, -(Rn), and @(Rn); or if (ii) its source mode is one of (Rn)+, @(Rn)+, -(Rn), and @-(Rn) and its destination mode uses Rn. For example:

- OPR Rn,(Rn)+—if the second operand is fetched from memory (the first operand needs no fetching) and autoincrement is performed *before* carrying out the operation, the incremented Rn will be used as the source operand. But if the operands (including register operands) are first stored into temporary registers as they are fetched, the original value of Rn would be used as the source operand. The latter is more intuitive, but the processor handbook makes no statement about this ambiguity.

Test recommended:

- For all double operand instructions, test all the 64 combinations of addressing modes that are ambiguous. Of course, we would need to define how the combinations should behave before we can test them.

Multiple, explicit and implicit operations on the same register

PC (register 7) is automatically incremented each time it is used to fetch a word from memory. It is used implicitly in some addressing modes while SP (register 6) and some memory locations (notably those reserved for trap vectors) are used implicitly by several instructions. Instructions that operate on these registers or memory locations explicitly as well as use them implicitly at the same time deserve special attention.

- A double operand instruction that uses the PC is ambiguous if (i) its source is PC and its destination is one of (PC)+, @(PC)+, -(PC),@(PC),X(R), and @X(R); or if (ii) its source is one of the list just given and the destination is PC.

Tests recommended:

- Test all the 12 ambiguous combinations of PC addressing modes.

Modification and decision on the same operand

If an instruction both modifies its operand(s) and uses it for a decision (branch, setting of condition codes etc.), then the relative order of the modification and the decision becomes critical.

Tests recommended:

- Test JMP (Rn)+ and JSR Rm,(Rn)+ which are both ambiguous.

Asymmetry/nonuniformity

Fundamental asymmetries/nonuniformities

- Registers are not byte addressable while all the memory locations are.
- Highest page of main memory is reserved for the I/O page.
- Some memory locations are special (e.g. processor status word, stack limit etc.).
- Some memory locations are not writable (e.g. some status registers).
- Some instructions implicitly use special memory locations (EMT, TRAP, BPT, and IOT).

Tests recommended:

- Make sure that the I/O page is in the right place.
- Make sure that all the special memory locations are there. For instructions that use special memory locations, make sure that they access the correct special locations when executed.

Other asymmetries/nonuniformities

- Logical instructions COM(B), BIT(B), BIC(B), and XOR have different condition code setting conventions. In COM(B), the V bit is cleared but the C bit is *not changed*.
- The autoincrement deferred addressing mode @(Rn)+ always increments Rn by 2, even for byte instructions, whereas (Rn)+ increments Rn only by 1 for byte instructions. Similarly, @-(Rn) always decrements Rn by 2, even for byte instructions, whereas -(Rn) decrements Rn only by 1 for byte instructions.
- Automatic sign extension in MOV_B -,Rn (see *Asymmetry/Nonuniformity*).
- MUL & DIV instructions store low order part of result into R v 1 (see *Asymmetry/Nonuniformity*).

Tests recommended:

- Test COM(B) for the correct setting of the C bit.

- Test @(Rn)+ and @-(Rn) for incrementing/decrementing Rn by *two*.
- Test sign extension in MOV_B -,Rn.

Logical complexity

There are not many complex instructions or features on the basic PDP-11. The MARK instruction and trace trap are probably the most complex features and the trap instructions (EMT, TRAP, BPT, IOT, RTI, RTT), SOB, JSR, and RTS instructions deserve some extra testing effort.

Tests recommended:

- Test the previous instructions/features to make sure that the right *sequences* of operations are performed when they are invoked.

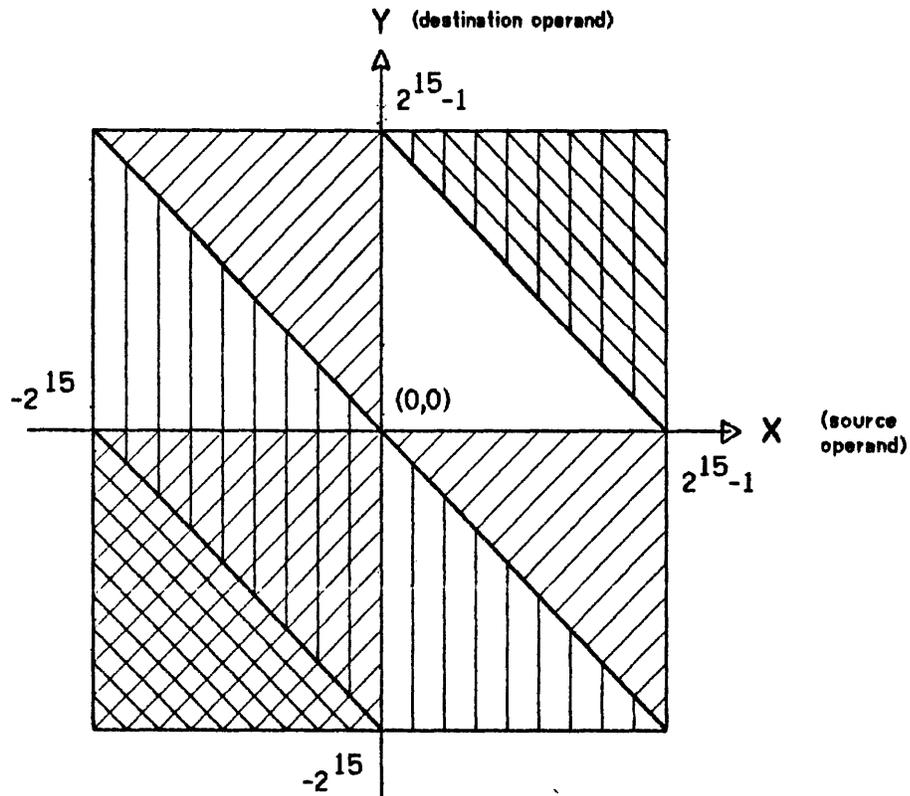
Boundary values

It is straightforward to figure out the boundary values for logical instructions—just test all four combinations for each bit position. It is often the case for arithmetic instructions, however, that there are too many boundary values and subsets must be chosen among them. Without going into detailed arguments, we assert that testing just a few key points on a boundary is almost as good as testing all the points on the boundary. This approach is illustrated below through a “boundary value analysis” of the ADD instruction.

The input domain is partitioned into different decision regions*** for each of the four condition codes N,Z,V, and C (which stand for Negative, Zero, overflow, and Carry respectively). For example, one region would consist of all input values that generate results which are less than zero and will therefore set the N bit while the complement of this region would consist of those input values that generate results which are not less than zero. There are well-defined boundaries between the partitions (see Figure 1). For example, the boundary values lying on the two sides of the longest N bit boundary are given by $y+z=1$ and $y+x=0$ respectively (Figure 2). Note that the partitioning is symmetrical with respect to the line $y=x$ because ADD is a commutative operation and that the partitions for different condition codes often have common boundary values. Besides correct setting of condition codes, one may also want to test for the following: the correct operation of each output bit†, correct generation and propagation of carry from each

*** A region may not appear contiguous on a two-dimensional graph. The input domain actually wraps around. For example, $2^{15}-1$ (011111₂) and -2^{15} (100000₂) are neighboring points. “Neighboring” means that the distance between the two points is one for *some distance measure*. In this case, the distance is measured by numerical difference. In other cases, we may want to use the geometric code distance measure in which neighboring points are those points that differ from the original point by one bit—hence there are n neighboring points for each point (where n is the word length).

† In the case of the ADD instruction, adding 00 . . . 00 & 11 . . . 11, 01 . . . 01 & 10 . . . 10, 10 . . . 10 & 01 . . . 01, 11 . . . 11 & 00 . . . 00, and 11 . . . 11 & 11 . . . 11 can test each output bit individually.



-  Region in which the C (Carry) bit should be set
-  Region in which the N (Negative) bit should be set
-  Region in which the V (oVerflow) bit should be set

Boundaries:

- N:** $y+x = 0$ & $y+x = -1$; $y+x = 2^{15}-1$ & $y+x = 2^{15}$; $y+x = -2^{15}$ & $y+x = -(2^{15}+1)$
- Z:** $y+x = 0$ & $y+x = -1$ & $y+x = 1$;
- V:** $y+x = -2^{15}$ & $y+x = -(2^{15}+1)$; $y+x = 2^{15}-1$ & $y+x = 2^{15}$;
- C:** $y+x = 0$ & $y+x = -1$; $y = -1, x \geq 0$ & $y = 0, x \geq 0$; $x = -1, y \geq 0$ & $x = 0, y \geq 0$;

Number of points on the boundaries

2^{18}

3×2^{16}

2^{17}

2^{18}

Figure 1—Boundary values for condition code settings of the ADD instruction.

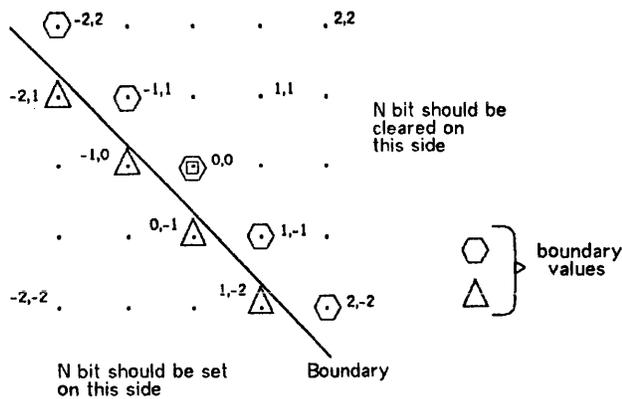


Figure 2—A close look at the N bit boundary.

position . . . Each of these have their own set of boundary values that need to be tested. In general, one first determines the things (actions) that one wants to check out, then partitions the input domain into decision regions for each particular action and finally picks out the boundary values as test data.

Ideally all boundary values of an instruction should be included as test points in testing the instruction. If one cannot afford to test all of them (e.g. in a 32 bit machine), a subset of "key points" could be chosen for testing. The minimum set of test points recommended are those values that are either at the "corners" of boundaries or at extreme points of the input domain. The existence of a boundary can be tested by crossing it from one boundary value to a neighboring value on the other side of the boundary. The process of selecting the key points is illustrated in Figure 3.

Tests recommended:

- Verify the operation (condition code settings, etc.) of each logical and arithmetical instruction for *all* its

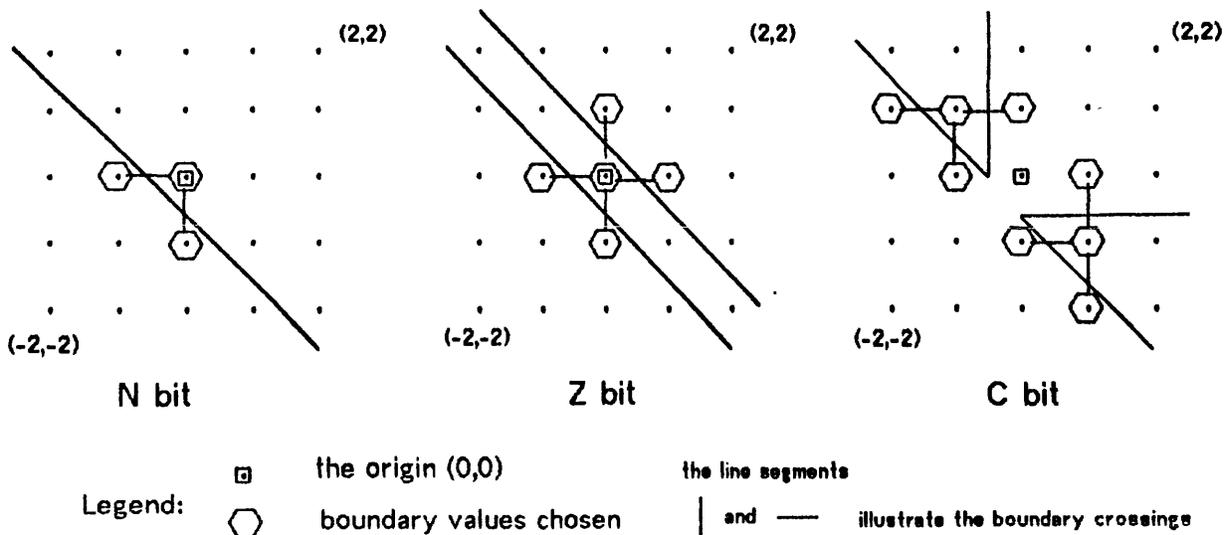


Figure 3—Three examples showing the selection of test data based on boundary crossings.

boundary values. If this is not practical, pick a subset of "key points." For logical instructions, the simplest test is to assume independence among different bits in a word and apply the input pairs of 000000 & 000000, 000000 & 177777, 177777 & 000000, and 177777 & 177777.§

Counter-intuitive and unusual features

- Complement (COM), a logical instruction, sets the C bit instead of leaving it untouched.
- The increment (INC) and decrement (DEC) instructions do not affect the carry.

Tests Recommended:

- Test the C bit setting of the COM(B) instruction.
- Test that the carry bit is truly unaffected by INC(B) and DEC(B) instructions.

Inconsistencies in nomenclature and style

- The usage of the "contents of" notation, (. . .), is inconsistent.
- The notation used to represent a register is also inconsistent.

Test recommended:

- None. In this case none of the inconsistencies is very serious. If one is writing a complete test program, however, then it would be worthwhile to pay special attention to even minor inconsistencies.

§ A better test is to apply some algorithmic patterns (like walking 0's and 1's) to systematically check for cross-coupling between bits.

Missing features

A good test program should try to test for the *existence* (not necessarily the complete correctness) of as many features as possible.

Tests recommended:

- The existence of all major features should always be checked out.
- As our resources permit, test for the existence of as many minor features as possible.

RESULTS AND CONCLUSIONS

The eight error categories are conjectures established based on their *potential* of causing implementation errors. To examine how well they capture actual errors, the tests recommended in the previous section were calibrated against two error histories. The first one is a list of incompatibilities among models of the PDP-11 and the second one is a list of errors found in an ISP "implementation." In both cases, only errors in implementing the basic architecture* are considered.

Comparison with published incompatibilities among various models

This error history is published by DEC itself⁵ and it lists known differences among five implementations of the PDP-11 architecture (see Appendix B). These are usually obscure errors that have slipped through the conventional validation tests. Among the 14 implementation errors, eight would definitely be caught by the recommended tests, three would probably be caught and the remaining three would likely slip through. All, however, fall into five of the eight categories and hence would likely be caught by more detailed tests (which requires a more detailed analysis of the PDP-11 architecture than what was done).

Comparison with errors found in the PDP-11 ISP

The ISP description of a computer architecture can be considered an implementation of that architecture through emulation on the "Register Transfer Machine."¹ A PDP-11 ISP description was recently written and debugged.** A rather complete history of the errors that have been discovered in the description has been kept. The comparison (see Appendix C) revealed that eight of the 13 errors would definitely be caught by the recommended tests. Two would probably be caught and the remaining three would likely slip

through. All, however, fall into the error categories and would conceivably be detected by more detailed tests.

Discussion

The above comparisons suggest the following:

1. An error-oriented architecture testing program written with the proposed categories as primary test targets can augment existing test schemes. In the first case, the recommended tests caught a significant percentage of obscure errors that have slipped through conventional tests. More encouraging is that all the implementation errors in both cases fall into the eight categories. Hence if someone devotes enough time (perhaps months, a reasonable investment for a major computer family) to develop an architecture testing program using the proposed methodology, most of these errors can conceivably be caught by the detailed tests.
2. Exercises like the "likely error analysis" presented can help architects to improve their specifications and reduce implementation errors caused by problems in the specification. Such exercises are also useful in spotting error-prone areas, which often cause difficulty in implementation as well as programming, in the architecture itself.

In retrospect, other than a few categories like *boundary values* which can conceivably be rigorously defined, most of the proposed categories have rather "soft" criteria and require human judgment in applying them. A lot more work is still needed to make them more specific and more amenable to automation. In addition, difficult areas such as floating point instructions have not been dealt with. We nevertheless hope that the proposed categories can serve as helpful guidelines for those who are going to write architecture testing programs. Analyzing an architecture specification to identify potential errors is a very time-consuming process, however, and the usefulness of any testing methodology ultimately depends on automation. Toward this end, research should continue on analysis techniques and the automatic generation of test data and test programs.

ACKNOWLEDGMENT

The author would like to thank Dan Siewiorek for initiating the study and helping along the way. Thanks are also due to Mario Barbacci, Len Shustek, Bob Sproull, Richard Swan, and the referees for their valuable suggestions.

REFERENCES

1. Barbacci, M. R., G. E. Barnes, R. G. Cattell and D. P. Siewiorek, *Symbolic Manipulation of Computer Descriptions: The ISPS Computer Description Language*, Technical Report, Dept. of Computer Science, Carnegie-Mellon University, 1978.
2. Birman, A. "On Proving Correctness of Microprograms," *IBM J. R. & D* Vol. 18, No. 4, 1974, pp. 250-266.

* Errors in features like user/supervisor/kernel modes, memory management, and floating point instructions are not considered.

** Originally written by Dan Siewiorek at CMU, for details see Reference 22.

3. Burr, W. E., A. H. Coleman and W. R. Smith, "Overview of the Military Computer Family Architecture Selection," *AFIPS Conference Proceedings* Vol. 46 (National Computer Conference) 1977, pp. 131-137.
4. Carter, W. C., W. H. Joyner and G. B. Leeman, "Automated Experiments in Validating Microprograms," *Int'l Symposium on Fault Tolerant Computing: FTC-5*, Vol. 5, 1975, p. 247.
5. Digital Equipment Corporation, *PDP-11 04/05/10/35/40/45 Processor Handbook*, Digital Equipment Corp. Maynard, Mass., 1975.
6. Digital Equipment Corporation, *Microcomputer Handbook*, Digital Equipment Corp., Maynard, Mass., 1976.
7. DeMillo, R. A., R. J. Lipton and F. G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," *Computer*, Vol. 11, No. 4, 1978, pp. 34-41.
8. Geller, M., "Test Data as An Aid in Proving Program Correctness," *CACM*, Vol. 21, No. 5, 1978, pp. 368-375.
9. Goodenough, J. B., and S. L. Gerhart, "Towards a Theory of Test Data Selection," *Proc. International Conference on Reliable Software*, 1975, pp. 493-510.
10. Hartwick, R. Dean, "Test Planning," *AFIPS Conference Proceedings*, Vol. 46 (National Computer Conference), 1977, pp. 285-294.
11. Hetzel, W. G., (ed.), *Program Test Methods*, Prentice Hall, 1973.
12. Hicks, H. T., "The Air Force Cobol Compiler Validation System," *Datamation*, 1969, pp. 73-81.
13. Hoehne, H., and R. Piloty, "Design verification at the register transfer language level," *IEEE Trans. on Computers*, Vol. 24, No. 9, 1975, pp. 861-867.
14. Hoyt, P. M., "The Navy Fortran Validation System," *AFIPS Conference Proceedings*, Vol. 46 (National Computer Conference), 1977, pp. 529-537.
15. IBM, *IBM System/370 Principles of Operation* IBM Corp., Form GA22-7000-4, 1976.
16. Joyner, W. H., W. C. Carter and G. B. Leeman, "Automated Proofs of Microprogram Correctness," *Ninth Annual Workshop on Microprogramming*, Vol. 51, No. 55, 1976.
17. Lai, Konrad K. K., *Computer Architecture Specification*, Technical Report, Dept. of Electrical Engineering, Carnegie-Mellon University, 1978.
18. Leeman, G. B., "Some Problems in Certifying Microprograms," *IEEE Trans. on Computers* Vol. C-24, No. 5, 1975, pp. 545-553.
19. McCaskill, Richard, and Wayne E. Sohl, *Study of Limitations and Attributes of Microprocessor Testing Techniques*, Technical Report, NASA, contract NAS8-31954 (final report), Macrodata Corp. (prepared for George C. Marshall Space Flight Center), 1977.
20. Miller, E. F., "Program testing: Art meets Theory," *Computer*, Vol. 10, No. 7, 1977.
21. Miller, E. F., "Program Testing," *Computer* Vol. 11, No. 4, 1978, pp. 10-12.
22. Parker, R. Alan, *A Guide to the PDP-11 JSP*, Technical Report 5403-232, RAP,NRL Prob 54B02-31, Naval Research Lab, 1977.
23. Russell, R. D., and K. Hall, "The PDP-11: A Case Study of How Not to Design Condition Codes," *The Fifth Annual Symposium on Computer Architecture*, Vol. 5, 1978, pp. 190-194.
24. Wagner, T. J., *Hardware Verification*, Technical Report AIM-304, Stanford Artificial Intelligence Lab., 1977.
25. Youngs, E. A., "Human Errors in Programming," *Int. J. Man-Machine Studies*, Vol. 6, 1974, pp. 361-376.

APPENDIX A

A brief description of PDP-11 addressing modes and instructions

Single operand instructions have the format *OPR destination* and usually perform $d \leftarrow op d$.

Double operand instructions have the format *OPR source, destination* and usually perform $d \leftarrow s op d$. Each operand can be accessed using one of eight addressing modes, giving

a cross-product of 64 ways of addressing operands in a double operand instruction.

Addressing Modes

<i>Modes</i>	<i>Symbolic</i>	<i>Description</i>
0	R	(R) is operand
1	(R)	(R) is address
2	(R)+	(R) is adr., increment R after fetching
3	@(R)+	(R) is adr of adr., incr. R after fetching
4	-(R)	decrement R before fetching, (R) is adr.
5	@-(R)	decr. R before fetching, (R) is adr. of adr.
6	X(R)	indexing, (R)+X is adr.
7	@X(R)	(R)+X is adr. of adr.

APPENDIX B

Architectural incompatibilities among five implementations of the PDP-11

The incompatibilities listed below are compiled from a list in *Microcomputer Handbook* published by Digital Equipment Corporation in 1976. The list, entitled "LSI-11, PDP-11 Programming/Hardware Difference List," listed the known differences among five implementations of the PDP-11 architecture: LSI-11, PDP-11 05/10, 15/20, 35/40, 45. In compiling the following, differences among features that are not part of the basic PDP-11 architecture are not considered. Next to each of the architectural incompatibilities is listed the error category that the incompatibility belongs to and an indication of whether it would have been caught by the tests that we have recommended.

<i>Incompatibility in</i>	<i>Category</i>	<i>Would It Be Caught With the Tests?</i>
1. OPR R,(R)+ or OPR R,-(R)	Ambiguity in Sequence	yes
2. OPR R,@(R)+ or OPR R,@-(R)	ditto	yes
3. OPR PC,X(R) or OPR PC,@X(R)	ditto	yes
4. JMP (R)+, JSR Rm,(Rn)+	ditto	yes
5. JMP R, JSR Rm,Rn (both illegal instructions) trap differently	Nonuniformity	yes
6. SWAB does not change V in some models	Missing Functions	yes
7. Bus addresses of the registers are special	Nonuniformity	yes

			each of the errors is listed the category that the error belongs to and an indication of whether it would have been caught by one of the tests that we have recommended.		
8. Power fail trap has different priority w.r.t. RESET instruction	Incomplete Spec.	no			
9. RTT instruction not implemented	<deliberate omission—doesn't count>		<i>Error</i>	<i>Category</i>	<i>Would It Be Caught?</i>
10. RTI behaves differently with the T bit set	Logical complexity	probably	1. DEC did not set V bit when destination was 100000.	boundary value	yes
11. Priority between trace trap and interrupt is different	Incomplete spec.	yes	2. MOVB did not sign extend byte moved to register.	asymmetry	yes
12. Trace trap will sequence out of WAIT instruction on some models	ditto	probably	3. SBC did not set the V condition code	missing fn.	yes
13. Direct access to Program Status Register (a special memory location) can change the T bit in some models	Nonuniformity	yes	4. SWAB did not assign the result of the SWAB back to memory	missing fn.	yes
14. Odd address/non-existent traps using the stack pointer	Incomplete spec.	no	5. Byte instruction using indexed deferred addressing mode didn't work correctly.	nonuniformity	no
15. Guaranteed execution of the first instruction in an interrupt routine	Incomplete spec.	no	6. MARK instruction reset the stack pointer wrong	logical complexity	yes
16. Odd address trap not implemented on the LSI-11	<deliberate omission>		7. PSW did not have bus address 177776.	nonuniformity or missing fn.	yes
17. Effect of bus errors on PC/register modification	<part of error handling, whether this is part of the architecture is arguable>		8. SOB dropped highest bit of offset	asymmetry	no
			9. ASRB instruction operates on wrong field that is off by one bit.	missing fn.	yes
			10. ASHC & ASH: C and V bits are not set correctly (spec. not clear).	incomplete spec.	no
APPENDIX C			11. MUL stores intermediate result in a 17 bit register instead of a 16 bit register	peculiar to ISP, would probably be caught by boundary value tests.	
<i>Errors found in the PDP-11 ISP</i>			12. DIV; when source register addr. is odd garbage is produced.	incomplete spec./inconsistency in style	probably
The errors to be listed are compiled from a collection of memos documenting the errors that have been found in the PDP-11 ISP after it has been released by its author. The original PDP-11 ISP was written by Dan Siewiorek at Carnegie-Mellon University and has been maintained by Alan Parker of NRL. For the purpose of this paper, we have left out errors concerning user/supervisor/kernel modes, memory management, floating point instructions, and those errors peculiar to ISP as a programming language. Next to			13. DIV: divide by zero did not set condition codes and abort.	missing fn.	yes

A survey of methods for intermittent fault analysis*

by YASHWANT K. MALAIYA and STEPHEN Y. H. SU

State University of New York
Binghamton, New York

INTRODUCTION

As digital circuits grow in complexity, and as they are used in more and more critical applications, their reliability becomes an important consideration. This directs the attention of researchers involved in this field to two basic questions—first, how to design more reliable systems; second, how can something be said about the reliability of a particular system?

Causes leading to permanent failures can easily be examined. A class of faults called permanent or solid faults was identified and it has been extensively studied. Some solid faults in some lines in digital systems behave as if they are permanently connected to a logic 0 or a logic 1. They are called stuck at 0 or stuck at 1 faults. Significant analytical results are available to handle such faults.

Another class of faults, called intermittent or transient faults, are responsible for a very significant number of failures. The effects of such faults appear and disappear randomly. It has been noticed that an overwhelming proportion (up to 90 percent) of faults can be estimated to be intermittent. As these faults are more difficult to isolate and fix, they account for most of the field maintenance costs. Ball and Hardy¹ in their study found that, while the test and maintenance costs due to the permanent faults tend to decrease drastically with time, the costs incurred per unit time due to intermittent faults change very little. McConnell and Siewiorek² report that in an LSI-11 system with 24k words of dynamic MOS memory, one would expect a transient failure every 100 hours, whereas a solid failure can be expected in every 7700 hours. Needless to say, the problems of intermittent failures are very significant.

In this paper, a brief survey of the significant results available today is presented. These aspects of intermittent faults will be considered. First, physical reasons for intermittent faults will be considered. Second, various suggested models for intermittent faults will be examined along with the basic assumptions. Testing and reliability analysis are considered next. Finally, design methods are examined which take intermittent faults into consideration.

APPROACH

Some have suggested that a distinction between transient faults and intermittent faults be made on a quantitative basis. We will, however, refer to the entire class of such faults as intermittent faults or transient faults interchangeably. We distinguish between an intermittent fault being *present* and being *active*. When an intermittent fault is present, it can either be *inactive* without affecting the correct operation; or it can be *active*, and can cause the system to operate incorrectly. The activity of *signal-independent* faults does not depend on the inputs or the present state of systems. These are easier to handle, much as the analyses done so far assume only signal-independent faults. The activity of signal-dependent faults may depend on the logical values present in the system. An intermittent fault is said to be *well behaved*, if it behaves as a permanent (stuck-at) fault whenever it is active.

As the activity of intermittent faults is random, it is not easy to characterize them. The randomness makes it necessary to use probabilistic methods. The activity can be characterized by a set of parameters, which may be estimated by using statistical methods. These parameters would be utilized in arriving at various analytical results. Probabilistic methods have been successfully used to model random occurrences of permanent faults in digital systems, and related parameters have been estimated. Such parameters have been empirically correlated with system complexity (gate and pin count for example).^{3,4} While the intermittent faults behave in a more complex fashion, the use of probabilistic methods should be expected to yield practically usable results. Often expected reliability measures⁵ are given in a design specification which implies use of probabilistic and statistical methods.

PHYSICAL CAUSES OF INTERMITTENT FAULTS

Physical causes of intermittent failures could be external to the system, like temperature, or internal like loose connections. A combination of the two types can be suspected in several cases.

Yen⁶ has reported that in four-phase MOS circuits, some nodes in some circumstances may have voltages lower than

* This work is supported by the Division of Mathematical and Computer Science of the National Science Foundation under grant No. MCS78-24323.

normal, making them very susceptible to noise. Noise can be suspected to be a significant cause of intermittent faults. All electrical devices are inherently noisy; noise generated by a device can sometimes be correlated with the temperature and the electrical characteristics.⁷ Additional noise could also result because of strong electromagnetic fields in the area, which often accompanies switching of large currents.⁸ Switching in the same system can cause noise. It has been noticed that significant amounts of noise can be generated due to refresh operation or instruction execution.⁹ One of the suggested models, for intermittent faults, characterizes noise on a bus.¹⁰

Loose or poor electrical connections can cause intermittent failures; such faults may be activated by either electrical or mechanical causes. Faults on the semiconductor chips are known to cause signal-dependent types of intermittent faults. Because of coupling between physically adjacent cells or faulty decoders, some patterns in a semiconductor memory may cause faults.¹¹ Some microprocessor chips have been found to possess instruction sensitivity.¹² Such faults will exhibit themselves intermittently. Some parts of an integrated circuit chip may get heated after some time period of use, and may display intermittently faulty behavior.

Temperature and radiation (especially in outer space) are examples of external causes. Proper operation of semiconductor devices is defined only in small ranges of these, and if they exceed the specified limits, or stay at the limiting value for a prolonged duration, components may behave erratically. Variations in the power supply can also cause intermittent failures. Airborne computing systems might encounter atmospheric electric discharges.

Usually causes of intermittent failures are not known. The only choice is to regard them as random processes. In a particular environment, their effect can be characterized by a specific set of parameters, which can be determined statistically or using empirical methods.

MODELING

Breuer¹³ suggested a two-state discrete-parameter model, using the assumption that the activity of the fault can be represented by a Markov process. His model provides a good starting point for studying the modeling of intermittent faults. If the outcome of a Markov process is known at an instant t , then for the duration following t , the history of the process before t is immaterial. If an intermittent fault is present, then the system can either be in state fault-not-active (FN) or in state fault-active (FA). As shown in Figure 1, two transitions for a state can be defined. Let the state of the system be described after every Δt seconds (called time-step). Then the parameters indicated in Figure 1 denote the probability of the appropriate transition in time Δt . If $p_0(q)$ and $p_1(q)$ are the probabilities of being in states FN and FA at time t_q respectively, then

$$(p_1(q+1), p_0(q+1)) = (p_1(q), p_0(q)) \begin{bmatrix} r & \bar{r} \\ s & \bar{s} \end{bmatrix}$$

Clearly the estimated values of both \bar{r} and s depend on the

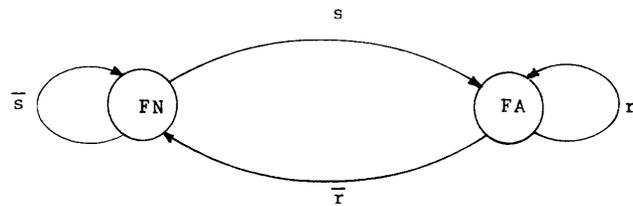


Figure 1—Two-state discrete parameter.

time-step chosen, and they have to be re-estimated if the time step is changed. The same problem is encountered if one uses Spillman's approach,¹⁴ as he uses the same discrete parameters as Breuer for his analysis and numerical comparison. This is restrictive, as the time-step is not an attribute of the fault, rather it is chosen to correspond to a particular clock-rate. This model, like all others to follow, assumes signal-independent intermittent faults.

A zero-order Markov model, which is simpler to use, has been utilized by Kamal and Page¹⁵ to find procedures for fault-detection when a digital circuit may have a multiple number of faults. In their model, the behavior of the fault is fixed at all times. They define a conditional probability of malfunction (e_i), for each intermittent fault f_i , which is the probability of the fault being active, given its being present. This model has been used by Kamal,¹⁶ Koren and Kohavi,¹⁷ and Shedletsky.¹⁸

Merryman and Avizienis¹⁹ use a model which characterized intermittent faults by an arrival rate, τ , which is assumed to be constant over the life of the system, and the transient duration (in which faults remain active) D_τ . The transient duration D_τ is random, with a density function f_{D_τ} . This density function was assumed by them to be

$$f_{D_\tau}(t) = \gamma e^{-\gamma t} \quad (2)$$

where γ is a parameter.

This is a lucky assumption, as it will allow us to correlate this model with the next one. This model has been used by Ng.²⁰

A continuous-parameter Markov model has been used by Su, Koren and Malaiya.²¹ The model is based on these causal considerations. If the system is in a fault-not-active (FN) state at a certain time, then the probability that it will be found in fault-active (FA) state after a short time Δt , should be proportional to the duration Δt . If a constant of proportionality λ is assumed, then $Pr\{\text{system will be in FA at } t + \Delta t | \text{it was in FN at } t\} = \lambda \Delta t$. Similarly, for a reverse transition,

$$Pr\{\text{system will be in FN at } t + \Delta t | \text{it was in FA at } t\} = \mu \Delta t \quad (3)$$

These two assumptions generate a continuous parameter Markov model shown in Figure 2. This model has several desirable characteristics. It is easily conceived, amenable to mathematical analysis and poses fewest restrictions. Later, this model will be used to correlate different models.

A continuous-parameter Markov model can be represented by a pair of differential equations. If $p_0(t)$ and $p_1(t)$

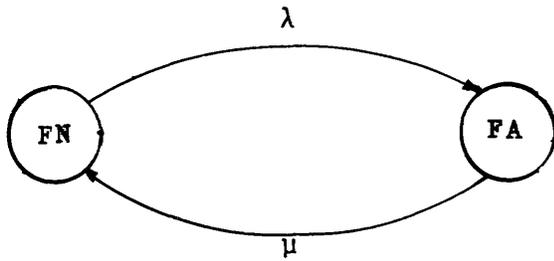


Figure 2—Continuous-parameter Markov model.

are the probabilities that the system is in state FN and FA at time t , respectively, then²²

$$\begin{aligned}
 p_0(t) &= p_0(t_0)e^{-(\lambda+\mu)t} + \frac{\mu}{\lambda+\mu}(1-e^{-(\lambda+\mu)t}) \\
 p_1(t) &= p_1(t_0)e^{-(\lambda+\mu)t} + \frac{\lambda}{\lambda+\mu}(1-e^{-(\lambda+\mu)t})
 \end{aligned}
 \tag{4}$$

Occasionally, transitional probabilities $P_{i,j}(t)$ are more convenient to use; $P_{i,j}(t)$ is the probability of being in state i at time t_0 . It should be noted here that $1/\mu$ is the average duration the fault remains active continuously and $1/\lambda$ is the average duration it is inactive.

Siewiorek, Canepa and Clark¹⁰ have used a model in which faults take the form of noise signals added to the bus signals. It is characterized by two parameters, F , which is the probability that a receiving device will read the bus line incorrectly in the presence of a fault; and N , the duration fault remains active (in bus cycles). These parameters were used to compute a reliability measure for a fault-tolerant multiprocessor.

An adaptation of Breuer's model is used by Lala and Hopkins.²³ They have used parameters α and δ , which are called frequencies of transition, to characterize the transitions between the two states. The ratio α/δ is referred to as the latency factor; the higher it is, the higher the percent of time fault is inactive and hence invisible.

These models can be correlated by using the continuous-parameter Markov model.²⁴ Breuer's discrete-parameter model will approach this model as the time-step becomes small. The parameters then can be compared when the time-step is assumed to be sufficiently small.

Using Equation 4 it can be seen that if $t \rightarrow \infty$, then $P_0(t) \rightarrow \mu/(\lambda + \mu)$ and $P_1(t) \rightarrow \lambda/(\lambda + \mu)$, i.e., the probabilities will assume "steady-state" values. If the state of the system is examined only after long durations (i.e., $e^{-(\lambda+\mu)t} \approx 0$), then the probabilities associated with the two states could be regarded as constant, and the model will reduce to Kamal and Page's model.

In Merryman and Avizienis' model, the parameter arrival rate τ corresponds to λ in the continuous-parameter model. However as τ is constant, a fault may "arrive," even though the fault which had arrived before had not disappeared. Clearly, for a correlation it must be assumed that the fault duration is small. Their parameter γ corresponds to μ .

Parameters F and N used by Siewiorek, Canepa and Clark can also be correlated to the continuous-parameter model.

The correspondence is, however, not as simple as in previous cases. The model employed by Lala and Hopkins is equivalent to the continuous-parameter model. These correspondences are summarized in Table I.

Varshney³⁵ has suggested some models which incorporate some memory in the fault behavior. One way memory can be incorporated is to describe the system by a multistate Markov model, in which some states correspond to faulty behavior and the rest to fault-free behavior. Other methods suggested by him use conditional probabilities $p(m/n)$; which is the probability that a fault-free duration is of length m given the preceding fault-free duration was of length n .

MODELS FOR RELIABILITY CALCULATIONS

The continuous-parameter Markov model just discussed assumes that the intermittent fault is present all the time. Sometimes an intermittent fault may become present only after a duration of time. This can be modeled by providing another state in which the fault is not present, and then providing the appropriate transition.

Koren and Su²⁵ have suggested the model shown in Figure 3. The fault is not present in state FNP, it is present but not active in state FN and it is both present and active in state FA. A somewhat more general model shown in Figure 4 is suggested by Malaiya and Su²⁶ in which transition is also allowed from state FNP to FA. This model is a result of merging two concurrent Markov processes.²⁴

The model of Figure 3 reduces into the well known model for permanent faults by letting $\mu \rightarrow 0$ and $\lambda \rightarrow \infty$. The same can be achieved by using the model of Figure 4, by simply letting $\mu = 0$.

In a somewhat similar model, Lala and Hopkins²³ allow a transition from FNP to FA. Transitions from FNP to FN are not allowed. This restricts their model, as the transition from FNP to FN should be possible.

Recovery is a commonly used way to counter-act for

TABLE I.—Relationships between Existing Models and the Continuous Parameter Model

Model	Parameter	Symbol	Correspondence to	Correspondence valid when
Breuer	Transition probability	s \bar{r}	$\lambda \Delta t$ $\mu \Delta t$	time step Δt is very small
Kamal & Page	Conditional probability of malfunction	e	$\frac{\lambda}{\mu + \lambda}$	time between two successive samplings is large
Merryman & Avizienis	Average arrival rate	τ	λ	fault duration is extremely small
	Average duration	$\bar{D}, 1/\gamma$	$1/\mu$	
Siewiorek, Canepa & Clark	Probability of incorrect read operation	F	$\frac{\lambda(1-e^{-(\lambda+\mu)t_c})}{\lambda+\mu}$	applicable to faults on the bus-line
	Duration	N	$1/\mu t_c$	
			$t_c = \text{clock period}$	
Lala & Hopkins	Frequencies of transition	α δ	μ λ	apparently equivalent

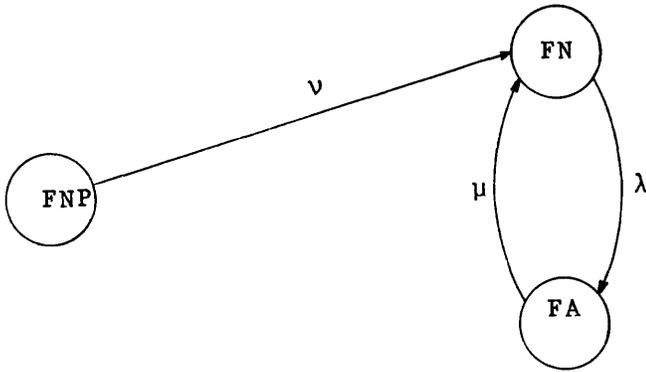


Figure 3—Model suggested by Koren and Su.

intermittent faults.²⁷ In processors, recovery is initiated after an intermittent fault is detected. Merryman and Avizienis¹⁹ assume that T_d , the time between fault occurrence and fault detection is random, probably governed by an exponential distribution law. T_r , the time between fault-detection and the end of the recovery is assumed to be constant. They have also used these parameters.

$$C_i \triangleq Pr \{ \text{recovery from transient} | \text{transient occurs} \}$$

$$1 \triangleq Pr \{ \text{transient fault is interpreted as permanent} \}$$

Lala and Hopkins²³ have used a parameter governing the transition from state fault-active (FA) to the state fault-not-present (FNP). This parameter can be called recovery rate. Therefore, their model remains Markovian even after inclusion of the recovery process. Ng²⁰ has also assumed that the overall process can be modelled by a Markovian process, as the recovery time can be assumed to be short. These assumptions are, however, open to question.

Modeling a multiple number of intermittent faults in the same system can be complicated. For Kamal and Page's model this does not pose much of a problem, as behavior is assumed to be independent of time. Su, Koren and Malaiya²¹ have assumed that only one of the possible intermittent faults can be present in the circuit. Malaiya²⁴ has modelled a multiple number of independent faults by independent Markov processes.

One important question that needs to be asked is "Given a model, how are the parameters to be estimated?" Some preliminary statistical methods have been considered in Reference 24. Parameters could be dependent on the kind of environment the system operates in; therefore, experiments

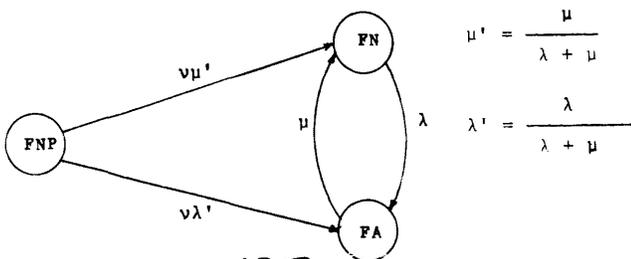


Figure 4—Model suggested by Malaiya and Su.

for parameter estimation must take place in a real or closely simulated environment. If any kind of empirical relationships exist between the parameters and the environmental attributes (like temperature), they could be used to predict the parameters in a certain environment, also they could be used for "accelerated testing."

Most models can be used to model the behavior at a certain node, as well as the behavior at the output port.²⁴ If a model is used to characterize the output-port behavior, the parameters will be dependent on not only the physical behavior of the fault; but also the inputs and the state of the system.

TESTING INTERMITTENT FAULTS

The integrated circuits must always be tested if they are to be used in a reliable system. Most types of IC testing is done for permanent faults only. The importance of testing intermittent faults is apparent, as was stated by Yen.⁶ However, one problem for testing intermittent faults is that the faults may not be active when the test is applied. Therefore, to be reasonably sure that an intermittent fault is not present in the circuit, the test would have to be repeated a number of times. The repeated testing would be terminated after either the fault has been detected or a prespecified time has been spent for testing. How often a test is to be repeated has been of significant concern, as the total testing time available is always limited.

Breuer¹³ defined a confidence level CL, which is the probability that a test would detect the presence of an intermittent fault. He then derived the number of times that the test has to be repeated. He has also suggested how tests can be generated to detect a particular intermittent fault.

Kamal and Page¹⁵ have considered detection procedures for combinational circuits which could be used for circuits with several possible intermittent faults; however, it is assumed that at most one intermittent fault can be actually present. It is also assumed that the faults are well behaved and signal-independent; these assumptions have been used by others, who have considered testing strategies for intermittent faults. They have suggested that the total number of times a fault is to be tested can be found using one of the two decision rules. One decision rule is to stop when the a-posteriori probability (i.e., the probability after testing) of an intermittent fault being present in the circuit, drops below a specified number, say, 10^{-6} . The other decision rule is to stop when a ratio of probabilities, called a likelihood ratio, becomes less than a small number called the threshold.

They have extended the results for the general case. Considering a set of intermittent faults (f_1, f_2, \dots, f_n) , and a set of tests (t_1, t_2, \dots, t_m) , a fault matrix $A=(a_{ij})$ can be defined such that

$$a_{ij} = \begin{cases} 1 & \text{if } t_j \text{ tests for } f_i \\ 0 & \text{if } t_j \text{ does not test for } f_i \end{cases}$$

Using this matrix, and an appropriate decision rule, an integer programming problem can be set up. The solution of

this integer programming problem would give the required number of repetitions of each test, so that the total number of repetitions is minimum.

The method is further developed by Kamal.¹⁶ He has suggested a procedure to identify the intermittent faults. After an intermittent fault is detected, a fault identification procedure is initiated. This is done by sequentially ruling out the possibility of having a particular set of intermittent faults. Finally only one fault is left as a possible candidate and is thus identified. He has shown that the total number of tests is finite; therefore, the identification procedure will eventually terminate.

Savir²⁸ has considered random testing for intermittent faults in irredundant (nonredundant) combinational circuits. Testing is random in the sense that the sequence of tests is random. The tests chosen for random testing are members of an "optimal generating set." To get an optimal generating set, one starts with the set of all possible tests. From this set a test is deleted, if another test can be found which detects the same faults or additional ones. Deletion is carried on until none of the remaining tests can be deleted, giving the optimal generating set. A non-linear optimization problem then can be set up. This will maximize the probability that a fault will be detected, given that the circuit is faulty. The optimization problem will then generate the relative frequency of various tests in the optimal generating tests. Finally the total number of tests required can be found by considering a pre-specified "escape probability," which is the probability that the circuit will be declared fault-free after testing, even though it is faulty.

Koren and Kohavi¹⁷ have considered the problem of locating intermittent faults in combinational networks. They have defined a cost function, which equals the number of distinguishing tests required to locate a fault. A procedure using dynamic programming is given, which generates a sequential decision tree minimizing the cost function. The sequential decision tree gives the order in which tests are repeatedly applied. They have suggested another method in which only "reasonably minimum" decision trees are generated, however, computation time is considerably saved.

Su, Koren and Malaiya²¹ have suggested that signal-independent faults in combinational circuits can be tested more efficiently by continuous testing. In continuous testing, tests are applied continuously for a duration of time, and outputs are monitored continuously (asynchronously). For both continuous testing and repetitive testing (which can be used for both combinational and sequential circuits), a duration of testing time is calculated, so that the probability of missing the faults is below a specified number. They have considered the optimum fault-detection experiment for the general case when a set of tests is available, each test detecting one or more faults. They have assumed that only one of the possible faults is present at a time; however, this restriction can be removed as shown in Reference 24.

RELIABILITY ANALYSIS

The objective of reliability analysis is to be able to predict the performance of a system with respect to time. If only

permanent faults are considered, one reliability measure is usually enough.

$$R(t) = Pr\{\text{a system will operate correctly at time } t\} \quad (5)$$

If intermittent faults are also considered, then several reliability measures can be defined. A somewhat stricter reliability measure, which can be called durational reliability, can be defined as

$$R_d(t) = Pr\{\text{system operates correctly during } (t_0, t_0 + t)\}. \quad (6)$$

This measure is suitable for critical systems, in which failures of short duration would render them useless. An example of such systems could be the navigational system of a missile, designed to hit very selectively.

In other cases instantaneous reliability defined below could be a more suitable measure.

$$R_i(t) = Pr\{\text{system operates correctly at } t\}. \quad (7)$$

Instantaneous reliability is also referred to as availability.³¹ This could be a more useful reliability measure, for example, for the communication control system of an artificial satellite.

The two reliability measures will be the same if only permanent faults are considered. When intermittent faults are also considered, they are different; in fact, no general relation to compute one directly from the other can be found. As one will expect,

$$\begin{aligned} R_d(t) &= R_i(t) \text{ for } t = t_0 \\ R_d(t) &< R_i(t) \text{ for } t > t_0 \end{aligned} \quad (8)$$

As permanent faults are a special case of intermittent faults, the existing results for permanent faults can be used to check the results for intermittent faults.

If the definition of durational reliability is modified as shown below, the resulting measure is called "survivability."¹⁹

$$S(t) \triangleq Pr\{\text{no fatal failure in duration } (0, t)\}$$

This definition allows for successful recovery operations.

Sometimes a reliability measure can be specified in terms of mission time.³¹ The *mission time* is the duration in which a reliability measure is above a given number, say 0.90. The mission time, thus is the expected duration, in which a system will operate reliably.

A scan of literature regarding the reliability of systems with permanent faults will reveal that there are several techniques in existence. Sometimes, parameters for individual components are the starting point,⁴ sometimes parameters chosen characterize a whole system.³¹ In the following discussion, it is assumed that the parameters are estimated at the output ports of the individual subsystems.

The reliability of hardware redundancy systems has been considered by Koren and Su²⁵ and Malaiya and Su.^{24,26} No repair or recovery is assumed. The different modules are assumed to be completely shielded from each other.

In Reference 25, the model of Figure 3 is used. Using this

model, analytical results are presented which enable one to compute the duration reliability of a simple Triple-Modular-Redundancy (TMR) system when each module may potentially possess some faults from a set of all possible faults. By assuming a set of parameters, substantial improvement in reliability by using a TMR system was found. Results are extended to N-module redundancy systems.

The slightly more general model of Figure 4 was used in References 24 and 25. Using this model, several hardware redundancy schemes have been examined to evaluate their reliability. It should be remembered that it is the methods to compute reliability which are more interesting; the numerical results hold good for only a specific set of values of parameters. They have examined the well known Triple-Modular-Redundancy (TMR) and N-Modular-Redundancy (NMR) schemes, the NMR/Bipurge scheme (called by Ng²⁰ as NMR/Simplex), the NMR/Unipurge scheme (self-purging,³² called NMR/S by Ng²⁰), hybrid redundancy scheme and the reconfiguration scheme.³⁷ In the NMR/Unipurge system only the faulty module is purged out. In NMR/Bipurge system, two modules (one good, one bad) are purged in order to maintain an odd number of active modules.

An expression for instantaneous reliability in most cases can be found in a straightforward way. The analytical results are presented in Table II. It can be easily verified that in each case if μ is set equal to zero, results reduce to the well known results for systems with only permanent faults.

Durational reliability is harder to compute. One can take either one of two possible approaches—one involves solving a system of differential equations iteratively, the other uses a closed form general expression to get an approximate figure. As the system of differential equations may be "stiff," i.e., solutions may become unbounded if the proper step-size is not used and the approximate value can be used to check if the solution is bounded. If it is not within the bound, step-size may be reduced, and computations re-started from the previous step.

The basic model of Figure 4 can be used to graphically represent a fault-tolerant scheme, which can be described by an equivalent system of equations. Figure 5, for example represents a simple 5MR scheme. Only one possible fault in any module is assumed. Each state in Figure 5 is labelled by the numbers, the first gives the total number of modules with an intermittent fault present in it, the second is the number of modules with the fault active in it. The system leaves a state (i, j) in one of four ways.

1. Fault in one module with fault active may become inactive. In this case state $(i, j-1)$ will be reached.
2. Fault in one module with fault inactive, may become active, state $(i, j+1)$ will be reached.
3. Fault in one module may become present, but remain inactive, state $(i+1, j)$ will be reached.
4. Fault in one module may become present and active at the same instant, state $(i+1, j+1)$ will be reached.

Notice that the instant any three modules have the fault active in them, the system lands in the failed state, which is labelled F in Figure 5. As durational reliability definition

TABLE II.—Instantaneous Reliability of Various Schemes

Scheme	$R_i(t)$
Single Module	$\prod_{i=1}^n [\exp(-\nu_i t) + \mu_i' \{1 - \exp(-\nu_i t)\}]$
TMR	$3G^2 - 2G^3$
NMR	$\sum_{k=0}^{N-1} \binom{N-1}{k} G^{N-k} (1-G)^k$
NMR/Bipurge	$\sum_{k=1}^{N-1} \left\{ \frac{N(N-2) \dots (N-2(k-1))}{k! 2^k} (1-G'^2)^k \cdot G'^{N-2k} \right\} + (1-G'^2)(G')^N$
NMR/Unipurge	$\sum_{k=0}^{N-2} \left\{ \binom{N}{k} (1-G')^k (G')^{N-k} \right\} + d \binom{N}{1} (1-G')^{N-1} (G')$ $0 \leq d \leq 1.0$
Hybrid	$\sum_{q=s}^0 \left\{ \binom{N+s}{N+q} (1-G')^{N-q} (G')^{N+q} \right\} + \sum_{k=1}^{N-s} \left\{ \binom{N+s}{N-k} (1-G')^{k+s} (G')^{N-k} \right\} + 3 \binom{N+3}{3} G' (1-G')^2 (1-G')^{N-3}$
Reconfiguration	$\sum_{k=0}^s \left\{ \binom{5}{k} (1-G')^k (G')^{5-k} \right\} + 0.5 \binom{5}{1} (1-G')^4 (G')$

$G = Pr\{a \text{ module is operating correctly at time } t\}$
 $= \prod_{i=1}^n [\exp(-\nu_i t) + \mu_i' \{1 - \exp(-\nu_i t)\}]$
 $G' = Pr\{a \text{ module operates correctly in duration } [0, t]\}$
 $= \prod_{i=1}^n [\exp(-\nu_i t) + [\nu_i \mu_i' / (\lambda_i - \nu_i)] \{ \exp(-\nu_i t) - \exp(-\lambda_i t) \}]$
 $n = \text{number of intermittent faults}$
 $N = \text{number of modules}$
 $s = \text{number of spares}$
 $k = \text{number of faulty modules}$
 $q = \text{number of good spares}$

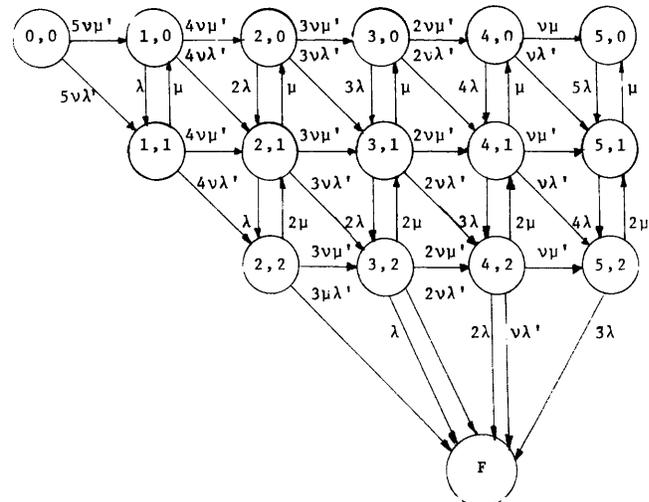


Figure 5—Model of a 5MR system with a single fault.

requires continuous correct operation, transitions back from the failed state need not be considered. The durational reliability of the system is given by the probability of the system being in a state other than the failed state. The reliabilities of various schemes are compared in Figure 6. Except for the case of the single module, all the schemes employ five modules.

From Figure 6 it should not be interpreted that the simple NMR scheme is always superior. For very different values of the parameters, some other schemes might be superior than the simple NMR scheme.²⁴ In the general case, there might be many possible intermittent and permanent faults with different parameters, then the best scheme for that value of parameters can be chosen by comparing curves for different schemes.

An algorithm has been presented²⁶ which would compute the reliability of various hardware redundancy schemes in

the general case when a mixture of permanent and intermittent faults are possible in a module.

Merryman and Avizienis¹⁹ have considered a recovery process in enhanced TMR systems. They have calculated the probability that a system will fail within the mission time. This probability is the sum of the probabilities of these two events (i) two modules become faulty at the same time (ii) with one faulty module, recovery in another fails. Recovery may fail if recovery is initiated before the fault becomes inactive, or if the recovery process is imperfect. They have presented results of numerical computation showing improvement in survivability in TMR systems if recovery is also used in addition to the massive redundancy.

Lala and Hopkins²³ have also considered the recovery process in a triple redundant system. They have modeled recovery as a transition, considering the overall process still to be Markovian. The same assumption is used by Ng.²⁰

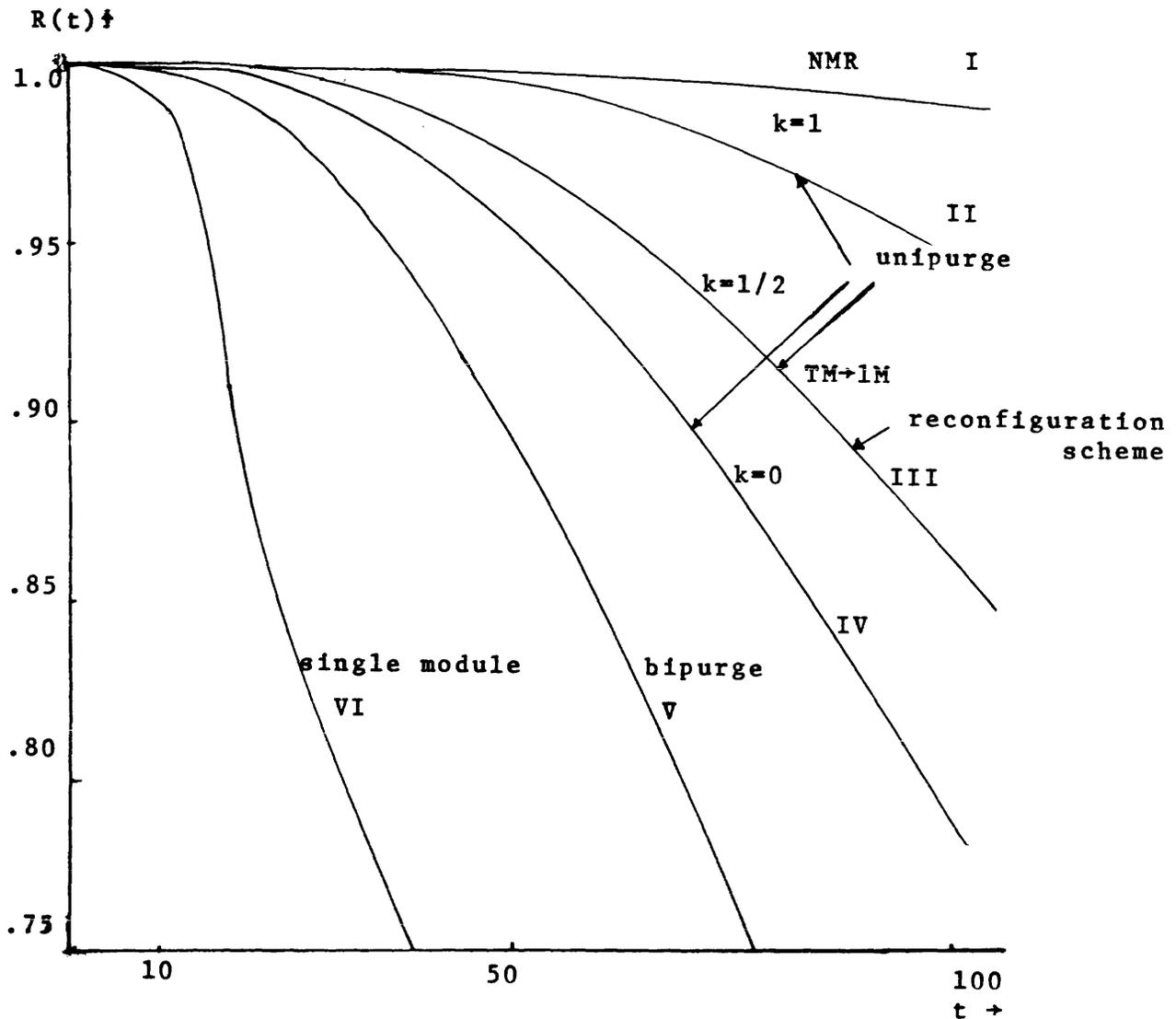


Figure 6—Comparison between various schemes.

Effects of this assumption, however, need to be investigated.

DESIGN CONSIDERATIONS

In a processor, an intermittent fault can become active and inactive again, however, its effect might not disappear immediately. If it effects the control part of the system, the system will be desynchronized and will continue to operate incorrectly until it gets synchronized again. In a fault-tolerant multiprocessor Cvm² desynchronization is reported to occur 25 percent of the times. Desynchronization is the most undesirable affect of intermittent faults. It should greatly improve reliability, if systems could be designed which are desynchronization-resistant and/or easily resynchronizable. Desynchronization-resistant designs may incorporate redundancy in the control part of the systems. Wakerly³⁴ has considered easily resynchronizable designs. Sequential circuits may be designed without global feedback, which possess special characteristics; such circuits can be resynchronized only by applying two successive vectors. Synchronization can also be easily achieved if extra reset inputs are provided. Such systems can then use TMR voting to keep them in synchronization.

Diagnosability of systems with interconnected units, which are capable of testing each other has been considered for permanent faults. Diagnosis capability may be an important consideration in a system design. For systems with permanent faults two measures of diagnosability are defined: a system is t_p -fault diagnosable without (with) repair if one test routine is sufficient to identify all (at least one) of the permanently faulty units, provided the number of such faulty units does not exceed t_p . Mallela and Mason³⁶ have defined a measure for intermittent fault diagnosability. A system is t_i -fault diagnosable when it is such that if no more than t_i units are intermittently faulty then a fault-free unit will never be diagnosed as faulty. Two different measures for intermittent fault diagnosability similar to permanent fault case can be defined, both of them, however, can be shown equal to the measure previously defined.³⁶ Mallela and Mason have also established least upper and greatest lower bounds for t_i in terms of t_p . A two-part procedure is given for determining t_i , which avoids exhaustive search.

FUTURE DIRECTIONS

We have only begun to know how to approach the problem of intermittent faults. While much of this area remains unexplored, one should expect to see results for the problems mentioned below.

Modeling and testing methods are suited for only signal independent faults. It can be safely said that a significant number of intermittent faults are signal-dependent. To obtain generalized modeling and testing methods is an unsolved problem. We need more insight into the physical nature of faults, specially when they are induced by noise.

Techniques to model burst type (faults which become

active and inactive many times in a short duration) and harbinger type (intermittent faults which eventually turn into permanent faults) are to be developed. In most analyses it is assumed that redundant modules are independent. However, the study at Carnegie-Mellon² has revealed that about 10 percent of all transient errors are simultaneous in different modules. This small fraction becomes significant when one considers the fact that such errors will cause failure in a simple TMR system. Probabilistic methods to handle such coupling between modules need to be developed. Methods to compute reliability, should be extended to include the possibility of simultaneous errors.

Methodology to estimate parameters in all cases have to be developed. The methods to estimate parameters can also be extended to check validity of the associated model, and will provide a means to correct a model if needed.

Suitable generalized hardware redundancy methods are needed, which will optimize the reliability performance when a set of intermittent as well as permanent faults are considered. Design methodology is needed to design easily synchronizable and/or desynchronization resistant systems.

REFERENCES

1. Ball, M., and F. Hardy, "Effects and detection of intermittent failures in digital systems," *Fall Joint Computer Conference Proceedings*, 1969.
2. McConnel, S., and D. P. Siewiorek, "C. vmp: the implementation performance and reliability of a fault-tolerant multiprocessor," Report #CMU-CS-78-108, Carnegie-Mellon University, March 1978.
3. Kasouf, G., "Evaluation of LSI/MSI reliability model relative to observed failure rates," SI-704, Aircraft Equipment Div., Utica, New York, January 1977.
4. Ingle, A. D., and D. P. Siewiorek, "Reliability models for multiprocessor systems with and without periodic maintenance," *Proc. FTCS-7*, Los Angeles, June 1977, pp. 3-9.
5. "Development Specification for the Fault-tolerant Spaceborne Computer (FTSC)," Space and Missile Systems Organization, Los Angeles.
6. Yen, Y. T., "Intermittent failure problems of four-phase MOS circuits," *IEEE Journal of Solid State Circuits*, Vol. SC-4, No. 3, June 1969, pp. 107-110.
7. Carlson, A. B., *Communication Systems*, McGraw-Hill, New York, 1975, pp. 118-128.
8. Kothemir, W. C., "The source and nature of transient surges," *IEEE Trans. Industry Applications*, Vol. IA-13, No. 6, November 1977, pp. 501-503.
9. Carey, B. J., "A power distribution and noise-suppressing element in MK4696 memory systems," Rogers Corp, Chandler, Arizona, Application Note # 1976.
10. Siewiorek, D. P., M. Canepa and S. Clark, "The architecture and implementation of a fault-tolerant multiprocessor," *Proc. FTCS-7*, June 1977, pp. 37-43.
11. Chiang, A. C. L., and R. Strandridge, "Pattern sensitivity of 4K RAM devices," *Computer Design*, February 1975, pp. 88-90.
12. Hackmeister, D., and A. C. L. Chiang, "Microprocessor test design technique reveals instruction pattern sensitivity," *Computer Design*, December 1975, pp. 81-85.
13. Breuer, M. A., "Testing for intermittent faults in digital circuits," *IEEE Trans. Computers*, Vol. C-22, No. 3, March 1973, pp. 241-246.
14. Spillman, R., "A Markov model for intermittent faults in digital systems," *Proc. FTCS-7*, June 1977, pp. 157-161.
15. Kamal, S., and C. V. Page, "Intermittent faults: a model and a detection procedure," *IEEE Trans. Computers*, Vol. C-23, No. 7, July 1974, pp. 713-719.
16. Kamal, S., "An approach to the diagnosis of intermittent faults," *IEEE Trans. Computers*, Vol. C-24, No. 5, May 1975, pp. 461-467.

17. Koren, I., and Z. Kohavi, "Diagnosis of intermittent faults in combinational networks," *IEEE Trans. Computers*, Vol. C-26, No. 11, November 1977, pp. 1154-1158.
18. Shedletsky, J. J., "Intermittent faults or design errors," presented in the *Intermittent Faults Workshop*, Johns Hopkins University, March 30, 1977.
19. Merryman, P. M., and A. Avizienis, "Modeling transient faults in TMR Computer Systems," *Proc. 1975 Annual Reliability and Maintainability Symp.*
20. Ng, Y. W., "Reliability modeling and analysis of fault-tolerant computers," Report No. NSF-MCS-7203633-76981, Computer Science Dept. UCLA, September 1976.
21. Su, S. Y. H., I. Koren and Y. K. Malaiya, "A continuous parameter Markov model and detection procedures for intermittent faults," *IEEE Trans. Computers*, June 1978.
22. Parzen, E., *Stochastic Processes*, Holden Day Inc., 1962.
23. Lala, J. H., and A. L. Hopkins, "Survival and dispatch probability models for FTMP computer," *Proc. FTCS-8*, Toulouse, 1978, pp. 37-43.
24. Malaiya, Y. K., "Modeling, testing and reliability analysis of intermittent faults in digital systems," Ph.D. dissertation, Dept. of Electrical Eng., U. S. U., Logan 1978.
25. Koren, I., and S. Y. H. Su, "Reliability analysis of N-modular redundancy systems with intermittent and permanent faults," to appear in *IEEE Transactions on Computers*.
26. Malaiya, Y. K., and S. Y. H. Su, "Reliability measures for hardware redundancy fault-tolerant digital systems with intermittent faults," submitted for publication.
27. Tasar, O., and V. Tasar, "A study of intermittent faults in digital computers," *Proc. NCC 1977*, pp. 807-811.
28. Savir, J., "Optimal random testing of single intermittent failures in combinational circuits," *Proc. FTCS-7*, June 1977, pp. 180-185.
29. Savir, J., "Testing for multiple intermittent failures in combinational circuits by maximizing the probability of fault detection," *Proc. FTCS-8*, 1978, page 212.
30. Landraut, C., and J. C. Laprie, "Reliability and availability modeling of systems featuring Hardware and Software faults," *Proc. FTCS-7*, 1977, Los Angeles, pp. 10-15.
31. Beaudry, M. D., "Performance related reliability measures for computing systems," *Proc. FTCS-7*, June 1977, Los Angeles, pp. 16-21.
32. Losq, L., "A highly efficient redundancy scheme: self-purging redundancy," *IEEE Trans. Comp.*, June 1976, pp. 569-578.
33. Su, S. Y. H., and R. Spillman, "An overview of fault-tolerant digital system architecture," *Proc. NCC*, June 1977, Dallas, Vol. 46, pp. 12-26.
34. Wakerly, J. F., "Transient Failures in Triple Modular Redundancy Systems with Sequential Modules," *IEEE Transactions on Computers*, Vol. C-24, No. 5, May 1975, pp. 570-573.
35. Varshney, P. K., "Intermittent faults in digital systems: Analytical models," submitted for publication.
36. Mallela, S., and G. M. Masson, "Diagnosable systems for intermittent faults," *IEEE Trans. Comp.*, Vol. C-27, No. 6, June, 1976, pp. 560-566.
37. Su, S. Y. H., and E. DuCasse, "A Reconfiguration Scheme for Tolerating Multiple Failures in Digital Systems," *Proceedings of International Computer Symposium*, Aug. 20-22, 1975, Taipei Taiwan, pp. 216-222, to appear in *IEEE Transactions on Computers*.

Architectural and design perspectives in a modular multi-microprocessor, the DPS-1

by KELLS A. ELMQUIST

InterSystems Inc.
Ithaca, New York

INTRODUCTION

For the last decade, one of the most active and exciting areas in computer architecture is the interconnection of computers to form parallel or concurrent systems. These systems are generally called "multiprocessors" or "distributed processors" and may range in organization from processors sharing a common memory to geographically isolated computer installations connected as a network. The low cost and ease of implementation of LSI microprocessors make them extremely attractive design possibilities for the implementation of general-purpose multiprocessor systems. Furthermore, the MOS technology used to implement the majority of microprocessors limits the instruction execution rates such that many applications are compute bound rather than limited by other system bandwidths. This indicates that a number of microprocessors may be effectively interconnected to increase the general system performance.

In this paper we will discuss the general system characteristics of all multiprocessor systems and attempt to derive a set of design requirements for a modular, microprocessor-based multiprocessor. Given this set of characteristics and design requirements, we will discuss two general interconnection schemes, the Global Bus and the Dual Port Memory, and analyze their suitability. The architecture and implementation of the DPS-1 modular multiprocessor will then be described and modeled in terms of limitations on system throughput and optimum cost-effectiveness.

MULTIPROCESSOR SYSTEM CHARACTERISTICS

Anderson and Jensen¹ have given a general taxonomy of computer interconnection structures which may be briefly summarized as follows: The basic element of communication between processors is the *message*. No distinction is made between different types of messages such as requests for service, data blocks, etc. The most basic taxonomic decision to be made in the design of a multiprocessor is whether messages will be transmitted *directly* from the source to the destination, or whether they are transmitted *indirectly*, requiring the intervention of some process which routes the message to a number of alternate destinations. The next decision to be made is whether messages will be transmitted

over paths which are dedicated or shared. A *dedicated* path is defined as one which is accessible from only two points, while a *shared* path may be accessed by an arbitrary number of points.

From these basic taxonomic decisions, we may classify the sundry multiprocessor architectures and analyze them in terms of certain system characteristics:

Modularity, the cost of making incremental changes in system capability. Modularity is most severely impacted by the choice between shared or dedicated message paths. Note that if all messages travel over dedicated paths, the cost of adding the *n*th processor requires the addition of *n-1* interconnection paths, whereas if all messages share the same path the cost is merely that of the processor.

Fault-tolerance, the system costs of different failures, the costs of their detection, and the costs of system reconfiguration to allow operation in degraded mode.

Bottlenecks, the performance limitations inherent in different message communication structures.

To these we may add:

Degree of coupling, the message transfer delay from sender to receiver.

Cost/performance ratio, impacted mostly by the costs of the interconnection paths.

DESIGN REQUIREMENTS

Given these system characteristics and their mutual optimization as criteria, we may postulate a set of design requirements suitable for implementation in LSI components either available now or available over the next few years:

1. The system must have low cost-modularity. The cost of adding one processing unit to the system should not greatly exceed the cost of the processing unit itself.
2. Each processing unit should have a local failure detect mechanism that would detect a local failure and suspend local operation before the processing module gained system access. Further, the system should be signaled of this condition.
3. The system should have a large address space, at least 1 megabyte but preferably 16 megabytes.

4. All data transfer protocols should accommodate both byte and word parallel transfers, to accommodate new 16 bit processors and peripherals.
5. System interrupt response should be optimized since real-time applications will be common.
6. All processors should be able to access all other processors and all resources global to the system.
7. Both direct and indirect message communication strategies should be allowed by the architecture. A great deal of research and experimentation is currently being conducted in the design of multiprocessor operating systems using both these communication methods, and the hardware architecture should not limit the installation of either type.
8. All contenders for shared paths should be capable of receiving dynamic priority allocation from the operating system.
9. Care must be taken to minimize loading on all shared paths.
10. Cost of the total system must be somewhat commensurate with the cost of its components. (\$1000 worth of processor modules and \$20,000 worth of interconnection would not be considered commensurate.)

ARCHITECTURE

For reasons of cost-modularity and general cost/performance (ratio considerations) with low bandwidth processors, complete interconnection via dedicated paths has been rejected. It can be seen that the cost of the interconnection scheme rapidly obscures the cost/benefits of micro processors, and that addition of one processor requires extensive system modification, and expense. Also, if we are to meet our design requirement that both direct and indirect message strategies be allowed, we must reject star configurations, since they do not allow direct message transfer. This leaves two major interconnection schemes, the time shared common bus and the multi-port memory.

The time-shared common bus has a number of cost advantages and meets a number of our design requirements. It has the lowest overall system hardware cost, and the least logical complexity of interconnection. The modularity of the common bus is excellent, both in terms of the cost of adding additional processor modules, and in terms of the homogeneous/non-homogenous nature of the modules. This allows modules with special, dedicated functions to be added with the same ease as a general purpose modules, and it allows the system designer to increase performance only where it is needed, relieving a bottleneck by modularly adding intelligence. In this case it becomes the responsibility of the system designer to partition the system requirements in such a way that a specialized processor justifies its cost, rather than being an under-utilized feature, or conversely, its absence an inherent limitation in a general purpose system.

If we dedicate one processor on the bus as a message switch, and all messages are routed through this processor, then we have configured the system for indirect message transfer. If, on the other hand, we treat all processors

equally, the system may be used with direct transfer strategies and no hardware reconfiguration is necessary. Note also that, in a common bus system, large messages may be communicated very quickly by the passing of address pointers.

The drawbacks to a common bus system are obvious. First, the bandwidth of the common bus is the upper limit on total system throughput; each processor's throughput must degrade every time a new processor is added to the system. Second, while the failure reconfiguration cost of common bus systems is very low and failure-effect with respect to processors is also very good, the failure-effect with respect to the common bus is usually catastrophic. Anderson and Jensen¹ make two comments pertaining to these drawbacks of common bus systems: First, that the common bus is not likely to saturate from communications alone, that is, if processors run programs in private or local memory, system bandwidth becomes very large before the common bus reaches saturation. And second, the low logical complexity tends to mediate somewhat the failure-effect in central message switching systems.

In order that processing modules be capable of running programs from local memory, and that we meet our design requirement that all processors be capable of accessing all memory, let us summarize the characteristics of a dual-port memory. A dual-port memory is a memory unit which may accept memory requests both from a local processor without creating a system bus access, and from the system bus, to which it appears as a normal single-port memory unit. Such a configuration has a number of advantages for a common bus multiprocessor. First, since programs may be run from local memory and the system bus may be reserved from global references or message transfer, there is the potential for a very high total system transfer rate. The bandwidth of the system bus is still the limiting factor in total throughput, but an order of magnitude increase in throughput is observed. Indeed, experience with the CM*² multiprocessor indicates that between 82 and 99 percent of all memory references are to local memory. The marginal cost of configuring memory units in this way, while being small to begin with, is far outweighed by the increase in performance. Second, the modularity of the common bus system, with all its advantages, is preserved. And third, since the memory is accessible to the whole system, the local processor need not become involved in transferring data to and from local memory. This is especially important in the transfer of long messages, where the message may be switched by simply passing an address pointer.

The DPS-1 multiprocessor uses a combination of both the common, time-shared system bus and the dual-port memory to achieve the majority of the design requirements.

INTERRUPTS IN MULTIPROCESSOR SYSTEMS

The parallel nature of multiprocessors makes their application to real-time computing problems very attractive. If this application is to be effective, however, the interrupt response of the system, that is, the worst case response time

to asynchronous service requests, and the ease with which the system may be reconfigured to provide service, must be optimized. Jaswa³ has determined four independent levels of interrupt applications in a real time multiprocessor, each level of which should be optimized. They are:

1. *Intra-processor communication and control.* This interrupt level responds to events which occur during an instruction execution by a local processor. An interrupt generated on this level generally informs the local processor that some unacceptable error condition has occurred, (illegal system bus reference, parity error, illegal memory reference, attempted write to protected memory, etc.) and that the processor should discontinue program execution and notify the operating system.
2. *Intra-system communication and control.* This interrupt level responds to service requests from system peripherals, whether they are local to the processor or global system peripherals. Optimization on this level involves such considerations as dynamic priority assignment, masking options, and processor selection for interrupt servicing.

3. *System executive control.* At this level the system executive (whether it is centralized or decentralized) may interrupt any local processor. The basic executive controls are (1) receive message at j , (2) execute task at j (3) pause.
4. *Interprocessor communication.* This level is used to initiate status and data transfers between processors. This level is only important in direct message transfer systems, in indirect systems it is contained in (3).

Mutual optimization of these interrupt levels indicates that a number of interrupt levels be available at each processing module, the highest priorities dedicated to error response and recovery, the next priorities dedicated to system executive control, and the remaining levels to peripheral requests, both on the local and global levels. If the shared system bus, however, has a number of interrupt vectoring levels, and if each processor is to be capable of responding to any level, then the number of priority levels at each processor module becomes unwieldy. Further, if every processor in the system is to be capable of responding to any system interrupt, either the interrupt service routines must be duplicated in each processor's local memory, or they

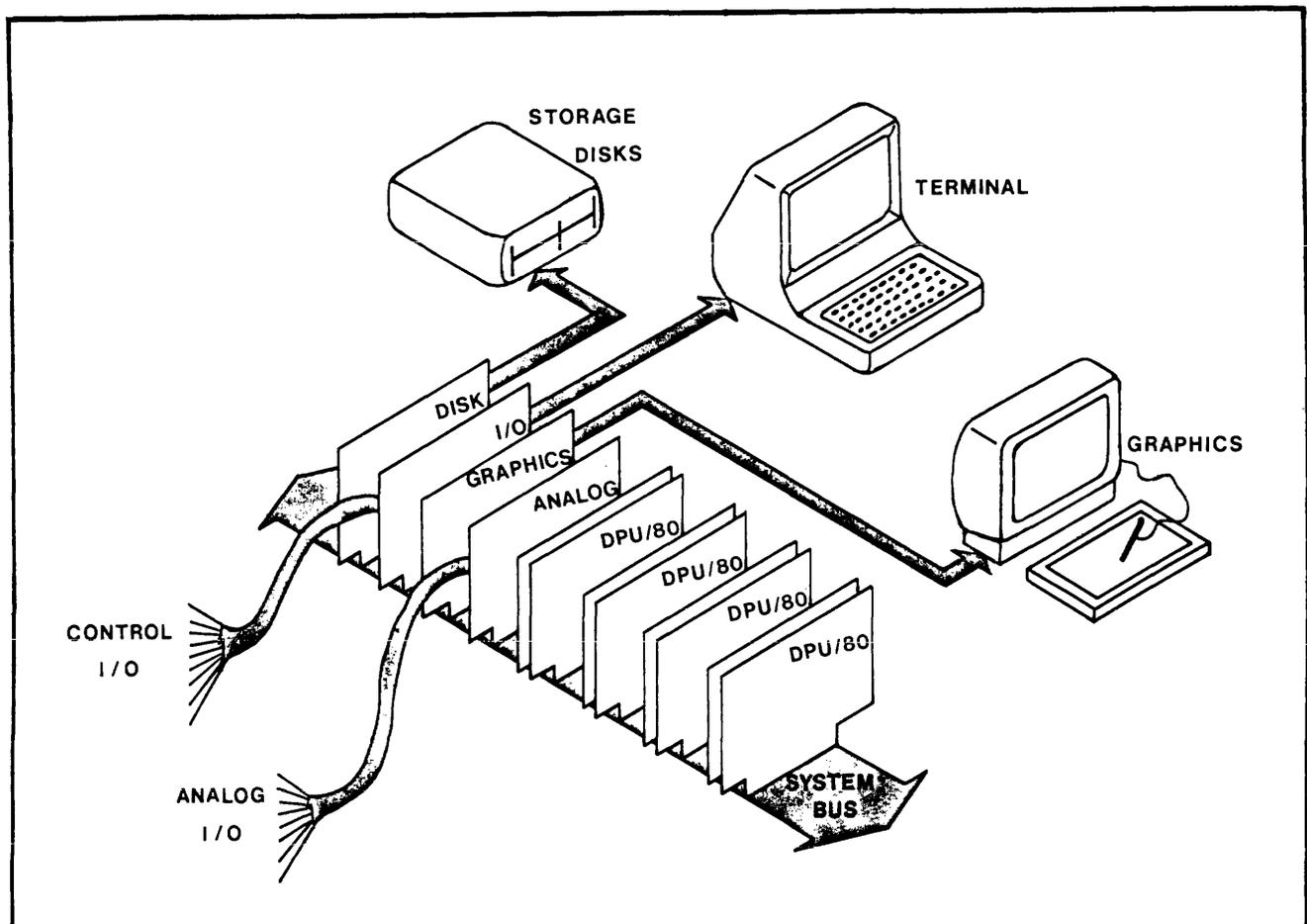


Figure 1—DPS-1 Distributed Processing System.

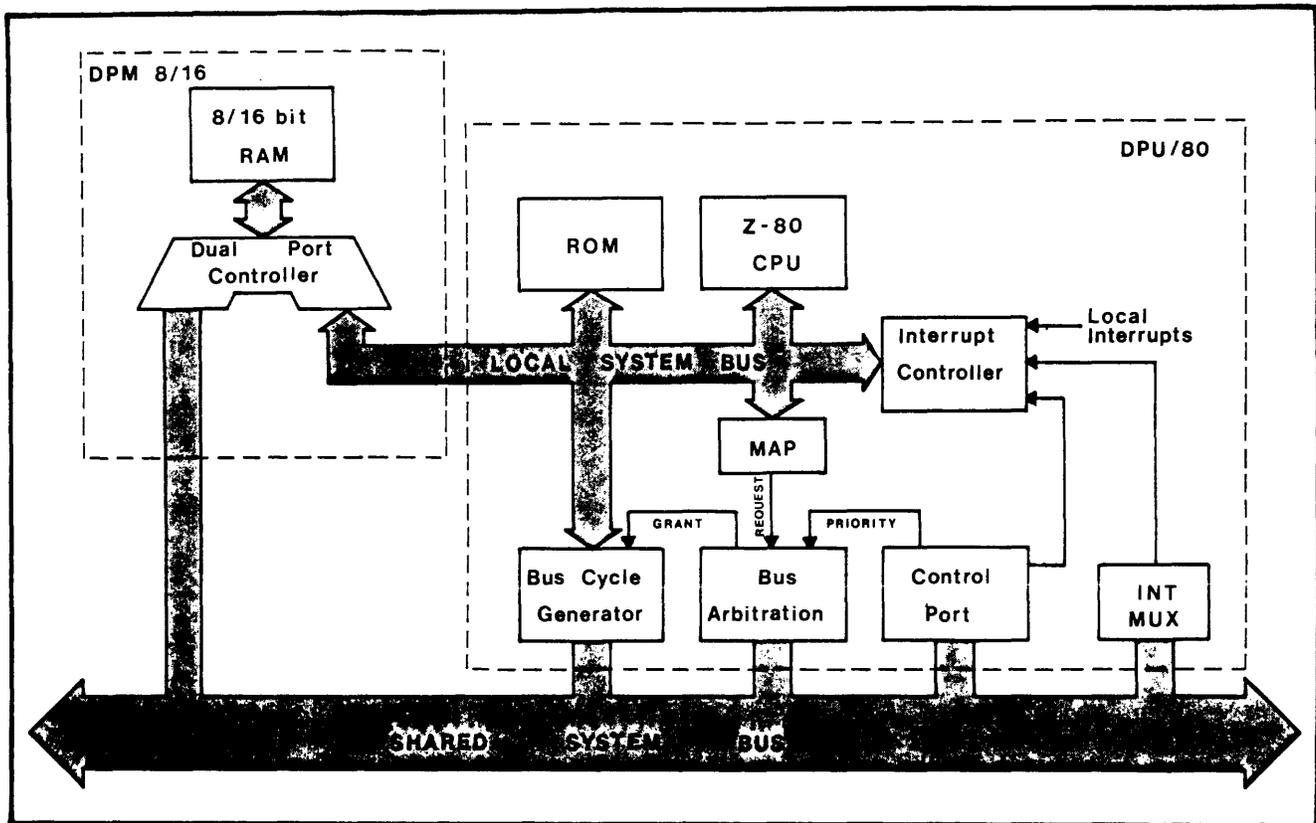


Figure 2—Distributed processing unit.

must be kept in global memory and run via the shared system bus, degrading total system throughput. Another problem with decentralized interrupt response is that it becomes cumbersome for the system executive to control the relationship between task priority and interrupt priority.

One common solution to these problems with decentralized interrupts is to assign one processor in the system as an interrupt processor. While this approach solves most of the problem noted above, it places two important restrictions on system interrupt response time. The first is that two processors become involved in handling a single interrupt, first the interrupt processor and then the processor whose task is concerned with the information. The second is that simultaneous servicing of two or more interrupts is no longer possible, since only one processor may respond to system interrupts. These two factors combine to seriously degrade the total interrupt response time of the system.

Our solution in the DPS-1 multiprocessor has been to implement a partially decentralized interrupt response strategy. There is an interrupt processor in the system capable of responding to all system vectored and system error interrupts, masking, rotating priorities, etc. Also, on each local processor a number of interrupt vectors are implemented. These respond to local error conditions, system executive interrupts, local peripheral service requests, and one vector level may be multiplexed among the global interrupt vectors under software control, or masked out altogether. By combining these strategies we thus provide the

flexibility of assigning any system interrupt to any processor for fast response, of simultaneous interrupt servicing, and also the simplicity of central interrupt servicing for all non-time-critical interrupts.

Further, we have specified two distinct types of interrupt response vectoring. In the first of these, locally vectored interrupts, an LSI interrupt controller supplies the response vector to the processor. This type of interrupt response does not require a global bus access for vectoring. In the second response mode, globally vectored interrupts, the responding processor accesses the system bus, placing the accepted vector level on the address bus, and the interrupting device places its response vector on the data bus. This mode is extremely useful when implementing newer LSI device controllers in interrupt driven systems. These chips have many internal conditions which may cause an interrupt, but without a globally-vectored response mode, the exact condition cannot be determined.

DISTRIBUTED PROCESSING SYSTEM ONE— IMPLEMENTATION OVERVIEW

The system (see Figure 1) is composed of up to 16 processing modules connected to a common, time-shared bus. The common bus is specified for a large (16-megabyte) address space, and both byte and word (16-bit) parallel transfers are specified such that both 8 and 16 bit masters and

slaves may co-exist in the same system. Bus Arbitration is performed in a parallel encoded format capable of very fast priority resolution (less than 100 nsec.) and proceeds on a single cycle basis unless a bus lock function is imposed by the processor currently in possession of the bus. Global memory units and global input/output devices may also be connected to the common bus.

Each processing module consists of a Distributed Processing Unit (DPU) which may contain an 8- or 16-bit CPU, a dual-port memory (DPM), and may be connected to a number of input/output devices. The dual-port local memory may be of any size up to 64 K bytes, and each processing module is assigned one 64 K page of total system memory. This memory unit, while normally accessed by its local processor, may be accessed by any processor in the system via the global bus, temporarily suspending the operation of the local processor. The memory unit is designed such that it is capable of both byte and word transfers, not only with respect to the global system bus, but also with respect to the local processor's bus. Note that this memory unit may be used as a two-port buffer between the global bus and a high speed data collection device (video frame grabber, etc.). Memory cycle times down to 70 nsec. may be attained.

Each processing module may be assigned its bus arbitration priority dynamically by the operating system. This is extremely important in real time systems, since any number of asynchronous events may alter the system-wide priority of running tasks. If priority were not dynamically allocated, tasks would have to be switched to another processing module, whose task may again be displaced, etc. It is estimated that the decrease in task switching overhead will improve the real time response of the system nearly an order of magnitude.

Eight levels of vectored interrupts are defined for the global bus, as well as two error interrupts, parity error and illegal memory reference. The interrupt processor may respond to all of these interrupts, as well as interrupts from local timers, or it may mask them and assign them to individual processing modules. Each processing module has eight levels of interrupts local to it. Two of these are assigned to local error conditions, an illegal system bus access and a local parity error. Three are assigned to interrupts from the system executive. Two are assigned to local peripheral service requests, and one may be software multiplexed to respond to one of the eight global interrupt vectors.

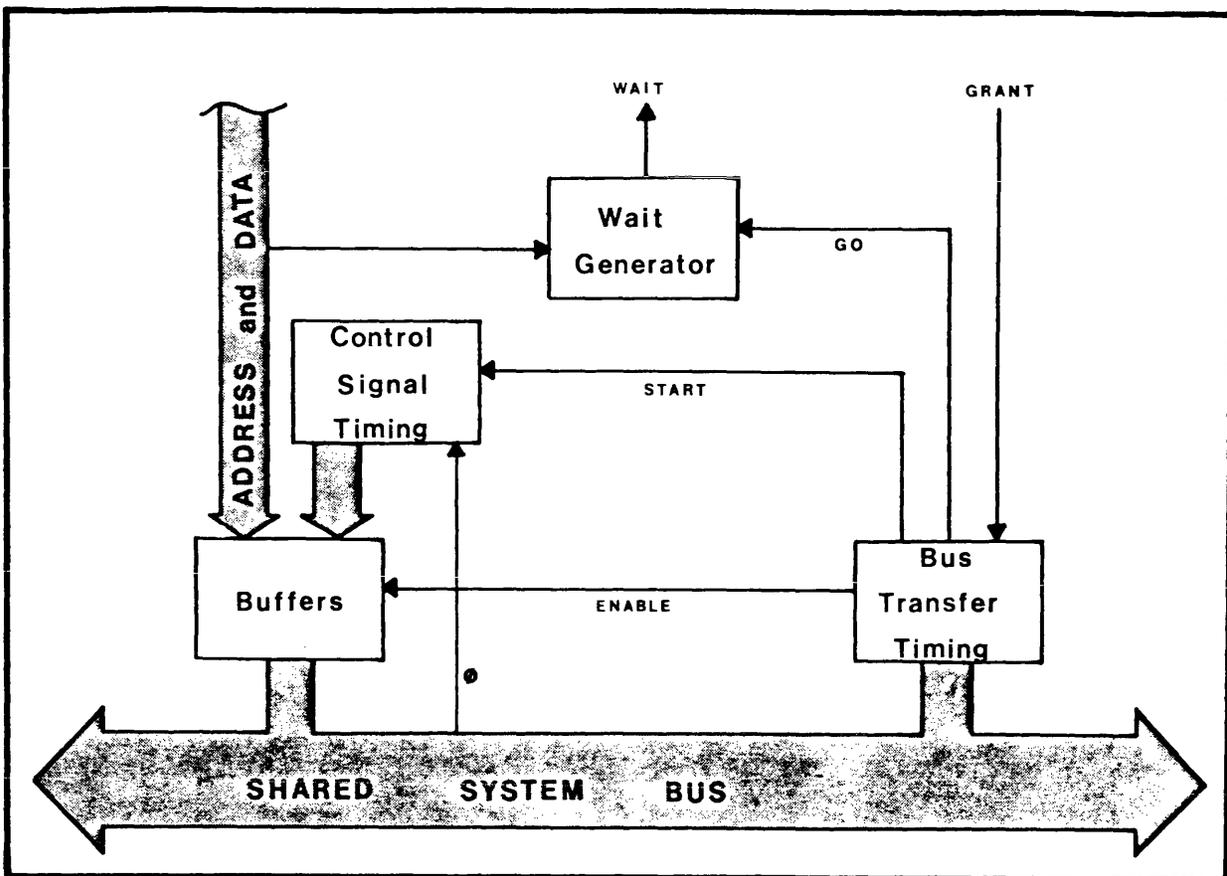


Figure 3—Bus cycle generator.

DPU/80

The distributed processing unit (see Figure 2) is based around the Zilog Z-80 microprocessor. Sockets for up to 8 K bytes of ROM are provided to hold an initialization procedure, a self-test/interface-test procedure, interrupt service routines, and a message communication kernel, for message transfers to the system executive. A power-on or reset jump to the first byte of the ROM is provided for system initialization. An LSI interrupt controller is dedicated to handling the local interrupts described above, and is interfaced directly to the local system bus. A control port, accessible via the system bus, is used to assign priority to the bus arbitrator and to gage system executive interrupts to the interrupt controller. A locking mechanism is provided such that either the global system bus or the dual-port memory can be locked, allowing implementation of indivisible test and set operations. Three elements in the design have been implemented in proprietary integrated circuits, these are: the bus arbitrator, the dual-port controller and the control signal timing generator.

Extended address management is accomplished for the Z-80's short (16-bit) address bus by dedicating a segment of local address space as a mapping area. This map is divided into two segments, and extension bit registers are provided for each segment, providing two bus access "windows." On the DPU/80 8K of local address space is dedicated as a map, giving two 4K bus windows. A local access to this area generates a bus request and places the local processor in a wait state. When the system bus becomes available, the bus arbitrator performs an arbitration with other requestors and if it has the highest priority, receives a bus grant. A bus transfer, under the control of the bus transfer timing unit (Figure 3) conducts the transfer of the bus to the processing module, starts the control signal timing, and releases the processor from the wait state. Note that the location of the mapping area, the location of the ROM and the location of the memory are all independently selectable, and that the bus address and the local address of the ROM are not related.

SYSTEM BUS SATURATION

The determination of total system throughput as a function of the number of processors in the system is of prime importance. System throughput is a function of the number of processors in the system, the throughput of each processor, and the amount of bus interference that exists with respect to contention for the common system bus. It is clear that saturation of the system bus will impose the upper limit on system bandwidth.

The maximum throughput of the system occurs when there is no contention for the shared system bus, and may be written

$$T = nP$$

where T is the total system throughput, n is the number of

processors in the system, and P is the throughput of each individual processor.

The amount of bus interference in the system is a function of the bus utilization requirements of the individual processors. To model this we must define a utilization parameter, b , as the fraction of available bus cycles required by each individual processor. If there were no local memory in the system, b would be very close to 1, and the bus would saturate very quickly. Experience with the Cm* multiprocessor² indicates that b ranges from .2 to .01 depending on the application, and averages about .1. Clearly the worst case for system throughput would be if all processors accessing the bus had to wait $(n-1)$ bus cycles before being granted access to the bus. This minimum throughput has been shown by Reyling⁴ to be:

$$T = \frac{nP}{1 + b(n-1)}$$

This relation is illustrated in Figure 4 for values of $b = .5$ and $b = .1$.

Average throughput is derived by considering all possible states, the probability of each, and the probability of a transition from one state to the next. Ravindran and Thomas⁵

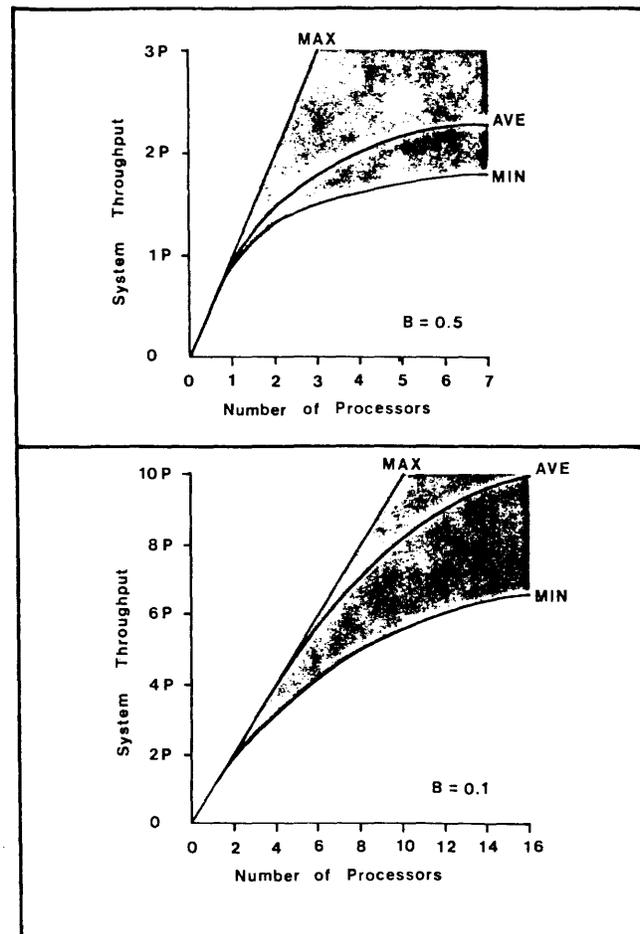


Figure 4—System throughput.

have shown this to be:

$$T_{ave} = P \sum_{j=1}^n j \sum_{i=1}^n p_i A(i, j)$$

where p_i is the probability of i processors requesting the bus, and $A(i, j)$ is the probability of a transition from state i to state j . This is also illustrated in Figure 4.

CONCLUSION

The implementation and architectural implications for the design of a cost-effective, microprocessor-based, modular multiprocessor, the DPS-1, have been discussed and illustrated. The design has been shown to be cost effective for both small and larger multiprocessor systems. A number of features have been included not found on other multiprocessors, such as dynamic priority allocation, wide address bus, co-existence of 8 and 16 bit masters in the same system, and a versatile, partially decentralized interrupt scheme. The system has excellent cost modularity, and may be configured for many levels of performance.

ACKNOWLEDGMENT

I would like to thank Jeff Moscow and Bill Hemsath, both of Cornell University, for their constant consulting on mul-

tiprocessor operating system design, and Steven Edelman, of InterSystems, for consulting on bus structures and systems design.

REFERENCES

1. Anderson, G. A., and E. D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," *Computing Surveys*, Vol. 7, No. 4, Dec. 1975.
2. Fuller et al., "Multi-microprocessors: an Overview and Working Example," *Proc. IEEE*, Vol. 66, No. 2, Feb. 1978.
3. Jaswa, R., "Designing Interrupt Structures for Multiprocessor Systems," *Computer Design*, Sept. 1978.
4. Reyling, G., "Performance and Control of Multiple Microprocessor Systems," *Computer Design*, March 1974.
5. Ravindran, V. K., and T. Thomas, "Characterization of Multiple Microprocessor Networks," *Digest 1973 IEEE Int. Conf.*
6. Enslow, P. H., "Multiprocessor Organization—A Survey," *Computing Surveys*, Vol. 9, No. 1, March 1977.
7. Lehman, M., "A Survey of Problems and Preliminary Results Concerning Parallel Processing and Parallel Processors," *Proc. IEEE*, Vol. 54, No. 12, Dec. 1966.
8. Hansen, P. B., "The Programming Language Concurrent Pascal," *IEEE Trans. Soft. Eng.*, Vol. SE-1, No. 2, June 1975.
9. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," *Comm. ACM*, Vol. 17, No. 10, Oct. 1974.
10. Baer, J. L., "A Survey of Some Theoretical Aspects of Multiprocessing," *Computing Surveys*, Vol. 5, No. 1, March 1973.
11. Gountanis, R. J., and N. L. Viss, "A Method of Processor Selection for Interrupt Handling in a Multiprocessor System," *Proc. IEEE*, Vol. 54, No. 12, Dec. 1966.

Work flow view of a distributed application

by J. R. HAMSTRA

Sperry Univac
Roseville, Minnesota

INTRODUCTION

Work Flow Management is a unified set of concepts for the definition, implementation and operation of Application Systems. A companion paper¹ to this one provides a more general treatment of the requirements motivating Work Flow Management. An Application System is a major function of the work of a computer system as perceived by the customer. Thus Application Systems often mirror the structure or work of the customer enterprise. In this paper we shall consider a credit card processing application, forming a major function of the work of the hypothetical Masterkey Credit Corporation (MCC). The portion of this application performed on the MCC distributed computer system is called the Credit-Cards Application System. MCC also uses its computer system for other applications, including personnel and payroll, financial information, and the development and maintenance of applications.

People relate to an Application System in various roles, including end user, systems analyst, programmer and operator/administrator. Figure 1 illustrates these relationships. Work Flow Management attempts to facilitate communication among these various parties including the Application System, by supporting high-level descriptions expressed in the Work Flow Definition Language.

Within an Application System certain kinds of work are performed repeatedly. This is called recurring work, as opposed to *ad hoc* work performed once. Recurring work, the primary concern of Work Flow Management, may be parallel (e.g. charge slips are processed concurrently in each of 10 regional data centers, and may indeed be processed concurrently within a single center) or cyclic (e.g., customers are billed monthly based in part on their previous statements) or both. Recurring work is more easily described by defining the underlying structure of the work, than by explicitly enumerating or generating the instances.

In general, a schema (pattern, diagram, schematic) provides a structured "template" of information for generating and controlling instances of complex entities. A Work Flow Schema is the description of an Application System expressed in the Work Flow Definition Language. It describes the structure of the recurring work of an Application System, with provision for the dynamic introduction of modifications and *ad hoc* work.

A Work Flow Schema is compiled into an internal form

interpreted by the run-time system. This produces, in effect, a customized applications executive providing a complete simulated environment, including facilities for production work, as well as application modification, testing and training, and auditing and recovery. Thus the Work Flow Definition Language can be classified as a simulation language, albeit a special purpose one since the classes of simulated entities are predetermined (see Figure 2).

In the remainder of this paper we will develop the basic Work Flow concepts and show how they are represented in the Work Flow Definition Language. Being an overview, this paper must omit some components, and the descriptions of those presented are necessarily simplified. Nevertheless, it is intended to give the reader some insight into the scope, style and power of both Work Flow Management and the Work Flow Definition Language.

WORK FLOW CONCEPTS

Functional distribution

As previously suggested, the work of the Credit-Cards Application System proceeds simultaneously in 10 regional data centers. These centers are logical processing environments perceived by Work Flow Management as Applications Environments called Regional-Data-Centers. The portion of the work of Credit-Cards performed within each Regional Data Center is called a Regional-Operations Sub-Application System.

Figure 4 illustrates these relationships, using the structure notation defined in Figure 3 and the symbols for Sub-Application System and Sub-Applications Environment shown in Figure 2. Note that the relationship between Regional-Data-Center and Regional Operations is transient, since the logical work of one Regional-Operations may be moved to a different actual Regional-Data-Center if some untoward event, such as a flood, renders the first Regional-Data-Center unavailable.

We have considered the first-level decomposition of the Credit-Cards Application System. At this level, it comprises similar functions (Regional-Operations) distributed among similar processing environments (Regional-Data-Center) in a pre-determined yet alterable manner. Work Flow Management encourages and supports the functional decomposition of Application Systems and the controlled distribution of the

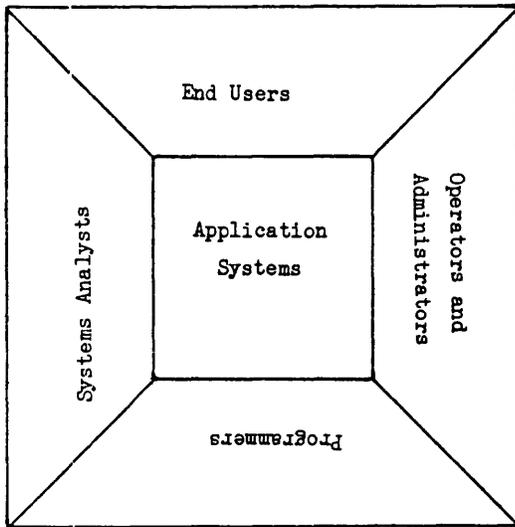


Figure 1—User roles and relationships.

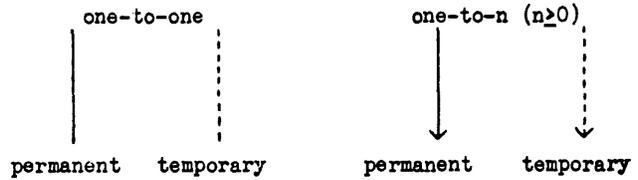


Figure 3—Schematic structure notation.

functional components. This contrasts with the emphasis on load-sharing or data base distribution found in many other approaches to distributed processing, although Work Flow Management does not preclude either of these. Indeed, it requires certain forms of data distribution. MCC distributes its work by dividing the United States into 10 regions and keying both credit card and merchant identification to these regions.

Considering the requirements of credit card processing within a Regional-Operations Sub-Application System, it is reasonable to decompose them into functions associated with the cardholders serviced by that region, called Cardholder-Operations, and functions associated with the merchants serviced by that region, called Merchant-Operations. MCC derives revenue from both Cardholder-Operations (membership fees and interest) and Merchant-Operations (service charges and discounts). Considering that each Regional-Operations is a separate profit center, while cardholders may use their cards anywhere in the country (MCC has no direct international operations), we conclude that management will desire, and accountants and auditors require, separate control of inter-regional financial transac-

tions. These functions are provided by an Inter-Regional-Operations Sub-Application System within each Regional-Operations. Figure 5 illustrates the structure of Regional Operations.

Although further decomposition of Credit-Cards into Sub-Application Systems is possible, it is not essential to this presentation and will not be pursued.

Data flow

Information processing is the work of transforming and communicating data. Availability of the data is both a necessary precondition and a major stimulus for performing this work. Thus information processing systems (organic and mechanical) are largely driven either implicitly or explicitly by data flow. Work Flow Management relies heavily on data flow to control the Application System. In the remainder of this section we consider the flow of charge information from the merchants to the cardholder accounts. This example will illustrate the key Work Flow concepts of Transaction Groups, Tasks and Queuing Points.

Merchants participating in the MCC system accumulate batches of charge slips. Each slip contains merchant and cardholder identification encoded in a suitable optical-character-recognition (OCR) font. A merchant will periodically deliver these batches to his MCC Regional-Data-Center, either directly or via his bank. The merchant is reimbursed for the total amount of these charges, less a computed discount. The amounts on each slip are OCR-encoded, then the batches are read into the computer system via an OCR-

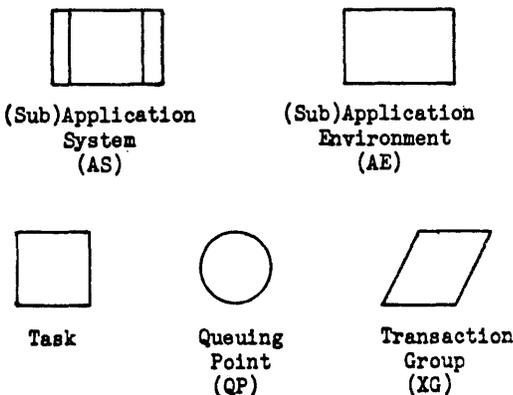
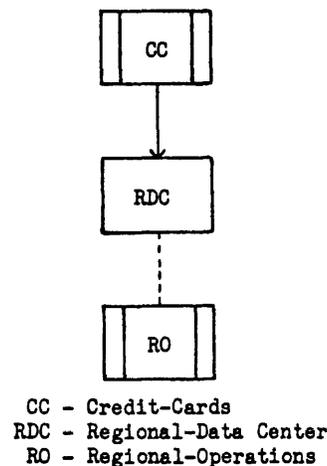


Figure 2—Schematic entity classes.



CC - Credit-Cards
 RDC - Regional-Data Center
 RO - Regional-Operations

Figure 4—Credit Cards structure.

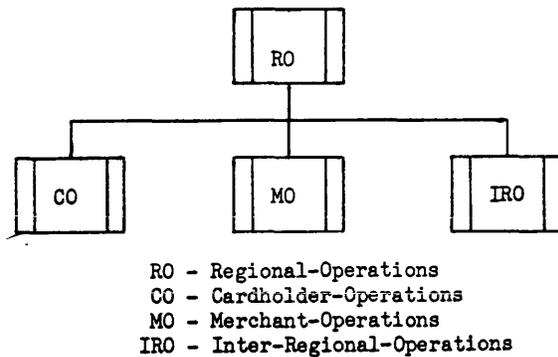


Figure 5—Regional-Operations structure.

reader. This is the first operation perceived by the Application System.

The internal form of a batch of charge slips is a Transaction Group called Sales-Inputs. A Transaction Group can be thought of as a bundle of transactions to be routed and processed together; however, its actual internal structure is considerably more complex than this, to satisfy the combined requirements of program data access and the Work Flow integrity/recovery architecture. A Transaction Group need not represent an external entity such as a batch of charge slips. Transaction Groups are the units of data flow within an Application System.

Sales-Inputs Transaction Groups are generated within the computer system by the operation of OCR-readers. Within a Merchant-Operations Sub-Application System this is represented by Optical-Character-Reader Tasks. A Task is a basic instance of work within an Application System. Tasks are the users of data, i.e., they produce, utilize and/or consume Transaction Groups. In addition to Optical-Character-Reader Tasks, the flow of Sales-Inputs Transaction Groups involves two other kinds of Tasks. The first is the crediting of the merchant accounts within Merchant-Operations, called Update-Merchant-Accounts Tasks; the second is the debiting of the cardholder accounts within Cardholder-Operations, called Update-Cardholder-Accounts Tasks.

Figure 6 illustrates the complete flow of Sales-Inputs Transaction Groups within the Application System, using the schematic structure notation. In addition, the arrows on the horizontal line show the direction of flow; more specifically they represent the sequence of transitions of the Sales-Inputs flow control state. After Update-Cardholder-Accounts the Sales-Inputs Transaction Groups are no longer necessary and they cease to be active entities within the Application System, although they are retained as archival entities for auditing and recovery purposes. We conclude this topic by noting that a Work Flow Schema contains a complete producer/consumer model of data flow within the Application System.

Commitment control

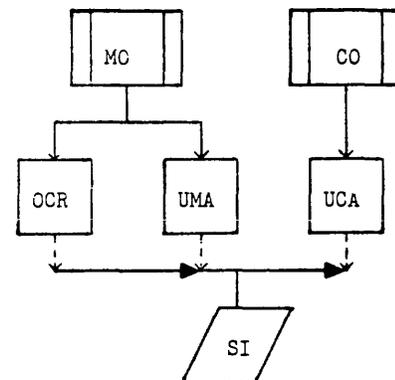
The foregoing treatment of the flow of charge information would be satisfactory were cardholders not permitted to make purchases outside their home regions. The absence of

this restriction poses complications not readily resolved even if Update-Cardholder-Accounts has access to a global cardholder data base distributed among the ten Regional-Data-Centers. First, all Regional-Data-Centers may not always be available, a fact that should not hinder local processing at other centers and one that we wish not to expose to the individual Tasks running elsewhere. Second, each Regional-Operations is a separate profit center requiring a separate reckoning of inter-regional financial transactions, which would be hidden in a single distributed data base. Third, we do not wish to submit transactions to a remote region until we have some degree of confidence in the results of the processing that produced them, nor do we wish to post transactions from a remote region without similar control. We will now address these problems.

The first part of the support for remote purchases is the introduction of Remote-Purchases Transaction Groups to effect the flow of this data among the various Regional-Operations as shown in Figure 7. Each Update-Cardholder-Accounts Task can optionally produce one Remote-Purchases Transaction Group for each of the nine Regional-Operations other than its own. This is called data flow fan-out. A new kind of Task called Remote-Cardholder-Updates accepts Remote-Purchases Transaction Groups from one or more other Regional-Operations and debits the local cardholder accounts accordingly. This is called data flow fan-in. Figure 7b illustrates this arrangement for three Regional-Operations.

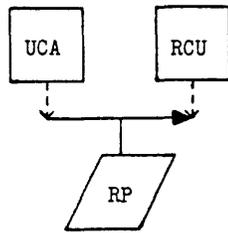
We now consider the support for inter-regional financial accounting of Remote-Purchases Transaction Groups. This consists of two kinds of Tasks within the Inter-Regional-Operations Sub-Application Systems, Outbound-Remote-Balancing to post outgoing Transaction Groups and Inbound-Remote-Balancing to post incoming Transaction Groups. Figure 7c illustrates the general flow of Remote-Purchases Transaction Groups.

Our treatment has regarded data flow as essentially par-

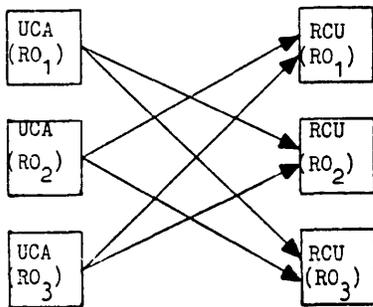


MO - Merchant-Operations
 CO - Cardholder-Operations
 OCR - Optical-Character-Reader
 UMA - Update-Merchant Accounts
 UCA - Update-Cardholder-Accounts
 SI - Sales-Inputs

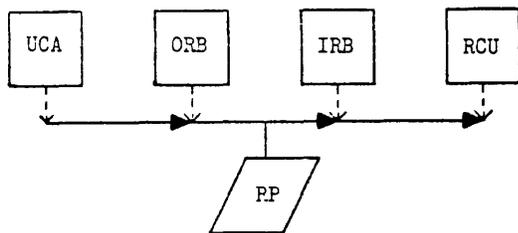
Figure 6—Flow of Sales-Inputs.



a.) Simplified Structure



b.) Specific Example



c.) General Structure

- UCA - Update-Cardholder-Accounts
- RCU - Remote-Cardholder-Updates
- RP - Remote-Purchases
- RC - Regional-Operations
- ORB - Outbound-Remote-Balancing
- IRB - Inbound-Remote-Balancing

Figure 7—Flow of Remote-Purchases.

allel and asynchronous. However, application integrity and recovery control require that certain phases in processing be synchronized. This requirement was first explicitly applied to the flow of Remote-Purchases Transaction Groups among Regional-Operations, but we have in fact relied on it throughout this presentation. Furthermore, the fact that work will proceed at different rates and with different levels of actual concurrency in different parts of the system and at different times imposes additional requirements on the synchronization of data flow. These requirements are subsumed under the general notion of controlled commitment.

Commitment (consigning, binding over for use) occurs whenever data produced by one Task is made available for use by other Tasks. This may occur when data in a shared data base is unlocked; however commitment in this manner is not very amenable to higher-level control, being neces-

sarily synchronized to the internal operations of the Task. Transaction Groups make the flow of data between Tasks explicit. This explicit flow can best be utilized for Application control if an external agency is interposed between the relinquishing of control of a Transaction Group by one Task and the acquisition of control by another. This agency is a Queuing Point.

Queuing Points are mail boxes for Transaction Groups. They serve as brokers to acquire and dispose of Transaction Groups for Tasks, the actual users of data. Thus they serve to decouple individual Tasks from what, when and where other Tasks exist. The availability of data at a Queuing Point may be the stimulus for scheduling work, or the data may be held at Queuing Points pending some other stimulus such as time or administrator action.

We can satisfy the remaining requirements for control of the flow of Remote-Purchases Transaction Groups with appropriate Queuing Points. Within the Inter-Regional-Operations Sub-Application System of each Regional-Operations Sub-Application System we establish one Remote-Region Queuing Point for each other Regional-Operations. The Remote-Region Queuing Points accumulate Transaction Groups bound for the other Regional-Operations. Upon a suitable administrator command, Outbound-Remote-Balancing removes each Transaction Group from the selected Remote-Region Queuing Point, posts appropriate information to the Inter-Regional-Operations database, and forwards the Transaction Group to Inter-Regional-Operations within the remote Regional-Operations.

Each Inter-Regional-Operations accumulates incoming Transaction Groups at a single Inter-Regional-Transactions Queuing Point. Upon a suitable administrator command, Inbound-Remote-Balancing removes each Transaction Group from the Inter-Regional-Transactions Queuing Point, posts appropriate information to the Inter-Regional-Operations data base, and forwards the Transaction Group to the appropriate place within this Regional-Operations. In the case of Remote-Purchases Transaction Groups, this is Remote-Cardholder-Updates.

Figure 8 illustrates the complete flow of charge information into the appropriate accounts, applying the principle that all Tasks are decoupled by Queuing Points. This figure shows the power of the schematic definition concept. It shows the flow of work through the Application System in an inherently parallel manner, yet it admits of the necessary synchronization and control. The Queuing Points provide for the unification of a network communications model (data flow) with a hierarchical processing model (data transformation).

The names of the entities in Figure 8 are actually names of types of entities within the classes of entities denoted by the shapes in Figure 2. At any given time there exist multiple occurrences of entities of each named type, e.g., multiple Tasks of type Update-Cardholder-Accounts and multiple Queuing Points of type Remote-Region. Note that there might even exist multiple Application Systems of type Credit-Cards, e.g. for testing within MCC, or because a software house has applied this application to companies other than MCC. Much of the power of the Work Flow Definition Language derives from its ability to define com-

plex application structures in terms of the underlying types of entities, while associating enough information with these named types to adequately control the actual occurrences. In the next section we will show how this is done.

WORK FLOW DEFINITION LANGUAGE

Global structure

We have presented the structure of an Application System as a schematic diagram, and we will now examine its expression in the Work Flow Definition Language. The following prose description of the language is presented to explain the appendices, which are intended to convey a better understanding of the language. Appendix A summarizes the conceptual structure of the Work Flow Definition Language. The language is basically block-structured with recursive nesting of the higher-level constructs, i.e., Sub-Application Systems and Sub-Applications Environments. The representation is free-form with punctuation and indentation optional.

(Sub)-Application Systems comprise application-blocks delimited by terminal constructs. Application-blocks describe nested spheres of control of work within an Application System. Application-blocks may contain, in addition to nested higher-level constructs, the declarations of types of lower-level entities of the categories internal, external and environmental. These will be described further in subsequent sections.

(Sub)-Applications Environments comprise environment-blocks delimited by terminal constructs. Environment-blocks describe processing environments which do not directly perform any work, although they may contain Sub-Application Systems which do perform work. Environment-blocks may contain a subset of the entities contained in application-blocks, internal and external entities being excluded.

Appendix B is a sample Work Flow Schema for the portion of the Credit-Cards Application System described in the second section. While this schema conforms to the structure shown in Appendix A, it contains assertions more detailed than shown in Appendix A. It is the assertions appearing as rules or policies associated with the structural entities, that

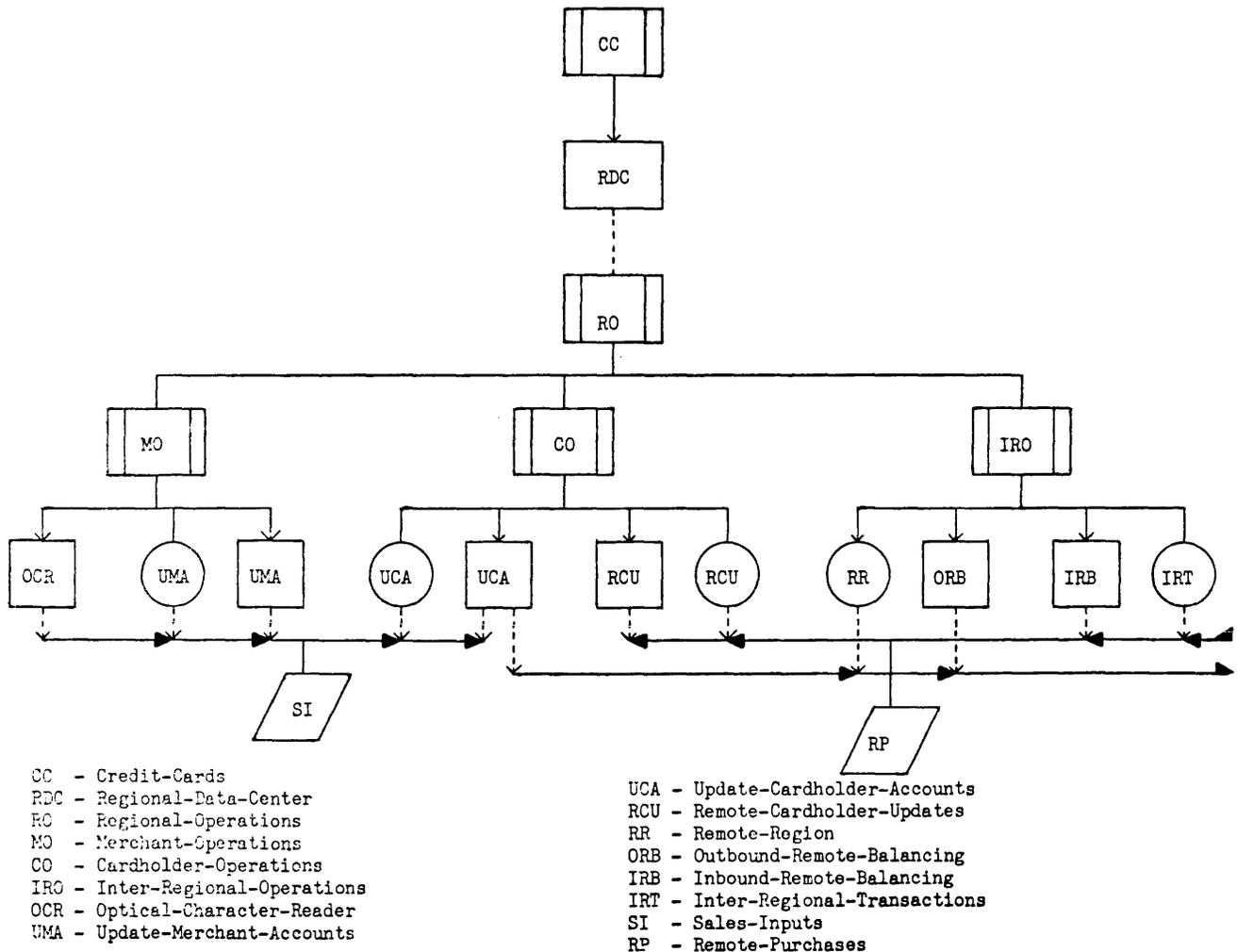


Figure 8—Complete flow of charge information.

give much of the meaning to the schema. These assertions apply generally or by default within the scope of the declarations containing them. Many more kinds of assertions can be made than are illustrated in the sample schema; however, the sample should illustrate the concept.

Internal entities

The classes of internal entity types in a Work Flow Schema are Transaction Groups, Application Modules, Queuing Points, Clocks and Calendars. These define the work performed internal to, and under complete control of, the Application System.

As previously stated, Transaction Groups have extensive internal structure, although this has been deliberately omitted from the example in the interest of clarity. This structure may be partly described in the Work Flow Schema, but is usually completely defined in a data schema. An extensive repertoire of control functions exists for Transaction Groups, and a few examples are given. Initiate creates a Transaction Group and attaches it to a Task. Terminate is the inverse of Initiate. Export transfers control of a Transaction Group from a Task to a Queuing Point. Import is the inverse of Export. Pass is an Import/process/Export sequence.

An Application Module is a program and related control information defining a type of Task. Tasks may be scheduled explicitly, or implicitly on data arrival. Assertions associated with the Application Module define the local data environment for the Task. These assertions may also generate a Queuing Point type of the same name.

Queuing Points are usually generated from Application Module declarations; however, they may be explicitly declared when specific routing and control functions are desired. These functions include, e.g., Hold, which inhibits the Import of data from the held Queuing Point, and Release, which is the inverse of Hold. Normally one occurrence of the named type of Queuing Point will be generated for each occurrence of the containing or generating type of entity, e.g., Application Module, Sub-Application System or Terminal Group. The occurs-clause in the Remote-Region declaration specifically controls the generation of occurrences of Remote-Region Queuing Points.

Multiple named Clocks and Calendars may exist within a Sub-Application System. Clocks and Calendars are entirely synthetic. While they may represent real-world time, they may, on the other hand, represent nothing more than logical state in some abstract event space. Further discussion of Clocks and Calendars is beyond the scope of this paper.

External entities

The classes of external entity types in a Work Flow Schema are Terminal Groups, User Groups, Conversations and Workstations. These define the work performed at the interfaces to, and under partial control of, the Application System. Terminal Groups are types of external interfaces to an Application System. They manifest themselves as types

of Tasks within the Application System. Queuing Points may also be generated for them. The information provided in the Work Flow Schema, in conjunction with communications configuration information, can be thought of as controlling a "daemon module" which defines the Task. These functions are provided by the implementor of the computer system, since it is generally unrealistic and undesirable to expect the customer to implement them.

User Groups and Conversations represent, respectively, types of external users of, and types of their interactions with, the Application System. External users may actually be organic (human), mechanical, or other Sub-Application Systems. They manifest themselves as types of Tasks and Queuing Points within the Application System, controlled as with Terminal Groups. Conversations manifest themselves as Transaction Group types within the Application System. Further discussion of User Groups and Conversations is beyond the scope of this paper.

Workstations represent locations for tracking data flow external to the computer system, and have utility primarily for external production control. Further discussion of Workstations is beyond the scope of this paper.

Environmental entities

The classes of environmental entity types in a Work Flow Schema are Data Bases, Scheduling Classes, Resource Budgets and Control Points. These exist primarily to resolve the mapping of the work of an Application System onto the supporting processing environments. Thus the actual nature of the named entity types is largely resolved beyond the scope of the schema.

Data Bases are identical to Transaction Groups except that their existence is controlled external to the Application System. The possibility that they may be shared with other (Sub)-Application Systems is also presumed. The internal structure of Data Bases is identical to that of Transaction Groups.

Scheduling Classes and Resource Budgets represent, respectively, the quality of service (e.g., priority, response time) and the quantity of service (e.g., resource consumption) provided for various units of work within an Application System. Further discussion of Scheduling Classes and Resource Budgets is beyond the scope of this paper.

Control Points are the means of exchanging Application System control information among interested parties. Further discussion of Control Points is beyond the scope of this paper.

CONCLUSION

Summary

In this paper we have introduced the general notions of schemas and Application Systems. We have presented the basic concepts of Work Flow Management, including functional distribution, data flow and commitment control. We have shown how they apply to a hypothetical credit card

processing application. Then we have presented the Work Flow Definition Language, considering the global structure of the language and its external, internal and environmental component entities. Finally, we have described a portion of the credit card processing application in this language, illustrating many of these constructs.

Any introductory presentation of a subject must omit certain components. Chief among those omitted here are:

1. Overall application control, including events, conditions, Clocks, Calendars and Work Flow Procedures.

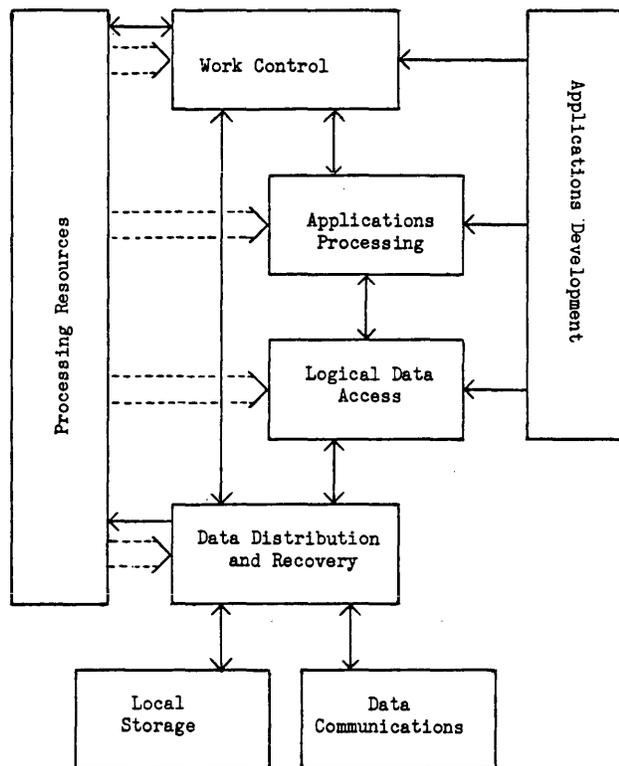


Figure 9—Overall system structure.

2. The administrators' and programmers' views of the system, inssofar as they extend beyond that of the systems analysts as reflected in the schema.
3. The end users' view of the system, including interactive conversation control and display management.
4. Production control, including work performed according to a schedule rather than on demand, and the correlation of internal data flow with external data flow.
5. Integrity control, including auditing, security and recovery.
6. Application development, testing and training.

Figure 9 illustrates the overall structure of a running Work Flow Management system and its relationship to other parts of a computer system. While not all components shown are part of Work Flow Management, they are all essential to a useful computer system. Work Flow Management brings these components together to provide a unified set of tools for the support of customer applications.

ACKNOWLEDGMENTS

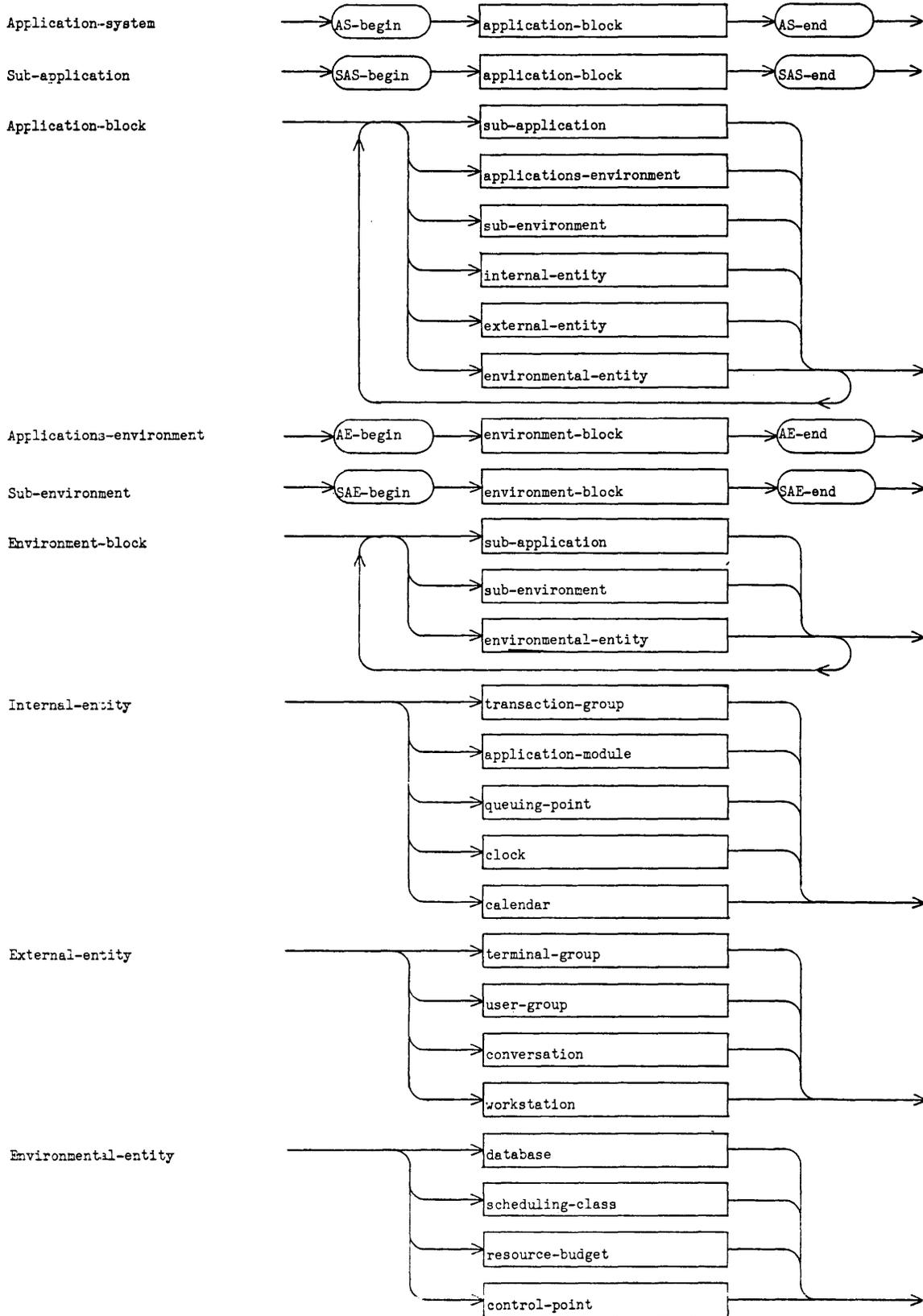
Many people have contributed to the ideas presented here, including the current members of the Work Flow group—Marv Beriss, Nigel Dolby, Tony Jenkins and Frank Streine. I am particularly indebted to Neville Black, who first proposed the idea of Work Flow Schemas, and to John Blasdale, who stated the requirements of credit card processing as a sample problem.

The views expressed in this paper are those of the author, and not necessarily those of Sperry Univac.

REFERENCE

1. Jenkins, A. P., "Commercial Data Processing for the 80s—An Integrated Approach," *1978 Fall Technical Symposium*, Sperry Univac, Session 3-3.

APPENDIX A. Work Flow Definition Language - Conceptual Structure



APPENDIX B

``This is a sample Work Flow Schema for the Credit Card scenario, as presented by J. R. Hamstra at the
 ``National Computer Conference on 6 June 1979.

``

``The following conventions are used in this preparation:

``

- `` 1. Key words are written entirely in capital letters.
- `` 2. Name words are written with their first letters capitalized.
- `` 3. Hyphenated words are denoted by -.
- `` 4. Compound names are separated by . .
- `` 5. Comments are delimited by `` ``; end of line also terminates comments.
- `` 6. Optional constructs are delimited by [].
- `` 7. Any other use of punctuation is optional.

APPLICATION [SYSTEM] Credit-Cards

[APPLICATIONS] ENVIRONMENT Regional-Data-Center

`` Each Regional-Data-Center is controlled external to the Application System.

`` The names enumerated here are externally resolved references.

OCCURS IN New-York,

Philadelphia,

Atlanta,

Chicago,

St-Louis,

Dallas,

Denver,

Los-Angeles,

San-Francisco,

Seattle

SUB-APPLICATION [SYSTEM] Regional-Operations

DATABASE Cardholder-Information END

DATABASE Merchant-Information END

DATABASE Inter-Regional-Information END

`` Databases are distinguished from Transaction Groups only by the fact that

`` their existence is controlled external to the Application System. Their

`` names are externally resolved references.

TRANSACTION GROUP Sales-Inputs END

TRANSACTION GROUP Remote-Purchases END

`` etc

SUB-APPLICATION [SYSTEM] Cardholder-Operations

USE Cardholder-Information

SUB-APPLICATION [SYSTEM] Cardholder-Services

`` etc

END [Cardholder-Services]

SUB-APPLICATION [SYSTEM] Credit-Management

`` etc

END [Credit-Management]

SUB-APPLICATION [SYSTEM] Cardholder-Accounts

[APPLICATION] MODULE Update-Cardholder-Accounts

`` These declarations will generate an implicit Update-Cardholder-Accounts

`` Queuing Point, in addition to the Application Module.

IMPORT AND TERMINATE EACH Sales-Inputs

INITIATE AND EXPORT OPTIONAL Remote-Purchases TO EACH Remote-Region

`` Inter-Regional-Operations contains the Remote-Region Queuing Points.

END [Update-Cardholder-Accounts]

[APPLICATION] MODULE Remote-Cardholder-Updates

`` These declarations will generate an implicit Remote-Cardholder-Updates

`` Queuing Point, in addition to the Application Module.

IMPORT AND TERMINATE EACH Remote-Purchases

APPENDIX B

```

    END [Remote-Cardholder-Updates]
    .. etc
END [Cardholder-Accounts]
END [Cardholder-Operations]
SUB-APPLICATION [SYSTEM] Merchant-Operations
    USE Merchant-Information
    SUB-APPLICATION [SYSTEM] Merchant-Services
    .. etc
END [Merchant-Services]
SUB-APPLICATION [SYSTEM] Merchant-Accounts
    TERMINAL GROUP Optical-Character-Reader
        INITIATE AND EXPORT Sales-Inputs TO Update-Merchant-Accounts
    END [Optical-Character-Reader]
    [APPLICATION] MODULE Update-Merchant-Accounts
        .. These declarations will generate an implicit Update-Merchant-Accounts
        .. Queuing Point, in addition to the Application Module.
        PASS EACH Sales-Inputs TO Update-Cardholder-Accounts
    END [Update-Merchant-Accounts]
    .. etc
END [Merchant-Accounts]
END [Merchant-Operations]
SUB-APPLICATION [SYSTEM] Inter-Regional-Operations
    USE Inter-Regional-Information
    ON INITIATE HOLD Inter-Regional-Transactions AND EACH Remote-Region
    QUEUING POINT Remote-Region
        OCCURS IN CURRENT Regional-Operations FOR EACH OTHER Regional-Operations
        ON RELEASE SCHEDULE Outbound-Remote-Balancing
    END [Remote-Region]
    [APPLICATION] MODULE Outbound-Remote-Balancing
        PASS EACH Remote-Purchases FROM Remote-Region
        TO Inter-Regional-Transactions,
        THEN HOLD Remote-Region
    END [Outbound-Remote-Balancing]
    QUEUING POINT Inter-Regional-Transactions
        ON RELEASE SCHEDULE Inbound-Remote-Balancing
    END [Inter-Regional-Transactions]
    [APPLICATION] MODULE Inbound-Remote-Balancing
        PASS EACH Remote-Purchases FROM Inter-Regional-Transactions
        TO Remote-Cardholder-Updates,
        THEN HOLD Inter-Regional-Transactions
    END [Inbound-Remote-Balancing]
    .. etc
END [Inter-Regional-Operations]
END [Regional-Operations]
END [Regional-Data-Center]
END [Credit-Cards]

```

The use of self-inverse program primitives in system evaluation

by JOHN E. MACDONALD, JR.
International Business Machines Corporation
 Poughkeepsie, New York

INTRODUCTION

Performance comparisons among two or more central processors of a computer system usually involve the execution of special, rather arbitrary test programs called benchmarks. A fundamental difficulty arises when more than one such benchmark is used in the comparison process, namely the absence of any common measure of the power or complexity of the contending benchmarks.

This paper defines such a common measure and shows how to use it. Essentially, we place additional conditions on the benchmark program which force it into the realm of information theory and allow us to apply the theoretical arsenal of results due to Shannon¹ and those who followed in his footsteps.

Our approach is different from that of Hellerman.² Briefly, Hellerman defines the work of a computational process as the information content, in the sense of Shannon, of the table-lookup implementation of the process. In making his definition, Hellerman further assumes that all possible inputs to the process have the same relative frequency. A critique of Hellerman's approach and Hellerman's response are found in References 3 and 4, respectively. Other efforts to assign a work and/or complexity measure to computations have been made by Savage⁵ and Johnson.⁶

Shannon's measure of the information content of a source depends only on the relative frequencies of the letters used by the source. In the simplest case, namely the zero-memory case, these relative frequencies are independent of any past output of the source. For a source alphabet of N letters we then have

$$H_2 = - \sum_{i=1}^N P_i \times \log_2 P_i \text{ (bits/letter)} \quad (1)$$

where P_i is the relative frequency of the i th letter, and H_2 is the average information content of the source. The subscript on H corresponds to the choice of the base of the logarithm function. Base 2 is by far the most popular choice, but any base would do. Note that a source with exactly two letters each used with relative frequency 0.5 would yield a value of 1.0 for H_2 . It is convenient to think of this as a standard source. The value of H_2 has a specific implication for the possible one-to-one coding of strings of source letters

into strings of standard binary symbols. Consider the following example.

Example 1—A source has two letters with

$$\begin{aligned} Pr(A) &= 0.9 \\ Pr(B) &= 0.1 \end{aligned}$$

Then

$$\begin{aligned} H_2 &= -(0.9 \times \log_2 0.9) - (0.1 \times \log_2 0.1) \\ &= 0.469 \text{ bits/source letter} \end{aligned}$$

Huffman⁷ has given an explicit method of encoding sequences of S source letters one-to-one into variable length sequences of binary symbols in such a way that the average number of binary symbols per source letter approaches H_2 as S increases. Huffman's technique yields in Example 1 the following:

S	Average Number of Binary Symbols per Source Letter
1	1.000
2	0.645
3	0.533

The actual codes produced by Huffman's method are given in the Appendix.

Example 2—A source has 3 letters with

$$\begin{aligned} Pr(A) &= 1/3 \\ Pr(B) &= 1/3 \\ Pr(C) &= 1/3 \\ H_2 &= -1/3 \times \log_2 1/3 - 1/3 \times \log_2 1/3 - 1/3 \times \log_2 1/3 \\ &= \log_2 3 \\ &= 1.586 \end{aligned}$$

S	Average Number of Binary Symbols per Source Letter
1	1.667
2	1.611
3	1.605
4	1.605
5	1.589

It is convenient to think of a standard channel as a device capable of transmitting one symbol per second when transmitting sequences generated by the previously defined standard source. The mappings from sequences of source letters into sequences of symbols for the standard binary channel is then one-to-one and hence invertible. To illustrate, consider the Huffman code for Example 1 with S equal to 3.

Source Sequence	Channel Sequence
AAA	0
AAB	100
ABA	101
BAA	110
ABB	11100
BAB	11101
BBA	11110
BBB	11111

The invertibility of such codes is guaranteed by the prefix property. The prefix property guarantees that no complete channel code is also the front end of another complete channel code when the code is read from left to right.

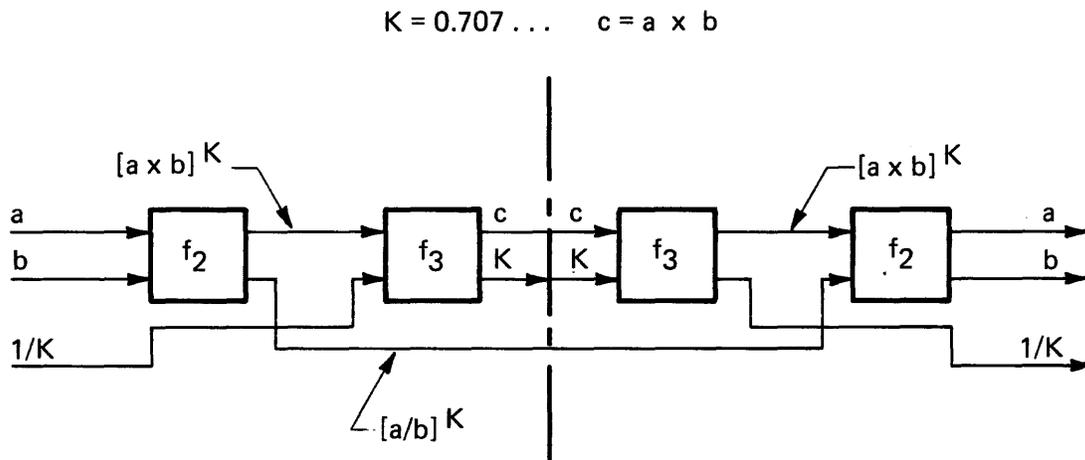
Consider a tandem arrangement of source, source-sequence-to-channel-sequence encoder, channel, channel-sequence-to-source-sequence decoder, and sink. Suppose we had a long sequence of source symbols from Example 1 to transmit to a remote point and a binary channel with a capability of 10 standard channel symbols per second. If we choose S equal to 3, we can say that we are transmitting information at a rate

$$\frac{10 \text{ (standard binary symbols)/second}}{0.533 \text{ (standard binary symbols)/source letter}} = 18.76 \text{ (source letters)/second}$$

We have tacitly assumed that any delay due to encoding and decoding has been made negligible through some buffering scheme.

If we want a theoretical upper bound on source letter transmission rate in the above calculation, we would substitute H_2 for 0.533, obtaining

$$\frac{10 \text{ (standard binary symbols)/second}}{0.469 \text{ (standard binary symbols)/source letter}} = 21.32 \text{ (source letters)/second}$$



Storage Contents

<u>Initial</u>	<u>After Step 1</u>	<u>After Step 2</u>	<u>After Step 3</u>	<u>After Step 4</u>
a	$[axb]^K$	c	$[axb]^K$	a
b	$[a/b]^K$	$[a/b]^K$	$[a/b]^K$	b
1/K	1/K	K	1/K	1/K

Figure 1—Network for Example 4.

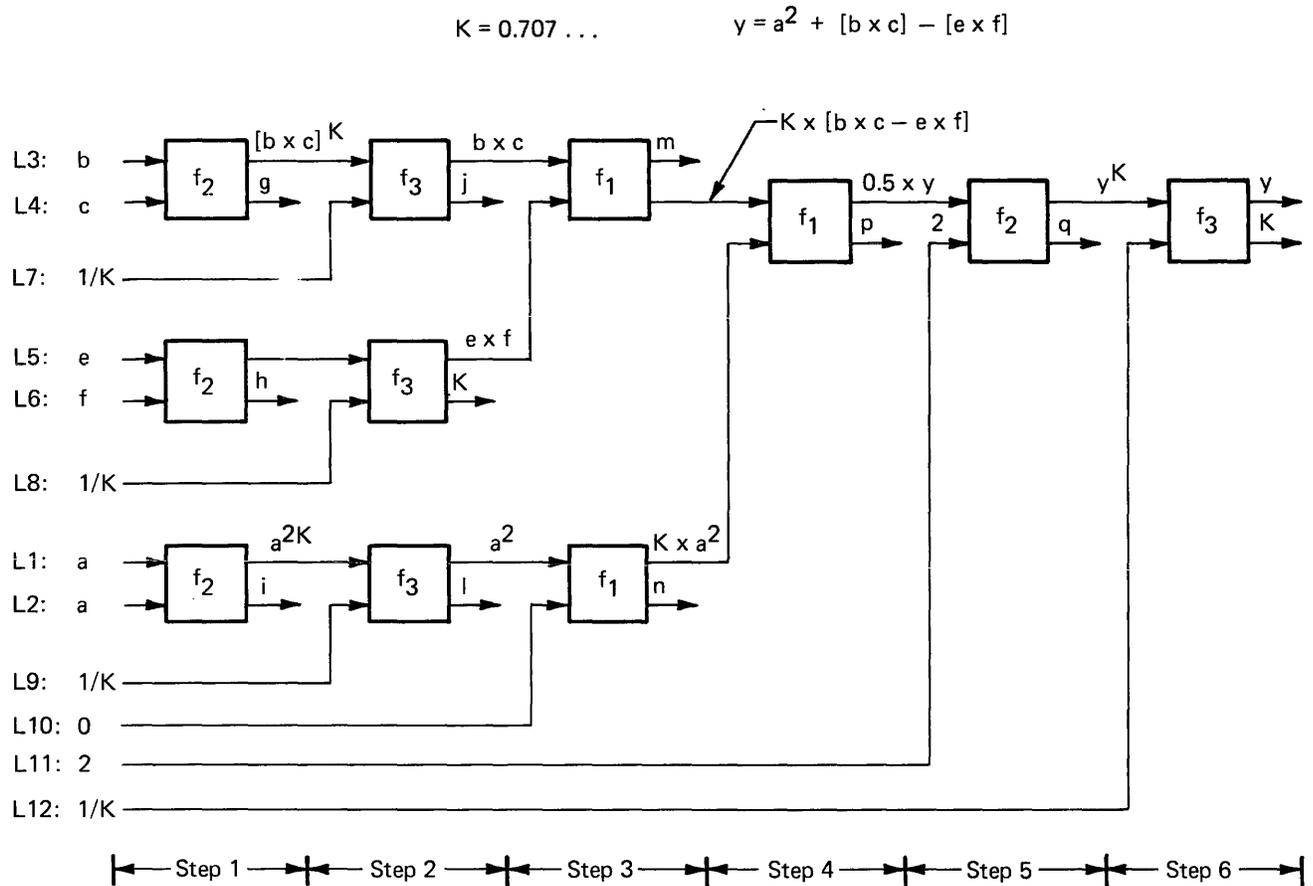


Figure 2—Direct network for Example 5.

Conversely, suppose we had access only to the source and sink. It would be possible first of all to compute H_2 for the source by means of Equation 1. Then, for any given implementation or implementations we could measure the average time to transmit source letters from source to sink. The reciprocal of this figure is the average transmission rate in source letters per second. If we then assume an internal mechanism in the implementation which produces ideal encoding (very large S), we can compute the internal channel rate as a figure of merit for a particular source and a particular implementation. It is very important to recognize that changing the source or the implementation or both will in general change the figure of merit value. This is crucial to our soon-to-be-made assertion that we can compare different benchmarks on the same computer system and/or the same benchmark on different computer systems and/or different benchmarks on different computer systems.

Thus, if we compute the H_2 for Example 1 as 0.469 and then measure an implementation for transmitting this alphabet at 21.32 source letters per second, the figure of merit would be

$$0.469 \times 21.32 = 10 \text{ (standard binary symbols)/second} \\ = 10 \text{ bits/second}$$

CREATION OF INVERTIBLE BENCHMARKS

A key requirement for the method described in the introduction is that the end-to-end system implement a one-to-one mapping from source letters into sink letters. Usually this mapping is the identity function in data transmission applications. When we attempt to apply the idea directly to a data processing benchmark, an unexpected difficulty presents itself. Namely, data processing is almost never a one-to-one mapping. Consider a two-input modulo 3 adder as a simple example.

Example 3— $c = (a + b)_{\text{mod } 3}$
c:

b	0	1	2	$H_2(a, b) = - \sum_{a=0}^2 \sum_{b=0}^2 p_{a,b} \times \log_2 p_{a,b}$
a	0	1	2	
0	0	1	2	
1	1	2	0	
2	2	0	1	

$$H_2(c) = -(p_{0,0} + p_{2,1} + p_{1,2}) \times \log_2 (p_{0,0} + p_{2,1} + p_{1,2}) \\ - (p_{1,0} + p_{0,1} + p_{2,2}) \times \log_2 (p_{1,0} + p_{0,1} + p_{2,2}) \\ - (p_{2,0} + p_{1,1} + p_{0,2}) \times \log_2 (p_{2,0} + p_{1,1} + p_{0,2})$$

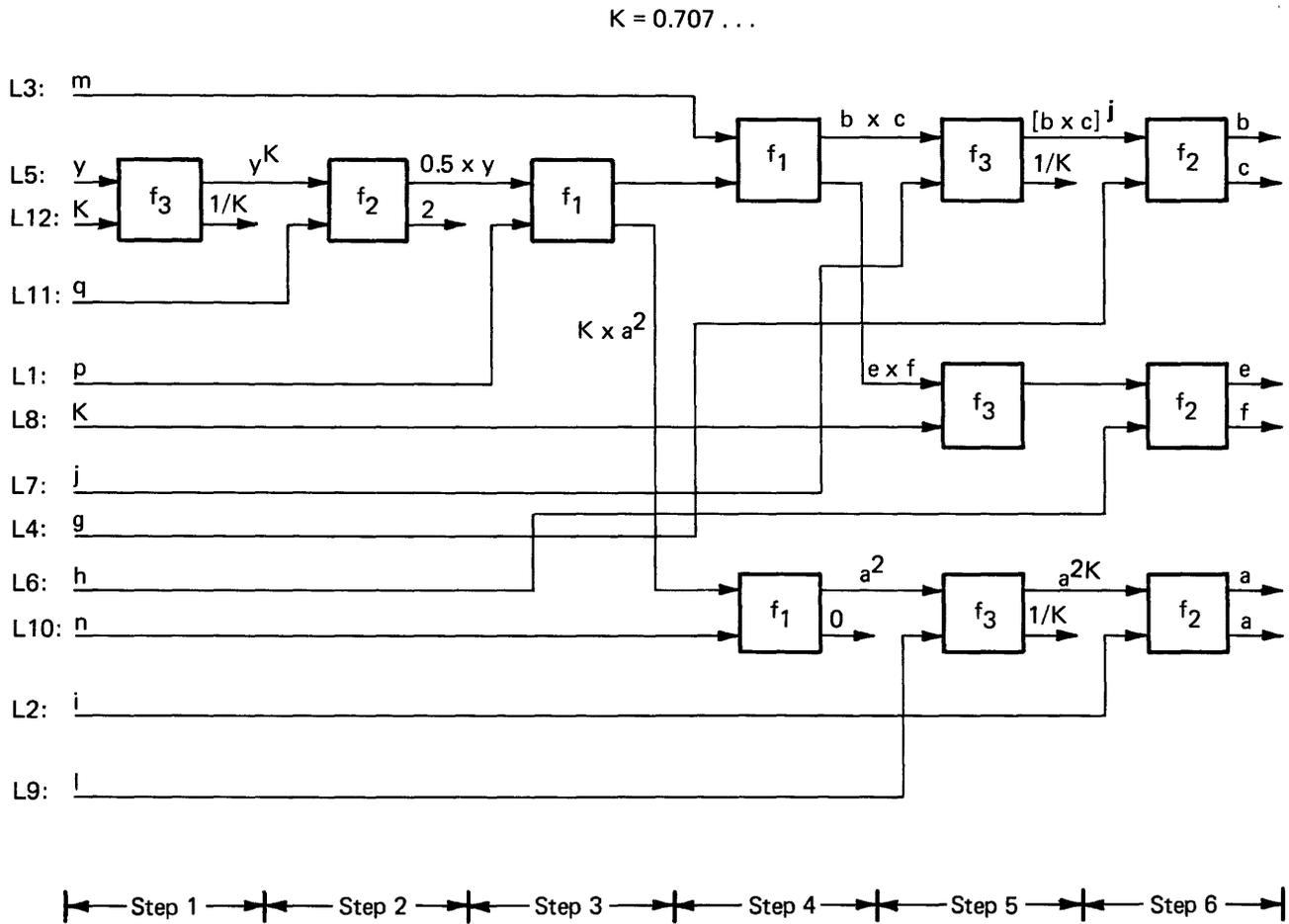


Figure 3—Inverse network for Example 5.

It is easy to show (see Appendix) that

$$-(p_x + p_y) \times \log(p_x + p_y) < -(p_x \times \log p_x) - (p_y \times \log p_y)$$

It follows that

$$H_2(c) < H_2(a, b)$$

We have thus illustrated a situation which has some of the elements of a paradox. To wit, data processing usually reduces information content. In the sense of Shannon, this is actually true. What is missing is the notion that the knowledge of $(a+b)$ is more valuable than the knowledge of (a, b) . Shannon's theory contains no concept of the value of information.

The seeming paradox is resolved if we insist that $(a+b)$ be computed in such a way that (a, b) can be recovered from $(a+b)$ plus auxiliary information. Then the entire end-to-end operation will be the identity operation and hence one-to-one, while at the same time the process produces the originally desired result $(a+b)$ as a sort of intermediate output. We know that (a, b) can be computed from $(a+b, a)$ or

$(a+b, b)$ or $(a+b, a-b)$ or endless other possibilities. Our choice is a set of primitive invertible programs (PIPs) which implement addition, subtraction, multiplication, division, exponentiation, and reciprocation.

Ordinarily, it would be a very difficult task to implement a benchmark in such a way as to guarantee invertibility while simultaneously computing a desired output. This task is rendered trivial by designing the PIPs in such a way that they are self-inverse. More specifically, each PIP has two inputs and two outputs, and implements

$$f(f(a, b)) = (a, b) \tag{2}$$

Thus any network of PIPs, however complex, has a simple mirror-image as its inverse network.

An obvious response to our requirements in the example above is to begin with (a, b) and produce $(a, b, a+b)$. We have rejected this approach because the required storage grows with the complexity of the computation. The PIPs which we present in the next section require a constant amount of storage during the entire benchmark computation.

SELF-INVERSE PRIMITIVE INVERTIBLE PROGRAMS

In all that follows, let K represent the reciprocal of the square root of 2. That is,

$$K=0.707\dots \tag{3}$$

The complete set of primitive invertible programs is given by

$$f_1(a,b)=(K \times (a+b), K \times (a-b)) \tag{4}$$

$$f_2(a,b)=((a \times b)^K, (a/b)^K) \tag{5}$$

$$f_3(a,b)=a^b, 1/b \tag{6}$$

Straightforward calculation shows that $f_1, f_2,$ and f_3 each exhibit the self-inverse property of Equation 2.

EXAMPLES OF INVERTIBLE BENCHMARKS

Example 4—Use PIPs to implement $c=a \times b$.

A network using two PIPs is shown in Figure 1 which yields the required result. The network requires three stor-

age locations. Figure 1 is drawn in such a way as to emphasize the mirror image property of the inverse network.

Example 5—Use PIPs to implement $y=a^2+(b \times c)-(e \times f)$.

A network employing 11 PIPs is shown in Figure 2. This solution requires 12 storage locations and six time steps. The PIPs associated with a given time step can be executed in any order or even simultaneously if the hardware permits. A trace of the storage location contents is shown in Figure 3. If the intermediate results are incidental to the required result, they have been given arbitrary names such as g, h, i, \dots, p, q in Figure 2 and Table I. The inverse of the network of Figure 2 is illustrated in Figure 3. A trace of the time steps associated with the inverse network is shown in Table II.

APPLICATION AND DISCUSSION

In the practical world of computer system selection, a few systems are candidates, each with its supporters. Similarly, a modest number of benchmarks are usually proposed.

TABLE I.—Direct network for Example 5.

Storage Location	Storage Contents						
	Initial	After Step					
		1	2	3	4	5	6
L1	p	p	p	$K \cdot a^2$	a^2	$a^{1/K}$	a
L2	i	i	i	i	i	i	a
L3	m	m	m	m	$b \cdot c$	$[b \cdot c]^K$	b
L4	g	g	g	g	g	g	c
L5	y	y^K	$0.5 \cdot y$	$K \cdot [b \cdot c - e \cdot f]$	$e \cdot f$	$[e \cdot f]^K$	e
L6	h	h	h	h	h	h	f
L7	j	j	j	j	j	$1/K$	$1/K$
L8	K	K	K	K	K	$1/K$	$1/K$
L9	l	l	l	l	l	$1/K$	$1/K$
L10	n	n	n	n	0	0	0
L11	q	q	2	2	2	2	2
L12	K	$1/K$	$1/K$	$1/K$	$1/K$	$1/K$	$1/K$

$$K = 0.707\dots$$

TABLE II.—Inverse network for Example 5

Storage Location	Storage Contents						
	Initial	After Step					
		1	2	3	4	5	6
L1	a	$a^{1/K}$	a^2	$K \cdot a^2$	p	p	p
L2	a	i	i	i	i	i	i
L3	b	$[b \cdot c]^K$	$b \cdot c$	m	m	m	m
L4	c	g	g	g	g	g	g
L5	e	$[e \cdot f]^K$	$e \cdot f$	$K \cdot [b \cdot c - e \cdot f]$	$0.5 \cdot y$	y^K	y
L6	f	h	h	h	h	h	h
L7	1/K	1/K	j	j	j	j	j
L8	1/K	1/K	K	K	K	K	K
L9	1/K	1/K	1	1	1	1	1
L10	0	0	0	n	n	n	n
L11	2	2	2	2	2	q	q
L12	1/K	1/K	1/K	1/K	1/K	1/K	K

$$K = 0.707\dots$$

It has been difficult in the past to put a common measure on the benchmarks. Thus, a standoff results when System A is faster on Benchmark 1 while System B is faster on Benchmark 2. It appears that the use of the techniques of this paper can help to resolve such a situation. For example, an aggregate benchmark could be constructed by weighting Benchmark 1 and Benchmark 2 according to some estimate of their relative frequency of execution in the actual job mix. For each benchmark an information content, H_2 , can be computed. Then for any given system the figure of merit in bits-per-second can be computed from measurements on the average number of source symbols transmitted per second by an invertible benchmark. A figure of merit for each competing system can then be computed by weighting individual benchmark figures of merit in accord with the relative frequency of execution. It seems reasonable to assign the implementation of the three PIPs for a particular system to the proponent or proponents of that system.

It is clear from our discussion that there are many PIP network implementations of any given benchmark. It may be possible in the future to find a systematic way of discovering the best implementation. However, any acceptable

definition of best should recognize the potential for trading speed for storage requirements. The PIP network approach exposes both of these factors to view and also allows us to count the necessary PIPs of each type.

Finally, it is possible that the idea of invertible benchmarks can be exploited in the direction of assisting in the development of checkpoint, rollback and restart properties for ordinary programs. Note that an invertible program can be suspended indefinitely after any step and resumed at leisure. This can be important to a system which encounters emergency shutdowns and also to a system which features a priority interrupt scheme.

REFERENCES

1. Shannon, C. E., "A Mathematical Theory of Communication," *Bell System Tech. J.*, Vol. 27, 1948, pp. 379-423, 623-656.
2. Hellerman, L., "A Measure of Computational Work," *IEEE Trans. Comput.*, Vol. C-21, May 1972, pp. 439-446.
3. Welch, T. A., "Comments on 'A Measure of Computational Work,'" *IEEE Trans. Comput.*, Vol. C-23, No. 2, p. 208.
4. Hellerman, L., "A Discussion of 'A Measure of Computational Work,'" *IEEE Trans. Comput.*, Vol. C-23, No. 2, pp. 208-211.

5. Savage, J. E., "Computational Work and Time on Finite Machines," *J. Assn. Comput. Mach.*, Vol. 19, No. 4, October 1972.
6. Johnson, R. R., "Some Steps toward an Information System Performance Theory," *ACM Performance Evaluation Rev.*, Vol. 1, No. 3, Sept. 1972, pp. 4-15.
7. Huffman, D. A., "A Method for the Construction of Minimum Redundancy Codes," *Proc. IRE*, Vol. 40, No. 10, September 1952, pp. 1098-1101.

APPENDIX

TABLE III.—Huffman Codes for Example 1

S=1				
Prob.	Input Seg.	Channel Code	Length	Prob. × Length
0.9	A	0	1	0.9
0.1	B	1	1	0.1
				1.0 bin.sym./source seq.
1.0/1=1.0 bin.sym./source letter				

S=2				
Prob.	Input Seg.	Channel Code	Length	Prob. × Length
0.81	AA	0	1	0.81
0.09	AB	11	2	0.18
0.09	BA	100	3	0.27
0.01	BB	101	3	0.03
				1.29 bin.sym./source seq.
1.29/2=0.645 bin.sym./source letter				

S=3				
Prob.	Input Seg.	Channel Code	Length	Prob. × Length
0.729	AAA	0	1	0.729
0.081	AAB	100	3	0.243
0.081	ABA	101	3	0.243
0.081	BAA	110	3	0.045
0.009	ABB	11100	5	0.045
0.009	BAB	11101	5	0.045
0.009	BBA	11110	5	0.005
0.001	BBB	11111	5	
				1.598 bin.sym./source seq.
1.598/3=0.533 bin.sym./source letter				

PROOF OF

$$-[p_a + p_b] \times \log(p_a + p_b) < -[p_a \times \log p_a] - [p_b \times \log p_b]$$

Assume

$$p_a \leq p_b \tag{1}$$

Then

$$\frac{1}{p_b} \leq \frac{1}{p_a} \tag{2}$$

and

$$\left[\frac{1}{p_b}\right]^{p_a} \leq \left[\frac{1}{p_a}\right]^{p_a} \tag{3}$$

So

$$\left[\frac{1}{p_b}\right]^{p_a} \cdot \left[\frac{1}{p_b}\right]^{p_b} \leq \left[\frac{1}{p_a}\right]^{p_a} \cdot \left[\frac{1}{p_b}\right]^{p_a} \tag{4}$$

$$\left[\frac{1}{p_b}\right]^{p_a + p_b} \leq \left[\frac{1}{p_a}\right]^{p_a} \cdot \left[\frac{1}{p_b}\right]^{p_a} \tag{5}$$

But

$$\frac{1}{p_a + p_b} < \frac{1}{p_b} \tag{6}$$

so

$$\left[\frac{1}{p_a + p_b}\right]^{p_a + p_b} < \left[\frac{1}{p_b}\right]^{p_a + p_b} \tag{7}$$

Combining (7) with (5) gives

$$\left[\frac{1}{p_a + p_b}\right]^{p_a + p_b} < \left[\frac{1}{z_a}\right]^{p_a} \cdot \left[\frac{1}{z_b}\right]^{p_b} \tag{8}$$

Taking logarithms in (8) completes the proof.

A loosely-coupled applicative multi-processing system*

by ROBERT M. KELLER, GARY LINDSTROM and SUHAS PATIL

University of Utah
Salt Lake City, Utah

INTRODUCTION

The architecture of highly-parallel machines has received increased attention from researchers over the past decade. At first, because of the machines' novelty, workers were content with proposing elaborate machine architectures without giving due consideration to how such machines would ultimately be programmed to exploit their available computational power. Experience with Illiac IV, Star-100, etc. has shown this to be a mistake. Indications are that programming languages deserve consideration at the earliest stages of architectural conception. Included in such considerations are issues such as storage and task management.

This paper describes a proposed machine, AMPS (Applicative Multi-Processing System). It features a *loosely-coupled architecture*, incorporating a large number (say 1000) of processors functioning independently to a large extent, but effectively interacting when necessary. Furthermore, the programs supported are not tied to the structure of the machine, thereby facilitating expandability. Such expandability is further enhanced by the particular physical organization to be described. The architecture of AMPS attempts to bring costs of communication among processing units to a manageable level by taking advantage of *locality of reference*.

Our architecture is currently in the development stage. We present in this paper some of our major philosophical considerations, along with an execution model for a subset of the machine language.

LANGUAGE ISSUES

Heretofore, research on highly-parallel machines has predominantly emphasized statically-structured, usually numerical, computation. We are orienting our design toward dynamically-structured, often symbolic, computations. Although we do not exclude numerical applications from AMPS, we are designing it to provide direct support for

languages such as Lisp which have been invented for symbolic computation. In fact, the machine language for AMPS itself resembles a dialect of Lisp. By retaining a close relationship between a higher-level language and its supporting machine language, debugging is facilitated. Furthermore, the *applicative* (i.e. based on *function application*) nature of our machine language obviates many pre-processing transformations of the type used to extract parallelism. Such transformations are really just means of extracting functional dependencies which are easily determined from an applicative program.

To further clarify, consider the task of counting the leaves of a binary tree. Figure 1a presents an applicative program for this task, whereas Figure 1b presents a corresponding non-applicative program employing a stack. Clearly, Figure 1a is easier to understand and its inherent parallelism is easier to detect than that of Figure 1b.

Lisp, with some minor extensions, such as *lenient cons* discussed in the seventh section (*cf.* References 9 and 12) seems to include all opportunities for exploitation of concurrency that proposed data flow languages do, and more. It provides concurrent operations on tree or graph data structures during their creation, and natural ways for dealing with conceptually infinite structures. By supporting such a language at the machine level, we also provide a natural means of communication between processes and their environment, e.g. file systems and I/O devices. Space limitations preclude discussing these issues here, but related ideas may be found in References 10 and 22. It is also worth mentioning that our machine language can directly support other languages which deal with infinite structures such as those in References 5, 2 and 13.

BASIC ARCHITECTURE

The physical arrangement of components in AMPS is a tree structure with two types of nodes. The scaled-down version in Figure 2 is merely meant to be suggestive, as we envision trees with 1000 or more nodes as being feasible in the next decade. Combined processor/memory units are attached as leaf nodes. The internal nodes of the tree structure are combined communication and load-balancing units. Other more specialized units might also be present, attached

* This work was supported in part by grants DCR-74-21822, MCS-77-09269 and MCS-78-03832 from the National Science Foundation.

a. Applicative Program

```

leafcount(t) = if atom(t)
                then 1
                else leafcount(left(t)) + leafcount(right(t))

```

b. Non-Applicative Program

```

leafcount ← 0;
stack ← {t};
while non-empty(stack) do begin
  pop T from stack;
  if atom(T)
    then leafcount ← leafcount + 1
    else begin
      push left(t) on stack;
      push right(t) on stack
    end
end
end

```

Figure 1—Applicative and non-applicative versions of leafcount.

closer to the root node for enhanced accessibility and utilization, but we do not further discuss this possibility here.

This paper makes the assumption of a binary tree for simplicity, although technology considerations suggest that a 4-ary or 8-ary tree might be more appropriate. For expedited communication, we may eventually include additional links laterally connecting the tree, possibly as suggested in Reference 8.

A processing unit in AMPS is roughly the size of a conventional micro-computer, but its architecture is substantially different. It is able to carry out local computation, particularly with respect to assembly and dissemination of information, and to initiate actions for fetching information from other nodes of the tree. It is able to execute program tasks sequentially or in an overlapped mode, and to allocate storage in response to the execution of *invoke* instructions. *Invoke* instructions create tasks which are then executed either in the local processing unit or in another processing unit, as system loading dictates (see the eighth and ninth sections).

The primary memory of the system is distributed among the processing units. Each processing unit has direct access to that segment (e.g. 64K words) of memory located within it. It also has access, through the *communication network*, to the segments of memory located at other processing units. Even though the memory is distributed among the processing units, there is only one *unified address space*. Given the address of a datum, any node in AMPS is able to logically access it without address translation. Such addresses do not appear at the assembly language level, as they are generated by the system dynamically. The internal nodes of the communication network are responsible for any required routing of data. Access to auxiliary memory and other forms of

external communication takes place through special-purpose leaf processors, which will not be discussed here.

COMMUNICATION NETWORK

The communication network in AMPS is designed to take advantage of locality of information flow, thereby reducing communication costs. Information first travels up the tree towards the root node until it encounters a node which spans the destination leaf, whence it proceeds down the tree until it finally reaches the desired destination. Thus, for sending or receiving information from neighboring leaves, it is not necessary for the information to travel the entire depth of the tree. Relatively local data flow therefore takes less time and the overall communication cost of the computation is thereby improved.

A second function of the communication network is to provide a reasonably balanced distribution of the computing load. Such a function is useful since the machine allocates tasks dynamically. Each node of our communication network periodically obtains *load monitoring signals* from its subordinates, which indicate their current degrees of utilization. When such signals indicate a sufficiently unbalanced state, the node can cause the transfer of uninitiated tasks from one subtree to the other (see the section on Load Balancing).

LOCALITY

One of the most important concepts of our architecture is an attempt to improve performance by exploiting locality of information flow. Locality is an established phenomenon in program execution, which should therefore be exploitable within applicative programs. Locality will be enhanced by the fact that functions are naturally apt to confine their references to certain portions of data structures. Secondly, repeated global references to the same data will become localized by a *caching effect* incorporated in the referencing scheme of AMPS. Note that the read-only nature of data in applicative systems greatly simplifies the implementation of such caches.

If computations which interact heavily with one another are allocated space in leaves that are a short average distance apart in the communication network, the overall time spent in information flow will be reduced. It is important to note that even if it is not possible to allocate space for a new computation in the storage space at the invoking leaf, the correctness of the overall computation will be maintained, even though the speed of the computation may be decreased. This is aided by the uniformly acceptable address space and the deferred binding of program blocks to physical addresses. Moreover, such locality should tend to cause traffic flow to decrease with node level, thereby balancing bandwidth with demand.

In designing a highly-parallel machine, care must be taken that costs associated with creating new tasks and communicating with them do not outweigh the speed advantage

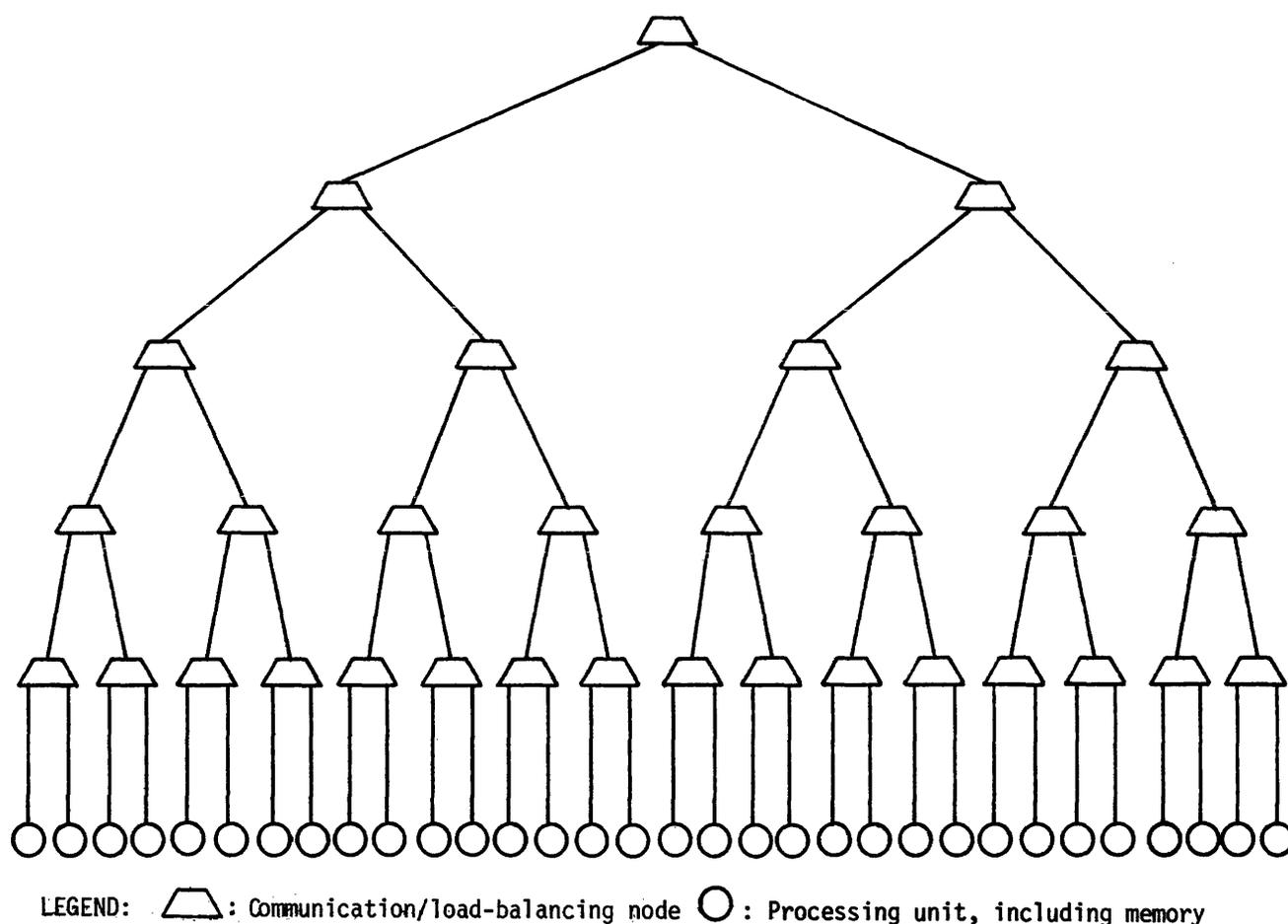


Figure 2—Physical form of AMPS.

gained from overlapped execution of these tasks. Consequently, our design prescribes that computation is divided into blocks in such a way that all computation local to a block (i.e. exclusive of communication with other blocks) will be done within one processing unit. Since such blocks are identical with the code blocks of the program (see the eighth section), locality is further enhanced by the clustering of connected operators within blocks. The global structure does not seek gains from parallelism on the level of, say, a simple arithmetic expression. Instead, this effect is achieved within the processing unit itself.

Another anticipated effect which will contribute to locality might be called the *seeding effect*. When a task A in execution creates a second task B, the latter may be allocated its storage in any of the processing units in which there is sufficient space. Since B may cause the creation of other tasks C1, C2, . . . , Cn, locality is enhanced if the storage for the latter is allocated in processing units near to that of B in the tree. Hence even if B is a long distance from A, thus incurring non-trivial communication cost between the two, this cost may be balanced out by the lower costs of communicating between B and C1, C2, . . . , Cn.

The seeding effect creates a tradeoff in resolving a choice of how far away a created task should be placed. It also

demonstrates the possibility of a certain amount of *natural re-localization* in recovering from bad task-placement decisions by the system. For example, even if B were placed in a congested area, the storage from completing tasks near B could eventually be reclaimed to provide more space for C1, C2, . . . , Cn. Although such a scheme will not always work with optimal efficiency, it will work until all space is in use.

INFORMATION FLOW

The characterization of information flow within the machine is dependent on the conceptual level being considered. For example, at the *task level*, we are concerned with the flow of operands among tasks, which we implement in a *demand driven* fashion. In the demand-driven scheme a task may actively seek additional pieces of data after its initiation. In contrast, most proposed data flow machines are primarily data-driven, in that an instruction never asks for data to be sent to it. Instead, it waits to receive data itself, and when all necessary operands have been received, it initiates the computation, the result of which is then sent to all other designated instructions.

At the *communication network level*, we find the infor-

mation flow separated into the flow of *tasks* (which at this level are always *invoke* instructions), *operands* (single data words), and *blocks* (multiple data words), and requests for the latter two. All such pieces of information are accompanied by a destination address. All information transmitted through the communication network is handled by *packet switching* (i.e. *store-and-forward*). Line-switching is not used because of the potential congestion caused by tying up long paths through the network.

A node of the communication network communicates to its parent through a form of handshaking. For block transfers, a *burst mode* of communication is used in which the handshaking occurs only before and after the entire block has been transferred, thus drastically reducing the associated overhead.

MACHINE LANGUAGE

As its language, AMPS executes a compiled dialect of Lisp called *FGL*, for *Flow-Graph Lisp*, or *Function Graph Language*. Although FGL programs are stored as blocks of compiled code, we prefer to present them using *function graphs*, as described in Reference 15. FGL allows us to display clearly the data flow between operators and thus the potential concurrency within programs.

A program in FGL consists of a main function graph, together with *productions* for programmer-defined functions. These productions specify how a node containing a function name (the *antecedent* of the production) is to be replaced by a function graph (the *consequent* of the production). FGL provides a repertoire of basic operators (e.g. the primitive functions of Lisp) that may be used in constructing function graphs.

For the sake of this presentation, let us suppose that data structures are *trees*, with integers and *nil* as atoms. Conditional evaluation is obtained through the use of the function *cond*, which causes the evaluation of its second or third argument, depending on whether its first argument is non-*nil* or *nil*, respectively.

Trees are built using the *cons* operator, which forms trees from atoms or other trees by connecting its arguments as subtrees of a common root. The selector functions *car* and *cdr* extract the left and right subtrees respectively of a tree built by *cons*. The *cons* of FGL is in fact *lenient cons*, which allows the machine to exploit concurrency which it could not with conventional *strict cons*.⁹ More precisely, FGL semantics provide that the identities $car(cons(x,y))=x$ and $cdr(cons(x,y))=y$ hold independent of whether *x* and *y* are convergent.

For simplicity, we do not discuss input of trees. Rather, we assume them to be resident at the beginning of the computation, built by an appropriate graph of *cons* operators and atoms. Conceptually, trees flow on the arcs of an FGL graph. In actual implementation, however, the flow of a non-atomic tree is represented by the flow of a pair of pointers to its subtrees. For the current presentation, iteration is implemented by recursion, in the manner of Reference 19. This can be shown to give automatically the same

concurrency-detection effect of "look-ahead" processors, which "unfold" iterations to achieve concurrency.¹⁴

To cause a result to be printed, a *demand* is generated at some *print* node in the function graph. This causes propagation of the demand to the operator feeding the *print*. When that operator ultimately produces a value, it will then be printed.

Evaluation consists of a combination of transmutations to the graph and the application of basic operators. In this sense, AMPS is a *reduction machine*,⁴ executing a *reduction language*.³ By exploiting the richer connectivity of graphs, we can avoid much of the *combinatorial explosion* which takes place in purely string-oriented reduction machines.

Figures 3 through 7 give examples of programs in FGL. Figure 3 presents a production for the function of Figure 1a, which counts the leaves of a tree. This example uses the *strict* operator (i.e. one which demands *both* of its arguments) *add* to cause the creation of instances of operators which can be evaluated concurrently. Figure 4 shows a possible snapshot of the program during its application to a specific tree. Several concurrent sub-computations are visible.

In Figure 5a, we present a *main program* which calls a recursively-defined function NATNUMS, the graph of which is presented in Figure 5b. Intuitively, NATNUMS(*n*) "computes" the infinite sequence $\{n, n+1, \dots\}$ by constructing its representation in the form of a tree as shown. In the context of the main program, the value printed is the second element of the sequence where $n=2$, i.e. $car(cdr(NATNUMS(2)))$. Adjoined to the graphs in Figure 5 are listings of their FGL assembly code, the meaning of which is explained in the next section. The parenthetic labels on the graph indicate the correspondence between the graph and the code.

Figure 6 presents a program which employs the function NATNUMS to generate the prime numbers in increasing order. The reader may wish to refer to the similar examples in References 2 and 13.

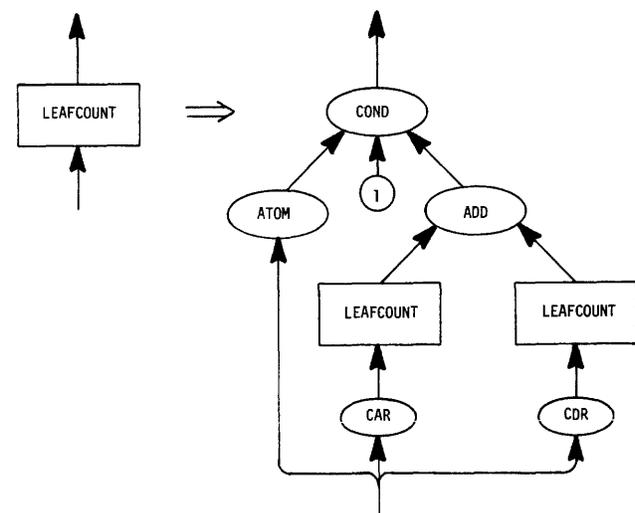


Figure 3—FGL production for the leafcount function.

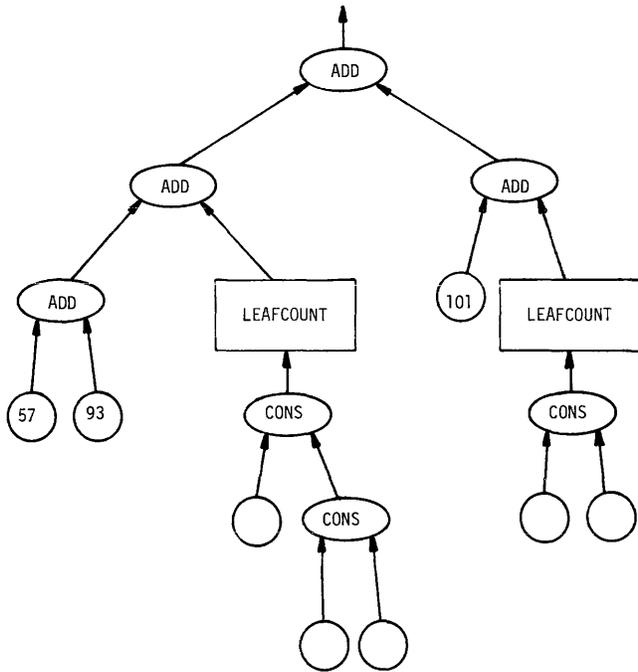


Figure 4—One possible snapshot of the program of Figure 3 during its computation on a tree. Unlabeled leaves are those of the original tree. The ultimate result will be the number 256.

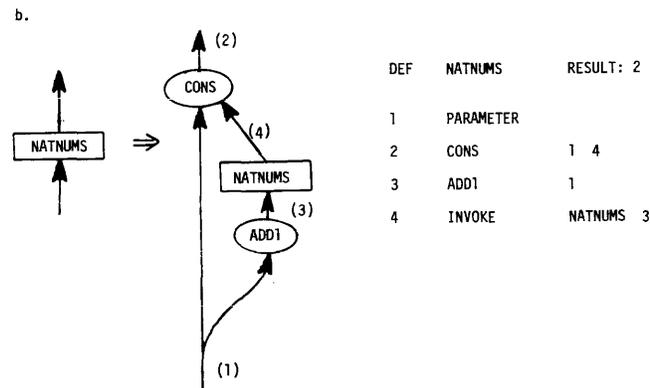
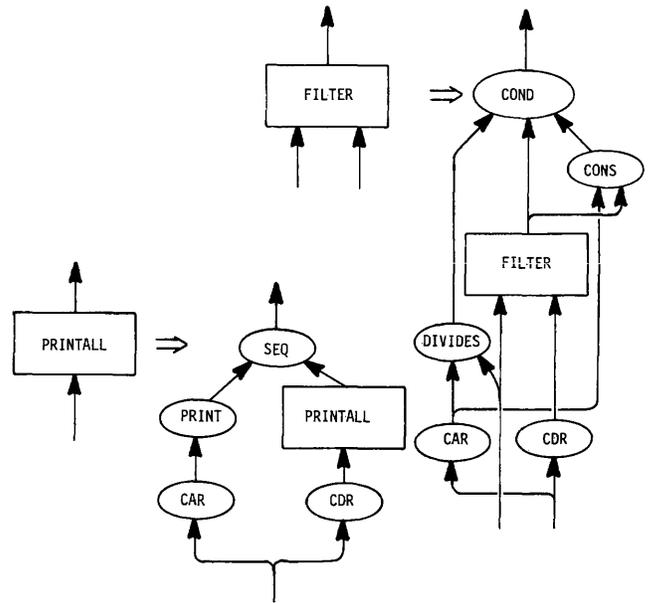
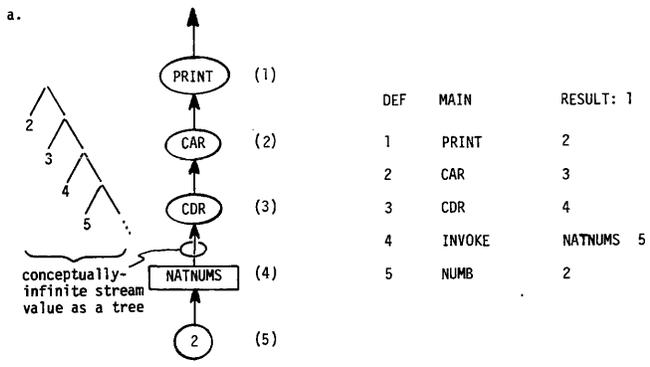
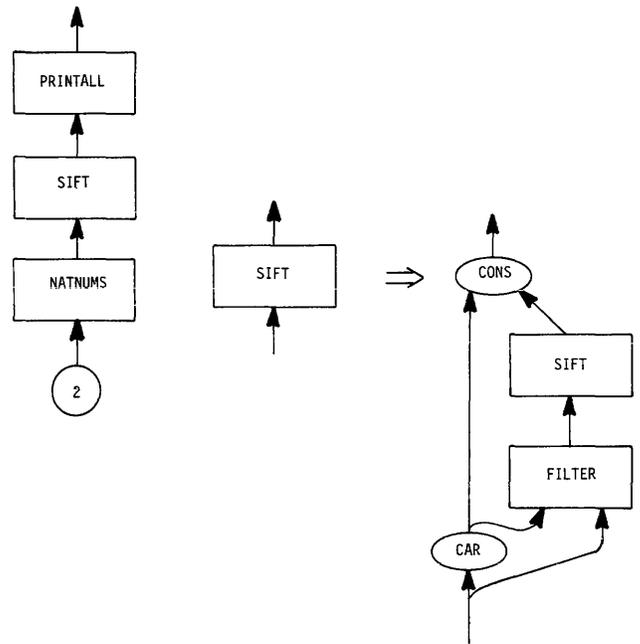


Figure 5—Sample FGL main-program (a) and production for defined-function NATNUMS (b) with assembly-language listings.

Figure 6—FGL program for printing prime numbers in increasing order. The production for NATNUMS is given in Figure 5. The primitive function *divides* yields *nil* unless its second argument evenly divides its first. The primitive *seq* causes its arguments to be evaluated in sequence.

It can be proved that every well formed interconnection of FGL operators computes a unique function (*cf.* Reference 16), even an interconnection involving cycles. Cycles provide one means of efficiently implementing bi-directional communication between two functions. Such an example is shown in Figure 7, which illustrates a program for the breadth-first production of all atoms in a tree. Detailed description of the recursively-defined functions *PASSATOM*,

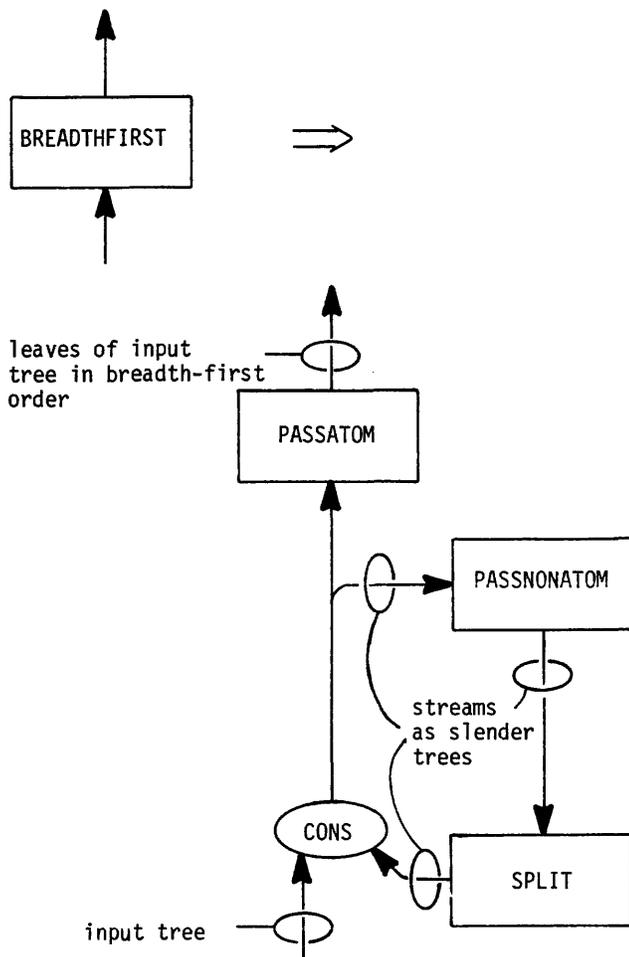


Figure 7—Production defining a function which produces the leaves of a tree in breadth-first order. Productions defining PASSATOM, PASSNONATOM, and SPLIT are not shown.

PASSNONATOM and SPLIT are not presented here. It should be noted that FGL employs fan-out of arcs to effectively avoid recomputation of the same value, an effect which must be obtained by recognition of common sub-expressions in some applicative languages.

In the next sections we describe, in more detail, program storage, task execution, typical operators, and production application via the special operator *invoke*. We do not discuss storage reclamation here, for lack of space.

PROGRAM STORAGE AND EXECUTION

All storage is allocated in *blocks*. The use of blocks makes storage management more efficient, and is consistent with trying to keep the *locality* of a computation contained within one processing unit. A block is either a *data block* or a *code block*. The contents of a code block form a linear representation of an FGL graph, which is copied as the source of

initial code to be stored in a newly allocated data block. This copying may be viewed as the application of an FGL production, i.e. replacing the antecedent node with its consequent graph. Each word in a data block is initially a code word or a literal value. A code word may get changed to a literal value during execution. The *ready bit* distinguishes whether the word is a value or a code word.

A code word in a data block corresponds to a node in an FGL graph. A value corresponds to what eventually appears on the output arc of that node. The code word contains the local addresses of words corresponding to nodes at the opposite end of its input arcs, i.e. the sources of its operands. Local addresses are relative to the start of a block. We assume here for simplicity that each operator has only one output arc, although such arcs may fan out as necessary.

In addition to specifying the input arcs of its operands, an instruction word may include *notifiers*, which are local addresses of operators which have this operator's output arc as one of their input arcs. These are usually set dynamically as demands occur, although it is possible to have them pre-set in the initial code.

In addition to an *operation code*, the following fields may or may not be present, depending on the nature of the particular operation:

1. Local addresses of *arguments* of the operator.
2. *Notifiers*, i.e. local addresses of *notified* code words, which are instructions which have demanded this instruction's value.
3. A single *global address*, used to provide linkage across blocks, and for specifying the code block in the case of an invoke instruction.

The *invoke* instruction, when demanded, causes the allocation of storage for a data block and the copying of a code block into that storage. The demand driven approach thus provides a natural means of deciding whether and when to trigger the invocation of a defined function, which requires the allocation of a storage block. An *invoke* instruction also initializes various linkage instructions which provide an interface between the nodes of the graph containing the antecedent of the production and those of the consequent, since local addresses cannot be used to provide this linkage. Details on these linkage instructions may be found in Reference 18. Linkage instructions are not shown in the code blocks in Figure 5, as they are supplied automatically by the assembler.

Regarding efficiency, the use of local addresses achieves economy in code word storage, and avoids relocation in copying. The copying of code blocks may be contrasted with approaches such as that in Reference 21, which interpret a pure code block without copying. The approach taken here is more effective in keeping references local to a processing unit. It also eliminates separate fetching of code and data words for each task. Due to the use of a burst mode, blocks are copied more efficiently than the same number of words individually.

TASK EVALUATION

Although code words may be conceptualized as representing functions, we must describe how these words are interpreted by the machine to cause values to be produced. The loosely-coupled aspect of task evaluation is achieved in AMPS through a *task list* organization, which allows many processors to partake in the evaluation of *tasks*, i.e. particular instances of operators with their associated data. The task list is decomposed into two separate lists which may be served independently. These are:

demand list—Contains addresses of operators for which evaluation is to be attempted.

result list—Contains addresses of operators, along with their corresponding values after evaluation.

The *invoke* tasks on the demand list are distributed to individual processing units by the communication network, which takes into account the current processor load profile. Only such tasks are considered for distribution, since it is only these which might profitably be executed in another processing unit, due to the communication costs incurred in their transmittal. Hence, each processing unit has its own *invoke list*, a sublist of the demand list containing only *invoke* instructions.

Figure 8 shows the organization of the task evaluation mechanism. The diagram is to be interpreted in an informal sense, and is less akin to conventional flowcharts than indicative of the flow of tasks in the system. Further details may be found in Reference 18.

Initially, the address of the word which will produce the "main result" is put on the *demand list*. The word itself is then fetched. The instruction specifies certain arguments, which are also fetched. If the arguments are all ready, the function is evaluated. If not, then demand is propagated to each unready argument not previously demanded by placing its address on the demand list and setting a notifier in it. A word may contain several notifiers indicating which instructions have demanded it. The presence of at least one notifier is used to signify a previous demand. Figure 9 sketches symbolically the flow of demand in the example of Figure 5, along with the corresponding production applications and evaluations which take place.

Once evaluated, a result value replaces the code word as *ready* data. Via the result list, any instructions which were specified by notifiers as awaiting this result as an argument are then notified by putting them on the demand list to be retried. Observe that all demanded operators remain accessible until they are replaced by ready data, either through being put on the demand list or through being referenced by a notifier of an accessible operator.

Forms of evaluation other than pure demand evaluation can be supported by judicious pre-setting of notifiers and demand bits and advanced placement on the demand list. Special operators are also available to the programmer which have the purpose of generating advanced demand to enhance parallelism, and for postponing demands to avoid premature allocation of data blocks.

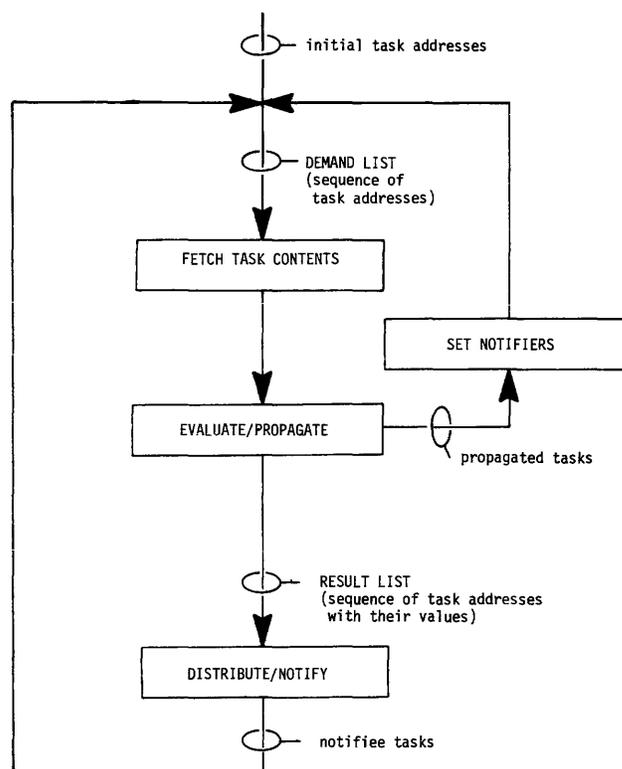


Figure 8—Simplified task processing flow.

PROCESSING UNIT ARCHITECTURE

We do not go into great detail here on the organization of individual processing units. As described in the previous section, each unit selects tasks from its demand list. While on this list, a task is represented by its address in memory. The content of this address is fetched and, if a code word, an attempt is made to evaluate it. This normally entails reference to one or more additional words in the memory.

Since a referenced word might reside in the physical memory of any processing unit, fetching may involve transmission of words through the communication network. In order that the processor need not be idle while such a fetch is taking place, we provide a *staging area* for buffering a set of such tasks while their operands are being assembled. The staging area is conceptually similar to a conventional *pipeline*, except that order of task execution is unimportant, all essential ordering being explicit in the program graph.

The size of the staging area is chosen to maintain reasonably good utilization of the function units within the processing unit, which carry out each operation once its operands are assembled. Of course, each function unit could itself be pipelined, depending on economic advantages which would accrue under particular application loads. Design of such a staging area is fairly routine and therefore will not be discussed further here.

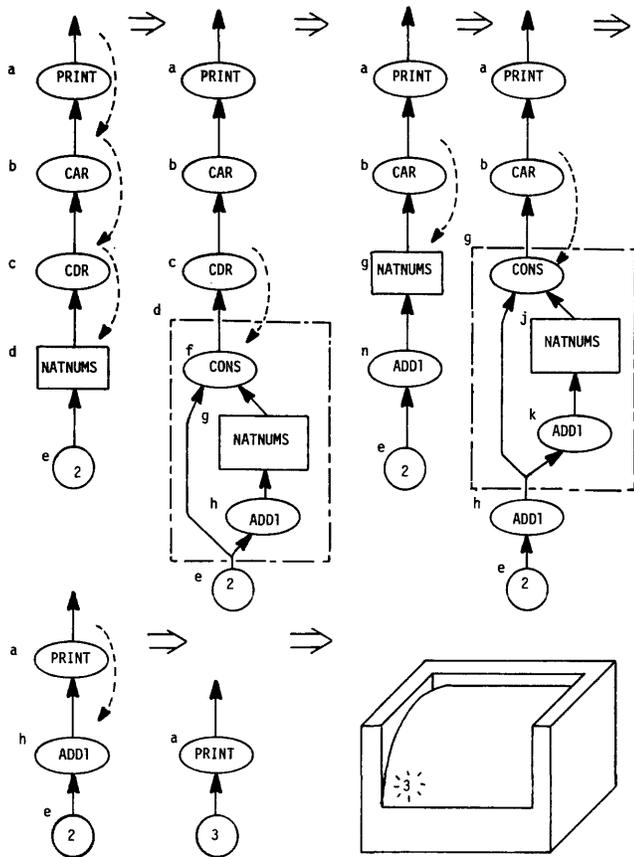


Figure 9—Snapshots of the evaluation of an FGL program. Perforated arrows illustrate demand propagation.

LOAD BALANCING

Load balancing occurs through the redistribution of tasks from the invoke list of one processing unit to that of another. This is a function of the communication network which is separate from, but topologically compatible with, the routing of operand data.

By the *load* at a processing unit, we mean the storage requirement of tasks on the invoke list at that unit. In a similar manner, we can define the *load* at any node of the communication network to be the sum of the loads at its spanned leaves, divided by the number of such leaves as a normalizing factor.

Again, to simplify the explanation, we are assuming that the communication network is a binary tree. Each node of the communication network maintains desired lower and upper limits, L and U , on the loads of its immediate descendants. These are functions of the amount of storage currently in use by the leaves spanned by those descendants. If the load of one descendant is above U and that of the other below L , the network attempts to shift tasks from the overloaded descendant to that of the underloaded one. If loads of both its descendants are above U , this will be communicated to its parent (if any), so that the latter may try to shift some of the load to one of its

descendants having load less than L . In this way, the balancing function is distributed throughout the communication network, with each node thereof applying the same balancing strategy.

The effectiveness of the balancing scheme relies on the loosely-coupled nature of the system. That is, no task is bound to a particular processor until storage is allocated for it.

COMPARISONS WITH RELATED MACHINES

The data flow machines of References 7 and 1 originally influenced the structure of AMPS. The communication network in AMPS plays the role of the arbitration and distribution network of the Dennis data flow machine. However, the processing units which assemble instructions and initiate information flow are of a higher level, as are the processors in Reference 1.

Even though the architecture of AMPS has a tree like structure, it is not a "recursive architecture" in the sense of Reference 6. The hierarchical structure and method of storage allocation in the Davis machine seem to impose certain constraints on the creation of new computations and on the flow of information in the machine. For example, when a processing element creates a task, the latter must be placed either in the space of the processor carrying out the application or in the space of a subordinate processor, even if all subordinates are crowded for space and the machine has other processors which have plenty of space. This problem does not occur in AMPS, due to the construction of the communication network, the uniformity of the address space, and load balancing.

A common feature of all of the previous architectures is that they are data-driven rather than demand-driven, as is AMPS. One might be led to think that the latter presents some additional overhead. However, closer examination reveals that FGL programs are often simpler in that they do not require explicit instructions for the gating of data.

In data-driven machines, a form of ready-acknowledge signalling is often used for transmission of data via storage words. This is, in fact, a special case of demand-driven computation, in which the demand for an operand is equated with readiness of its recipient. The demand-driven approach seems to provide more flexibility in the relationship between elaboration of a programmer-defined function and the evaluation of its arguments. It is also clear that the demand-driven feature is a necessity in supporting lenient *cons*. On the other hand, demand-driven computation could possibly be *engineered* on other architectures by treating demands as data, but this seems cumbersome.

The tree-structured reduction language machine of Reference 20 is fundamentally different in its operation from AMPS. In the former machine, an expression to be evaluated is mapped symbol by symbol onto the leaves of the tree. In AMPS, an expression would first be converted to function graphs which then reside in the memory space of one or more processing units of the machine.

Our system has in common with those just cited the desire

to integrate architectural, communication and language considerations. This is one of the ways it differs from superficially similar systems, such as Cm*.²³ In Cm*, parallel processing is based on the concept of interacting sequential processes that run on conventional processors (DEC PDP-11's), while AMPS is capable of directly evaluating function graphs, integrates considerations of evaluation and local balancing, and directly supports communication among tasks.

CONCLUSIONS AND FUTURE RESEARCH

We have stated that machine architectures should be developed with greater attention paid to ultimate programmability. With this motivation, we have presented a loosely-coupled parallel processing system AMPS which executes an applicative language, FGL. We have sketched the internal representation of programs in AMPS and the execution of programs on it.

Our implementation seems to be the first detailed one presented for Lisp programs on a parallel machine. An implementation has been described qualitatively in Reference 11. However, their description relates mainly to the issue associated with *colonel* versus *sergeant* tasks, sergeants being distinguished from colonels as tasks whose evaluation may never actually be required, but which provide a potentially useful way of employing otherwise idle processors. In contrast, all tasks in the machine described here are of the colonel variety, whose existence may be traced to certain *strict* operators, such as *add* in the leaf count example of Figure 3. On the other hand, subtle details, such as occur in the implementation of a *global notifier* scheme, have been discovered in the course of designing our evaluator.¹⁸ How such subtleties interact with an implementation which does offer sergeant tasks remains a topic for future investigation.

The ideas presented here were derived after considering many possible alternatives. We may, of course, elect to return to one or more of these alternatives after more experience in programming the machine has been gained. A simulator for the evaluation model has been written in Pascal to assist in such a venture. Thus far, the simulator has been used to verify that the evaluation mechanism works and to experiment with additions to the language FGL. Construction of a Simula-67 simulator for the tree architecture is now in progress. We have no immediate plans for construction of a physical realization of the machine.

Issues remaining to be investigated include the necessary support for FGL in terms of storage reclamation and scheduling. We are currently contemplating how best to organize the distributed heap for efficient medium-term data storage.

We have preliminary results on how to deal with other features of Lisp, such as *funargs*, *setq*, and *prog*. A description of the handling of *funargs* appears in Reference 18. Efficient access of array-like structures is handled by extending *cons* from pairs to tuples and providing an indexed selector function.

A related issue is whether *indeterminate* computations should be supported, as there are some indications that they

permit efficiency gains not otherwise achievable.¹⁷ Under investigation also is the use of machine operators for the support of resource allocation. These have been programmed into the experimental simulator and seem to fit very naturally with our method of evaluation. The usefulness of applicative programs in allowing graceful backup when a processing unit fails also remains to be exploited. Thus many issues, at levels from detailed processor construction to more fundamental language problems, await us.

ACKNOWLEDGMENTS

Comments by Al Davis, Milos Ercegovac, Mark Franklin, Kim Gostelow, Jed Marti, Elliott Organick and the anonymous referees are appreciated. A discussion with Gilles Kahn provided incentive to use demand-driven computation.

REFERENCES

1. Arvind and K. P. Gostelow, "A computer capable of exchanging processors for time," *Proc. IFIP '77*, June 1977, pp. 849-853.
2. Ashcroft, E. A., and W. W. Wadge, "Lucid, a nonprocedural language with iteration," *CACM*, Vol. 20, No. 7, July 1977, pp. 519-526.
3. Backus, J., "Programming language semantics and closed applicative languages," *Proc. ACM Symp. on Principles of Programming Languages*, 1973, pp. 71-86.
4. Berkling, K. J., "Reduction languages for reduction machines," *Second Annual Meeting of Computer Architecture*, 1975, pp. 133-138.
5. Burge, W. H., *Recursive programming techniques*, Addison-Wesley, 1975.
6. Davis, A. L., "The architecture and system method of DDM-1: A recursively-structured data driven machine," *Proc. Fifth Annual Symposium on Computer Architecture*, 1978.
7. Dennis, J. B., and D. P. Misunas, "A preliminary architecture for a basic data-flow processor," *Proc. 2nd Annual Symposium on Computer Architecture*, December 1974, pp. 126-132.
8. Despain, A. M., and D. Patterson, "X-tree: A tree structured multiprocessor computer architecture," *Proceedings of the Fifth Annual Symposium on Computer Architecture*, 1978.
9. Friedman, D. P., and D. S. Wise, "CONS should not evaluate its arguments," in Michaelson and Milner (eds.), *Automata, Languages, and Programming*, Edinburgh University Press, 1976, pp. 257-284.
10. Friedman, D. P., and D. S. Wise, "Aspects of applicative programming for file systems," *Sigplan Notices*, Vol. 12, No. 3, pp. 41-55, March 1977.
11. Friedman, D. P., and D. S. Wise, "Aspects of applicative programming for parallel processing," *IEEE Trans. on Computers*, Vol. C-27, No. 4, April 1978, pp. 289-296.
12. Henderson, P., and J. H. Morris, Jr., "A lazy evaluator," *Proc. Third ACM Conference on Principles of Programming Languages*, 1976, pp. 95-103.
13. Kahn, G., and D. MacQueen, "Coroutines and networks of parallel processes," *Proc. IFIP '77*, 1977, pp. 993-998.
14. Keller, R. M., "Look-ahead processors," *Computing Surveys*, Vol. 7, No. 4, December 1975, pp. 177-195.
15. Keiler, R. M., "Semantics of parallel program graphs," University of Utah, Dept. of Computer Science, Tech. Rept. UUCS-77-110, July 1977.
16. Keller, R. M., "Denotational models for parallel programs with indeterminate operators," E. J. Neuhold (ed.), *Formal description of programming concepts*, North-Holland, 1978, pp. 337-366.
17. Keller, R. M., "An approach to determinacy proofs," University of

- Utah, Dept. of Computer Science, Tech. Rept. UUCS-78-102, March 1978.
18. Keller, R. M., G. Lindstrom and S. Patil, "An architecture for a loosely-coupled parallel processor," University of Utah, Dept. of Computer Science, Tech. Rept. UUCS-78-105, October 1978.
 19. McCarthy, J., "Towards a mathematical science of computation," *IFIP '62 Proc.*, 1963, pp. 21-28.
 20. Mago, G. A., "A network of microprocessors to execute reduction languages," Department of Computer Science, University of North Carolina, March 1978.
 21. Patil, S., "An abstract parallel-processing system," M. S. Thesis, MIT Dept. of Electrical Engineering, June 1967.
 22. Ritchie, D. M., and K. Thompson, "The Unix time-sharing system," *CACM*, Vol. 17, No. 7, July 1975, pp. 365-381.
 23. Swan, R. J., S. H. Fuller and D. P. Siewiorek, "Cm*—A modular, multi-microprocessor," *AFIPS Proc.* Vol. 46, June, 1977, pp. 637-644.

A prototype data flow computer with token labelling

by IAN WATSON and JOHN GURD

University of Manchester
Manchester, England

INTRODUCTION

Computer users in many areas of scientific study appear to have an almost insatiable desire for processing power. Many of the computations exhibit a high degree of parallelism which is not exploited in conventional computer structures. Considerable interest has been shown in parallel computer architectures in the last decade in order to make use of this property.

The two main approaches toward this goal have become known as SIMD (array processor) and MIMD (multiprocessor) architectures which are typified by the ILLIAC IV and the C.mmp systems respectively. It is not appropriate to describe these architectures in detail here, but it should be noted that a number of inadequacies have come to light as these systems have been applied to many problems. There are two basic difficulties which appear in both architectures in slightly different guises. The first concerns the arbitrary choice of a fixed number of processing units in an architecture (64 in the case of ILLIAC IV, 16 in the case of C.mmp). The difficulty that is occasioned by this choice derives from the fact that it is often inconvenient, if not impossible, to organize the problem to be solved so that it is exactly the "width" of the architecture. The second problem arises when sections of essentially serial programs are processed. The problems in SIMD architecture are clear; in MIMD systems the difficulty usually appears when a critical area of memory becomes "locked-out" while many processors are trying to access it. Evidence is accumulating that relatively little of the potential speedup of SIMD and MIMD systems can actually be achieved for a broad spectrum of problems: Minsky's conjecture,¹ Flynn's analysis² and Enslow's view of operating system overheads in multiprocessors³ all attest to this opinion.

More recently, interest has arisen in data driven computation for the expression of programs (known as Data Flow). A computation is expressed as a data dependent directed graph with nodes representing computational operations and the arcs representing the flow of data. The nodes become ready for execution when all their input data are available. The expression of parallelism in terms of data dependencies rather than in spite of them, leads to a far more natural and flexible picture of parallel program execution.

The theoretical basis of data driven computation can be traced to the paper of Karp and Miller in 1966.⁴ An extension of this line into machine architecture was pioneered by the team led by J. B. Dennis at MIT.^{5,6}

The architecture described in this paper follows the same broad principles as the MIT work but introduces the concept of token labelling as a mechanism to support re-entrant code structures. It is believed that this leads to a more efficient use of storage resources and provides a greater potential for the utilization of parallelism. Another important aspect of the design is the use of pseudo-associative store to perform token matching; this is achieved by reducing the nodes to a simple machine instruction level with a maximum of two inputs.

DATA FLOW PRINCIPLES

In order to illustrate the principles of a directed graph representation, consider the 'Butterfly' of a Fast Fourier Transform shown in Figure 1. The expressions being evaluated are:

$$A' = A + C \cos \alpha + D \sin \alpha$$

$$B' = B - C \sin \alpha + D \cos \alpha$$

$$C' = A - C \cos \alpha - D \sin \alpha$$

$$D' = B - D \cos \alpha + C \sin \alpha$$

These can be built up from six simple nodes performing the functions of addition, subtraction, multiplication, sine, cosine and duplication of a value. A node becomes executable when its input values (normally called tokens) are available. Following this principle it can be seen that, assuming at least four execution units, the computation can be performed in seven steps as indicated by the number adjacent to the nodes. The total number of nodes in the graph is 21 and therefore the computation has an average parallelism of three.

The graph shown uses only simple operational nodes. In order to specify a more comprehensive computational model it is necessary to introduce a much wider set of primitives which provide control functions such as conditional branch-

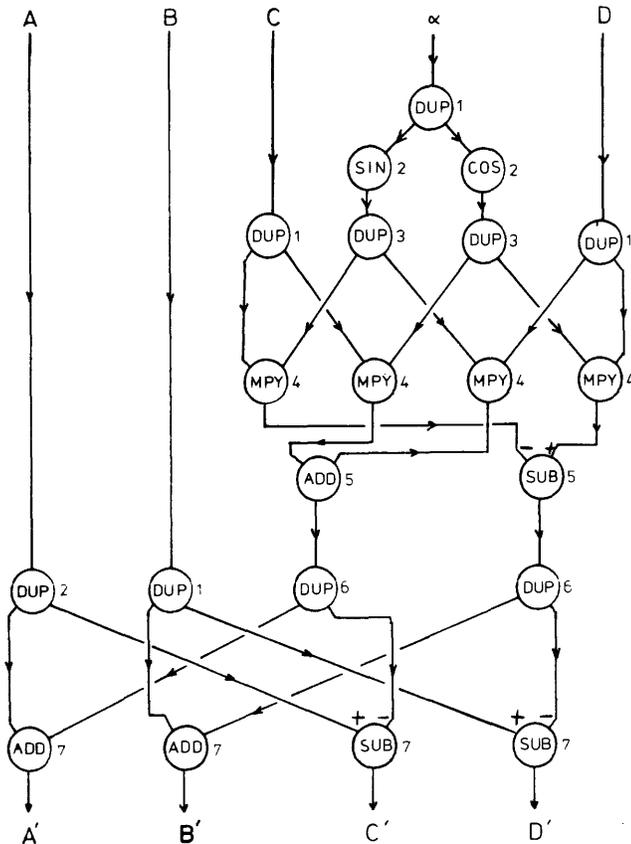


Figure 1—Graph of an F.F.T. Butterfly.

ing. A complete description of a practical order code is beyond the scope of this paper but can be found in other literature.⁷

PROBLEMS OF RE-ENTRANCY

A complete Fast Fourier Transform is an iterative computation performed on a large array structure. Ignoring, for the moment, practical implementation mechanisms, consider the problems associated with using the graph of Figure 1 as a complete description of the arithmetic computation. The input values are not simple scalars but arrays of the form $A[1] \dots A[N]$, $B[1] \dots B[N]$, etc., for each iteration. In a flexible parallel machine, there is no reason why further iteration steps may not proceed even though one iteration has not been completed for all points in an array. The inputs to the graph can then be of the general form $A[n]_i$, $B[n]_i$, etc., where n represents the array index and i the iteration level. It is then no longer possible to declare a node executable by the presence of any two tokens on its input as they may belong to totally different parts of the computation.

There are three possible solutions to this problem:

1. The use of a re-entrant graph is prohibited; each point of the array and each stage of the iteration must be described by a separate graph.

2. The use of the graph is limited by allowing one token to reside on an arc of the graph at any one time. This can be achieved by only allowing a node to become executable when both its input tokens are present *and* no token exists on its output arc.
3. The tokens are assumed to carry with them their index and iteration level as a label. The rules are extended to require two tokens with the same label before a node can be declared executable.

The first mechanism may require large amounts of code storage and, in problems where the iteration depth is only known at run time, the necessity for dynamic code generation. It is believed that this could be a significant overhead in a practical system. The second implies a sequential but pipelined use of the code. This will severely reduce the exploitation of potential parallelism.

The labelling method permits the use of pure static code and enables maximum usage of any parallelism which exists in the problem specification. This is clearly at the expense of the extra information which needs to be carried by each token.

Assuming that it is desirable to utilize the maximum amount of parallelism available, then the limitation of one token per arc must be rejected in the general case. Of the remaining two methods, it is our contention that token labelling will result in much lower overheads in a practical system.

One major use of re-entrant code, that of the procedure, has not yet been mentioned. Parallel invocation of procedures can be supported by ensuring that a unique identifier is allocated to each input token at the procedure call.

It is not possible in this paper to describe all the implications of token labelling. Techniques are required to support nested iterations, recursive procedure calls and index manipulation in data structures. These can be achieved by nodes which operate on the label fields of tokens rather than the data values. More comprehensive information on the labelling concept can be found in descriptions of our own work⁷ and that of Arvind and Gostelow at the University of Irvine, California, U.S.A.⁸

THE MACHINE ARCHITECTURE

Although it may be profitable to use a Data Flow computational model to describe parallel activities in a wide range of architectures, it is likely that a hardware structure which reflects the model directly will result in the most efficient implementation.

The basic elements of a Data Flow machine must contain parallel processing units which perform the nodal operations, a stored description of the directed graph and a mechanism for collecting and matching tokens. The first two requirements can be realized using standard processing and storage techniques; the matching operation is more complex.

During a computation, tokens will be produced by processing units which must await the arrival of other tokens with identical labels which are directed to the same node.

They can then form an executable package to be allocated to a free processing unit. This storage and matching function can be simplified greatly if the maximum number of node inputs is limited to two; an associative storage technique can then be used.

Figure 2 shows the basic architecture of the Manchester Data Flow Prototype which uses these principles.

The Instruction Store is a random access memory which holds the directed graph description. Each entry is in the form of a nodal operation and the addresses of the subsequent nodes to which the output token(s) will be directed. Note that in this practical implementation, nodes are able to specify two output destinations for their result. It is also possible to specify a literal value as one input to the node.

The processing units are microprogrammed microprocessors with a distribution and arbitration system. The distribution system, on receipt of an executable package, will select any processor which is free and allocate the nodal operation. The arbitration system controls the output of tokens from the processors.

The switch provides input and output for the system. Initial tokens are directed to the starting nodes of the computation. A special destination address in the final nodes of the graph allows tokens to be output.

The Token Queue is a First In First Out buffer which equalizes data rates around the system.

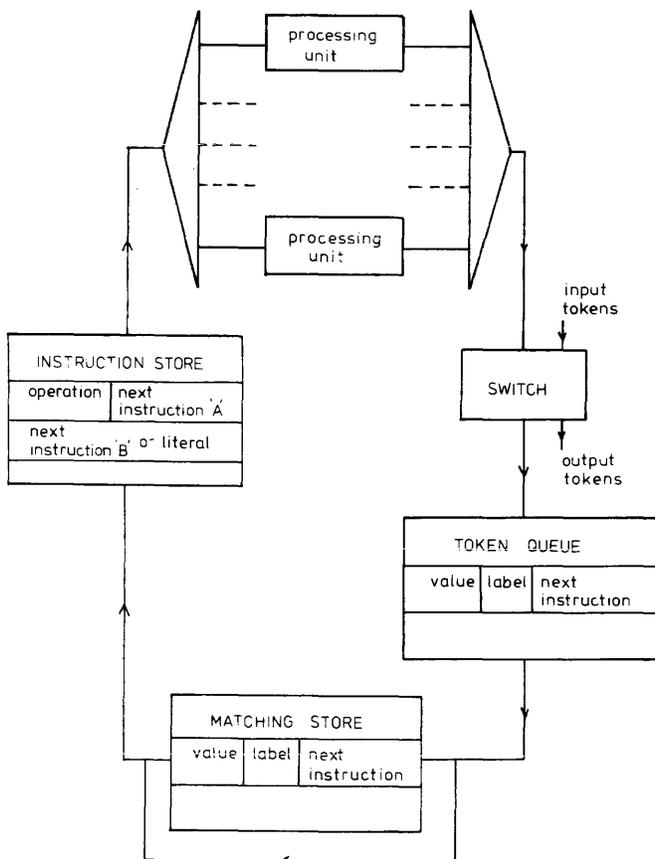


Figure 2—Basic architecture.

The Matching Store is associative in nature, although it is implemented using conventional random access store with hardware hashing techniques. The associative field is formed from a concatenation of the label and next instruction fields; the value field is the token value. There is a requirement in a practical instruction set for single input nodes which require no matching operation. A control digit in the next instruction information allows a bypass of the Matching Store in this case.

In order to execute a program, the graph description is entered in the instruction store. The initial data tokens are input to the Token queue via the switch. As an example of this, Figure 3 shows a very simple graph to form the sum of the product of two pairs of numbers, together with an indication of the Instruction Store entries and initial token formats. The label is not used in this case and is thus set to zero.

The tokens, on reaching the front of the Token Queue, can access or bypass the Matching Store dependent on the Next Instruction Information. An access to the Matching Store will cause an associative search of the store. If a token is found with the same label and Instruction Address it is removed to form a token pair. If no match is found, the incoming token is written to the store.

Token pairs from the Matching Store, or single tokens which have bypassed it, now access the Instruction Store

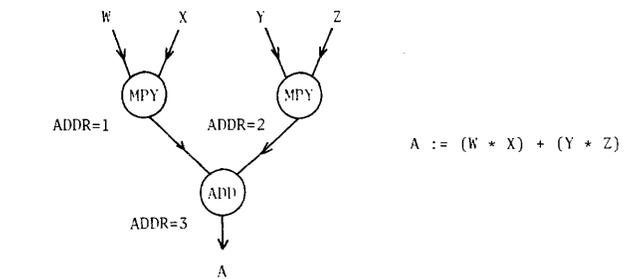


Figure 3—A simple graph and its machine representation.

and form an executable package. This is distributed to any free processor for execution. Tokens produced by the processors are entered on the back of the Token Queue via the Arbitration Unit and Switch.

This operation proceeds in a pipelined manner around the ring structure until the computation is complete and any output tokens have been produced. Each unit communicates with its successor and predecessor by a two-way handshake interface. This ensures that if, for example, all processing units are busy, the ring operation is suspended until the necessary resources become available.

In order to facilitate the description of the architecture, certain simplifications have been made. It does, however, contain the underlying principles of the prototype Manchester Data Flow machine.

IMPLEMENTATION OF THE MACHINE UNITS

The token queue

The Token Queue is implemented using Random Access Memory with pointers indicating the front and back of the queue. The store is divided into two independent stacks which permit concurrent read and write operations. Although it is not intended to include storage hierarchies in the prototype design, it is felt that use could be made of shift register devices if a much larger queue were required.

The matching store

The pseudo-Content Addressable Memory required for the matching operation uses hardware parallel hashing techniques similar to those described by Goto and Ida.^{9,10} However, as the order of propagation of results around the ring is unimportant, use can be made of an overflow hashing mechanism rather than a re-hash. The advantage of this is that the average store access time can be reduced and a backing store structure can be introduced.

The instruction store

No special techniques are necessary in the Instruction Store. It is simply a linear Random Access Memory which is addressable by the Next Instruction Information field.

The processing units

Ten Processing Units are constructed from Schottky 'bit-slice' microprocessor elements. They contain alterable microprogram store in order that the instruction set can be changed with ease. All processors are connected to a common bus with a distribution mechanism to allocate executable instructions to free processors and allow the output of results. A preliminary study of arithmetic operations has indicated that an average instruction execution time of $3\mu\text{S}$ can be achieved in each processor.¹¹

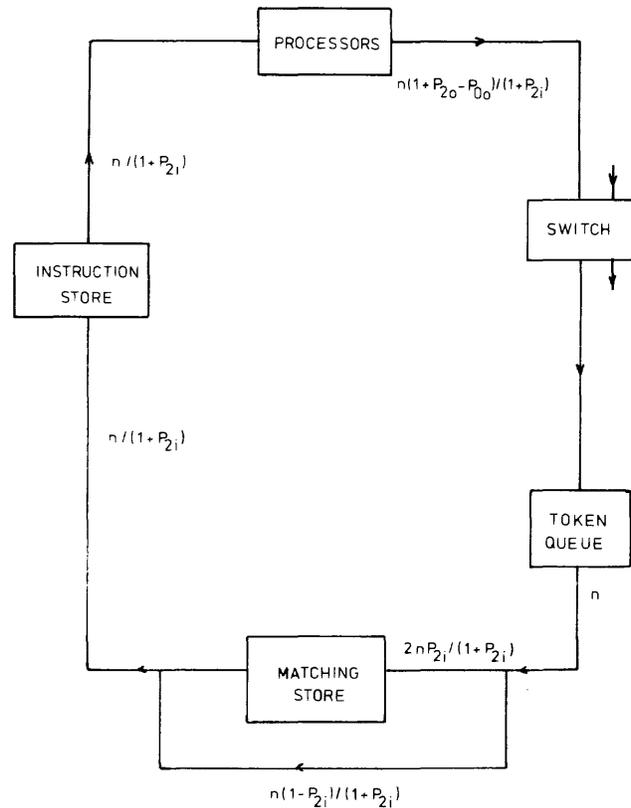


Figure 4—Data rates.

SYSTEM PERFORMANCE

The processor speed has already been stated as an average instruction time of $3\mu\text{S}$. If all ten processors can be utilized fully then an instruction execution rate of 3.3 MIPS can be achieved. It is necessary to estimate the storage speeds which are consistent with this.

With reference to Figure 4 we define four values:

- n Tokens per unit time read from the token queue.
- P_{2i} Probability of an instruction requiring two inputs.
- P_{2o} Probability of an instruction producing two outputs.
- P_{0o} Probability of an instruction producing zero outputs.

The following relationships can be derived:

- | | |
|--------------------------|------------------------------------------------------------------------|
| $2nP_{2i}/(1+P_{2i})$ | Tokens will access the matching store. |
| $n(1-P_{2i})/(1+P_{2i})$ | Tokens will bypass the matching store. |
| $nP_{2i}/(1+P_{2i})$ | Pairs of tokens will exit from the matching store. |
| $n/(1+P_{2i})$ | Instruction accesses will be made, and executable instructions formed. |

$n(1+P_{z_0}-P_{o_0})/(1+P_{z_1})$ Tokens will be produced by the processors, pass through the Switch, and be written to the token queue.

The constraint imposed by the processing unit is that:

$$n/(1+P_{z_1})=3.3 \times 10^6$$

For a typical program it appears, from simulation experience, that the following values of the probabilities are realistic for typical problems.

$$P_{z_1}=0.5$$

$$P_{z_0}=0.6$$

$$P_{o_0}=0.1$$

Using these we see that:

$$n=3.3 \times 10^6 \times 1.5 = 4.95 \times 10^6$$

From this we can derive the following required operation times for the units:

1. *Token Queue Read*—Given by $1/n=202nS$.
2. *Matching Store Access*—A token reaching the matching store, requires one read cycle plus one write cycle whether or not it is a successful match. Therefore the times are given by

$$\begin{aligned} \text{Read time} + \text{Write time} &= (1+P_{z_1})/2nP_{z_1} \\ &= 303nS \end{aligned}$$

3. *Instruction Store Read*—Given by $(1+P_{z_1})/n=303nS$
4. *Switch Operation*—Given by $(1+P_{z_1})/n(1+P_{z_0}-P_{o_0})=202nS$
5. *Token Queue Write*—Same as Switch Operation= $202nS$

From these figures it can be seen that the storage units must have access times of the order of 200nS to maintain the execution rate of ten processors. This speed can be readily achieved by low cost MOS storage devices currently available.

ARCHITECTURAL EXPANSION

In order to obtain very high speeds from any parallel computer system it is necessary to exploit parallelism in processing, storage and information transfer. The critical "bottlenecks," particularly in MIMD machines usually appear in the form of crossbar switches, common highways or common stores through which all the processors in the system may wish to communicate. The reason for this can be traced to the need for a processor to demand, and require rapidly, data from any other part of the system. In addition, there is a necessity to control access to data which has yet to be formed; this can also introduce significant communication overheads.

In a data driven environment, a processor is not required

to perform a section of a computation until all the data are presented to it as an executable package. Rapid data access to other parts of the system and access control are then unnecessary. This suggests an architecture containing a mechanism which accepts tokens from all processors and then distributes them to parallel but independent storage and processing resources.

In the Manchester architecture, this can be achieved by extending the input/output switch to become the intersection point of many identical "rings." A strategy is adopted, using the label and instruction address fields, which distributes tokens from different parts of the computation across the parallel rings.

The switch could be a crossbar, but this would suffer from the same problems as in a MIMD machine. Due to the pipelined nature of the rings, the requirement is for a high throughput rate rather than a fast transfer across the switch. A parallel pipelined structure can therefore be used which does not create a "bottleneck" in the system. The prototype will only contain a single "ring" as described previously. The switch will be constructed to enable the connection of further identical rings at a later date.

A more comprehensive description of the implementation and estimated performance of the multi-ring architecture can be found in other literature.⁷

PROGRAMMING A DATA FLOW MACHINE

This section is intended to provide a brief outline of the methods available for programming a Data Flow machine.

It would seem that the natural way of expressing a directed graph would be via a graphical language, and such languages have been investigated.¹² However, textual languages are far more familiar and it can be argued that features such as data structures are easier to express into a textual form.

One approach is to take a conventional language and translate it into a Data Flow graph. The principles involved in such a translation were originally suggested by Miller and Rutledge.¹³ More recently, Whitelock¹⁴ has developed an experimental compiler for a subset of PASCAL which compiles code for the Manchester machine.

Another class of languages, the Single Assignment Languages, are more naturally suited to the expression of parallelism in a Data Flow form. Some of these languages, for example Id,¹⁵ TDFL¹⁶ and LAPSE,¹⁷ have been developed specifically for this purpose. Other similar languages such as LUCID¹⁸ have developed independently with emphasis on the proof of correctness of programs.

It is too early to forecast with certainty which of these approaches will be most fruitful. The Manchester single assignment and conventional compilers produce code for an architecture simulator; they are being evaluated at the present time.

It is certain that a Data Flow machine can be programmed without great difficulty, with no requirement for a knowledge of the underlying architecture which supports the ex-

ecution. The exact form of languages which will gain acceptance is yet to be decided.

CONCLUSIONS

The expression of computations in data dependent graphical form provides a natural method of determining the parallelism which is present. In the design of practical execution mechanisms, problems exist in the implementation of re-entrant code. An architecture has been proposed which attempts to overcome many of these problems by introducing a label which is carried by every data token. This, together with a pseudo-associative token matching store, results in an architecture which can be constructed at low cost using components which are readily available.

A prototype machine is in the process of design and its performance is being evaluated by simulation. It is not intended that this prototype should be of very high speed but should provide a research vehicle for studies of the potential of Data Flow machines for the solution of real problems. The architecture is extensible by using copies of the basic design and this will proceed if the initial investigations are successful.

No attempt has been made in this paper to address the problems of programming a Data Flow machine. Research is, however, being conducted both at Manchester and many other places which promises to produce high-level languages which will allow a machine independent formulation of parallel programs.

REFERENCES

1. Minsky, M. and S. Papert, "On Some Associative, Parallel and Analog Computations," *Associative Information Techniques* (E. J. Jacks, ed.), Elsevier, 1971.
2. Flynn, M. J., "Some Computer Organizations and their Effectiveness," *I.E.E.E. Transactions on Computers*, Vol. C-21, No. 9, September 1972, p. 948.
3. Enslow, P. H., "Multiprocessor Organization—A Survey," *A.C.M. Computing Surveys*, Vol. 9, No. 1, March 1977, p. 103.
4. Karp, R. M. and R. E. Miller, "Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing," *SIAM J. Applied Mathematics*, Vol. 14, November 1966, pp. 1390-1411.
5. Dennis, J. B. and D. P. Misunas, "A Preliminary Architecture for a Basic Data Flow Processor," *Proc. 2nd Ann. I.E.E.E. Symposium on Computer Architecture*, January 1974, p. 126.
6. Dennis, J. B., D. P. Misunas and C. K. Leung, "A Highly Parallel Processor Using a Data Flow Machine Language," *CSG Memo 134*, Laboratory for Computer Science, M.I.T., January 1977.
7. Gurd, J. R., I. Watson and J. R. W. Glauert, "A Multilayered Data Flow Computer Architecture," Internal Report, Development of Computer Science, University of Manchester, February 1978.
8. Arvind and K. P. Gostelow, "A Computer Capable of Exchanging Processors for Time," *Information Processing 77*, North Holland, 1977, p. 849.
9. Goto, E. and T. Ida, "Parallel Hashing Algorithms," *Information Processing Letters*, Vol. 6, No. 1, February 1977.
10. Ida, T and E. Goto, "Performance of a Parallel Hash Hardware with Key Deletion," *Information Processing 77*, North Holland, 1977.
11. Zurawski, J., "The Design of a Processor Array for a Data Flow Architecture," M.Sc. Dissertation, Department of Computer Science, University of Manchester, England, 1977.
12. Dennis, J. B., "First Version of a Data Flow Procedure Language," *Lecture Notes in Computer Science*, Vol. 19, 1974, p. 342.
13. Miller, R. E. and J. D. Rutledge, "Generating a Data Flow Model of a Program," *IBM Technical Disclosure Bulletin*, Vol. 8, No. 11, April 1966.
14. Whitelock, P. J., "A Conventional Language for Data Flow Computation," M.Sc. Dissertation, Department of Computer Science, University of Manchester, October 1978.
15. Arvind, K. P. Gostelow and W. Plouffe, "The (Preliminary) ID Report," *Technical Report 114*, Department of Information and Computer Science, University of California, Irvine, May 1978.
16. Ackerman, W. B., "Preliminary Data Flow Language," *CSG Note 36*, Laboratory for Computer Science, MIT, March 1978.
17. Glauert, J. R. W., "A Single Assignment Language for Data Flow Computing," M.Sc. Dissertation, Department of Computer Science, University of Manchester, January 1978.
18. Ashcroft, E. A. and W. W. Wadge, "Lucid, a Nonprocedural Language with Iteration," *CACM*, Vol. 20, No. 7, July 1977, p. 519.

A view of dataflow*

by KIM P. GOSTELOW and ROBERT E. THOMAS

University of California
Irvine, California

INTRODUCTION

In 1946 John von Neumann outlined an organization for computers¹ that has dominated the languages and architecture of machines to this day—the familiar sequential, one-word-at-a-time instruction stream which modifies the contents of a memory. Although the von Neumann model has proved to be a viable and powerful approach to computation, we have chosen to explore other models of computation to determine if they offer advantages in ease of programming, exploitation of concurrency and performance. A primary motivation is new technology such as large scale integration (LSI) which has greatly expanded the range of choice in computer design.

Dataflow is an alternative model of computation which is particularly promising. The basic principles of dataflow are asynchrony and functionality, and thus are in distinct contrast to the von Neumann model. Readers familiar with look-ahead processors² such as the IBM 360/91 and the CDC 6600/7600 will find that the principles of dataflow are not new. However, our goals in exploiting the principles of dataflow are of a more fundamental nature than the goals of the above systems. Rather than using dataflow simply to improve the performance of von Neumann processors, we have adopted the semantics of dataflow as the base semantics of our system. A primary reason for this direction is a desire to explore the full generality of dataflow. Another reason, perhaps of greater importance, is our impression that the functional nature of dataflow simplifies the semantics of programming languages and thus may reduce the cost of software (especially in the case of multiprocessor systems). Our approach is first to design a fully-integrated system before attempting to construct hardware. This includes the design of a base machine language, a preliminary high-level language,³ a user protection facility,⁴ and a high-level exception handling facility,⁵ all of which are based on the semantics of dataflow.

The following sections discuss details of the principles of dataflow with emphasis on a method of interpretation developed at Irvine. The general version of this interpreter is known as the *unfolding interpreter* and is described in the following section. (Details on a specific unfolding interpreter

are available elsewhere.³) The third section presents implementation techniques for dataflow systems while the fourth section discusses some principles of multiprocessor design currently being developed.

BASIC PRINCIPLES OF DATAFLOW AND THE UNFOLDING INTERPRETER

The present section concentrates on the logical implications of dataflow semantics without regard to physical implementations, efficiency, etc. These latter topics are discussed in later sections.

Asynchrony and Functionality

The following introduces dataflow by showing the correspondence between constructs in the high-level dataflow language *Id* (for *Irvine dataflow*) with schemata in a graphical dataflow machine language. The goal is twofold—first to show by example that programs need not be written in dataflow machine language, and second, to provide some intuition for understanding the execution of dataflow programs. We wish to emphasize that our purpose is to present the basis of dataflow and not to discuss the syntax of a particular dataflow language (*Id*), or the details of a particular machine language.³

Consider the following *Id* constructs:

```
s ← (initial sum ← 0
     for i from 1 to n do
       new sum ← sum + f(i)
     return sum);
```

 (2.1)

```
procedure sum (i, k)
(return (if i > k then 0
       else f(i) + sum(i + 1, k)));
```

 (2.2)

```
s ← sum(1, n);
```

both of which can be expressed in mathematical terms as

$$s = \sum_{i=1}^n f(i)$$

Statement (2.1) is an assignment statement whose right-hand side is an *Id* loop expression. The statements in (2.2) are a

* This work was supported by NSF Grant MCS76-12460: The UCI Dataflow Architecture Project.

procedure definition followed by an application of that procedure. Each of these constructs has a number of inputs—a value for n , a definition for the function procedure f , and in the case of the sum procedure definition, the value of i . In addition, both (2.1) and (2.2) produce the same result. We abstract these two definitions of “sum” by considering each to be a “black box” as shown in Figure 1a. Each dark spot in the figure represents the presence of a data item referred to as a *token*. For now, we can consider a data item to be an instance of an integer, real, or boolean value.

The mechanics of computation within a black box can be ignored as long as three conditions are met: 1) a complete set of input values (i.e. tokens) is consumed, 2) the computation within the box has no effect on other computations except perhaps to compete for resources (i.e. there are no semantic side-effects), and 3) a complete set of result tokens is always produced if the computation terminates. A black box meeting these requirements is a *function*. The basis of dataflow is the definition and operation of interconnected functions. One way, for example, to interconnect functions is by composition (Figure 1b). Other dataflow interconnection schemes including cycles have been devised^{3,6,7} but are not discussed here.

As opposed to the sequential, one-instruction-at-a-time memory cell semantics of the von Neumann computer, the basic principles of dataflow are:

1. Operations execute when and only when the required operands are available (asynchrony).
2. Operations are functions (there are no side-effects).

These principles imply that the order of execution of two functions, such as e and g in Figure 1b, is irrelevant since the computations internal to e and g cannot interact. Thus e and g can be computed concurrently. Such concurrency, present in the interconnection graph itself, is called *static parallelism*. A more interesting example of the asynchrony achievable in dataflow occurs when a function is executed more than once, either by iteration or by recursion (for example, function f in (2.1) and (2.2) previously). As shown in Figure 1c, suppose that a dataflow machine replicates the function f and its input and output lines for as many times as f is executed. Since f has no side-effects, each copy of f can be computed in any order or concurrently. This concurrency is called *dynamic parallelism* since the concurrency potential depends on the number of repetitions (determined at execution time) of the function f . Dynamic

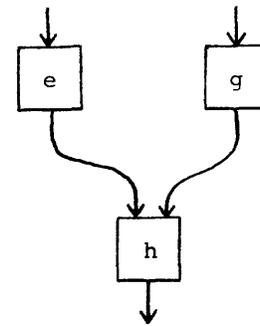


Figure 1b—Composition of functions

parallelism is of particular interest because it can affect the time complexity of an algorithm. For example, suppose the time complexity of function f in (2.1) and (2.2) is $O(m)$ (i.e. assume that f has an additional parameter m). Then on a sequential machine the time complexity of either (2.1) or (2.2) would be $O(nm)$. However, on a dataflow machine capable of dynamic parallelism, the processing time complexity would be $O(n+m)$ because the time required is $O(n)$ to generate the n instances of f , plus $O(m)$ to simultaneously compute all instances of f (assuming $O(n)$ processors are available), plus $O(n)$ again to sum the resulting values. The total is $O(n+m+n)=O(n+m)$, where for simplicity we have ignored some important considerations such as communication conflicts.

It is important to note that the input and output lines of a function are replicated for as many times as the function is executed. This implies that at most one token will ever travel on any given line instance, thus preserving functionality. In addition, the situation shown in Figure 2 is precluded by the “single-assignment rule” present in the high level language.

The replication of f and its input and output lines does not alone ensure dynamic parallelism since data dependencies in the program may inhibit it. For example, if the previous definition of sum is changed to

```

s ← (initial sum ← 0; x ← 5
for i from 1 to n do
  new x ← f(x);
  new sum ← sum + new x
return sum)
    
```

that is

$$s = \sum_{i=1}^n x_i$$

where $x_i = f(x_{i-1})$ for $x_0 = 5$ and $1 \leq i \leq n$

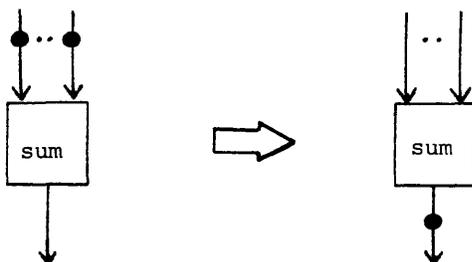


Figure 1a—Abstraction of the Id construct sum

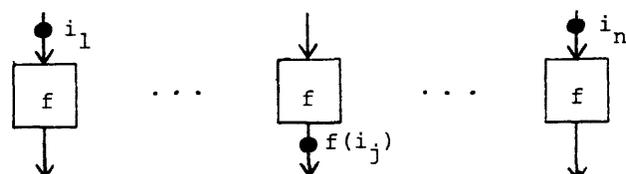


Figure 1c—Instances of function f

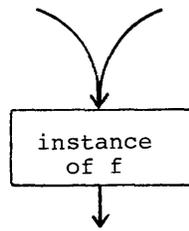


Figure 2—An illegal connection

then the input to f depends on the value computed by the previous instance of f . That is, the instances of f must be computed sequentially.

The Unfolding Interpreter

Using the simple notions of asynchrony and functionality discussed above, we present an interpreter which manages a *context* for each value produced and consumed in the system. The purpose of context management is to logically separate and direct the values to the proper instance of each function.

At this point we note that other dataflow interpreters have been defined^{6,7} which rely on either fixed-size buffers and request/acknowledge communication between functions, or the assumption of an unbounded FIFO queue between each interconnected function. The unfolding interpreter is capable of far more asynchronous operation than these other interpreters because of the function copying it performs.

Each execution instance of a function is called an *activity* and is uniquely identified by an *activity name*. An activity name comprises two parts denoted $u.l$ where u is the context part and l is a unique label referencing the description of the function to be computed by that activity. In addition, the referenced function description specifies the destination labels to be used for transmission of result tokens. A *dataflow object program* is a set of labeled function descriptions. The actions of the interpreter can now be stated:

1. Tokens generated by the execution of activities are grouped by activity name.
2. When the input tokens to an activity become available, the activity is executed according to its description.
3. Output tokens are produced by tagging the values resulting from the execution of an activity with the destination's activity name. The u part of the destination activity name is derived from the u part of the activity name of the producing activity according to a set of rules (some examples are given below). The l part is derived from the output destination information which is part of the description of the function executed by that activity. Note that the act of computing a destination activity name is equivalent to creating a "logical line instance" extending from the producing activity to the destination activity.

To illustrate, consider the dataflow object program in Figure

3a. Let the activity name of an instance of e be $u.l$. The rule for function composition says that the context of the output is identical to the context of the input, and the label of the output is specified by the description of the function being executed, i.e. the program code. This results in activity $u.l$ producing an output token with destination activity name $u.t$. Now consider the case when an activity itself comprises (smaller) activities, for example a procedure call, which is provided for by the base machine language primitives A , $BEGIN$, END , and A^{-1} as shown in Figure 3b. This figure shows the creation of a new set of activities resulting from the application (call) of procedure f . Note that the description of procedure f is one of the input values to the procedure application box. (We will not be concerned here with the representation of procedure values.) Activity name generation for procedure call is as follows:

- *The A (activate) primitive*—Assume that the activity name of an instance of A is $u.l$. Since the procedure call represents a change in context, the A primitive "stacks" the context part u within the new activity name, thereby creating a unique context for the activities within f . The activity name produced is $u'.beginf$ where $u'=u.l$. Also by convention, the A primitive groups into one vector value all of the input arguments so that exactly one input argument token is always delivered to the newly created instance of f .
- *The $BEGIN$ primitive*—The purpose of $BEGIN$ is to distribute the input arguments to the activities within f with no further change in the context u' .
- *The END primitive*—The END primitive "unstacks" the context u' to reveal the outer context and the label l . It then constructs the activity name $u.t$ (the activity to which the result of the procedure is to be returned) which can be accomplished in a number of ways. For example, t could be computed from l according to an agreed-upon rule. Also, by convention, the END primitive combines all output values onto one token for transmission to the A^{-1} activity.
- *The A^{-1} (terminate) primitive*—The purpose of the A^{-1} primitive is to distribute the results of f to activities in the outer context with no further change in the context u .

Though not presented here, other schemata for the un-

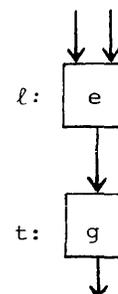


Figure 3a—a Composition of functions

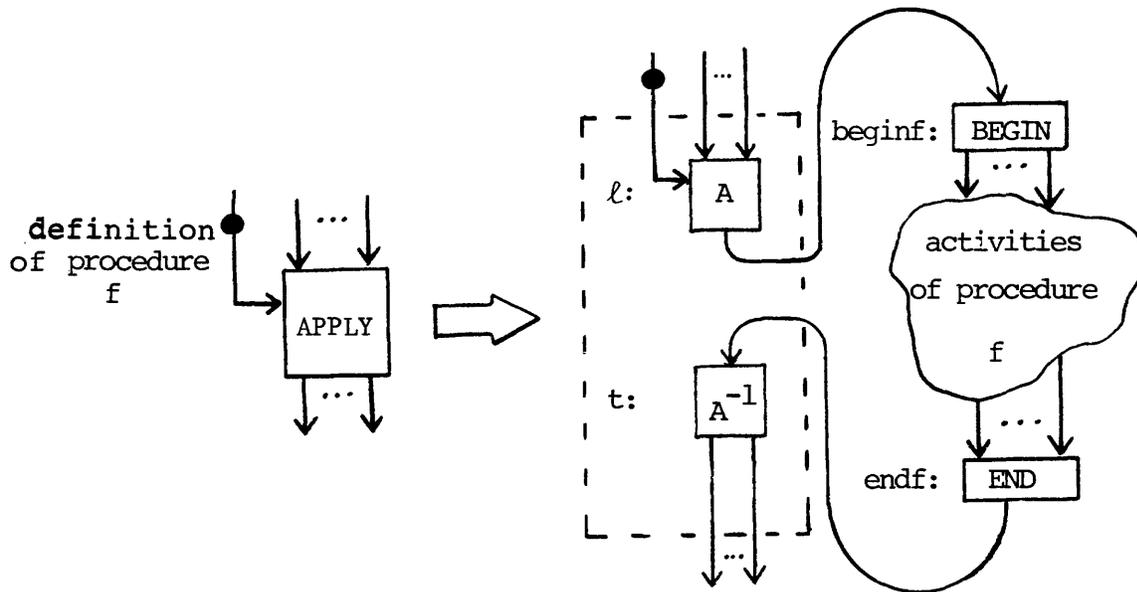


Figure 3b—Application of procedure f

folding interpreter have been devised.³ In particular there is a *loop schema* which “unfolds” the loop body (including nested loops) to expose dynamic parallelism. (The unfolding interpreter gets its name from this capability.) The loop schema depends on adding a new field, *i*, to the context part of an activity name to yield (u.i).*i*. Like recursion, each iteration of a loop exists in a distinct context generated simply by incrementing the *i* field in the activity name.

IMPLEMENTATION SCHEMES

In this section we discuss techniques for efficient implementation of dataflow. Although von Neumann computers may be used to implement these techniques within a dataflow system, we rigidly maintain that the semantics of dataflow are the only semantics visible external to the system—a principle we consider vital to the success of dataflow.

Dataflow Structures and Memory

Operation of the unfolding interpreter requires many copies of program code. Logical copies are sufficient and can be created simply by copying the pointer to a physical copy since all code (and data) is read-only. (Note that the label *l* in an activity name is equivalent to a pointer.) Of course in a multiprocessor environment, having just one physical copy may imply a bottleneck. In this case, we consider it the responsibility of a particular implementation to selectively make physical copies (in distinct memories) to reduce the bottleneck.

Similar remarks hold for the transmission and replication of values larger than simple integers, reals, booleans, etc. The need for logical copies is especially evident when a

value, such as an entire matrix, is transmitted between two functions and the receiving function utilizes only a small part and discards the rest. Also, a common programming task is the production of a data object which differs in only small ways from another (perhaps large) input data object. Because dataflow values can never be modified, an entire new object must be created, making the straightforward copy-all approach quite expensive.

Dennis has shown⁶ that the amount of copying can be reduced by properly defining a SELECT and APPEND operation on “structured” data residing in a conventional memory. A *dataflow structure* is a set of (selector:value) pairs where a *selector* is an integer or string and a *value* is any dataflow value (including another structure). A dataflow structure is always a tree (e.g. Figure 4a). The SELECT function (subscripting) has two arguments, a structure value and a selector, and it yields the value at the specified selector. The APPEND function has three arguments: a structure value, a selector, and a value to be appended to the given structure at the specified selector. APPEND does *not* modify the given structure but instead makes a logical copy of it with the new selector and value appropriately placed. This

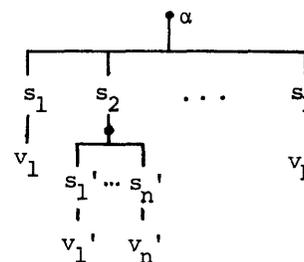


Figure 4a—A dataflow structure

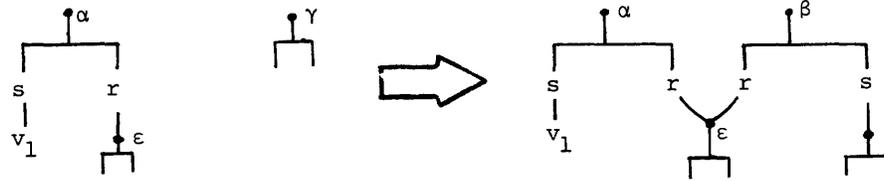


Figure 4b—APPEND (α, s, γ) $\rightarrow\beta$

can be implemented (with pointers) such that a physical copy need be made only of the “top level” of the original structure value. Thus sub-structures can be shared between any number of structure values without violating dataflow semantics. A simple example is given in Figure 4b where both structures (logical trees) physically share the sub-structure at selector r.

Since the definition of dataflow structures precludes the construction of internal cycles, a simple reference count scheme can be used to reclaim structures no longer needed. The reference count method is also helpful in detecting the special case of an APPEND to a structure when there is only one logical copy of that structure (i.e. reference count equals one). In this case APPEND can quickly produce its output by simply updating the old structure in place.

In the following we consider several explicit representations for dataflow structures. When a dataflow structure has contiguous integer selectors, a vector of contiguous memory words may be used where one value (or a pointer to a value) is stored per memory word. This is termed *array* representation. It is easy to see that in array representation a (one level) SELECT can be done in constant time while APPEND requires $O(n)$ time (for copying), where n is the number of words in the result memory vector.

A second representation, termed *selector vector*, is a fairly compact representation when string or sparse integer selectors appear in the dataflow structure. Again a contiguous memory vector is used but the (ordered) selectors are explicitly stored with the values. SELECT can then be done in $O(\log n)$ time using a binary search while APPEND requires $O(n)$ time where n is the number of selectors.

A third representation of a dataflow structure is a modification to a “balanced” tree scheme such as an AVL tree, B-tree, or B*-tree.⁸ In the following we have selected a

specific B-tree, the 2-3 tree, to illustrate the concept. A 2-3 tree is a tree in which every vertex that is not a leaf has either two or three sons, and every path from the root to a leaf is of the same length.⁹ A dataflow structure and its 2-3 tree representation is given in Figure 4c. Each internal vertex of a 2-3 tree contains the value of the largest selector appearing in its sub-tree. These values are used in the SELECT (and APPEND) operation to guide a modified binary search requiring $O(\log n)$ time, where n is the number of leaves in the tree. APPEND can also be done in $O(\log n)$ time,¹⁰ where none of the vertices of the original tree are disturbed (except perhaps for reference counts) and most of the original 2-3 tree is shared between the argument and result structures without affecting functionality. The 2-3 tree representation also promotes efficient concatenation of dataflow structures (an operation quite useful in programs such as quicksort, fast Fourier transform, etc.) with constant time required in the best case and $O(\log n)$ time required in the worst case, given certain restrictions are met in the input structures.¹⁰ However, a significant disadvantage with 2-3 tree representation is the extra memory required for the internal vertices of the tree; and while asymptotic behavior is good, $O(\log n)$, the constant factor in the equation also could be significant.

The reader may note that the design of an efficient dataflow memory system involves problems (i.e. memory management, garbage collection, choice of representations, etc.) which also occur on conventional systems. Currently these tasks are reprogrammed to some extent by each application program that is written. Our feeling is that by embedding these tasks within the system as close to hardware as is practical, a significant burden is removed from the application programmer, and if a good design is obtained, average performance will improve.

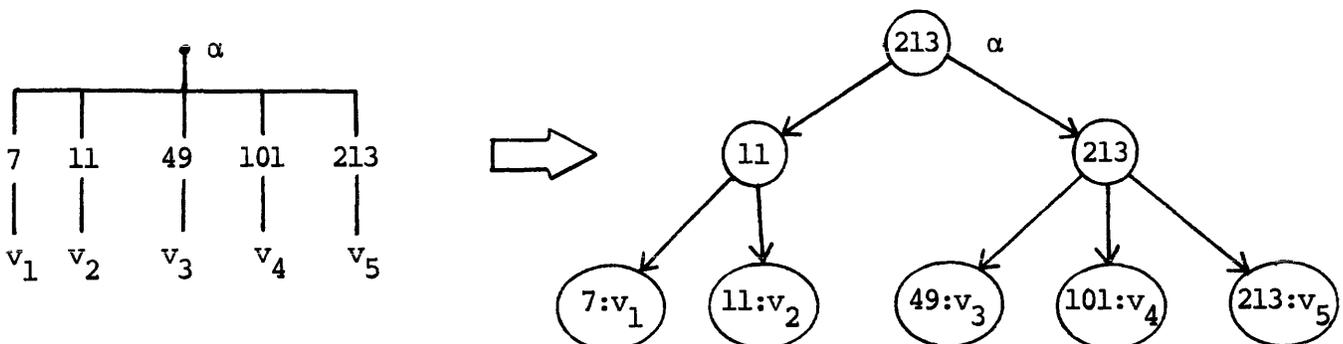


Figure 4c—Dataflow structure represented as a 2-3 tree

Implementation of the Unfolding Interpreter

One problem with the theoretical unfolding interpreter is the unbounded length of activity names, the primary purpose of which is to logically separate the tokens so that the inputs to each copy of each function are uniquely determined. For this purpose a unique number, N , for each context combined with the label l is sufficient. For example, starting with activity name $N.l$ for the A activity in Figure 3b, the context is changed by obtaining a new unique number N' to form activity name $N'.beginf$. In addition, an association is made in memory between N' and the activity name $N.t$; alternatively a token carrying $N.t$ can be sent from the A to the END activity. In either case, the return from the inner context is accomplished by the END activity which fetches the information $N.t$ associated with the number N' and uses it as a logical "return address" to the outer context.

The major question in this approach is the method of generating and managing the unique numbers. If enough bits (say, 60) are used to store N , uniqueness of N can be guaranteed to extend over the life of the system. However, if the system guarantees that none of the activities (and their associated tokens) exist from the previous use of N , a value N can be reused and significant savings achieved. This is generally not a problem in context management on sequential machines; however, due to the asynchrony of dataflow it is possible, for example, that an END activity finishes execution before all of the activities in the procedure have finished, even when the system guarantees that all such activities will eventually finish. One workable solution is to prevent END from finishing execution before all other activities and tokens of that procedure application have been consumed. With this restriction, a simple scheme such as a tree of stacks (cactus) can be used to implement unique number management with reusable names.

A second problem with the unfolding interpreter is the requirement for system-wide unique labels for object code. As is common in conventional systems, a tree-structured directory system with path specifications may be used to implement dataflow labels. If desired, path specifications can be included within activity names. For example, l can be split into two fields $p.s$ where p is a pointer to a procedure and s is a function (statement) number in that procedure.

MULTIPROCESSOR DESIGN PRINCIPLES

Although dataflow principles can be advantageously applied to conventional systems, we believe that new concepts in computer architecture must be developed to take full advantage of the concurrency and functionality of the dataflow model. This final section is largely speculative because of the difficulty of accurately predicting the performance of proposed architectures. However, we have simulated variations of a specific dataflow architecture and the results are reported in detail elsewhere.¹¹ Moreover, since the design of this architecture contains much detail and changes rapidly, we summarize our experience with it in the form of tentative principles for the design of one

possible form of dataflow machine. These principles are not new and have been applied to many systems. However, such a statement serves as a point of comparison with other views.

Our goal is to design a general purpose computing system which

1. Can effectively distribute small pieces of a computation over many processors in the machine;
2. Is modular enough so that additional blocks of processors can be easily added to increase the capacity of the machine;
3. Has a measure of fault tolerance so that hardware failures may decrease performance but will not necessarily halt the machine (i.e. fail-soft);
4. Does not require knowledge of the number and configuration of processors to write programs which effectively utilize these resources;
5. Does not depend on expensive interconnection schemes (e.g. crossbar switch) or extremely fast circuit speed for good performance;
6. Can support a number of simultaneous users.

The first principle of multiprocessor design (evidenced by simulation results) is the program-dependent tradeoff between distribution and localization of computation. Distribution may allow concurrent execution of a program, but it also tends to increase communication costs. Thus for any particular architecture and computation, there exists some optimal degree of distribution (perhaps as little as one processor) for which execution time is minimized. Locality (e.g. "the working set" in paging systems) is an established principle of conventional systems. Moreover, we believe that locality will be present to an even greater extent in dataflow due to the absence of side-effects and due to the high degree of structure imposed by our high level dataflow language. To take advantage of locality, we must consider two features of a dataflow system: 1) how the topology of the architecture allows reduced communication costs when physical locality is present and 2) how program locality is preserved in the mapping to physical hardware.

One topology which supports locality is a hierarchy of modules. For example, a primitive module could be a processor with memory which can execute any dataflow machine language instruction or group of instructions, including an entire dataflow program (the extreme case of locality). Primitive modules are connected to form larger modules which are then connected, etc., such that any module can be considered to consist of some processing power and some associated memory. This structure supports locality to the extent that communication within any given module can be made less costly than communication off the module.

A second aspect of locality is the mapping of program locality to physical locality in the machine. Assume each processor in the machine to have a distinct physical address. We have found that the activity names themselves constructed by the unfolding interpreter contain much of the locality information present in the source program. For example, activity names with the same context part belong to

the same instance of a procedure (or loop). In addition, the labels l in the object program can be assigned by the compiler such that numerically-close labels suggest close connection of functions. These pieces of information can be used by an activity *assignment function* which maps from logical activity names to physical processor addresses. The resulting physical address is placed on each output token to guide its transmission in the communication network. The selection of an appropriate assignment function is similar to the problem of selecting an appropriate hash function for a scatter table but with the additional consideration of preserving locality where appropriate. Note that the assignment function and the communication network perform a partial sorting of activity names by physically directing tokens to their destinations. The final sorting is done by each processor on only those activity names which map to its physical address.

As a practical matter, designing a fairly good assignment function which distributes computation while preserving locality is not too difficult to do (at least for the small selection of programs we have executed on our simulator). However, the selection of an optimal assignment function is a difficult problem requiring further investigation. We also note that it is possible for the machine to tune its assignment function(s) at execution time for improved performance.

The second principle of multiprocessor design we have adopted is that the communication delay (ignoring conflicts) between any two processors should be no more than $O(\log n)$ where n is the total number of processors. If we rule out complete interconnection schemes, this principle also suggests a hierarchical interconnection of modules. However, a tree is not the only structure with the $O(\log n)$ property. Two other examples are the boolean n -cube¹² and the interconnection network of Wittie.¹³ Both of these networks can be viewed as trees in which sufficient additional connections have been made such that the root node has become indistinguishable. (In other words, pick any node in the network; then appropriate connections can be deleted so that a tree remains.) Although much investigation remains to be done before selecting a particular interconnection network, we feel that a more highly connected structure than a tree is appropriate for two reasons: 1) the extra connections provide some measure of fault tolerance and 2) more flexibility is allowed to map the logical tree structure of many programs into the many physical trees present in the network. Currently we are favoring a modification of the Wittie network due to its lower implementation cost.

The last principle of multiprocessor design is recognition of the potential benefits of redundant copies of data and program code. Conventional systems have already developed this concept to some extent, primarily in the area of virtual memories and high speed caches. However, dataflow can take further advantage of redundant copies because values are never modified. We have three goals in pursuing the concept of redundancy: 1) to improve performance through concurrent access of data in distinct memories, 2) to improve performance through a caching scheme which localizes data to where it is most used, and 3) to identify each data copy so that if one copy is damaged, a search can

be instituted to obtain another valid copy. For those readers interested in possible mechanisms to achieve some of these goals, Reference 11 should be of some help.

CONCLUSIONS

The decision to incorporate the full generality of dataflow is not without its costs. We have seen that the principles of dataflow sometimes suggest implementations which make "inefficient" use of memory. Of course better implementations may yet be found but we suggest that problems with memory be viewed in the context of the following points: 1) the full generality of dataflow is not always required—for example a program like matrix multiplication can be executed within the semantics of dataflow and still require little more memory than does a conventional machine, 2) duplication of data and code can have various benefits such as concurrent memory access and the possibility of recovering from hardware faults, and 3) the cost of hardware and memory is decreasing while the cost of software and system failures will probably continue to increase. Thus in a few years the "efficient" use of memory in many situations might be viewed quite differently.

An introduction to dataflow is not complete without mentioning other issues and capabilities. Dataflow *streams* and *managers* are available to program history-sensitive applications such as airline reservation systems, resource management, etc. directly in a high-level language;¹⁴ in addition, abstract data types are available.

REFERENCES

1. Taub, A. H. (Ed.), *Collected Works of John von Neumann*, Vol. 5, The Macmillan Company, New York, 1963, pp. 34-79.
2. Keller, R. M., "Look-ahead processors," *ACM Computing Surveys*, Vol. 7, No. 4, December 1975, pp. 177-195.
3. Arvind., K. P. Gostelow and W. Plouffe, "An asynchronous programming language and computing machine," *Dept. of Information and Computer Science Technical Report 114A*, University of California, Irvine, CA, 1978.
4. Bic, L., "Protection and security in a dataflow system," *Dept. of Information and Computer Science Technical Report 126*, (Ph.D. Dissertation) University of California, Irvine, CA, 1978.
5. Plouffe, W., "Exception handling and recovery in a dataflow system," Ph.D. Dissertation, Dept. of Information and Computer Science, University of California, Irvine, CA, (in preparation).
6. Dennis, J. B., "First version of a data flow procedure language," *Symposium on Programming*, Institut de Programmation, University of Paris, Paris, France, April 1974, pp. 241-271 (also *MAC Technical Memorandum 61*, LCS/MIT, Cambridge, MA, May 1975).
7. Davis, A. L., "The architecture and system method of DDM: a recursively structured data driven machine," *ACM/IEEE Fifth Annual Symposium on Computer Architecture*, Vol. 6, No. 7, April 1978, pp. 210-215.
8. Knuth, D. E., *The Art of Computer Programming*, Vol. 3, *Sorting and Searching*, Addison-Wesley, Reading, 1973, pp. 451-480.
9. Aho, A. V., J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, 1974, pp. 145-155.
10. Thomas, R. E., "A comparison of methods for implementing dataflow structures," *Dataflow Architecture Project Note 35*, Dept. of Information and Computer Science, University of California, Irvine, CA, May 1978.

11. Gostelow, K. P. and R. E. Thomas, "Performance of a dataflow computer," *Dept. of Information and Computer Science Technical Report 127*, University of California, Irvine, CA, 1979.
12. Sullivan, H. and T. R. Bashkow, "A large scale, homogeneous, fully distributed parallel machine, I." *ACM/IEEE Fourth Annual Symposium on Computer Architecture*, Vol. 5, No. 7, March 1977, pp. 105-117.
13. Wittie, L. D., "Efficient message routing in mega-micro-computer networks," *ACM/IEEE Third Annual Symposium on Computer Architecture*, Vol. 4, No. 4, January 1976, pp. 136-140.
14. Arvind., K. P. Gostelow, and W. E. Plouffe, "Indeterminacy, monitors, and dataflow," *Proc. Sixth ACM Symp. on Operating Systems Principles*, November 1977, pp. 159-169.

A hardware-independent virtual architecture for PASCAL*

by VISWANATHAN SANTHANAM

Wichita State University
Wichita, Kansas

INTRODUCTION

Since its introduction in 1971, PASCAL language has received rapid acceptance as a structured programming tool in a wide variety of applications ranging from computer science education to systems programming in production environments. It is, therefore, not surprising that numerous reports relating to implementation of PASCAL compilers on various computers continue to appear in the current literature (References 1,4,6,12 for example). Many of these implementation reports relate to adapting an existing compiler to another computer. In many instances, these compilers are themselves written in PASCAL which makes it easy to adapt them using one of these procedures:

1. Implement the target machine of the original compiler as a virtual machine in the current hardware by means of an interpreter or microprogram. Now, the original compiler and its output can be executed on the current machine.
2. Alternatively, change the code generation phase of the original compiler to generate object code suitable for the current hardware. Then cross-compile the modified compiler (under the original compiler) and transport the object code to the current computer.

The second alternative is superior if efficiency is important. In fact, many implementors take a two-phase approach whereby the compiler is adapted quickly by the first approach, and it is refined later by the second approach to achieve better performance. When such a transition is made from the original virtual machine code to a more suitable object code, the best choice is not always the native instruction set of the present hardware. There are a number of situations where another virtual machine may be a better choice. For example, the hardware may be microprogrammable, in which case a well designed virtual machine could not only yield compact object programs but could also lead to better execution speeds. Another situation where virtual machine code may be preferred over native code is where a number of small PASCAL programs execute concurrently and interactively. This latter situation arises in instructional

environments where object code compactness and diagnostic capabilities outweigh execution speed.

The objective of this paper is to present a virtual machine design for PASCAL without basing the design on any specific hardware. The abstract machine is called VAMP for "Virtual Architecture Made for PASCAL," and its features are based on the PASCAL language, but they may also be adapted for use with other similar languages, such as ALGOL, SAL¹³ and BCPL.¹¹

There are a few other virtual architectures, some reported in the literature while the others are handed down from one implementor to the next, which are designed expressly for PASCAL-like languages. Notable among these are the P-machine¹⁰ and, more recently, EM-1.¹³ The P-machine is based on a 30-bit word-oriented processor, and it does not lend itself for efficient adaptation to other computers, such as the IBM 360/370. Nevertheless, many of the present PASCAL compiler implementors have adapted it in some form or the other due to the availability of a well written well documented compiler generating code for it. The architecture of EM-1 is also hardware-dependent to the extent that the code address space is assumed to be organized into eight-bit bytes. A more important difference between EM-1 and the current work is in the basic objectives. EM-1 is an architecture to match the needs of PASCAL-like languages and minimize the object code size of programs within the constraints of a specific underlying hardware. VAMP is an abstract machine designed only to match the needs of PASCAL-like languages, regardless of the hardware used to implement it. An implementor of VAMP has the choice to trade code size for execution speed, execution speed for firmware size and so on. Furthermore, VAMP is designed to meet the needs of full PASCAL language and not a typical subset of it. Once again, an implementor may omit selected features of VAMP if only a subset of PASCAL is to be implemented.

Despite the differences in the basic objectives, many of the architectural features of VAMP were motivated by similar features in these other virtual machines. In fact, as we point out in the section on implementation guidelines, an efficient implementation of VAMP will perhaps incorporate many of the ideas contained in the descriptions of these virtual machines.

The hardware-independent virtual architecture and the rationale for the design choices are presented in the next

* This research was partially funded by a grant from the Wichita State University Research Committee.

section. Some guidelines on how to adapt VAMP for a specific hardware are included in the third section of this paper.

ORGANIZATION OF VAMP

We describe in this section the organizational details of an abstract machine called VAMP for "Virtual Architecture Made for PASCAL." As the name implies, the features of VAMP are oriented specifically toward efficient support of PASCAL. However, because of the many similarities between PASCAL and other block-structured languages such as ALGOL, it is possible to adapt many of the features of VAMP for use with these other languages as well. The reader is assumed to have basic familiarity with PASCAL or a similar language. An excellent treatment of PASCAL can be found in References 14 and 9. Features of concurrent PASCAL are outlined in Reference 3.

The description of VAMP to be presented in this section is hardware-independent. The unit of memory, the forms of data representation, the number and purpose of hardware registers and other similar details regarding the processor on which VAMP is to be implemented, are unspecified. Of course, these and other hardware characteristics of the processor will determine the performance of VAMP, but they do not dictate the feasibility of implementation.

The term "efficient support," used earlier in this section, needs elaboration. The primary concerns of a language-based virtual machine design include (a) ease of compiler implementation, (b) object code compactness and (c) speed of execution. By design, there is a direct correspondence of features between VAMP and PASCAL. This correspondence significantly reduces the complexity of a compiler by eliminating the need for complex register and storage assignments that other compilers may need. At the same time, the design allows compilers to make trivial optimizations, such as combining a conditional branch with a preceding test, to reduce number of instructions. Also, by design, the number of object instructions per source statement is much smaller for VAMP than for a processor with a general-purpose instruction set. Such a low ratio of object code to source statement assures a certain degree of object code compactness regardless of the underlying hardware. In addition to this, the implementor can usually trade object code size for execution speed and vice versa, which is another degree of freedom that a virtual machine designed for a specific hardware does not provide.

An added advantage exists when a hardware-independent virtual machine is chosen as the target machine for compilers. Programs can be compiled into truly portable "object code" (as opposed to intermediate code that must be processed by a non-trivial translator) that is readily assembled into executable code for a specific implementation of VAMP.

General features

VAMP has five address space types that are accessible to a program. These are identified as CODE, PARM, DATA,

STACK and HEAP. There is one occurrence each of the CODE and HEAP spaces per program, whereas there is one occurrence each of the PARM, DATA and STACK spaces per active block. (An "active block" is one whose procedure has been called but has not returned yet.) The CODE space contains object instructions and constants. The reason for not providing an independent 'constants' space is to allow separate compilations of procedures. The HEAP is intended for dynamic allocation of variables through the use of the PASCAL data type *pointer*. A PARM space contains the actual parameters of the block; a DATA space contains the local variables; and a STACK serves as the work area for the procedure. The size of the CODE space is fixed. The sizes of PARM and DATA spaces of a given block can be determined at compilation time. An estimate of maximum size for a STACK space can be determined at the time of activation of the block. The size of HEAP is not specified by the program, but it is usually determined by the availability of memory in the operating environment.

It is not intended that these address spaces be allocated and managed independently. A typical implementation may integrate these address spaces into a single partition of memory (See References 1,10,13 for example). Because of this, we assume that every address space is realized from the same basic memory which is made up of *cells*. The size of a cell is unspecified, but it must store at least two distinct values. It is required that the cell be effectively addressable. Direct addressability in hardware, though not a requirement, should yield better performance.

The rationale for introducing five different address spaces is that it provides higher flexibility in compiling PASCAL programs and it divides information into logical compartments. CODE space, for example, may be write-protected. Though PARM and DATA spaces are fixed in size for a given block in PASCAL, some special procedures (such as READ, WRITE etc.) may be written to handle a variable number of parameters. VAMP allows for this possibility. Similarly separating STACK space from DATA space allows the allocation of dynamic arrays at block entry time. These features are not standard in PASCAL, but we anticipate the need for some non-standard functions in most systems for better efficiency or for being able to interface with external non-PASCAL environments.

Data representation

VAMP implements five basic data types—*integer*, *natural*, *real*, *pointer* and *packed naturals*. All these types, except the pointer, may have one or more variable size representations in memory. The size of a variable length data item will be specified in the address used to access it (see *operand addressing* in this section). The minimum size for the types integer, natural and real is one cell. The maximum sizes, also called *standard sizes*, for these types are set by the implementor. For elements of the packed naturals type the minimum size may be a fraction of a cell, and this too is determined by the implementor. The maximum size for an element of this type is the same as that of the type

natural. The variable size representations just described apply only to addressed operands; operands on any stack are always represented in their standard sizes. Furthermore, there are well defined procedures for conversion of a data item from one size to another.

Integers and reals are defined in the traditional manner with well defined procedures for the usual conversions in both directions. Natural is an unsigned integer with the requirement that its standard size representation be identical to the standard size integer with the same magnitude. Once again, well defined procedures are provided for conversion from a natural to integer, and from a non-negative integer to natural.

With the exception of the type *pointer*, all the basic data types of PASCAL (all scalars, their sub-ranges and reals) are to be implemented with the integers, naturals and reals of VAMP. The rationale for variable size representations of these types is based on this principle. This is essential if we wish to achieve reasonable storage efficiency when representing such a wide range of basic types with the small set of VAMP data types. If an implementor chooses a fairly large unit of storage (e.g., 16 bits or more) as the cell, then there will be less need for variable size representation. On the other hand, if the cell is chosen to be a small unit of memory (say, eight bits or less) better packing of data in memory can be achieved, but variable-size representation is inevitable. VAMP design leaves the choice up to the implementor by specifying one as the minimum number of representations to be implemented in each category.

Even after judiciously choosing an 'optimal' size for a cell, an implementor may find it hard to pack data satisfactorily, especially in relation with *packed arrays* of PASCAL. For example, the choice of an eight-bit byte for the cell may satisfactorily implement most basic data types, but it may be unsuitable for implementing a packed array of booleans, or a packed array of scalar such as (MON, TUE, WED, THU, FRI). The need for efficient packing also arises in conjunction with the common implementation of the PASCAL type *set* as a bit map (or packed boolean array). VAMP provides for the type packed naturals to achieve a higher level of storage compaction in these cases at the expense of packing and unpacking overhead at run time. The number of direct manipulation facilities is also limited when an array of naturals is so packed.

The VAMP pointers are used for two main purposes. They implement the PASCAL type *pointer* and also represent *var* parameter in procedure calls. A VAMP pointer is required to be of fixed size and capable of representing an address in any of the currently active address spaces. This includes the CODE space, the HEAP space, the DATA, PARM and STACK spaces of all active blocks. The implementor may place a limit on the number of such active blocks to be allowed.

Operand addressing

When an instruction needs an operand it is located by an *address*. The VAMP address may locate an operand either

as an *immediate* operand or by identifying an address space and a cell offset. An immediate operand may be in-line, following a suitable prefix in the address, or it may be obtained by popping the current stack top element. The former mode is useful in compiling operand references to small constants whereas the latter mode provides a powerful method for realizing single and zero-address operations from a standard two-address or one-address instruction. For example, VAMP includes only one "store integer" command with two addresses—one for the source operand and the other specifying the destination in memory. The same instruction can also "pop" the current top of stack integer to memory by simply specifying a "stack-top-immediate" mode for the source operand. This is not only intended to reduce the size of the interpreter needed to implement VAMP, but also yield compact object programs as this facility is available to all operations that use addressed source operands. The potential increase in the size of addresses themselves can often be eliminated by judiciously encoding the address modes.

An addressed operand (as opposed to an immediate operand) may reside in any of the address spaces that are directly accessible to the current procedure. This includes the CODE space, the current STACK space and the DATA and PARM spaces of the most recently activated blocks at lexicographic levels lower (outer) or equal to the current level. (Operands on the other stacks are strictly local to the blocks owning them, while operands in the HEAP space can only be accessed via pointers). This means that a VAMP address need only reference $2^n + 4$ address spaces while executing a procedure at level n (counting from 0 for the program level). The implementor may place a limit on the maximum value for n .

A memory operand may also be addressed indirectly by a VAMP pointer. In this mode of addressing, known as *indirect addressing*, the final operand may reside in any of the currently active address spaces. However, the pointer itself must reside in one of the directly addressable spaces or be popped from the current stack top. (There is no provision for "in-line immediate" pointers, as PASCAL doesn't provide for pointer constants other than *nil*). It should also be noted that there can only be one level of indirect addressing in any one address computation as a pointer is not capable of indicating an indirect mode.

In all cases except when the operand is obtained as an immediate operand from the stack top, the address also includes an operand size indicator. The number of different sizes represented in an address may vary depending on the operand type that it locates and it is determined by the implementor. This freedom to choose many different lengths for one type (say, natural) and possibly only one length for another type (say, real) makes VAMP design a flexible one without adding unconditional overheads to the interpretation process. In the case of indirect addressing, the size information in the address applies to the final operand, not the intermediate pointer. This is why VAMP pointers are required to be of a fixed size.

Independent of how the final address is arrived at, the cell offset of that address may be modified by adding a natural

called *index*. If indexing is indicated in the address, the current stack-top natural is popped and saved before proceeding further with the address computation. As in the case of indirect mode, only one level of indexing is permitted in VAMP. If both indexing and indirect addressing are specified only the final operand address can be indexed in one operation. This design favors object code compactness for the more common cases of *var* parameter arrays and pointer-based arrays over the case of array of pointers.

The reason for popping the index value from the current stack top rather than deriving it from an "index register" is many-fold. Since all arithmetic is carried out on the current stack-top, the index value is resident on stack-top to begin with. Furthermore, the introduction of an index register will require the implementor to come up with equivalent hardware features on his machine to implement VAMP. Such features may not exist in his machine. Even if high-speed registers are available in the hardware they could be better used as extensions of the stack top rather than set aside for indexing only. On the other hand, if the hardware includes registers that are intended for the sole purpose of indexing, the VAMP interpreter could still "pop" the index values to those registers and carry out the final indexing from there. Finally, the lack of need for complex register assignments takes a major burden off the compiler.

The various formats of a VAMP address are depicted in Figure 1 as a "variant record" tree. The nodes represent tag

fields and information fields, and the branches discriminate the cases of tag fields. It should be pointed out that the variant record representation is merely a way to illustrate the various address modes of VAMP, not a required implementation scheme.

Instruction formats and execution

The object instructions of a PASCAL program reside in the CODE space of VAMP. There is a "register" in VAMP called PC (for Program Counter) that contains the cell offset of the next instruction. The VAMP interpreter always refers to this register to locate the next instruction to execute.

Each instruction begins with an *opcode* which defines the operation. There may be additional fields following the opcode depending on the functions. These fields may be *literal fields* or *address fields*.

A literal field provides the value of an instruction parameter directly in-line. Typically these are integer and natural fields, represented by *I* and *N* respectively, whose values are known at compilation time. Each literal field is of a pre-defined length set by the implementor. An address field is a VAMP address whose format was described in the preceding subsection.

It is not required that the individual subfields of an instruction be some integral number of cells in length, even though such an implementation may significantly reduce the chores of interpretation. It is, however, required that the total length of any instruction be an integral multiple of cells, as the PC is not capable of addressing subunits of a cell.

The execution of a VAMP instruction proceeds in two phases—*fetch phase* and *execute phase*. The fetch phase is always completed before the execute phase begins. In the fetch phase, the current instruction is scanned from left to right 'decoding' individual fields as they occur. Decoding here means extracting literal fields into some standard length internal registers, evaluating address fields, and fetching the operands or simply computing an effective address, whichever is needed by the operation. Table I lists all the instructions of VAMP using a special notation which is described below.

The "fields" column of the table lists the component fields of an instruction. The order of the fields is important as is the mnemonic used to describe it. Table II provides a complete list of field mnemonics and the associated fetch phase functions they involve. For example, an *AI* refers to an addressed integer source operand. An *SI* refers to an integer operand storage address; that is, an immediate value may not be returned from address computation. Sometimes a trailing decimal digit is appended to a field mnemonic to distinguish it from another field of the same type within the same instruction.

The column labeled "function" outlines the execute phase functions of an instruction. The symbol "←" stands for assignment of value to an operand in storage. The notation "(SI)←AI" stands for "assign the integer value AI to memory location SI." The term *itos* refers to the integer which is currently at the top of the stack. Similarly *ntos*, *rtos* and

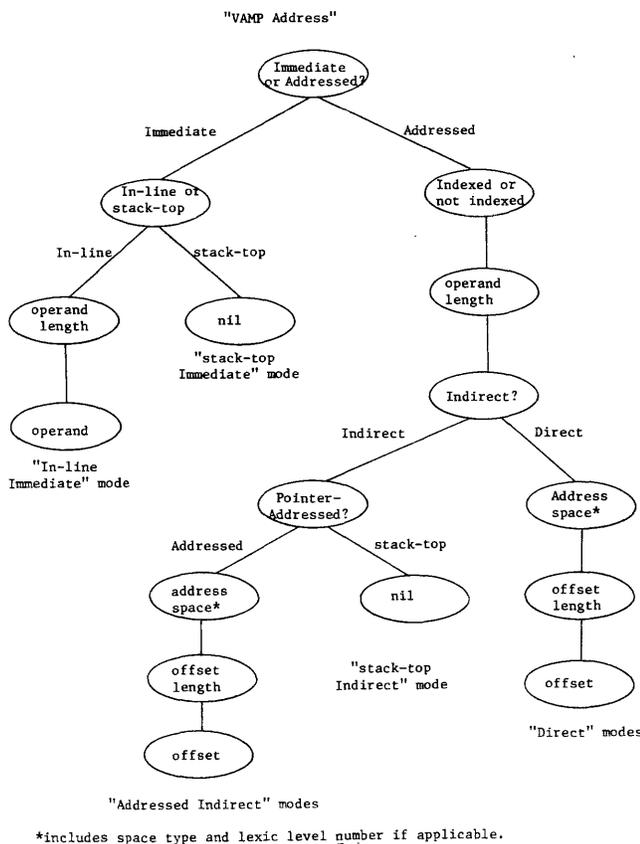


Figure 1—Address modes for VAMP.

TABLE I

Opcode Mnemonic	Opcode Description	Fields	Function
Arithmetic Group			
ADD	Add	AI	$itos \leftarrow itos + AI$.
Similarly SUB, MUL, Div (real result), IDIV (integer quotient), MOD; and ADDN, . . . , MODN for naturals			
ABS	Absolute	AI	$iPush(AI)$.
Similarly NEG (negate); NEGN (negate natural).			
ADDR	Add Real	AR	$rtos \leftarrow rtos + AR$
Similarly SUBR, MULR and DIVR.			
ABSR	Absolute Real	AR	$rPush(AR)$.
Similarly NEGR.			
EQS	Equal?[Skip]	I.AI	If $I < > 0$ then [if $ipop = AI$ then $PC \leftarrow PC + 1$]; else [if $ipop = AI$ then $nPush(1)$, else $nPush(0)$].
Similarly NES, LTS, GES, GTS and LES; and EQNS, NENS, . . . , LENS for natural operands.			
Also, EQRS, NERS, . . . , LERS for real operands (with "AI" replaced by "AR" and "ipop" by "rpop").			
ITOR	Integer to Real	AI	$rPush(AI \text{ converted to real})$.
Similarly TRUNC and ROUND for a real operand AR.			
SUBTIS	Subscript Translate Integer[skip]	AI.SX.I	SX: points to a record of i1: integer (lower bound), all of predefined i2: integer (upper bound), lengths. n3: natural (multiplier). If $AI < i1$ or $AI > i2$ then; else [$nPush((AI - i1) * n3)$; $PC \leftarrow PC + 1$].
Similarly SUBTNS for a natural operand AN.			
Logical Group			
ANDS	And?[skip]	I.AN	If $I < > 0$ then [if $npop$ and AN then $PC \leftarrow PC + 1$]; else [if $ntos$ and AN then $ntos \leftarrow 1$, else $ntos \leftarrow 0$].
Similarly ORS.			
NOTS	Not?[skip]	I	If $I < > 0$ then [if not $npop$ then $PC \leftarrow PC + 1$]; else $ntos \leftarrow not$ $ntos$.
SKT	Skip on True	I	If $npop$ then $PC \leftarrow PC + 1$.
Similarly SKF.			
Data Movement Group			
PUSHI	Push Integer	AI	$iPush(AI)$.
Similarly PUSHN, PUSHR, PUSHP (Push Pointer).			
PUSHPN	Push Packed Natural	AN.SPN	$nPush(elem(SPN)[AN])$.
STORI	Store Integer	AI.SI	$(SI) \leftarrow AI$.
Similarly STORN, STORR, STORP.			
STORPN	Store Packed Natural	AN1.AN2.SPN	$elem(SPN)[AN2] \leftarrow AN1$.
STISIS	Store Subrange Integer[Skip]	AI.SI1.SI2.N	SI1 points to an array of two integers i1; Lower bound, i2; upper bound. If $AI < SI1$ or $AI > SI2$ then; else [$(SI2) \leftarrow AI$; $PC \leftarrow PC + N$].
Similarly STSNS (Store Subrange Natural [Skip]).			
MVC	Move Cells	SX1.AN.SX2	For $j := 0$ to AN-1 do $cell(SX2)[j] \leftarrow cell(SX1)[j]$.
PACKN	Pack Naturals	SN.AN.SPN	For $j := 0$ to AN-1 do $elem(SPN)[j] \leftarrow elem(SN)[j]$.
Similarly UNPKN.			
EQCS	Equal Cell String?[Skip]	SX1.AN.SX2.I	$temp \leftarrow (cell(SX2)[j] = cell(SX1)[j])$ for $j := 0$ to AN-1); If $I < > 0$ then [if temp then $PC \leftarrow PC + 1$], else $nPush(temp)$.
Similarly NECS, where "temp" is replaced by "not temp" in the 2nd step of function.			
GTCS	Greater Cell String?[Skip]	SN1.AN.SN2.I	$temp \leftarrow (for \text{ some } 0 \leq k \leq AN-1, cell(SX2)[k] > cell(SX1)[k] \text{ and } cell(SX2)[j] = cell(SX1)[j] \text{ for } j := 0 \text{ to } k-1)$; If $I < > 0$ then [if temp then $PC \leftarrow PC + 1$], else $nPush(temp)$.
Similarly LTCS.			
Also, GECS and LECS with $temp \leftarrow$ condition for GTCS (LTCS) or condition for EQCS.			
Similarly EQPNS, NEPNS, GTPNS, LTPNS, GEPNS and LEPNS for packed natural strings with SN1, SN2 replaced by SPN1, SPN2 and "cell" replaced by "elem"			
INCPNS	Inclusion Packed Natural?[Skip]	SPN1.AN.SPN2.I	$temp \leftarrow (elem(SPN2)[j] \geq elem(SPN1)[j])$ for $j := 0$ to AN-1); If $I < > 0$ then [if temp then $PC \leftarrow PC + 1$], else $nPush(temp)$.
Similarly EXCPNS with ">=" replaced by "<=".			
DIFPN	Difference Packed Natural	SPN1.AN.SPN2	For $j := 0$ to AN-1 do $elem(SPN2)[j] \leftarrow elem(SPN2)[j] - elem(SPN1)[j]$.
Similarly UNIPN (union packed natural) and INTPN (intersection packed natural) with ">" replaced by "or" and "and" respectively.			

TABLE I.—(Continued)

Opcode Mnemonic	Opcode Description	Fields	Function
Flow Control Group			
SKIP	Skip	I	PC←PC+I.
BRA	Branch Absolute	AN	PC←AN.
FORUP	For Up	N	Top of stack contains i1,i2: integers with i2 on top. if i1≤i2 then[iPush(i1); i1←i1+1] else [ipop;ipop; PC←PC+N].
Similarly FORDN.			
CASEI	Case Integer	SI	SI locates an array of records of n1: natural of predefined length, n2: natural of predefined length, il: array [1 . .n1] of integers of length specified by SI. temp:=ipop; j:=1; repeat PC←PC+n2[j]; for k:=1 to n1[j]do if il[j,k]=temp then exit until n1[j]=0.
Similarly CASEN (Case Natural).			
Procedure/Function Linkage Group			
Note: The statements regarding procedures in this group apply equally to functions except when noted otherwise.			
CALL	Call Procedure	SX	SX points to record of naturals of predefined lengths, n1: lexicographic level number of target procedure. √ n2:offset in CODE space of target procedures entry point. If n1≤current procedure's lexicographic level number then all PARM, DATA and STACK spaces at the levels n1 through current are saved and made unavailable to the target procedure. The current value of PC and the current level number are also saved. Current level←n1; PC←n2.
ENTRB	Enter Block	SX	SX points to a record of 3 naturals: n1, n2, n3 of independent predefined lengths. A PARM space of n1 cells is allocated at the current level. n1 cells are moved (with deletion) from the caller's STACK top to this PARM space. A DATA space of n2 cells and a STACK space of n3 cells are also allocated. The stack is initialized to empty.
RET	Return from Procedure	AN	AN defines the number of cells to be transferred from the base of the current PARM space to the top of the calling procedure's stack (value returned by a function). The current PARM, DATA and STACK spaces are freed, the saved address spaces are made accessible to the calling procedure and PC is loaded with the return address in the calling procedure.
EXITB	Exit Block	SX	SX points to a record of naturals: n1,n2,n3 of predefined lengths. n1 defines the lexicographic level number of the target procedure to which control is to be transferred. The target block is the most recently activated block of that procedure. The DATA, PARM and STACK spaces of that block and all those at lower(outer) levels are made accessible to the target procedure. Cells from the top of the target procedure's stack are deleted until only n3 cells remain in its STACK space. Finally, PC←n2.
Miscellaneous Group			
ADJS	Adjust Stack	AI	The top of stack pointer is adjusted by AI cells. If AI is positive, this is equivalent to allocating space on the stack; if AI is negative, words are deleted from the stack.
GENP	Generate Pointer	SX	A VAMP pointer to the address SX is generated and pushed on the current stack.
GENHP	Generate Heap Pointer	AN	A VAMP pointer referring to the HEAP space with an offset AN is generated and pushed on the current stack.

ptos are used to refer to natural, real and pointer operands at the top of the current stack. These terms are to be distinguished from *ipop*, *rpop* etc. which include an actual popping of the respective operands from the stack. The function *iPush* (*x*) refers to pushing the value of *x* as a standard integer on the current stack; similarly for *nPush*, *rPush* etc.

When a memory operand is an element of an array it is referred to by the notation *elem*(address) [*j*]. Thus, "*elem* (*SPN*)[1]←*AN*" means assign the natural *AN* to the first element of the packed naturals array beginning at *SPN*. The reader's full understanding of these notations is essential for the discussion of the instruction set design that follows.

TABLE II

Field Mnemonic	Field Description	Fetch Phase Function
AI	Addressed integer	Returns a standard size integer.
AN	Addressed natural	Returns a standard size natural.
AR	Addressed real	Returns a standard size real.
AP	Addressed pointer	Returns a VAMP pointer.
SI	Storage address of integer	Returns an effective address and length of an integer.
SN	Storage address of natural	Returns an effective address and length of a natural.
SR	Storage address of real	Returns an effective address and length of a real.
SP	Storage address of pointer	Returns an effective address of a VAMP pointer.
SX	Storage address	Returns an effective address of an unspecified data type.
SPN	Storage address of packed natural	Returns an effective address and element length for a packed natural array[0..max].
I	Literal Integer Field	Returns a standard size integer.
N	Literal Natural Field	Returns a standard size natural.

Instruction set design

Throughout the design of the VAMP instruction set, careful thought was given to

- Simplifying code generation in the compiler.
- Eliminating unnecessary shuffling of stack top elements.
- Providing opportunities for simple, but effective code optimization.
- Providing instructions that are capable of handling the most general program constructs, yet yield efficient code for simpler, more common cases.

We illustrate each of these features with examples.

Code generation is simplified by the provision of such features as "store sub-range Integer" (which checks the integer value against its bounds before storing), "Subscript Translate" (which provides for converting a subscript into a cell offset or index after bounds checking), "Move Cell" (which is used in record and array assignments), "pack and unpack naturals" and "For up/down." These features of the virtual machine can not only reduce object code size but make it simpler to translate some of the more complex constructs of the language.

It should be pointed out in this context that the PASCAL case statements are not always to be translated into the case instructions provided in VAMP. The case statements often can be compiled into far more efficient code than a linked list of case values and case addresses. These efficient translation schemes usually involve the use of "branch tables," which option is available in VAMP and should be exercised. But in the most general case, there may not be any better scheme than to compare the case expression against individual case values in some sequential order and branching when a hit is found. These are the situations under which the VAMP case instructions are useful.

The design of the instruction subfields and their order also contributes toward this simplicity and can save unnecessary re-ordering of top of stack elements. Consider compiling an

assignment statement such as:

$$a[i] := b[i+1]$$

It is most naturally translated into this code sequence

```
compute i on the stack
translate it into an index for a
compute i+1 on the stack
translate it into an index for b
store b (indexed) into a (indexed).
```

Thus, the order of appearance of source and destination addresses in the store instruction saves us from having to reverse the indexes on the stack. This also holds if we had to push b on the stack explicitly (e.g. for type conversion) before "popping" it to a . A similar reasoning was applied to the design of every VAMP instruction.

The VAMP instruction set allows the compiler to generate "optimized code" without the need for complex optimization passes. Consider a simple *if . . . then . . . else* in PASCAL:

```
if a>b then s1 else s2
```

An unoptimized sequence of object code may appear as follows:

```
push a
push b
push boolean a>b
Branch on boolean=false to s2.
etc.
```

If a and b are of compatible types, the "push b " step can be avoided. But if we recognize also that a branch is to follow we can generate "compare and skip" instruction directly instead of "push boolean." If the skip offset field is large enough, one could even eliminate the "Branch on boolean" that follows. Similar savings can also be demonstrated in the case of *repeat . . . until* loops. Another case

of simple optimization occurs in assignment statements involving simple expressions on the right hand side and a compatible variable type on the left hand side, such as:

```
a[j]:=0
```

or

```
c[3]:='?'
```

In these cases one can generate a single store instruction (not counting the index computation) rather than a push/pop sequence.

A reader familiar with other stack machines may find some 'standard' stack operations missing. The lack of 'Pop' statements is one such case, which we have shown as equivalent to a 'store' with 'stack top immediate' mode of addressing for the source operand. Similarly a 'Duplicate' can be achieved by a 'Push' with the source address referring to the current STACK space. An 'exchange' is not provided as the need for it should be rare. Even then a sequence of push and store (pop) operations can yield the desired result.

The design of VAMP is oriented toward implementing the full PASCAL language, not a "typical subset" of it. For example, the PASCAL *goto* that allows leaving the current block and transferring control to a statement in a surrounding block is one of the more difficult features of the language to implement. The "Exit Block" feature of VAMP aids in transferring control out of a block in an orderly manner. On the other hand a simple use of the *goto* can most often be realized in terms of a "Skip" or a "Branch" and, perhaps, an "Adjust Stack" instruction. Similarly, "sub-range store" instructions may be generated only when bounds checking is desired; the ordinary "store" instructions can yield better speed if performance is important.

Before concluding the discussion on the VAMP instruction set, we wish to point out the design consideration for some of the less obvious instructions. The "For up/down" instructions are executed after pushing the initial and final values of the control variable on the stack. These values are always interpreted as integers which, on the stack, are indistinguishable from naturals. The initial value integer will also serve as the "control variable image" during future iterations. When the "For" statement is executed, a copy of this image variable is provided on top of the stack unless the end condition is met. This is so that an appropriate store instruction can be used to store it into the actual control variable by the first instruction in the loop. This avoids the need for typed "For" instructions.

The "call procedure" instruction specifies the lexicographic level number and the CODE space address of the target procedure. Three other pieces of information—the sizes of the PARM, DATA and STACK spaces for the target procedure—are specified in an "Enter Block" statement which should be the first instruction in the target procedure. These parameters may appear as a record within any of the address spaces normally accessible to that procedure except its own PARM, DATA and STACK spaces.

The "Exit Block" instruction is intended to "clean up"

the activities of all intervening procedures when a *goto* transfers control from a called procedure to a label in a calling procedure. It also provides for "cleaning up" of the target procedure's STACK space to the state expected by the next instruction in that procedure. This is achieved by specifying the number of cells that should remain on the stack when that instruction is begun. It should be noted that the size of the current stack is a unique number for a given point in the program and it can be determined at compilation time.

The "case" instructions are implemented using a linked list of case values and associated case addresses (see Figure 2). As we pointed out earlier, most common instances of the PASCAL *case* statement will probably lend themselves to translation to more efficient VAMP code than the proposed "case" instructions. A compiler should reserve this general form for those *case* statements that don't easily translate to another more efficient form of code.

The features provided for manipulation of packed naturals are motivated by the expected use of this VAMP type. If character strings were mapped into (unpacked) array of naturals then the "compare cell string" type of instructions will be used for string comparison which is permitted in PASCAL. On the other hand, if they are mapped into the packed naturals type, then the "compare packed natural string" type instructions will apply. The "Inclusion/Exclusion" comparisons and the "Difference/Union/Intersection" operations with packed naturals are provided for the set comparisons and set operations that are permitted in PASCAL. The "Pack/Unpack" facilities aid implementing the corresponding "standard procedures" of PASCAL (see Reference 9).

The next section deals with the issues of adapting VAMP to a specific hardware.

IMPLEMENTATION GUIDELINES

No matter how well it is designed in abstract, an architecture such as VAMP must be adapted judiciously to a given hardware before claims of better efficiency can be made. In this section, we attempt to list some of the major trade-offs that an implementor of VAMP will be faced with, and offer some guidelines on how to make such trade-offs. We also present examples of possible adaptations for two contrasting architectures, namely the IBM 360 and the Burroughs B1700.

Adapting VAMP for a specific hardware

The choice for "cell" is by far the most crucial one to make. It must match the underlying hardware. It should be at least one byte for byte-addressable machines like the 360; it could be as small as a bit for the B1700s. For machines with smallest addressable unit of memory that is several bits long (e.g. CDC 6600 with 60-bit word) the choice should be guided by other factors such as code compaction versus execution speed. Any choice of less than 60 bits for the

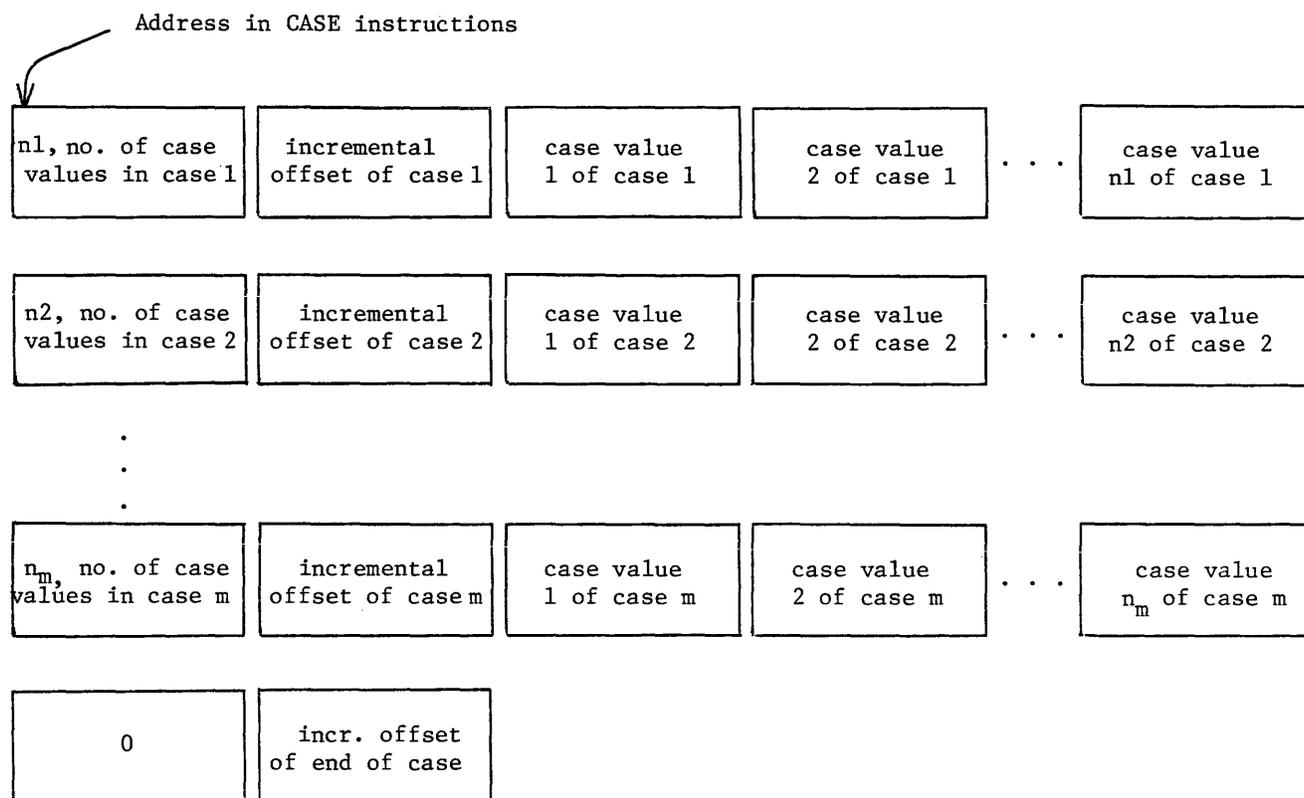


Figure 2—The CASE instructions of VAMP.

CDC 6600 will require software "simulation" of addressability and probably lead to some speed degradation.

The best choice for the unit "cell" is not always the smallest addressable unit of the hardware. On the B1700, for example, choosing a larger unit such as eight bits (or 12 bits) could lead to a reduction in the average instruction size due to fewer bits needed per VAMP address. Furthermore, on some machines, operands may have to be aligned at specific address boundaries for efficient access. One way to counter this problem is to define cell to be an "aligned unit" of storage. The compiler that generates code for the specific implementation of VAMP could then assign aligned addresses to data types as it processes their declarations. If we were to generate "portable" VAMP code, then the data map must be included with the object code. The translator that would convert this code into a specific VAMP code can take care of alignments and related address adjustments needed.

The next major choices are in the standard sizes for the basic data types. For the naturals, integers and reals the arithmetic capabilities of the hardware dictate the most suitable sizes. The standard size for a VAMP pointer is determined by the choices for the maximum number of active blocks and the maximum size of an address space.

After this, the implementor should consider what types are to have variable size representations and how many. For the type real, there is less to be gained by a variable size

representation, as PASCAL does not permit sub-ranges of reals as a data type. Integer sub-ranges are permitted and therefore shorter representations could lead to storage compaction. The most significant gains are to be expected from variable size naturals, as all of the following PASCAL types are to be represented as naturals—standard scalars—*char*, *boolean*, user-defined scalars, sub-ranges of *char*, and non-negative sub-ranges of integers. This is why a single representation for the natural is likely to result in poor use of the memory. For the 360, two or three representations (besides the standard one) may be useful—one, two and three-bytes. For the B1700 with eight-bit cells and a 24-bit standard natural, two smaller representations of eight bits and 16 bits may be valuable.

The next issue to tackle is whether or not "packed naturals" are to be really packed. For VAMP machines with eight-bit cells, packing of naturals could yield significant storage savings in relation with packed boolean arrays. Additional sub-cell representations, such as two-bit or four-bit packed naturals, may not be justifiable unless main storage is scarce and/or hardware addressability permits it easily. Based on this observation, we could choose a one-bit packed natural representation for the 360s and one-bit and four-bit representations for the B1700.

The choice for VAMP address formats is confronted next. As Figure 1 suggests, there are a number of sub-decisions to be made here. Do we wish to allow variable size offsets?

TABLE III

VAMP Feature	IBM 360	Burroughs B1700
Machine Name	VAMP/360	VAMP/1700
Hardware Memory	Byte addressable; 16M bytes memory	Bit addressable; Max. 64K bytes (model dependent)
Definition of "cell"	8-bit byte	8-bit field
Standard sizes		
naturals & integers	4 cells	3 cells
reals	4 cells	6 cells (real arith. not supported in hardware)
Variable sizes		
real	no	no
natural	1 cell, 2 cells, 4 cells	1 cell, 3 cells
integer	2 cells, 4 cells	2 cells, 3 cells
Alignments		
stack	full-word	none
real	full-word	none
natural	half-word for 2-cell full-word for 4-cell	none
integer	half-word for 2-cell full-word for 4-cell	none
Packed Naturals		
Permitted?	Yes	Yes
Sizes	1-bit	1-bit, 4-bit
Vamp Address	See Appendix I for details	See Appendix II for details
Max. static nesting	128	7
Max. address space size	16M cells in CODE, HEAP spaces 64K cells in all others	64K cells
Variable offsets?	Yes	Yes
Variable in-line imm.?	Yes	Yes
Pointer	see Appendix I for details	same as for VAMP/ 360
Max. dynamic nesting	4096	4096
Size	4 cells	4 cells

Do we wish to allow variable size (in-line) immediate operands? Many of these issues are dealt with in the same manner as we did previously. They usually boil down to a trade-off between execution speed and object code compaction. The issue of compiler complexity could also contribute to a decision of this nature, but in the author's opinion, a certain degree of complexity in compilation should be tolerated if it could lead to run-time savings.

Table III summarizes these major implementation choices for the IBM 360 and the B1700. These choices are neither "optimal" nor "absolute." They are merely intended to serve as a "first-cut" choice for adapting VAMP. Code generated from these design choices can then be monitored for efficiency and fine-tuned accordingly.

It is most appropriate to point out that there are a number of special considerations that should be taken into account during the adaptation of VAMP. Tanenbaum¹³ lists the average frequencies (static and dynamic) with which the common features of PASCAL-like languages are used in systems programs. This information along with Huffman's⁸ coding techniques will help the implementor achieve better compaction and execution speed for VAMP programs.

CONCLUSION

The design of VAMP was motivated by the numerous implementations of PASCAL and similar languages with virtual target machines, many of which were designed for specific (real) machines and adapted by others with little or no change to match their own hardware. There was no systematic way to distinguish features made necessary or desirable by PASCAL from those that were dictated by the specific hardware. The purpose of this work has been to collect those features directly attributable to the needs of PASCAL language and present them as an abstract hardware-independent virtual machine which we call VAMP.

The most direct use of the architecture presented in this paper will be in conjunction with target language design for PASCAL compilers. Another interesting possibility that it presents is the notion of a "universal target language." If we carefully design such language and write a compiler to generate code in that language, it could make PASCAL programs "portable" at or near object code level. The author is currently involved in the design of such a language and a translator that will produce the VAMP/1700 (see the third section) code from it. The results of this investigation will be reported in a later paper.

There are a number of other directions for further research in this area. While the popularity of PASCAL is growing at an unprecedented rate, there are a number of efforts underway to refine the language. One of the weak points of present PASCAL is that there are no provisions for exception-handling in the language. This deficiency makes it less attractive for use in development of crucial programs such as operating systems. If and when such a facility is accepted into the language, VAMP would have to be redefined. While the need for organized exception-handling may be seen even in the absence of corresponding provisions in the higher-level language, we chose not to include any such features in VAMP in the spirit of being concerned more or less with the "standard" features of PASCAL.

The works of Bowles,² Goodenough⁵ and Hill⁷ may provide directions on how to implement exception-handling along with VAMP. Reports of difficulties and how they were overcome could provide valuable information for extending PASCAL to include this important feature.

There are other topics for investigation that are perhaps more ambitious. Almost every computer system, mini or maxi, supports a multitude of high-level languages. Typically, there are the popular languages such as COBOL that the end user needs, and there is a "software development language" (to use one vendor's name) which is suitable for writing parts of the operating system and utilities. Because of the diversity in the features of these languages, a single virtual machine may not be ideally suited for all of them. Yet, the software developers could save a lot of effort and time if a minimal union of these virtual machines can be designed. It is not known, for example, whether an acceptable virtual machine can be designed to support COBOL, FORTRAN and PASCAL. Answers to such questions will be highly beneficial to software houses writing and selling portable software products, especially compilers.

ACKNOWLEDGMENTS

The author wishes to express his gratitude to the referees of this paper for their valuable criticism. Special thanks are due to Herb Mayer whose suggestions, comments and enthusiasm greatly influenced the quality of this paper.

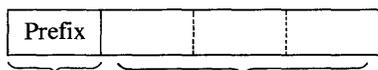
REFERENCES

1. Ammann, U., "On code generation in a PASCAL compiler," *Soft.-Prac. and Expr.*, Vol. 7, 1977, pp. 391-423.
2. Bowles, K. L., "Real time & Business extensions to UCSD PASCAL," Institute for Information Systems, Univ. of Calif. at San Diego, LaJolla, CA., April 1978.
3. Brinch-Hansen, P., "The programming language concurrent Pascal," *IEEE Trans. Soft. Engr.*, Vol. SE-1, No. 2, June 75, pp. 199-207.
4. Bron, C., and W. De Vries, "A PASCAL compiler for PDP11 minicomputers," *Soft.-Prac. and Expr.*, Vol. 6, 1976, pp. 109-116.
5. Goodenough, J. B., "Exception handling: Issues and a proposed notation," *Comm. ACM*, Vol. 18, No. 12, Dec. 75, pp. 683-696.
6. Grosse-Lindemann, C. O., and H. H. Nagel, "Postlude to a PASCAL-compiler bootstrap on a DECSys-10," *Soft.-Prac. and Expr.*, Vol. 6, 1976, pp. 29-42.
7. Hill, I. D., "Faults in functions in ALGOL and FORTRAN," *Computer J.*, Vol. 14, No. 3, Mar. 72, pp. 315-316.
8. Huffman, D. A., "A method for the construction of minimum-redundancy codes," *Proc. IRE*, Vol. 40, 1952, pp. 1098-1101.
9. Jensen, K., and N. Wirth, "PASCAL: User Manual and Report," Springer-Verlag, 2nd ed., 1975.
10. Nori, K. V., U. Amman, L. Jensen and H. H. Nageli, "The PASCAL (P) Compiler: Implementation Notes," *Berichte des Institutes fur Informatik, ETH Zurich*, Vol. 10, 1974.
11. Richards, M., "BCPL: A tool for compiler writing and systems programming," In *Proc. AFIPS SJCC34*, AFIPS Press, Montvale, N.J., 1969, pp. 557-566.
12. Russell, D. L., and J. Y. Sue, "Implementation of a PASCAL compiler for the IBM 360," *Soft.-Prac. and Expr.*, Vol. 6, 1976, pp. 371-376.
13. Tanenbaum, A. S., "Implications of structured programming for machine architecture," *Comm. ACM*, Vol. 21, No. 3, March 78, pp. 237-246.
14. Wirth, N., "The programming language Pascal," *Acta Informatica*, Vol. 1, No. 1, 1971, pp. 35-63.

APPENDIX I

VAMP/360 address format and pointer design

1. Address Format:



1 byte 0-3 bytes
depending on prefix.

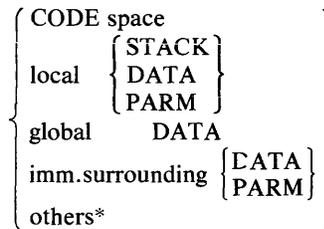
Prefix:

- (hex) 00 - immediate, stack-top.
- 01 - 03: immediate, in-line, 3 operand length variations.
- 04 - 0F: immediate, in-line, special constants such as 0,1-1,10, char ' ', pointer nil, etc.

Remaining modes; $h1\ h2$ (hex), with $h1 \geq 1$.

$$h1=1-F: \left\{ \begin{array}{l} \text{stack-top indirect} \times 3 \text{ op.length variations} \\ \left\{ \begin{array}{l} \text{direct} \\ \text{indirect} \end{array} \right\} \times 2 \text{ offset length variations} \\ \qquad \qquad \qquad \times 3 \text{ op. length variations} \end{array} \right\}$$

$h2=xn$: where x is "indexed-or-not" bit and $n=0-7$:



*others \Rightarrow 2nd Prefix byte including the following information:

- a. lexicographic level number (7 bits)
- b. DATA or PARM space (1 bit)

Offset Length variations:

- length 1 = 1 byte.
- length 2 = 2 bytes for all address spaces except CODE.
- 3 bytes for CODE space.

Operand length Variations:

Depends on operand type (see Table III).

2. Pointer Format:

Fixed length = 4 bytes.

First byte (or 2 bytes) contents:

(hex) $0x = \text{nil}$

$1x = \text{CODE}$ (next 3 bytes define offset)

$2x = \text{HEAP}$ (next 3 bytes define offset)

$3xx = \text{DATA}$ (xx defines dynamic block number, next 2 bytes define offset)

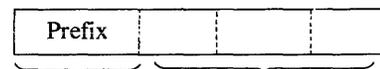
$4xx = \text{PARM}$ (same as for DATA)

$5xx = \text{STACK}$ (same as for DATA)

APPENDIX II

VAMP/1700 address format design

Address Format: (with cell = 8-bits)



1 cell 0-2 cells
depending on prefix

Prefix;

(hex) 00: immediate, stack-top
 01: immediate, in-line.
 02-0F: immediate, in-line special constants such as
 0,1,-1,10,char " ", pointer nil, etc.

$\left\{ \begin{array}{l} \text{stack-top indirect} \\ \text{CODE, direct} \\ \text{Local STACK, direct} \\ \text{Local STACK, indirect} \end{array} \right\}$	$\begin{array}{l} \text{X indexed? X 2 op.} \\ \text{Lengths} \end{array}$
----------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------

Remaining codes for "addressed" modes—

1x: with $x=0-F$ from

Remaining codes yx interpreted as follows:

$y=2-F$ from (7 lexic levels X DATA or PARM)
 and $x=0-F$ from
 (indexed? X 2 op. lengths X 2 offset lengths X indirect?)

Design of a high-level language machine*

by G. J. BATTAREL and R. J. CHEVANCE

Cii-Honeywell Bull
Louvenciennes, France

INTRODUCTION

There is at present a broad consensus that a capability-based architecture is the best approach to safe systems.^{5,3,6} However, at the bare hardware level, a machine should provide for a set of basic mechanisms (e.g. processor assignment, manipulation of queues of processes, definition of and manipulation on domains) without prejudging of any policy to be applied to these elements. The internal structure of the objects referred to by capabilities, the way they are organized into capability lists and the evolution of these lists in relation to various events appearing throughout the life of a program and of its possible activations should appear at some higher level, so that they can be modified without disturbing the hardware. This demands flexibility (the capability for emulating, for instance), adaptability (to future modifications) or the ability to define a machine which could be orientated, on demand, toward a certain class of applications (see for instance, the Burroughs B1700 approach). In the following, we shall be concerned mainly with this level, which we shall call "virtual architecture level." Considering a basic architecture, which we describe briefly, we define a virtual architecture for higher-level languages (namely PL/1, COBOL, FORTRAN and the system implementation language LIS,¹³) called the HLL-machine in this discussion. The objectives of the HLL-machine are the following:

- To define a virtual architecture as clean and homogeneous as possible, on which compilers may become more simple, and debugging aids more powerful.
- To define a minimum of intermediate languages, ideally, just one, for supporting PL/1, COBOL and FORTRAN.
- To define a set of run-time mechanisms enforcing safety.
- To obtain more compact object programs, as compared to third generation computer systems, so as to reduce the working sets of programs.
- To abide by the error confinement principle, which states that a procedure should not have at its disposal

more capabilities than actually required for its execution.

- To improve overall system performance.

CONCEPTS AND TERMINOLOGY

All the system resources are defined as *objects*, an object being the unit of naming, sharing and protection. An object is referenced through a *capability*, which specifies access rights for this object and contains a reference to its realization. Capabilities cannot be created, modified or destroyed but at the basic architecture level; they must be protected against unauthorized modification and are grouped into C-lists, which are distinct from the objects, called *data sets*, which are accessible to the user.**

The addressing scheme of the basic architecture provides a process with a capability list, two call/return stacks—CS (Capability Stack) for capabilities and VS (Value Stack) for other information, and a set of base registers. A procedure is considered as composed of four parts:

1. Definition of a set of objects.
2. Definition of entry points into this procedure.
3. Instructions manipulating the objects defined in 1.
4. References to entry points of other procedures.

Some of the objects defined in a procedure are purely local to it (not accessible from the outside), while others (called external objects), may be shared among several procedures. A procedure, which is an object of the cataloging system, may be referenced by means of one of its entry points.

In the following, we define a domain as a connected partial subgraph of a graph G of the above kind. A domain is therefore a construction on procedures.

Example—Given four procedures P_1 , P_2 , P_3 , P_4 and assuming P_1 references P_2 and P_3 , P_2 references P_3 , P_3 references P_2 and P_4 , and P_4 references P_2 and itself, the graph G is as shown in Figure 1a. Possible domains for this graph are G itself, or G_1 or G_2 as shown in Figures 1b and

* This study was sponsored by the French DRME, Direction des Recherches et Moyens d'Essais, under contract 74/450.

** Due to considerations of data representation compatibility with other systems, the tagged architecture scheme was not retained, in spite of its merits.⁷ The partition principle has been retained.

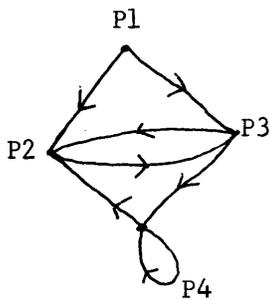


Figure 1a—Graph G.

1c, respectively. A *program* is a domain in which an entry point of a specific node of the graph has been specified; this node then corresponds to the main procedure. A *process* is a particular activation of a program; control enters the main procedure specified by a program and walks along edges of the graph G .

In a procedure, as it is constructed by a compiler, references to external objects or entry points are symbolic: actual access to an object implies that symbolic references be transformed into (virtual) addresses. Two extreme possibilities may be considered for these transformations:

1. Prior to any program specification, all references are transformed into addresses—the complete domain (corresponding to the graph G above) is constructed at one time; this corresponds to the well known static linking strategy.
2. The transformation of references into addresses is performed on demand—the domain is constructed edge-by-edge during the process activation. This corresponds to the dynamic linking strategy, as it is implemented in MULTICS,¹⁰ for instance.

Any linking policy appears as a particular way of constructing domains.

With the notion of domain in mind, a classification of objects can be issued according to their scope and lifetime. The *scope* of an object corresponds to the existence of an entry for this object in the cataloging system. It may be

- *System-wide*—For instance, a file which is accessible by several processes.
- *Domain-wide*—Local to a domain (e.g. a STATIC EXTERNAL variable in PL/1).
- *Internal*—I.e., local to a procedure.

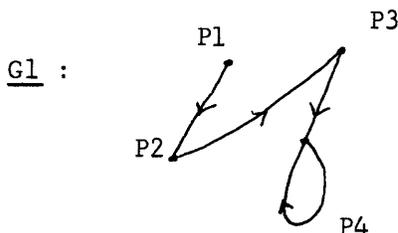


Figure 1b

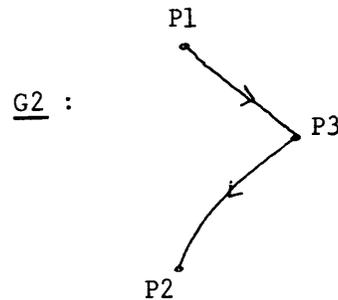


Figure 1c

The *lifetime* of an object is related to the existence of a realization for this object; inside its scope, an object may or may not have a realization.

ADDRESS SPACE—ACCESS PATHS

In order to achieve protection and error confinement, the informations which constitute the address space of a process have to be grouped inside distinct areas, according to three characteristics of the objects they are to contain:

- Scope
- Lifetime
- Access rights

In the first step, we define the areas which are necessary for a PL/1-like language, then we define relations between these areas—that is, the access paths to information.

Addressing space areas

The different areas are introduced in the order of their creation, from the compilation of a procedure until its execution in a specific domain.

- Procedure areas are defined at compile time. They consist of three parts:
 1. A CODE area containing object code instructions.
 2. A DC area containing descriptors and constants.
 3. A LINK area, a list of capabilities containing references to this procedure realization and entry points, and to the objects that this procedure may manipulate whose scope is not internal.

The first two areas are distinguished because their access attributes are different—CODE is 'execute-only,' whereas DC is 'read only.'

As long as the procedure is not destroyed, a single copy of CODE and DC areas will ever exist (object code is reentrant), whereas a new copy of the link area is created every time the procedure is linked to a new domain.

- Domain area. A domain must have its own (domain)

LINK area, which is a concatenation of copies of procedure LINK area. This is necessary because some objects referenced by a LINK are domain-wide, and the resolution of the reference to such objects in a LINK area is domain-dependent.

- Execution time areas. A distinction is introduced according to the lifetime of the objects.

Three classes—known from PL/1—are introduced:

- Static. An object of this class has a single realization all over the process activation.
- Automatic. The lifetime of an object in this class is related to the activation of blocks and/or procedures.
- Controlled. The existence of a realization which may consist in several 'generations' is under programmer's control.

The following areas are introduced:

- ST, static area, contains the realization of static objects whose scope is not larger than the domain. System-wide objects are realized in individual areas called DS (Data Sets). The ST area, created when a domain is activated, is enlarged every time a procedure is activated for the first time. As for the domain LINK area, only a part of the ST area—called *slot*—is accessible at a given time; it corresponds to the currently executing (external) procedure.
- Objects of the automatic class are realized in a value stack VS if they are local to the domain, or referenced from a capability stack CS if they are not (then their realization is in a DS area).
- Controlled objects cannot be system-wide; they are realized in a value heap area, VH, to which some garbage collection algorithm must be associated.

- Dangling references. A dangling reference is a reference to an object which currently has no realization in the address space. A safe implementation must detect such a circumstance, which may arise whenever the lifetime of an access path to an object is larger than the lifetime of this object. The problem is complicated since by means of pointers, parameters or overlay definitions, several access paths to an object may coexist. A solution consists of introducing tombstones⁹ indicating whether a realization for the object currently exists, and forcing all the access paths to the object to use this tombstone.

For automatic objects, tombstones may be defined on a block basis. As regards controlled objects, one tombstone per generation is required. A major characteristic of tombstones is that, once allocated, they can never be reused. Due to this, tombstones are realized in a particular area, called the tombstone space (TS).

Access paths

Three kinds of information are manipulated in the addressing space—addresses, descriptors and values. Values may

appear in ST, VS, VH, or DS areas, according to their scope and lifetime.

For protection purposes,¹ objects are associated with descriptors; these may be completely defined at compile time, then they are implemented in the read-only DC area, or they are not fully defined before execution, then they have to be constructed dynamically in VS or VH areas, according to the object class (automatic or controlled).

Access to an object is gained through indirections across the address space areas. These may communicate between each other according to certain restrictions. Three kinds of communications have to be considered—capabilities, addresses and pointer values. The Appendix exhibits the authorized communications between areas, from which address formats can be derived. Before describing these formats (in the third section) a few words about pointer values and program structure are necessary.

Pointer values

The use of pointer values, if they are to be implemented just as addresses, is a means of violating the rules of languages concerning the manipulation of variables in relation with their types. One way to avoid this is to associate not only a reference to an object, but also a descriptor of this object, to a pointer value.⁴ By this means, a descriptor is associated with every access path to a variable, and therefore control can be exercised on the type of access which is attempted to it whatever access path is used.

Program structure handling

Block structure is taken under consideration by means of activation records (see for instance, the BURROUGHS B6700¹¹) on capability and value stacks, CS and VS. An activation record contains the following information:

- Dynamic links (DC, DV) to the block which activated this block.
- Type—block, procedure.
- Lexicographic level.
- Exit information indicating through which statements control can leave the block.
- Reference to the tombstone associated with the activation record.
- Condition enabling information.
- Display information characterizing the activation records of the blocks which statically encompass this block.
- Loop control information.
- Parameters area (see below).
- Realization of automatic objects (values and/or descriptors).

As regards parameters, since formal parameters define a new access path to an actual parameter, they should be considered as pointer values, so that the protection requirements just defined can be met.

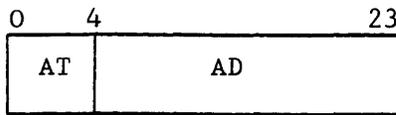


Figure 2—Operand addresses.

ADDRESS FORMATS DESCRIPTORS

Addresses

Three kinds of addresses are introduced, which allow a realization of the access paths in accordance with the restrictions defined in the appendix. Bit patterns are given as an indication.

- Operand addresses appear inside instructions, as shown in Figure 2, where

AT: Specifies the area and the kind of address defined in AD—direct or indirect.

AD: Is a displacement within the area or slot defined by AT. If AT specifies a stack, then AD is a pair (11, d) indicating the lexicographic level and a displacement inside the activation record.

- Internal addresses are purely dynamic and are used for special purposes (i.e. tombstones), as shown in Figure 3, where

AT: Same as before.

V: Validity field—If set to 0, indicates a dangling reference.

d: Displacement in the area specified by AT.

- General addresses may be created at compile time in the descriptor area DC, or at a run-time, to implement pointer values in particular. They may contain up to three kinds of information:

- The descriptor of an object or a reference to it (D).
- A reference to the next area of the access path (R).
- An additional displacement (P) used either when the reference R does not lead directly to the object, or for aggregate elements (see the following).

These three informations are not always required and four types of addresses are defined:

- Type 0 contains field D.
- Type 1 contains fields D and R.

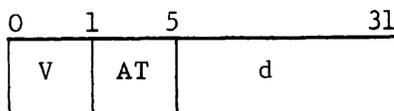


Figure 3—Internal addresses.

- Type 2 contains fields R and P.
- Type 3 contains fields D, R and P. This type is used to represent pointer values.

Type 3 addresses are shown in Figure 4, where

T: Type of the address.

AT: Specifies the area and the kind of address. There is a particular AT value which indicates that R contains a value, and not an address; this is used for intermediate results.

V: Validity field; V=0 indicates a null pointer value.

Descriptors

The data structures which are taken under consideration are elementary items, arrays and aggregates.

- Elementary item descriptors appear in the D field of Type 1 or Type 3 general addresses. They specify the different attributes of the object (arithmetic, string. . .). For strings, they contain the length of the area allocated to the string. For arithmetic data, base, scale, mode and precision are encoded.
- Array descriptors define both the array structure and the array element. This allows the detection of out-of-bounds references, and unauthorized manipulations on an element. A descriptor for a two-dimensional array is represented in Figure 5, where

N: Number of dimensions.

RVO: Relative virtual origin, i.e. displacement to the (virtual) element with subscripts 0, 0, . . . , 0.

LB, UB: Lower and upper bounds of subscripts for this dimension.

M: Displacement from one element to the next one in the same dimension.

- Aggregate descriptors consist of Type 0 addresses (corresponding to the main aggregate and any sub-aggregate) and a collection of Type 1 or Type 3 addresses (one for each terminal aggregate component). The Type 0 address contains the number of elementary components in the (sub-) aggregate, and a displacement to the descriptor of the first component. An example is shown in Figure 6. By this mean, an item is accessed in the same manner, whether or not it belongs to an aggregate. An array of aggregates is described as an aggregate of arrays.

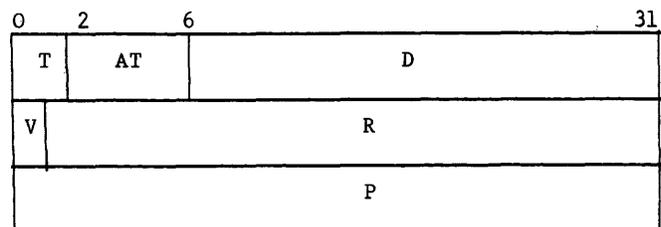


Figure 4—Type-3 addresses.

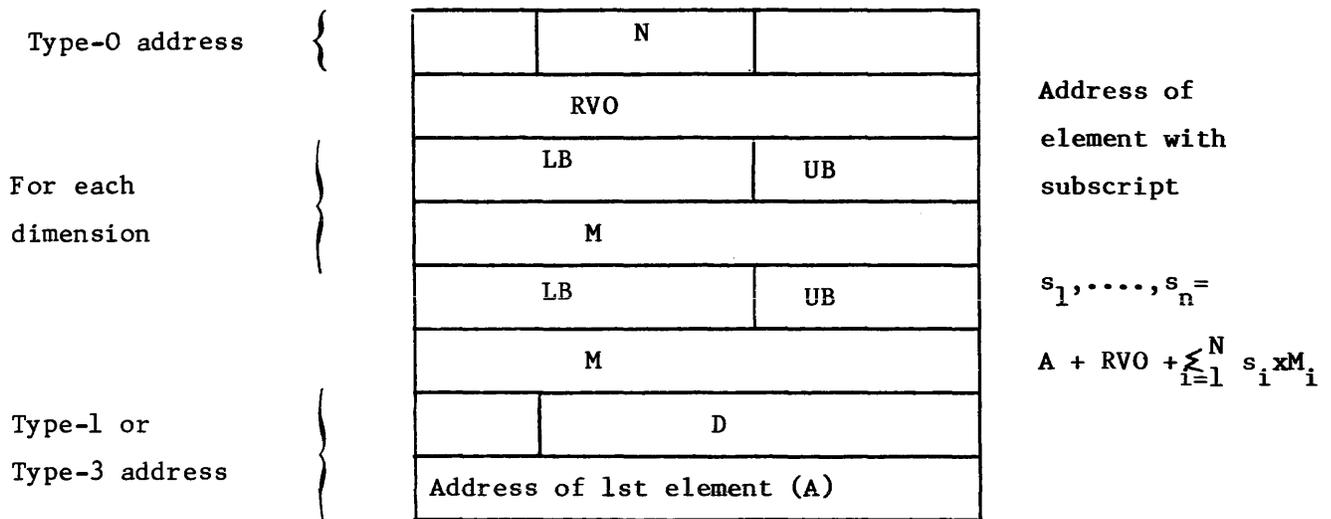


Figure 5—Array descriptor.

INSTRUCTION SET

The first question in designing the instruction set concerns the number of intermediate languages—should one define a single intermediate language, as it is done in most systems, or a high-level language, as it appears in the Burroughs B1700? Considering microprogram memory size, maintenance problems, training of software people and development costs, the best solution is to have a single intermediate language. As regards PL/1, FORTRAN and COBOL, a design based on PL/1 includes nearly all the language constructions of COBOL and FORTRAN. Only a few additional features are necessary to cope with particular constructions. It would be necessary to estimate the performance loss—if any—when executing COBOL or FORTRAN programs onto a PL/1-based architecture.

The following objectives must be kept in mind when designing an instruction set:

- *Safety*—The error detection must be as precise as possible, and appear as soon as possible.
- *Efficiency*—The most frequently used language constructions should be treated efficiently. Thus, the designer must have some knowledge about how the language is actually used, which implies run-time measurements in source programs.¹²
- The instruction set should provide facilities for compiler designers, and for the implementation of high level debugging aids.
- *Generality*—A few general mechanisms should be used, rather than locally-optimal implementation tricks.

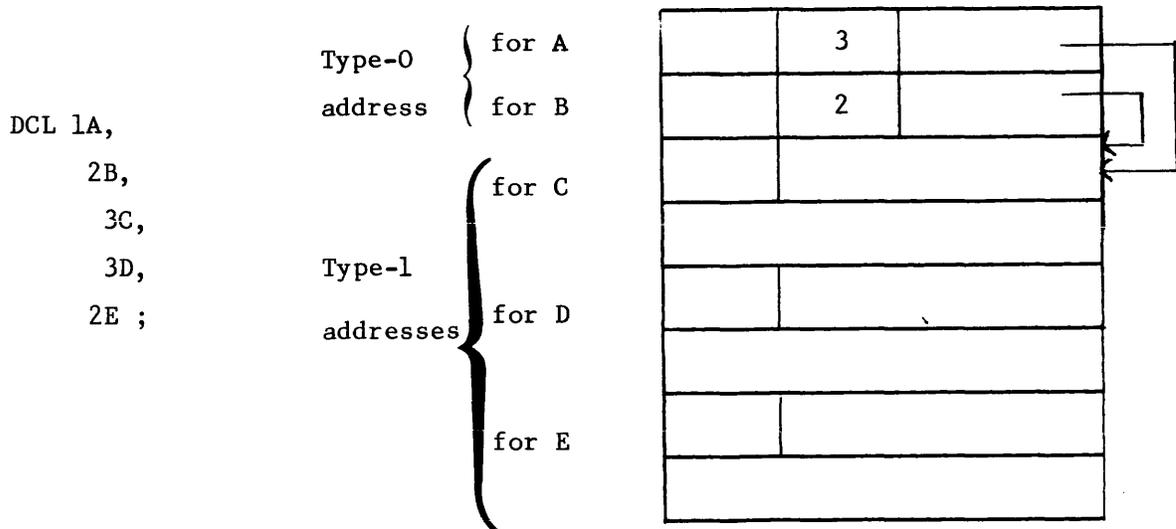


Figure 6—Aggregate descriptors.

Let us consider the various forms of instruction streams and related hardware architectures.

- *Stack machine and zero-address instructions.* This category may be illustrated by the Burroughs B6700—single or double precision data items are stacked, otherwise descriptors are stacked. Such a structure requires that operand values or descriptors be explicitly pushed on the top of the stack prior to any computation.
- *General registers and one-address instructions*—IBM/370 or CDC 6600 can be found in this category. Its major interest lies in an optimal handling of intermediate results. It requires that registers be explicitly loaded by instructions. In a descriptor-based architecture, it is mandatory that a description register be associated to each general (data) register. As a practical consequence, the complete register must be large enough to contain the largest data item, which may be too costly for small to medium machines.
- *Three-address machines* are reminiscent of the pioneer days; they may be illustrated by the Burroughs B3700 system. They are well suited to variable length items handling but two problems require special attention—intermediate result handling and array accessing.

Intermediate results or constants are represented as a special form of general addresses called IDV (Immediate Descriptor and Value). An IDV contains both the description and the value of an item, which saves one indirection. Some instructions may specify the creation of an IDV as the result of an operation; this is specified in the instruction opcode (I option). Source program measurements show (see, for instance, Reference 8) that the arithmetic expressions are generally extremely simple and do not involve any intermediate results. This is an argument in favor of three-address code, which generates instruction sequences which are more compact (and, therefore, interpreted more efficiently), as long as no intermediate result is required, with the counterpart of a higher cost for the execution of more complex expressions.

Again, according to program measurements, it appears that one reference among three is to array element; this means that array handling instructions should be given special care. On stack or general register machines, a rather long sequence of instructions is required. We suggest that two instructions be introduced:

- *build index list*
 $BINDL\ n,\ OPA_0,\ OPDA_1,\ \dots,\ OPDA_n$

which builds a list of n index values with operands $OPDA_1, \dots, OPDA_n$ at location OPD_0 , and

- *index*
 $INDEX\ OPDA_1,\ OPA_2,\ OPA_3$

which selects an element from the array defined by $OPDA_1$, with the index list found in OPA_2 , and stores it in OPA_3 . The instruction also performs subscript range checking, ac-

ording to the information found in the array descriptor. In the examples of instructions exhibited below, OP stands for *OP*erand, $OPDA$ for *OP*erand *D*escriptor *A*ddress, and OPA for *OP*erand *A*ddress. Instructions may be classified as follows:

- *Computational*
Examples:
 1. $ADDI\ OPDA_1,\ OPDA_2,\ OPA_3\ (OP_3 := OP_1 + OP_2)$
 I in $ADDI$ indicates that an intermediate result (IDV) is created at the location specified by OPA_3
 2. $INC\ OPDA_1,\ OPDA_2\ (OP_2 := OP_2 + OP_1)$
 3. $DECI\ OPDA_1\ (OP_1 := OP_1 - 1)$
- *Computational data movement*
Examples: $MVNZ\ OPDA_1$ moves numeric zero to OP_1 .
- *String operations, which require the use of descriptors*
Examples:
 1. $CAT\ [I]\ OPDA_0,\ OPDA_1,\ OPDA_2$ is the concatenation of two strings
 2. $SUBSTR\ [I]\ OPDA_0,\ OPA_1,\ OPA_2$ selects a substring in OP_0 , according to two integer values (origin and length) found in OP_1 and creates string descriptor for the string in OPA_2 .
- *String move instructions*
Examples:
 1. $MOVE\ [I]\ OPDA_1,\ OPDA_2\ (OP_2 := OP_1)$
 2. $MVS\ OPDA_1$: sets string OP_1 to 'spaces'
- *Index manipulation*—see $BINDL$ and $INDEX$ above.
- *Branching*—several forms of branching are considered. Relative to the current instruction, indirect (to cope with COBOL's PERFORM and ALTER statements), local to a block, or outside the current block. Note that this last case implies a lot of housekeeping work, especially on the stacks. A typical branch instruction contains operand(s) address(es) and branching address.
Examples:
 1. $BRLESS\ OPDA_1,\ OPA_2,\ BADDR$
 (if $OP_1 < OP_2$ then go to $BADDR$)
 2. $BRS\ OPDA_1,\ BADDR$
 ($OP_1 = \text{spaces}$ then go to $BADDR$)
- *Procedure and block control*—Call instructions specify the address of a list of parameter descriptors represented as an aggregate, and the address of an entry descriptor. For safety purposes, leaving a block resets its activation record to zero. When entering a block, the activation record is initiated by means of computational data, or string, or address movement instructions.
- *Address movement instructions* are the only way to alter the contents of a pointer value.
Examples:
 1. $SETPTR\ OPDA_1,\ OPDA_2$ forces pointer OP_2 to reference item OP_2 . The pointer value inherits the item description, and the reference to the item may be indirect, if it is accessed through a tombstone (then the pointer will reference this tombstone).
 2. $RESETPTR\ OPDA_1$ resets pointer OP_1 to the null value.

- Miscellaneous instructions contain among other things allocate and free statements on the heap, and instructions for the dynamic construction of descriptors.

CONCLUDING REMARKS

The instruction set we have briefly presented leads to compact object programs—a ratio of two has been observed for typical COBOL programs,² as compared with the IBM 370 code. Obviously, this instruction set should be tuned according to further measurement and to hardware and/or firmware requirements and constraints. The design of the HLL-machine has shown that a unique architecture may support PL/1, COBOL and FORTRAN. A great attention has been given to safety problems (e.g. dangling references) and programs could be run safely on this architecture. As regards efficiency, this architecture cannot be considered as realistic unless specially tailored hardware and/or firmware be defined for it.

REFERENCES

1. Battarel, G. J., and R. J. Chevance, "Requirements for a Safe PL/1 Implementation," *SIGPLAN Notices*, May 1978.
2. Chevance, R. J., and T. Heidet, "Static Profile and Dynamic Behaviour of COBOL Programs," *SIGPLAN Notices*, April 1978.
3. Denning, P. J., "Fault-Tolerant Operating Systems," *Computing Surveys*, Vol. 8, No. 4, December 1976, pp. 359-389.
4. Ecma, ECMA/Tc 10/ANS I-X3J1 PL/1 BASIS 1/12, July 1974.
5. England, D. M., "Architectural Features of System 250," *Infotech State of the Art Report 14: Operating Systems*, Infotech International Ltd., Maidenhead, England, 1972, pp. 395-428.
6. Fabry, R. S., "Capability Based Addressing," *CACM*, Vol. 17, No. 7, July 1974, pp. 408-412.
7. Feustel, E. S., "On the Advantages of Tagged Architecture," *IEEE Trans. Computers*, Vol. C-22, July 1973, pp. 644-656.
8. Knuth, D. E., "An Empirical Study of FORTRAN Programs," *Report No. CS-186*, Stanford University, 1970.
9. Lomet, D. B., "Schema for Invalidating References to Freed Storage," *IBM Journal of Res. and Devlt.*, January 1975, pp. 26-35.
10. Organick, E. I., *The Multics System: An Examination of its Structure*, MIT Press, Cambridge, Mass., 1972.
11. Organick, E. I., *Computer System Organization: The B 5700/B 6700 Series*, Academic Press, New York, 1973.
12. Wortman, D. B., "A Study of Language Directed Computer Design," *Technical Report CSRG-20*, University of Toronto, 1972.
13. LIS, *The System Implementation Language LIS*, CII-HB Documentation, 4549 E/EN.

APPENDIX—COMMUNICATIONS BETWEEN ADDRESS SPACE AREAS

TO \ FROM	DC	ST	LINK	VS	CS	TS	VH	DS
DC	0	1	1	1	1	0	0	0
ST	0	0	0	0	0	0	0	0
LINK	0	0	0	0	0	0	0	0
VS	0	1	1	1	1	1	1	0
CS	0	0	0	0	0	0	0	1
TS	0	0	0	1	1	0	1	0
VH	0	0	0	0	0	0	1	0
DS	0	0	0	0	0	0	0	0

0=forbidden; 1=allowed

Horizontal arrows indicate possible beginnings for an access path; vertical arrows indicate possible terminations

TO \ FROM	ST	LINK	VS	CS	TS	VH	DS
Pointer value (in ST, VS or VH)	1	1	1	1	1	0	0
LINK, CS	0	0	0	0	0	0	1
TS	0	0	1	1	0	1	0
DS, ST, VH, VS	0	0	0	0	0	0	0

A programming language for high-level architecture

by YAOHAN CHU

University of Maryland
College Park, Maryland

and

EDWARD RAY CANNON

International Computing Company
Bethesda, Maryland

The machine language of a computer is the programming language that the bare hardware can accept and interpret. In a von Neumann architecture, it is essentially a set of machine instructions and data formats. In a high-level computer architecture,^{7,8} the machine language is a high-level programming language since the hardware high-level architecture accepts and interprets this high-level language. Therefore, the programming language for a high-level computer architecture is the *high-level machine language*. For this reason, the programming language to be presented in this paper is called the *HLM language*.

Since the constructs of a high-level machine language affect intimately the constructs of the interpreting high-level computer architecture, the design of a high-level architecture begins with the design of a high-level machine language. This paper presents the design of a high-level machine language, while a separate paper describes the high-level architecture which implements this high-level machine language.

DESIGN CONSIDERATIONS

Before the HLM language is described, major design considerations that result in the choices of the particular data types, structures, operations and constructs are presented. These considerations serve as guidelines for the language design and are as follows:

- a. The overall consideration is *understandability* and *simplicity* of the language.
- b. The HLM language should have adequate language constructs for writing programs in applications of system programming, data processing, scientific computing and process control, since it is a *programming language*.
- c. The HLM language should have language constructs that typical programmers can *understand* well so that they can effectively use the language.
- d. The HLM language should have language constructs that facilitate the writing of *reliable* programs.

- e. The HLM language should have language constructs that help in creating a *simple* and *clean* interactive direct-execution architecture^{4,8} for its implementation.
- f. The HLM language should not be *over-designed* merely for the sake of seeking power and elegance of the language.
- g. The HLM language should have adequate language constructs for writing *high-level software*, since the high-level architecture may have software. For example, the HLM language may be used to write an interpreter for a high-level or a very-high-level language.
- h. The HLM language is designed with a particular regard to *microprocessor* implementation.
- i. The HLM language may become a *family of programming languages*. It will be a family because member languages will have a uniform syntax and a similar structure, but with a different choice of data types. Each member language will be simpler to implement since it will meet a limited need, yet it will be adequate for its programming purpose.

In case of a conflict among the previous considerations, a compromise is guided by the overall consideration.

PROGRAM

A program declares data and specifies data operations for data processing or computing. To specify complex operations, sequences of data operations are needed. Control operations are needed to sequence the data operations. These data operations and their sequencing in a program create the data flow and the control flow, respectively, of the program. This section introduces the concept of data flow and control flow of a program, and then describes the program elements and program structure of the HLM language.

Data flow and control flow

In any program, there are two flows—the data flow and the control flow. The data flow is the flow of the data

changes in the program as a result of data operations; it is described by *data flow statements*. The control flow is the flow of data operations sequences when the program is being executed; it is described by *control flow statements*. Examples of a data flow statement and a control flow statement are the assignment statement and the "if" statement, respectively.

In the HLM language, the data flow and control flow are organized so that all data flow statements are *embedded* in control flow statements. In this way, the order of executing the data flow statements is entirely directed by the control flow statements. Visibility of both the data flow and control flow to the HLM language programmer is one of the unique language features of the HLM language; this helps the programmer to understand program execution better.

Program elements

The program elements of a high-level language are those constituents which make up a viable program. The program elements of the HLM language are:

- a. Macro definition
- b. Data declaration
- c. Procedure definition
- d. Control flow statement
- e. Data flow statement
- f. Comment statement

Macro definitions describe the macros of a HLM program. Each macro definition specifies some program text for "substitution" when the macro name is later called. An example is shown in Figure 1 where macro *x* denotes "123," and macro exchange (*y,z*) denotes "y:=y+z."

Data declarations declare the names of the data storages together with their types and structures. An example is shown in Figure 1 where buffer *i* of data-type number of two digits, buffer *j* of data-type string of 10 characters, buffer *m* of data-type status of 'TRUE,' 'FALSE,' and 'dont_know;,' and array *k* with 10 array elements of data-type number of two digits are declared.

```

/*This is an example of the HLM language program*/
MACRO x=123 ENDM;
MACRO exchange(y,z)=y:=y+z ENDM;
BUFFER i OF NUMBER[2],
      j OF STRING[10],
      m OF STATUS[TRUE,FALSE,dont_know];
ARRAY k[10] OF NUMBER[2];
PROCEDURE a;
  BLOCK i:=INPUT(0); ENDB;
  CALL b(i);
  BLOCK OUTPUT(0):=i; ENDB;
ENDP a;
PROCEDURE b(BUFFER x OF NUMBER[2]);
  BLOCK x:=x*8; ENDB;
ENDP b;
ENDPROGRAM;
    
```

Figure 1—An example showing program elements and program structure.

Procedure definitions describe the procedures of a HLM language program. Each procedure consists of control flow statements and optionally local data declarations. An example is shown in Figure 1 where Procedures *a* and *b* are defined. Procedure *a* has three control flow statements but no parameters. Procedure *b* has buffer *x* of data-type number of two digits as the parameter which is called by reference. It has only one control flow statement.

Data flow statements specify data operations. Control flow statements specify the data flow statements to be next executed. Two types of control flow statements are shown in Figure 1—BLOCK and CALL. There are three BLOCK statements, each specifying one or more assignment statements to be executed. There is one CALL statement which calls Procedure *b*.

The comment statement provides an explanatory remark for improving readability and documentation. It is a string of characters from the character set enclosed by a pair of symbols, /* and */, and may appear anywhere a space character can except inside of a character string. An example of the HLM program is also shown in Figure 1. A more detailed example is shown in Reference 9.

Program structure

The program structure of the HLM language is chosen to be simple because simplicity is an overall design consideration. As illustrated in Figure 2, a HLM language program consists of macro definitions, data declarations, procedure definitions, and a statement to indicate the program end. An example of program structure is also shown in Figure 1. The order to the appearance of these program elements may be changed as long as a name is declared before it is referenced. The first procedure definition is the main procedure where program execution begins. The other procedures are to be called by the main procedure directly or indirectly. (A procedure is called indirectly if it is called through more than one procedure calls.)

Data

In designing a programming language, there are several aspects of data⁵ that need be considered—data types, data structures, data operations and data flow. In addition, there are control data types, control data structures and control data operations.

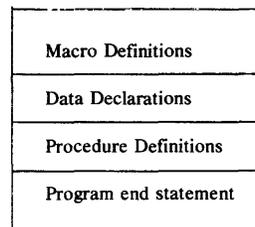


Figure 2—Program structure for a HLM language program.

Data types are the primitive elements which may be numerical, physical, or others that are chosen to represent the data. They determine the scope of the language to describe the data. Data structures are the program elements which actually contain the data; they offer the facility in the language to access the data. Data operations are those operations which operate on the data types and data structures. Merely providing a data type or a data structure in a programming language without an adequate provision of data operations does not make that data type or that data structure particularly useful. The "programmability" of programming language is often limited by the data operations available for the data types and data structures. Data flow refers to the changes of the values in the data structures. Control data types and control data operations refer to those data types and data operations that affect the control flow during the program execution. These aspects of the HLM language are discussed next.

Data types and operations

There are many data types in current programming languages. Examples are data types of real and label in Algol 60, fixed and floating-point numbers in Fortran, record in Cobol, printers and files in PL/1, integer and string in SIMPL-T and byte and address in PL/M. In Pascal, data types may also be defined by the programmer. In the HLM language, five data types are chosen—decimal number, character string, byte string, time and status.

Decimal numbers represent numerical objects. Character strings represent symbolic objects. A byte string represents

a bit string; however, a byte string is restricted to bit strings of multiples of eight-bit lengths. Time represents real time from which relative time, incremental time and delay can be derived. Status represents the condition after a test. A special case for status is the boolean status, which has the values of TRUE and FALSE.

In the HLM language, data operations for the five data types are shown in Table I. The five arithmetic operations are +, -, ×, /, and modulo; they are provided for decimal numbers, byte strings and time units. The four logical operations are .A., .O., .E., and .C. which represent logical and, or, exclusive-or, and complement, respectively. They are provided for byte strings. The six shift operations are SHL, SHR, ROL, ROR, RCL and RCR which represent logical shift left and right operations, rotate left and right operations, rotate-with-carry left and right operations, respectively; they are available only for the data type of byte string.

Concatenation, finding-length, finding-substring and conversion operations are provided for both character strings and byte strings. In addition, delay and conversion operations are provided for data type of time units, and the set operation is provided for data type of status.

Data structures and operations

There are many data structures commonly used in programming. Examples are arrays, queues, stacks, tables, trees, files and list structures.⁸ In the HLM language, data structures are declared in data declarations. Examples of data declarations are shown in Figure 3. The available data structure types are buffers, one-dimensional arrays, stacks, ports, clock, and files.

A buffer is a data storage which may store any one of the five data types (it is equivalent to the simple variable in a programming language where the variable stores a mathematical object). An array is a linear list of a fixed number of array elements which have the same data type. Each array element can be referenced by a subscript. Only one-dimensional arrays are adopted to keep the hardware array structure simple. A stack is a linear list of a varying number of stack elements of the same data type. It is a first-in-last-out structure. Only one end of the stack is accessible for insertion and deletion of stack elements.

A port is a "data storage" through which input data or output data flow. A clock is a special type which generates the real time, and a timer is a special type where time units are being accumulated. Both accept only the data type of time units. A file is a list of related file elements commonly called records. (The records physically reside on an external storage device.) Each record may be structured or unstructured. A structured record consists of fields and subfields and it can be of fixed or variable length. An unstructured record can be interpreted by means of a structured template.

The data operations for the data structures are shown in Table II. In brief, there are two operations for buffer, four for array, seven for stack, two for port, eight for file and one for clock.

TABLE I—Data Type Operations and Data Flow Statements

Data Type	Data Operations	Data Flow Statements (examples)
decimal number	5 decimal arithmetic operations	a: =b+c/d;
character string	Concatenate strings x and y Find length of string x Find sub-char string in string x, starting at ith char for j chars Convert char string to byte string	x:=x y; i:=LENGTH(x); y:=SUBCHAR(x, i, j); a:=CONVBYTE(x);
byte string	4 logical operations 5 byte arithmetic operations 4 shift operations Find subbyte string in string f, starting at ith byte for j bytes Convert byte string to char string Find length of string x	i:=j.A.k; k:=j+k; i:=SHR(j, a); f:=SUBBYTE(f, i, j); x:=CONVCHAR(a); i:=LENGTH(x);
time units	Convert x msec into date-and-time Delay n time units 5 integer arithmetic operations increment/decrement buffer x by 5 time-units	datetime:=CONVTIME(x); a:=a DELAY n; t1:=t2+t3; X:=INC 5; X:=DEC 5;
status	set buffer x to status on	see note

Note: This is a control operation which is to be specified by a control flow statement as shown in Table IV.

Any data structure may be subdivided into any number of fields or sub-fields which are organized by level numbers. Examples of data structures with fields are shown in Figure 3.

Following conventional usage, the names of data structures in a data declaration may sometimes be referred to as *variables*.

Control data types

A control data type is the data type which affects the flow of control in program execution. For example, the value of a test in an if-statement is a control data type since this value causes a change of control flow. In a conventional programming language, the control data types are not specially recognized; they are not considered as separate data types.

In the HLM language, there are two control data types—status and time units; they have been shown in Table I. A status is a datum stored in a data storage specifically for control flow modification. The datum can be numeric, boolean (i.e. TRUE or FALSE), or symbolic (such as “on” or “complete”). The time unit represents such units as hours, minutes, seconds and microseconds of time. For convenience, it may be abbreviated as *tut*.

Data flow statements

A data flow statement specifies one or more data operations. It consists of a sequence of operands and operators. The syntax of a data flow statement requires it to begin with a data-operator keyword such as PUSH and SIZE. Examples of data flow statements are shown in Tables I and II,

where each data flow statement begins with a data storing operation indicated by the data operator “:=”. The data storing and data reference operations together with the input and output data flows, the timing reference and the operator precedence are subsequently described.

Data reference

Data reference is the appearance of a declared name in a data flow statement. It is a data operation since the appearance means the access of the value of the name. It is the most common data operation. If an operator were chosen to represent this operation, this operation could be “fetch *a*” or “access *a*” where *a* is a declared name. No operator is used to represent this data operation in programming languages. This practice is followed in the HLM language.

Data storing

Another often-used data operation is data storing which stores an operand into a data storage. The operand is the value of a declared name or the value of an expression; the data storage is a declared data structure. If an operator is to be chosen to represent the data storing operation, this data operation could be: “store *a* into *b*” or “store *a+b* into *c*,” or “assign *a* into *c*.” Common practice in programming language uses an assignment operator such as “:=” to represent it. In the HLM language, this practice is followed. However, the assignment statement or statements are enclosed by the reserved control flow delimiters, BLOCK and ENDB to enhance the visibility of the data flow statements and allow a simpler implementation.

```

ARRAY      program__memory[65,536] OF BYTE__STRING[1];
ARRAY 01   am[am__size],
           02 type of CHAR__STRING[1],
           02 name OF CHAR__STRING[10],
           02 locn OF NUMBER[2],
           02 ptr1 OF NUMBER[2],
           02 ptr2 OF NUMBER[2];
BUFFER     am__mara OF NUMBER[3],
           am__free OF NUMBER[3],
           x OF TIME__UNIT[MSEC];
STACK 01   nestack, /*nesting stack*/
           02 type OF CHAR__STRING[1],
           02 name OF CHAR__STRING[10],
           02 locn OF NUMBER[2],
           02 ptr1 OF NUMBER[2],
           02 ptr2 OF NUMBER[2];
STACK
BUFFER     estack OF STATUS[syntax.execute]; /*execution stack*/
           calledproc OF CHAR__STRING[10],
           fp__loc OF NUMBER[2];
CLOCK 01   wallclock,
           02 MONTH OF NUMBER[2], /*note that MONTH, DAY, YEAR, HOUR,*/
           02 DAY OF NUMBER[2], /*and MIN are reserved words*/
           02 YEAR OF NUMBER[2], /*since only reserved words are capitalized*/
           02 HOUR OF NUMBER[2],
           02 MIN OF NUMBER[2];

```

Figure 3—Examples of data declarations.

TABLE II—Data Structure Operations and Data Flow Statements

Data Structure Type	Data Operations	Data Flow Statements (examples)
buffer	reference buffer x store into buffer x	y:=x; x:=y;
array (one-dim.)	reference array element x[n] store into array element x[n] find number of elements of array x search array x for argument y	y:=x[n]; x[n]:=y; y:=SIZE(x); y:=SEARCH(x) FOR (y);
stack	reference nth element from top of stack x push down y into stack x pop up from stack x into buffer y test stack x for empty find number of elements of stack x initialize stack x to empty search stack x for argument y and return index from stack top for the first occurrence of y. (return -1 if none found)	y:=x[n]; x:=PUSH(y); y:=POP(x); x=EMPTY; y:=SIZE(x); x=EMPTY; z:=SEARCH(x) FOR (y);
port	input from port 1 into buffer cmd output '@' to port 0	cmd:=INPUT(1); OUTPUT(0):='@';
file	open input file x open output file x close and save file x close and delete file x read DIRECT file x record n write DIRECT file x record n read SEQUENTIAL file x write SEQUENTIAL file x	OPENIN(x); OPENOUT(x); CLOSESAVE(x); CLOSEDELETE(x); READIR(x[n]); WRITEDIR(x[n]); READSEQ(x); WRITESEQ(x);
clock	read clock x in declared time units	READ(x);

Input and output

In the HLM language, the input data or the output data flow through a pseudo-data storage called "port." An I/O device is connected to a particular port; this connection is determined by the system configuration. There are a maximum of 128 input ports and 128 output ports; the port is represented by the reserved words INPUT or OUTPUT, and the port number is indicated by a numerical subscript. Examples are shown in Table II.

Clock

A clock is a data source which generates the data type of time. It generates real time for time referencing and for synchronizing hardware processors. Reserved word CLOCK is used to represent the real time clock. A data declaration is needed to declare the names and time_units desired for

one or more clocks, though all the clocks give the same real time. A reading operation to CLOCK gives a reading in its declared time_units.

Expressions

An expression is concatenation of operands and operators to specify a sequence of operations. A numerical expression is a sequence of numerical operands and arithmetic operators. In the HLM language, there are five types of expressions corresponding to the five data types. The order of evaluating the operations in an expression follows the precedence of the operators. In the HLM language, the precedence follows common convention.

CONTROL

Control refers to those language constructs that may change the path of program execution. There are several aspects of control that need to be considered—control types, control operations, control flow and control structures. These aspects for the HLM language are discussed next.

Control types

The control type refers to the manner in which the control flows during program execution. In a conventional programming language, the control flow follows the written order of the statements of the program; this type of control is sequential. There are other types of control. For example, PL/1¹ has the concurrent type of control in the name of multiple-tasking. PL/M³ has the interrupt type of control to meet the needs of a microprocessor system. CDL² permits non-sequential execution of statements.

In the HLM language, two control types are selected—sequential and interrupt. The sequential type of control permits the execution of control flow statements according to the order of the control flow statements in a procedure. The execution begins with the main procedure which calls directly or indirectly the other procedures.

The interrupt type of control permits the suspension of the control flow and transfers the control flow to a predefined interrupt procedure. This interrupt procedure is then executed and returns to where the control flow was interrupted. The interrupt type of control also permits interruption by software.

Boolean expression

The application of a relational operator (i.e. =, ≠, >, <, ≥, ≤) to two data storages of the same type yields an expression of the control type boolean. All six relational operations are available to four data types, and only two (=, ≠) are available to the remaining data type STATUS. (See Table III.)

TABLE III—Control Data Operations and Boolean Expressions

Data Type	Control Data Operation	Boolean Expression (examples)
decimal number	6 relational operations 4 boolean operations	$x < y$ AND $y > z$
character string	6 relational operations 4 boolean operations	$x < y$ AND $y > z$
byte string	6 relational operations 4 boolean operations	$x < y$ OR $y < z$
time units	6 relational operations 4 boolean operations	$x < y$ OR $y < z$
status	2 relational operations (= and ≠) 4 boolean operations	$x = \text{on}$ $x = n$ AND $y = \text{off}$

The value of a boolean expression can be negated by prefixing it with the unary boolean operator NOT. Two boolean expressions can be joined by one of the three binary boolean operators (AND, OR, or XOR) to form a more complex boolean expression.

Note that all boolean expressions must contain at least one relational operator with one exception. If a data storage of type STATUS's value is either the status constant TRUE or FALSE then it is considered to be a boolean expression.

Control flow statements

Control flow refers to the order in which the control flow statements are executed. Since the data flow statements are embedded in the control flow statements, control flow also refers to the order in which the data flow statements are executed.

In the HLM language, there are two control types—sequential and interrupt. Skeleton control flow statements are shown in Table IV. The control flow statements for the sequential control type are:

- a. Block statement
- b. If statement
- c. While statement
- d. Call statement
- e. Return statement
- f. Macro statement
- g. Procedure statement
- h. Set statement

Each of these control flow statements is single-in-single-out. These simple control constructs make the HLM language a structured programming language.

The control flow statements for the interrupt control type are:

- a. Interrupt statement
- b. Disable statement
- c. Enable statement

TABLE IV—Control Operations and Control Flow Statements

Control Type	Control Operation	Control Flow Statement (skeleton)
sequential	grouping	BLOCK ... ENDB;
	alternative	IF ... THEN ... ENDI; IF ... THEN ... ELSE ... ENDI;
	looping	WHILE ... DO ... ENDW;
	macro declare	MACRO ... ENDM;
	procedure declare	PROCEDURE ... ; ... ENDP; PROCEDURE ... RETURNS ... ; ... ENDP;
	procedure call	CALL ... ;
	procedure return	RETURN; RETURN ... ;
interrupt	set a status	SET ... TO ... ;
	execution interrupt	INTERRUPT ... ;
	disable interrupt	DISABLE ... ;
	enable interrupt	ENABLE ... ;

The interrupt statement suspends the current control flow, and transfers it to a specially defined procedure. The enable and disable statements enable and disable the ability of the interrupt to occur, respectively.

Control structure

Control structure refers to the manner in which the declarations and statements are organized to steer the control flow. For modularity, declarations and statements are allowed to be grouped together in a certain manner. For example, the compound statement in Algol-60 and the do-end group in PL/1 allow the grouping of statements. In Algol-60 and PL/1, both statements and declarations may be grouped into a so-called block structure; the block structure may permit a complex interaction between the data flow and control flow of a program.

In the HLM language, simple control structure is sought-after. There is no ALGOL-like block structure. Only three control structures are permitted—nesting structure, macro substitution and procedure structure. The nesting structure allows the nesting of two or more control flow statements (e.g. *if* statement enclosing a *while* statement). The macro and procedure structure are described in the next sections.

Macro

As mentioned before, macro definitions in a HLM language program describe the macros of the program. Each macro definition specifies some program text for substitution. The program text is substituted at where it is called. Syntax of this program text is not recognized until after the macro is called and the textual substitution is made.

In the HLM language, macro definitions within a macro definition are not allowed for simplicity in implementation. Nevertheless, they serve the useful functions of text substitution and text compacting.

Procedure

As mentioned before, procedure definitions in a HLM language program describe the procedures of the program, and the first procedure is the main procedure.

Procedure structure

There are three relations among the defined procedures, which are the procedure defining, calling and returning relations. These relations are called collectively the *procedure structure*.^{5,6}

In the HLM language, simplicity of the procedure structure is sought-after. As a result, a "simple" procedure structure is chosen which has the following characteristics:

- a. No procedure may be defined within another procedure (i.e. no nested procedure definitions).
- b. No procedure may be defined or called recursively.
- c. Each procedure always returns to where it is called.

The simplicity of the previous procedure structure contributes toward simpler program writing and in turn more reliable programs.

Procedure parameters

A procedure may need no parameters if all variables are declared globally. However, in the HLM language, there can be local data declarations and parameters are permitted in a procedure definition. The parameter can be the name of any data structure of any data type. However, it can be neither a procedure name, nor a macro name. All parameters are called *by reference* or *by value*.

It should be noted that procedure parameters contribute to the interaction between the data flow and the control flow of the program. This interaction is needed, but it should be minimized and made visible for the purpose of contributing toward reliable programs.

Function

It is common to have functions in a programming language. For example, there are function subprograms in Fortran and typed procedures in Algol-60. In the HLM language, no specific designation is provided for functions. For most commonly-used functions, they may be pre-declared as data operators. Otherwise, a function is regarded as a special case of procedure where the procedure returns with a single value for any data type and a call statement is still required.

LEXICALITY

Lexicality deals with the symbols and codes of the language. It plays an important role since it greatly influences

the appeal of the language to the potential users as well as the implementability of the language on a computer system. In this section, we choose the character set and code. We describe how constants are written. We specify how names are spelled, and what the operators and reserved words are.

Character set and code

The seven-bit ASCII X3.4-1968 code consists of the following two types of symbols:

- a. **Single character symbols** which consist of 52 letters, 10 digits and 33 special characters.
- b. **Control symbols** which consist of 33 reserved words.

An escape function represented by symbol ESC is provided in order to allow the ASCII code to be expanded beyond the 128 characters.

In order to guarantee that the character set is capable for information interchange between computer systems and communication systems, the 95 single character symbols of the seven-bit ASCII code are chosen as the character set of the HLM language. From this character set, the constants, the identifiers and the operators are formed.

Constants

Constants are data values. The constants for the data types of the language are described below.

Numerical constants

A number is a numerical constant. For the data type of decimal numbers, a numerical constant is written as a decimal number, which can be positive or negative. A decimal point may exist if it is needed. Examples of numerical constants are:

-579, 579.0 and +57.95

The permissible range of decimal numbers is to be chosen during implementation; the default range is chosen to be 15 digits including a sign.

Character-string constants

A character-string constant is a string of characters enclosed by a pair of apostrophes. In case an apostrophe is a character of the string, this apostrophe is written as a double-apostrophe. Examples of character-string constants are:

`ABC\$*+ =/?` and `ABC` `DEF` `GHI`.

The size of a string is limited by the available memory space.

Byte-string constants

A byte-string constant consists of one or more bytes in concatenation (i.e. 8, 16, 24, etc. bits). It can be written in hexadecimal, octal, or binary form. The constant must be preceded by letter H, Q, or B and followed by a hexadecimal, octal or binary number in a single-quote pair. Examples are:

H'A1B2C3', Q'234567', Q'255', and B'10101010'.

Status constants

A status constant is an identifier or a decimal number. It first appears when the status is defined in a data declaration. It represents a condition to be used as an operand in a boolean expression in either the SET, IF or WHILE control statements. Examples of status constants are: 321, on, off, TRUE, FALSE, and ready_queue_empty.

Time constants

A time constant is a numerical value in time units. The time units can be microseconds, seconds, minutes, hours, days, months, years or combinations thereof. Examples are 10 microseconds, 15-hour/20-minute/25-second, 1977/april/15. The time units of a clock are declared in a clock declaration. The generic term for a time-unit is coined to be a "tut."

Identifiers

Identifiers are the names for data types, data structures, procedures, macros, statuses, subscripts, fields, etc. Examples have been shown in Figure 3. An identifier name is a character string consisting of letters, digits and character underscore; with the first character being a letter. Although there is no limit to the length of the identifier name, the identifiers in a program are distinguished by the first eight characters. This choice makes the size of the hardware's symbol table reasonable.

Operators

Operators are the terminals of the language. They consist of single-character operators, two-character operators, and reserved words. There are 20 single-character operators, 7 double-character operators, and 84 reserved words. Note that a blank is also a delimiter which is required for separating symbols.

A FAMILY OF LANGUAGES

The aforementioned programming language can be organized into a family of member languages. In this way, the

programmer needs to learn only a member language and the compiler/interpreter can be simpler. It is a family because all the member languages have the same program structure, control structure and data structure. Their differences are in the choice of the data types and control types and their associated data and control operations.

A member language for software implementation may choose all the five data types and both control types since it needs all of the descriptive power of the language. A member language for computation needs only the data types of number and status and the sequential control type. A member language for data processing needs to choose the data types of number, character string, file and status and the sequential control type. A member language for real-time control needs the data types of number, byte-string, status, time and both control types. A member language for microprocessors needs the data types of byte-string, status and time and both control types; this choice of data and control types is made to match the microprocessor capability.

For each of the member languages of the family, a compiler or an interpreter or both may be constructed. The implementation can be made simpler for the whole family by providing a family of software modules so that the compiler or interpreter of a member language could be built from a subset of these software modules.

CONCLUDING REMARKS

A programming language serves as a bridge between the programmer and the computer hardware. In the past, a programming language may have been designed for its power and elegance. The language as a result may have become hard to understand, long to learn and costly to implement. The language may well have been over-designed. The design of the HLM language has been an engineering undertaking with both the programmer's viewpoint and the language implementation taken into consideration.

To the programmers, the language is designed with a primary consideration of simplicity and understandability. The following language concepts and constructs have been adopted.

1. The concept of the data flow and control flow in a program and visibility of their interaction.
2. Inclusion of the data types of status and time and their data operations.
3. Data structures can all be declared with fields and subfields.
4. Data operations are provided for all data types and all data structures.
5. Treatment of inputs and outputs as data sources and sinks, respectively. Input and output data transfers are treated as input and output data operations, respectively.
6. Provision of a real-time clock is provided together with data operations.

7. Simplicity of control structure. There are no block structures, no recursive procedures, no nested procedure definitions and no nested macro definitions.
8. The use of single-in-single-out control flow statements.
9. Provision of both sequential and interrupt control types.

For implementation considerations, the language is designed to give a simple interpretation model. Costly and unnecessary constructs are avoided or eliminated. Implementation by both compilation and interpretation is considered. The following are the specific considerations:

1. Separation of the data flow and control flow interpretation.
2. Simplified model as a result of no block structures, no recursive procedures, no nested procedure definitions and no nested macro definitions.
3. Presence of reserved words at the beginning of each declaration or statement.
4. Declaration of the number of digits, characters and bytes for the data types of number, character string and byte string, respectively.

The following language constructs should be excluded for the sake of simpler implementation, but they have not because of more effectiveness in programming:

1. Provision of field and subfield definition for any of the data structures.
2. Provision of data operations for all of the data types and data structures.
3. Provision of procedure parameters.

The HLM language is now under an experimental and evaluation phase. During this phase, the language is used for writing programs. An interactive interpreter for a member language is being designed and implemented for experimental and evaluation use. The HLM language will be revised and refined as a result of this phase.

ACKNOWLEDGMENT

This research is supported by Grant MCS-75-05505-A01 from the National Science Foundation to the Department of Computer Science of University of Maryland.

REFERENCES

1. "PL/1 (F) Language Reference Manual," GC28-2801, IBM System/360 Operating System, IBM Corporation.
2. Chu, Y., "CDL—A Very High-level Language," *Technical Report TR-317*, Department of Computer Science, University of MD, July, 1974.
3. "8080 and 8080 PL/M Programming Manual," Intel Corporation, Rev. A, 1975.
4. Chu, Y., and E. R. Cannon, "Interactive High-level Language Direct-execution Microprocessor System," *IEEE Transactions on Software Engineering*, June, 1976, pp. 126-134.
5. Chu, Y., "Introducing a Software Design Language," *Proceedings of the Second International Conference on Software Engineering*, San Francisco, October, 1976.
6. Chu, Y., "Procedure Structure," *Technical Report TR-486*, Department of Computer Science, University of Maryland, October, 1976.
7. Chu, Y., "Architecture of a Hardware Data Interpreter," *Proceedings of the Annual Computer Architecture Symposium*, Silver Spring, Maryland, March, 1977.
8. Chu, Y., "Direct-Execution Computer Architecture," *Proceedings of IFIP Congress 77*, Toronto, Canada, August, 1977.
9. Chu, Y., and E. R. Cannon, "A Programming Language for HLCA," TR-534, Department of Computer Science, University of Maryland, May 1977.

Data management in distributed data bases*

by C. V. RAMAMOORTHY and BENJAMIN W. WAH

University of California
Berkeley, California

INTRODUCTION

The recent advances in large-scale integrated logic and memory technology, coupled with the explosion in size and complexity of the application areas, have led to the design of distributed architectures. Basically, a *Distributed Computer System (DCS)* is considered as an interconnection of digital systems called *Processing Elements (PEs)*, each having certain processing capabilities and communicating with each other. This definition encompasses a wide range of configurations from an uniprocessor system with different functional units to multiplicity of general-purpose computers (e.g. ARPANET). In general, the notion of "distributed systems" varies in character and scope with different people.³⁰ So far, there is no accepted definition and basis for classifying these systems. In this paper, we limit our discussion to a class of DCSs which have an interconnection of dedicated/shared, programmable, functional PEs working on a set of jobs which may be related or unrelated.

Due to the information explosion and the need for more stringent requirements, the design of efficient coordination schemes for the management of data on a DCS is a very critical problem. Data on a DCS are managed through a data base. A *Data Base* is a collection of stored operational data used by the application systems of some particular enterprise,^{6,12} and a *Distributed Data Base (DDB)* can be thought of as the data stored at different locations of a DCS. It can be considered to exist only when data elements at multiple locations are interrelated and/or there is a need to access data stored at some locations from another location. Due to the ever-increasing demand for on-line processing, there is a need for decomposing very large data bases into physically or geographically dispersed units and/or integrating existing data bases held in physically isolated nodes into a single, coherent data base that will be available to each of the distributed nodes.

In this paper, the design issues and solutions for resource management of data on a DDB are studied. The different aspects of resource management are categorized in the next section. These management issues are part of the issues related to the operational control of a DDB and are con-

cerned with the management of data as resources. They can be divided into three related levels, namely, the query level, the file level and the task level. The query level is concerned with the processing of user queries and requests so that parallelism in processing can be maximized, and the amount of communications on the system can be minimized. On the file level, the related issues are the compression of data files for efficient storage and communication, as well as the placement and migration of files for efficient accesses. On the task level, the objective is to schedule the requests so that overlap in processing can be maximized. These issues and some of the corresponding solution algorithms are studied in detail in the third to sixth sections respectively. Finally, the seventh section provides some concluding remarks.

RESOURCE MANAGEMENT OF DATA IN DDBS

There are many issues in the design of a data base, among which are the issues in logical organization, architectural designs, operational control and evolution. These issues have been discussed in Reference 31 and will not be repeated here. A summary of the issues in the design of a DDB are shown in Figure 1. In this paper, the resource management issues of data and files on a DDB are studied. The specific data management issues investigated are:

Query decomposition on DDBs

A *query* is an access request made by a user or a program in which one or more files have to be accessed. When multiple files are accessed by the same query on a DDB, these files usually have to reside at a common location before the query can be processed. Substantial communication overhead may be involved if these files are geographically distributed and a copy of each file has to be transferred to a common location. It is therefore necessary to decompose the query into sub-queries so that each sub-query accesses a single file. These sub-queries may then be processed in parallel at any location which has a copy of the required file. The results after the processing are then sent back to the re-

* Research supported by Ballistic Missile Defense Contract DASG60-77-C-0138.

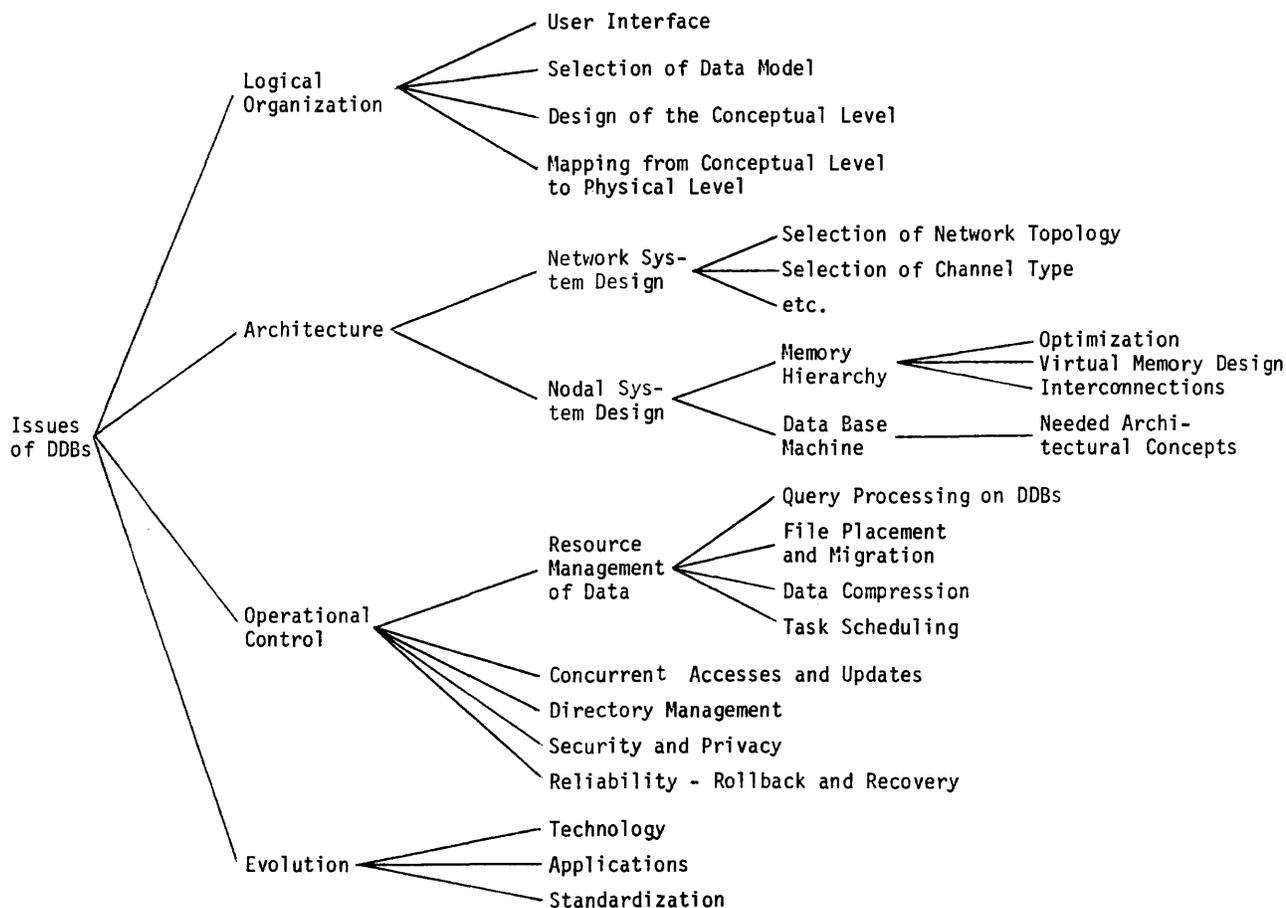


Figure 1—Classification of issues in distributed data base systems.

questing location. It is generally true that the amount of communications needed to transmit the results is much smaller than the amount needed to transmit the files. This approach has been proposed in the design of the centralized version of INGRES⁴¹ and is extended to the design of SDD-1,⁴² and distributed INGRES.⁸ However, in some cases decomposition is impossible and some file transfers are still necessary. In order to avoid these extra transfers, a technique is proposed in the third section so that redundant information is added to the files and non-decomposable queries can still be processed without any file movements.

Data compression

Data compression is any reversible encoding technique that produces a measurable reduction in the size of the data encoded. By reversible, it is meant that the original data is recoverable from the compressed form. Due to the growth in the size of information processing, it is necessary to develop good data compression techniques which reduce the size of the stored information and the amount of inter-node communications. This issue is discussed in the fourth section.

File placement and migration

This issue relates to the distribution and migration of data base components, namely, files and control programs, on the DDB with the objective of minimizing the overall storage, migration, updating and access costs on the system. A file assignment algorithm is proposed in the fifth section.

Task scheduling

Requests on the DDB must be scheduled so that high parallelism and overlap can be achieved. The request may be a single word fetch or it may be a page or file access. The parallelism on the DDB is important because in order to attain high throughput, the parallel hardware and resources must be efficiently utilized. The control of task scheduling can be distributed or centralized. In distributed control, each node may act independently and coordinate with each other. In centralized control, there is a primary node in which all scheduling control will be performed there. The decision of which is the better control mechanism depends very heavily on the interconnection structure and the

communication overhead involved. This issue is discussed in the sixth section.

The relationships among the various data management issues are shown in Figure 2 where a relation \rightarrow is said to exist between two design issues \hat{a} , \hat{b} , i.e. $\hat{a} \rightarrow \hat{b}$, if the solution of \hat{b} is transparent to the solution of \hat{a} . That is, the solution of \hat{a} is not affected by the solution to \hat{b} , but not vice versa. The solution to \hat{a} can therefore be developed independent of \hat{b} . In Figure 2, it is seen that generally, task scheduling is transparent to file placement and migration which in turn could be transparent to data compression and query decomposition. Algorithms for data compression and query decomposition can therefore be developed independently. In developing algorithms for file placements and migrations, the solutions for data compression and query decomposition should be taken into account. However, in most cases, assumptions can be made about their solutions and the file placement and migration problem can be solved independently. For example, it may be assumed that all queries which access multiple files may be decomposed into sub-queries that access single files. The file placement and migration problem for multiple files is therefore decomposed into many single file optimization sub-problems. It must be noted that other operational control requirements may also impose restrictions on the solutions to the data management issues. For instance, different reliability requirements may demand different lower bounds on the number of copies of a file on the DDB; different concurrency control mechanisms may have different costs on the file placement problem; etc. Reasonable assumptions must therefore be made about these operational control requirements in order to determine their effects on the resource management issues and to solve these issues independently.

QUERY DECOMPOSITION ON DDBS

The approach using query decomposition is geared towards relational data bases.⁴ In a relational data base, data

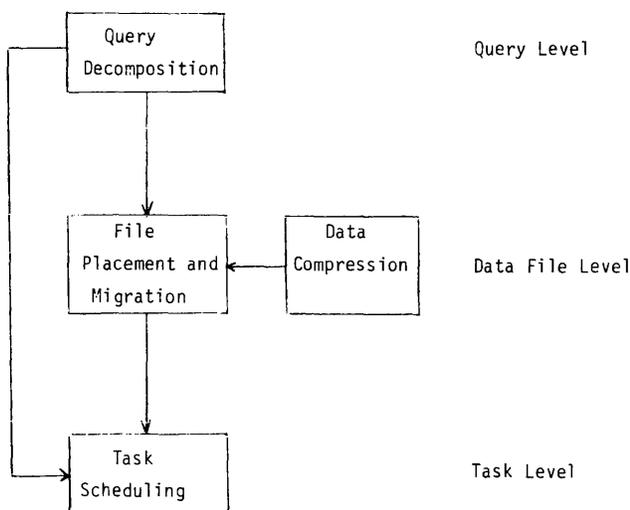


Figure 2—Relationships among various data management issues.

is viewed as relations of varying degree, the degree being the number of distinct domains participating in the relation. Each instance of a relation is known as a tuple, which has a value for each domain of the relation. Thus a relation can be simply represented in tabular form with columns as domains and rows as tuples. In query decomposition, optimization is performed on the processing of a single query originated at a node. The objective is to decompose a multiple relation query into as many single relation sub-queries as possible so that data (relation) movements from one node to another can be minimized.^{41,42,8,13} However, there exists non-decomposable queries which require all the relations that they access to be present at a common location. A large number of relation transfers may be needed if these relations are geographically distributed. In order to avoid these extra relation transfers, a technique utilizing redundant information is proposed here. Instead of decomposing queries that access multiple files, it may be sufficient to provide redundant information in each relation so that multiple relations do not need to reside at a single location before the query can be processed. This will be illustrated later in this section. We begin by first examining the different types of queries on a relational data base.

A query on a relational data base consists of two parts: the part specifying the domains of the relation to be retrieved and the part specifying the predicate which is a quantification representing the defining properties of the set to be accessed. Let S be a relation of domains $s\#$, $sname$, $status$, $city$; and SP be a relation of domains $s\#$, $p\#$, qty . The queries on a relational data base can be classified into the following categories:⁶

- *Retrieval Operations*

- a. *Single Relation Retrieval*—The predicate representing the defining property of the set to be retrieved is defined on the same relation as the set.
E.g. GET (S.sname): S.city="Paris" AND S.status>10
- b. *Multiple Relation Retrieval*—The predicate, as well as the set to be retrieved, may be defined over multiple relations.
E.g. GET (S.sname): (SP.s#=S.s# AND SP.p#="P₂")
Relation SP and S must be available simultaneously before the query can be processed.

- *Storage Operations*

- a. Single Relation Update
- b. Multiple Relation Update
- c. Insertion
- d. Deletion

- *Library Functions*

These represent more complicated operations on the predicate than the equality operations, e.g. counting the number of occurrences, selecting the maximum/minimum etc.

A query which is defined over multiple relations is not decomposable into single relation sub-queries when it has a

logical relation defined over a common domain of these multiple relations. For example, the query:

GET (S.sname): (SP.s#=S.s# AND SP.p#='P₂')

is not decomposable into single relation retrievals because there is a logical relation "=" which is defined over a common domain s# of the relations S and SP. These relations must be available simultaneously at a common location before the retrieval or update operations can be performed. It is noted that the common domains of these multiple relations actually represent multiple copies of the same domain on these relations (although the information they contain may not be identical). A lot of transfers can be eliminated if their common information is represented in both relations. For example, in processing the query:

GET (S.sname): (SP.s#=S.s# AND SP.p#='P₂')

on two geographically separated relations, S and SP (Figure 3a), it may be necessary to transfer relation S to the node where SP resides and then process the query there or vice versa. However, if the information SP.s#=S.s# are compiled beforehand into the two relations (Figure 3b), then it is only necessary to send the query to the location where S or SP resides and the query can be processed there.

This technique poses several problems. First, it is necessary to take one extra bit for each tuple in order to compile this piece of information. If the amount of information to be added is large, (e.g. when the number of different predicates defined on a common domain of two relations is large), the size of the extra storage space may be significant. Second, when the common domain of one relation is modified, it is necessary to "multiple update" all the common domains of the other relations in the data base. Referring to Figure 3b, if an extra tuple with s#=2 and sname="Boston" is added to relation S, then it is necessary to update the SP.s#=S.s# information in relation SP because relation SP contains a tuple with s#=2. If updating activity is frequent, then the "multiple update" cost is large. Third, this technique requires that the data base designer to be able to estimate the amount of additional information to be compiled into the relations. A possible technique is to pre-analyze the type of predicates used in retrievals and updates and to determine what are the essential information to be compiled into the relations. A compromise should be made between introduc-

S	s#	sname	SP	s#	p#
	1	New York		1	A1
	3	San Francisco		2	A1
	5	Chicago		3	A2
				4	A2
				5	P2

Figure 3a—Relations S and SP.

S	s#	S.s#= SP.s#	sname	SP	s#	S.s#= SP.s#	p#
	1	1	New York		1	1	A1
	3	1	San Francisco		2		A1
	5	1	Chicago		3	1	A2
					4		A2
					5	1	P2

Figure 3b—Relations S and SP with (S.s#=SP.s#) information compiled into the relations.

ing extra information with additional storage space and higher cost in multiple updates and reducing the amount of relation transfers. It would be advantageous for the more frequently used predicates and less advantageous for others.

DATA COMPRESSION

With the increase in the amount of information processing, it is important to keep the utilization of the memory high. The information content of data stored in large alphanumeric data bases is usually low. Further, as the processing becomes distributed, the communication overhead of transferring data from one location to another is usually substantial. In order to keep the utilization of the storage sub-system high, and to keep the amount of data transferred over communication links low, data compression is a natural solution to the problem. However, the use of compression codes which remove the redundancy of data seems to be in direct conflict with the use of redundant coding, e.g. parity check codes, which increase the reliability. What is needed then is an exploration of efficient error limiting codes which can be applied to compressed data and an analysis of the error rate of various compression schemes.

Desirable properties of compression codes

In designing a compression code, it should possess to some degree each of the following properties:

1. The technique should be reversible, i.e. the original data should be fully recoverable from the compressed form. This property can be relaxed in certain situations when the data is repeated elsewhere, e.g. the keys in a directory structure are usually repeated across levels.
2. The coding scheme should cause a measurable reduction in the size of the stored data. In comparing compression codes, a standard measure called percent compression is generally used.

$$\text{percent compression} = \frac{[\text{size of input data}] - [\text{size of output data}]}{[\text{size of input data}]} \times 100\%$$

3. The technique should be reasonably efficient to implement.

4. The technique should be general enough to be equally applicable to all alphanumeric data files.

Two other properties which are often desirable in compression schemes are:

5. The prefix property, i.e. no code is the prefix of another code. This assures that the decoder never has to backup on any portion of the text.
6. Lexicographic ordering property, i.e. if the input data is in a sorted order, then after encoding, the output data is still in sorted order. This property is useful for indexes.

Existing compression techniques, which possess part or all of the above properties, can be classified into the following board categories: (1) Run length encoding; (2) Differencing; (3) Statistical encoding; (4) Value set schemes.

Run Length Encoding—In a data base, there are frequent occurrences in which the data occur in a continuous sequence of identical characters, e.g. sequence of zeroes. This sequence can be replaced by the character followed by a count. Run length encoding is a technique by which a string of continuous characters or a "clump" are replaced by a repeat flag for the character followed by the size of the "clump" or run length. In practice, however, since very long clumps are highly improbable, one can limit the run length encoded and combine the flag and length in a single byte. This is the technique used in WYLBUR.¹⁰ Run length encoding of a single character type is potentially the most successful, with diminishing returns for more characters. Huang has discovered an upper bound for the entropy of run length encoding.¹⁹

Differencing—Differencing refers to techniques which compare a current record to a pattern record and retain only the differences between them. It is particularly successful with large files of records with fixed length alphanumeric fields where most corresponding fields are the same or are blanks and zeroes. This is the approach normally used for sequential files, where the pattern record is taken by the previous record in the file. When differencing is applied to direct access files, however, the first record of each block is left uncompressed and used as a pattern for the remaining records in the block. The unit of information on which differencing is performed can be the bit, the byte, the field or some logical data in the record. Byte-level differencing is the most common case since byte access is convenient and cheap. In field-level differencing, bit maps are often used to indicate the presence or absence of a field when identical to the previous. Two examples of the use of differencing in relational data base systems are Titman's experimental system³⁸ and the Peterlee Relational Test Vehicle.³⁹

Statistical Encoding—Statistical encoding is a transformation of an input alphabet so that it is assigned a code bit string whose length is inversely proportional to the frequency of its occurrences in the text. Since different characters occur with different frequencies, a statistical encoding scheme will usually compress the text. Huffman coding

scheme²⁰ is an optimum, elegant and simple algorithm to assign variable length bit codes with the prefix property to characters, given their frequencies of occurrence in a text. There are other techniques such as the Hu-Tucker Algorithm,¹⁸ which has both the prefix and the lexicographic ordering property. The major drawbacks of statistical encoding are that it does not exploit the natural radix of the computer (e.g. byte, word, etc.), and it does not take into account some special characteristics of the data, e.g. strings of repeating characters, and the distinction between numeric and character data. A solution to this is the use of fixed length encoding which manipulate data in units of byte.^{32,2} Further, the fact that the size of each character is variable also causes problems when the data are modified and the reliability of the data is difficult to assure because the character stream would not be recognizable once a bit is destroyed.

Value Set Schemes—A value set scheme in a data base system is a coding scheme in which repeated storage of data elements in their full character representation is avoided. Instead, each data element is stored once in the system and all subsequent occurrences of the same data element are referred to the first stored occurrence. An example of this technique is shown in the MacAIMs Data Management (MADAM) System¹⁴ in which a reference number is assigned to a new entering data element and all subsequent operations on the data element use the reference number. However, the fact that reference numbers are unique only within a relation could lead to problems in the reliability of the data management system and the integrity of the data. The MADAM System also uses a binary tree scheme for maintaining reference numbers which is inefficient for insertion and costly in storage space for large sets of data. There are other schemes which represent a better tradeoff between storage efficiency and processing efficiency.²⁴

The decision of which code to use is highly dependent on the applications. For example, in a data base where the order of data is not important, the lexicographic ordering property is not important. The required properties of the applications must therefore be identified by the designers before the code is selected.

Future directions of research

While there are many reported results on data compression, the future directions of research are seen to be concentrated in the following areas:

Identify and characterize data redundancy—In a data base, there are many levels of data. For example, there are the file level, the record level, the field level and the byte level. The type of data redundancy at each level must be identified. This would aid in selecting data compression schemes best suited to the particular type of redundancy. Further, it leads us to the possibility of multi-level compression schemes, wherein data is compressed through a set of cascaded stages. Each level of the data is possibly compressed using a different technique. The compression code must be selected so that it minimizes the effects on other levels of the data.

Develop a comparison model for various compression schemes—The comparison model must be able to measure the amount of storage reduction and the computation cost for encoding and decoding. A simple measure is the percent compression defined earlier. The computation cost can be broken down into the CPU cost, the memory usage cost and the input/output cost. In order to calculate the storage reduction for a given compression scheme, the number of encodable units of tokens in a record or file must be predicted. This can be obtained from an assumed input distribution such as uniform distribution, normal distribution or Zipf's distribution at the given level of data.

Study adaptive Huffman coding techniques which respond to update activity—As the data base gets updated, the initial Huffman code assignment based on the a priori character frequency distribution may no longer be optimal. A threshold for the expected compression ratio has to be determined which can dynamically reassign the variable length codes for the new frequency distribution. Further, the threshold selected should not cause excessive re-coding. The problem of updates which change the size of the data, and the reliability problems should also be studied.

Investigate the feasibility of implementing, in a microprocessor, a simple self-measuring self-adjusting encoder/decoder—Experience has shown that the current implementation of data base systems are I/O bound within a node and communication bound on the DCS. A microprocessor encoder/decoder, by performing compression and decompression, would cause communications to be done more efficiently, at the same time distributing or relieving this function from the processor sub-systems. Such a device would perform the following functions: (a) encode and decode data; (b) measure and adjust the code assignments; (c) detect errors and automatically re-initiate the operation; and (d) control concurrent accesses. The advantage of this design is that it would make data compression transparent to the rest of the system.

In conclusion, the use of data compression allows data to be stored more efficiently and data communication to be done with shorter messages. However, many issues relating to the feasibility, the design of coding techniques, the reliability of the resultant codes, the implementation issues, etc., must be solved. It is contended that such solutions do not exist now and future study is necessary.

PROGRAM/DATA PLACEMENT AND MIGRATION

Definition of the problem

The problem is defined as follows: given a number of computers that process common information files, how can one allocate the files so that the allocation yields minimum overall operating costs. This problem has been called the *File Allocation Problem (FAP)*.⁹ A more general problem is the *Dynamic File Allocation Problem (DFAP)* in which the files are allowed to migrate over time so as to adapt to changing access requirements. The solution to this problem

is affected directly by the query decomposition strategies and rather lightly by the data compression techniques. If the query is always decomposable into single file sub-queries, then the placement of each file may be optimized independently. Otherwise, the distribution of the files on the DCS must be optimized jointly and this increases the complexity of the problem significantly. On the other hand, data compression techniques generally affect the amount of data requested at a node and therefore the cost of an access is governed by the type of compression techniques used. By making certain assumptions on the query decomposition strategy and the compression technique, the FAP can be studied independently.

Motivations for file placement and migration

The major reason for allocating multiple copies of a file to certain parts of the system at certain times and the unnecessaryness of keeping a copy of every file at every node all the time is because users have localities of access in any time interval. At any particular time, a file may be used by a group of users and it will continue to be used by the same group for a certain length of time. For a particular user, the file that he wants to access may be available locally, in which case, he can access the file with very little cost. If the file is not available locally, he would have to pay a cost in terms of delay in accessing the file and also introducing traffic in the network before he can make the access. It is under this situation that we should consider moving a copy of the file to his node. Introducing a new copy would also increase the cost in terms of storage space and the additional overhead in locking and concurrency control. Therefore, the decision of whether to introduce a new copy of a file involves a balance of the cost between the two cases. The costs, e.g. communication costs, are a function of the topology of the system, the type of communication protocols used and most importantly, the extensiveness of usage at a particular node. Further, as the request frequencies change, the file allocation on the system must also change accordingly. However, in this case, the cost in migrating the file from one node to another must also be taken into account in the file placement algorithm.

Previous work

Most of the previous studies on optimization are based on static distribution, that is, the allocation does not change with time. Some variation of dynamic distribution involves the application of static algorithms whenever need arises. A summary of the previous researches in this area is shown in Table I. These algorithms are very expensive to run in real time. A particular solution to this problem involving a 30 site network required about an hour on an IBM 360/91 computer.¹⁶ The difficulty in optimization is also exemplified in Reference 33. Moreover, most of the algorithms are shown

TABLE I.—A Survey of Previous Researches in File Placement/Migration

	Network Flow Techniques		Mathematical Programming & Exhaustive Searches					Heuristic	
	Stone ²⁴⁻²⁷	Jenny ^{21,17}	Chu ³	Casey ¹	Levin & Morgan ^{25,29}	Ghosh ¹³	Foster et. al. ¹¹	Loomis & Popek ^{26,27}	Mahmoud & Riordon ²⁸
Assumption	Complete relations among objects: No redundant copies of objects.	Complete relations among objects.	Complete relations among objects: File access is poisson.	All objects independent.	Only Program-data relation exists between objects.	All objects independent.	Star network: All objects independent.	Complete probabilistic relation among objects.	Indep. obj's: Query & ret'n traffic divided equally among alloc. nodes.
Parameters	Avg. amount of comm. traffic among obj.: Execution cost on a computer: Overhead in migration.	Functional equations represents constraints on which process placements depend; Communication demands between processes.	Storage cost: Transmission cost: File length: Request rate between files: Update rate between files: Maximum allowable access time: Storage capacity.	Storage cost: Query trans. cost: Update trans. cost: Query rate between nodes: Update rate between nodes.	Communication cost for query: Communication cost for update: Traffic rate for query/update from a node to a file via a program.	Data base with multiple target segment types: Queries with multiple target segment types	Queuing time & service time for transactions: Storage capacity: Avg. no. of messages in network: Avg. local processing: Average file length: Access frequency: H/W, S/W characteristics.	Inter-node trans. cost: Node capability: File length: Processing needs of file: Prob. of a request acc. an object: Prob. of a request/update is incident on a node: Prob. of 2 objects processed in parallel.	Communication cost: File storage cost: Query/update traffic & corresponding return traffic for each file at each node: Availability requirements.
Algorithm used	Network flow	Network flow & predicate calculus.	Integer programming	Path search on cost graph	Path search on cost graph	Combinatorial search thru. possible sol.	Queueing network alg.: Integer prog.	Clustering	Int. prog. or add-drop heuristic
Remarks	Static: Optimal for 2 processors: Sub-optimal for 3 processor system: Can calculate critical load factor for 2 processor system.	Do not consider multiple copy allocation: Min-cut alg. produces optimal subprocess groupings: Minimize communication overhead.	Algorithm very complex: Consider delay from network queuing approach.	Algorithm efficient: Independence of objects reduces allocation of multiple file to single file.	Algorithm efficient: Definite access relations among objects reduces the allocation of multiple file to single file.	Maximize no. of segments that query can retrieve in parallel from different nodes: Do not model communication delays.	Minimize difference from optimal branching probabilities: Algorithm complex.	Dynamic network behavior ignored: Maximize potential for parallelism.	Obtain both capacity assignment for links & file placements: Should consider query to be routed to nearest node & not distributed equally among all nodes.

to be NP-complete.**⁹ Although polynomial algorithms could exist for some special cases of the problem, e.g. the allocation of files in a two-processor system,³⁴ their use in practical applications is very limited. This result suggests that the distributed system designer should focus his attention to efficient heuristics.

Heuristics for file distribution on a DDB are usually interactive algorithms. A feasible solution can be generated. Users of some decision algorithms then have to decide whether to improve the solution or not and how to improve it. The disadvantages of these types of algorithms are that they usually find a local optimum instead of a global optimum and the validation of the algorithm is very difficult. For most cases, the heuristics can be shown to perform

satisfactorily for some example values, but the algorithm is so complex that its worst case behavior is very difficult to determine. We first classify the three most commonly used heuristics, then we will discuss the application of a file assignment algorithm on this problem.

Hierarchical designs

This is a heuristic procedure in which attention is first restricted to the more important features of a system. In a file allocation problem, attention can first be restricted to geographical regions. After analysis has been performed and the files have been distributed to different geographical regions, attention can be directed to the less important details such as allocating files within a geographical region. This stepwise refinement procedure can continue down many levels. At each level of optimization, it is hoped that the effects on the optimization of the current level from the levels above and the levels below are very small. Neverthe-

** NP-complete problems²² is a class of problems for which there are no known optimal algorithms with a computation time which increases polynomially with the size of the problem. The computation times for all known optimal algorithms for this class of problem increase exponentially with problem size, i.e., if n represents the size of the problem, then the computation time goes up as k^n where $k > 1$.

less, iterations and design cycles may exist to refine the solution.

Clustering algorithms

Clustering algorithms are horizontal design processes which have a similar objective as hierarchical algorithms, namely, to reduce the complexity of the analysis in a large system. In a DDB, clusters can be formed on geographical distribution of access frequencies. The files are then allocated to clusters. The file allocation within a cluster may further be refined as in hierarchical algorithms.^{26,27}

Add-drop algorithms

In applying this algorithm, a feasible distribution of files is first found. The total cost of the system can be improved by successive addition or deletion of file copies. When a feasible solution with a lower cost is found, it is adopted as a new starting solution and the process continues. Eventually, a local optimum is reached in which addition or deletion does not reduce the cost. The whole procedure can be repeated with a different starting feasible solution and several local optima can be obtained. By taking the minimum of all the local minima obtained, it is hoped that we can get very close to the global optimum.²⁸

The above techniques are by no means complete. A combination of these techniques may be chosen by the designer. In the next section, we introduce a file assignment heuristic which utilizes some of the principles of add-drop algorithms.

File assignment algorithm

In this section, we present an algorithm which can be used to optimize the file placements on a DCS. The assumptions that we use in developing the model are:

1. **File accesses are independent**—By this, it is meant that there are no interactions among the files and all the accesses on the system are single file accesses. The placements of each file can therefore be optimized independently.
2. **It is assumed that all the constraints on the system can be represented in the form of costs.** For instance, paths linking two nodes in the network which violate some constraints such as the response time constraint, have a high inter-communication cost induced on them.
3. **It is assumed that for a certain time interval considered, it is divided into periods.** The file access behavior for a period are assumed to be estimated at the beginning of the period and the access behavior for the subsequent periods cannot be estimated at that point. With this assumption, it is possible to optimize the file allocations of each period independently and is not necessary to use dynamic programming to optimize the allocations for all the periods as done in Reference 25.

No assumption is made on the length of each period. Their lengths need not be identical and may be determined dynamically. The algorithm described in this section determines the file placements for each period, but no provision is made for determining the length of each period.

The symbols used in the model are:

n	number of nodes in the distributed system
a, b, c	indices for files
t	length of the current period of consideration, T
q_i^a	a random variable indicating the total amount of query accesses (including updates) at node i to file a (since we are optimizing each file independently, we will not write the superscript a in the remaining part of the discussion)
α_i	a random variable indicating the fraction of queries at node i that are updates to file a
$S_{i,j}$	per unit cost of accessing file a from node i to node j
$M_{i,j}$	per unit cost of multiple updating file a from node i to node j
$N_{i,j}$	per unit cost of moving file a from node i to node j
f_i	per unit cost of storing file a at node i
l_a	length of file a in bits
$Y_i =$	$\begin{cases} 0 & \text{if file } a \text{ does not exist at node } i \text{ during period } T \\ 1 & \text{otherwise} \end{cases}$
$X_i =$	$\begin{cases} 0 & \text{if file } a \text{ does not exist at node } i \text{ during period } T-1 \\ 1 & \text{otherwise} \end{cases}$
$K_0 =$	$\{j: Y_j=0\}$ = set of nodes without a copy assigned
$K_1 =$	$\{j: Y_j=1\}$ = set of nodes with a copy assigned
$K_2 =$	$\{j: J_j=\text{unassigned}\}$ = set of nodes unassigned
$K =$	$K_0 \cup K_1 \cup K_2, K =n$ (cardinality of K)

Consider the problem of allocating file a on the system at the beginning of period T of length t , the total amount of the retrievals and updates in this interval are estimated to be $q_i(1-\alpha_i)$ and $\alpha_i q_i$. The per unit cost of assessing, updating and transferring file a from node i to node j are $S_{i,j}$, $M_{i,j}$, and $N_{i,j}$ respectively. We assume that whenever a user at node i makes a request to a file not residing at node i , he will make the access at a node which has a copy of the file and with the lowest cost of access from node i . Our objective is to minimize the cost in the system. Our objective function is:

$$Z = \sum_{i=1}^n q_i(1-\alpha_i) \min_{j, Y_j=1} S_{i,j} + \sum_{j=1}^n Y_j(f_j + \min_{i, X_i=1} N_{i,j})l_a + \sum_{j=1}^n \sum_{i=1}^n Y_j \alpha_i q_i M_{i,j}$$

The first term in the above equation represents the query access cost; the second term represents the fixed cost of the period (cost of storage + cost of file transfers at beginning of

period); while the third term represents the multiple update cost of the system. We can rewrite the integer program as follows:

$$Z = \sum_{i=1}^n Q_i \min_{j, Y_j=1} S_{i,j} + \sum_{j=1}^n Y_j F_j \quad (1)$$

where

$$Q_i = q_i(1 - \alpha_i) \quad (2)$$

$$F_j = (f_j + \min_{i, X_i=1} N_{i,j}) l_a + \sum_{i=1}^n \alpha_i q_i M_{i,j} \quad (3)$$

subject to

$$Y_i = 0, 1 \quad (4)$$

The file assignment algorithm proposed here consists of the following basic parts:

1. Property or condition to assign or not to assign a copy of the file to a node.
2. Computation of a representative value for a candidate problem. (The state of a candidate problem is made up of the states of allocation to the n different nodes of the DCS. In general, the n nodes of the DCS can be partitioned into three sets, K_0 , K_1 and K_2 .) The function of the representative value is to illustrate the minimum of the candidate problem without actually enumerating all the allocations for the unassigned nodes.
3. Stopping the criterion.

The general steps of the algorithm are shown in Figure 4. We discuss each of these steps briefly here.

M-1. This is to initialize the candidate problem—all nodes are unassigned at this point. The candidate list, which is a list of states where an extra node from K_2 is added to K_0 or K_1 and its corresponding representative value, is assigned the empty set.

M-2 to M-5. These four steps essentially achieve the following: a node is selected from the un-assigned set, K_2 , and is assigned a copy or not assigned a copy of the file. A representative value, which is chosen to be a lower bound estimated by solving the integer program (Equation 1) without the integrality constraints (Equation 4), is calculated for each of the corresponding candidate problems. The derivation of the linear programming lower bound for a candidate problem is shown in the appendix. The computed lower bound and the corresponding assignments are attached to the candidate list. These steps are then repeated for each of the nodes in K_2 .

M-6. This step selects, from the candidate list, the candidate problem with the minimum lower bound and the corresponding assignment of nodes and use it for the next iteration. Steps M-2 to M-6 therefore have selected a node and have decided whether a copy should be placed at that node. This node is removed from the K_2 list.

M-7. The steps M-2 to M-6 are repeated until the K_2 list is empty. The overall computational complexity of the

algorithm is $O(n^4)$. To further illustrate the steps of the algorithm, it is applied on the following example.

Suppose the following matrix represents the query cost $S_{i,j}$ for a five-node system.¹

$$S = \begin{bmatrix} 0 & 6 & 12 & 9 & 6 \\ 6 & 0 & 6 & 12 & 9 \\ 12 & 6 & 0 & 6 & 12 \\ 9 & 12 & 6 & 0 & 6 \\ 6 & 9 & 12 & 6 & 0 \end{bmatrix}$$

Let

$$Q = [Q_i] = [24 \quad 24 \quad 24 \quad 24 \quad 24]$$

and

$$F = [F_i] = [168 \quad 180 \quad 174 \quad 126 \quad 123]$$

By enumerating the $2^5 - 1$ possible allocations, it is found that a copy of the file should be allocated to nodes 1, 4 and 5 giving a cost of 705. The detailed application of the heuristic is shown in Figure 5, giving a solution of 717. In general, this method will give a solution very close to the optimal solution and the computation complexity is very low when compared with that of generating the optimal solution. The five examples on a 19-node problem solved by Casey⁴ are compared with the solutions using the proposed heuristic and is shown in Table II. It is seen that the results do not deviate substantially from the optimum solutions. The results indicated here are somewhat preliminary. For the sake of simplicity, a more complicated algorithm is not presented. This algorithm, together with the analytical results and the theoretical studies will be presented in a future paper.

TASK SCHEDULING

Definition of the problem

This problem is related very strongly to the problem of query decomposition. After the query has been decomposed, the *Query Scheduling Problem (QSP)* is to sequence the processing of the sub-queries on the DDB for a given distribution of the files on the DCS defined by the FAP. Depending on the ways in which the sub-queries are processed, QSP can further be classified into *Sequential Query Scheduling Problem (SQSP)* and *Parallel Query Scheduling Problem (PQSP)*. In SQSP, the sub-queries are processed

TABLE II.—Comparison between Casey's Solutions on a 19-node Problem and the Solutions of the Proposed Algorithm

Problem	Update/Query Percent	Casey's Optimum Cost	Cost using Proposed Algorithm	Time on CDC6400 (sec.)
1	10	117596	123073	7.7
2	20	188738	200971	7.7
3	30	242581	246107	7.7
4	40	291790	298690	7.7
5	100	431720	615342	7.7

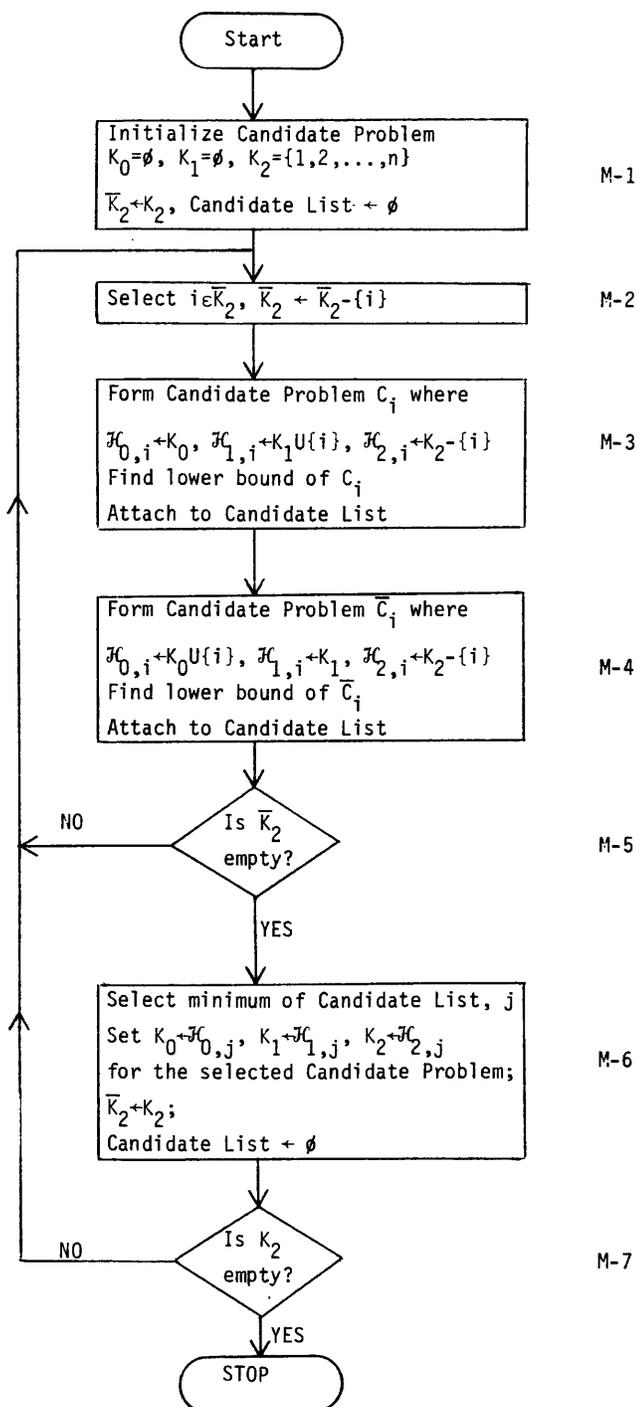


Figure 4—File assignment algorithm.

in sequential order. Using the results produced by the processing of the previous sub-query, the processing of the present sub-query will produce some results to be used by the next sub-query in sequence. If the files used by the sub-queries are separated geographically, intermediate results have to be transferred over communication lines. The objective is to minimize the amount of communications re-

quired. In PQSP, multiple queries are sent to different nodes and they are processed in parallel. The results after the processing are sent back to the requesting location. In this case, the response time may be smaller because all the communications are done in parallel (it is assumed that the major overhead is in communications and not in processing). For a compromise between the amount of communications and the response time, a combination of sequential and parallel query processing may be used. The QSP is a very similar problem which has been studied in other areas, e.g. the deterministic scheduling of multi-processors, the scheduling of requests in a computer system, etc. The results obtained there may therefore be extended to this study.

In order to solve the QSP, the notion of task must be defined. A *task* is defined to be a simple request which uses a resource for a finite amount of time. A request is said to be simple if no other resource is needed during the processing of this request. A complex request can always be broken down into a sequence of simple requests. A resource on a DDB can be physical, such as a communication channel, a processor, etc., or it can be logical, such as a file. The tasks are usually governed by a precedence graph so that a task cannot be processed until its predecessor has finished processing. The task scheduling problem is to sequence the processing of the tasks, subject to precedence constraints, so that some overall optimization criterion is satisfied. The criterion can be the maximum completion time of all the tasks if the objective is to maximize the throughput of the system; or it can be the sum of the completion times of all the tasks if the objective is to minimize the average response time.

To schedule the processing of queries, they are first decomposed into multiple tasks and the tasks are subsequently scheduled. The general task precedence graph for the processing of a query in the PQSP which requires the use of geographically distributed files is shown in Figure 6. On a DCS, the communication overhead, which includes time to set up the communication path and the queuing delay to transmit the messages, is usually much larger than the processing overhead for a query. Therefore, the time required to process a task at a node in Figure 6 is usually negligible as compared to the time to pass the results over the communication sub-system. The communication overhead on a DCS is dictated by the configuration of the interconnection mechanism. Many models have been designed to study the behavior of these delays, e.g. in Reference 23.

The QSP is usually solved with distributed control, that is, there does not exist a primary node which schedules all the processing of the queries on the DCS. Further, complete information for optimal scheduling are usually not available due to the high overhead in distributing them. The tasks are usually scheduled at each node sub-optimally without assembling all the necessary information before the scheduling.

Assumptions used to simplify the problem

Certain assumptions are often used so that the problem can be simplified.

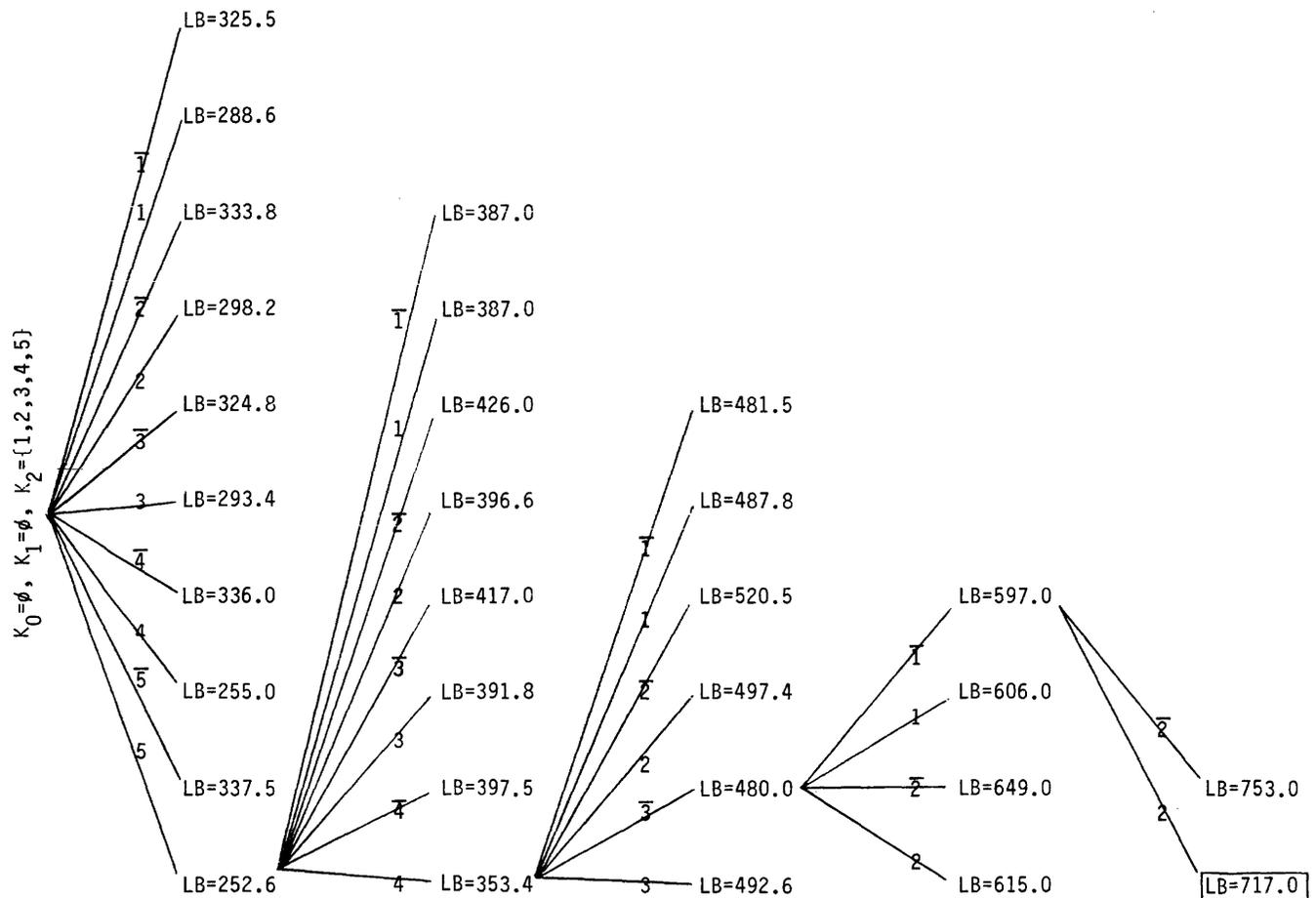


Figure 5—Application of file assignment on Casey's 5 node example.

Communication overhead

The processing overhead is usually much smaller than the communication overhead and they are usually ignored. This assumption will eliminate many tasks in the precedence graph.

Static versus dynamic algorithms

Static algorithms schedule a set of tasks available at the time of scheduling and a set of tasks that are known to arrive at fixed future times. The schedule does not change during the duration of the processing of these tasks. On the other hand, dynamic algorithms are more flexible and they re-schedule all the available tasks whenever a new task comes in. The advantage of dynamic algorithms is that they allow task initiations to be dynamic and do not restrict the schedule to the order determined initially, but they have the disadvantage of larger overhead. With the use of dynamic algorithms, the assumption that there are precedence constraints among the tasks can also be relaxed. Whenever

a task enters the system, all the tasks in the system are re-scheduled dynamically. The choice between the use of static and dynamic algorithms is system dependent. If the arrivals of requests are indeterminate, then dynamic algorithms are usually better. On the other hand, if the arrivals of requests can be determined precisely, then static algorithms should be used. The choice between static and dynamic may also be dictated by the overhead in each type of algorithm, and a judicious choice must be made by the designer.

Deterministic versus probabilistic processing time

The processing time for a task can be assumed to be deterministic or probabilistic. In the deterministic case, it is possible to determine which order can best satisfy the optimization criterion and therefore all the tasks can be scheduled in a specific order. However, in a probabilistic case, it is difficult to do so when the processing times of all the tasks are governed by a common distribution. Certain assumptions, e.g. the distribution of job size, have to be made before an analytical evaluation is possible.⁵ The theory of

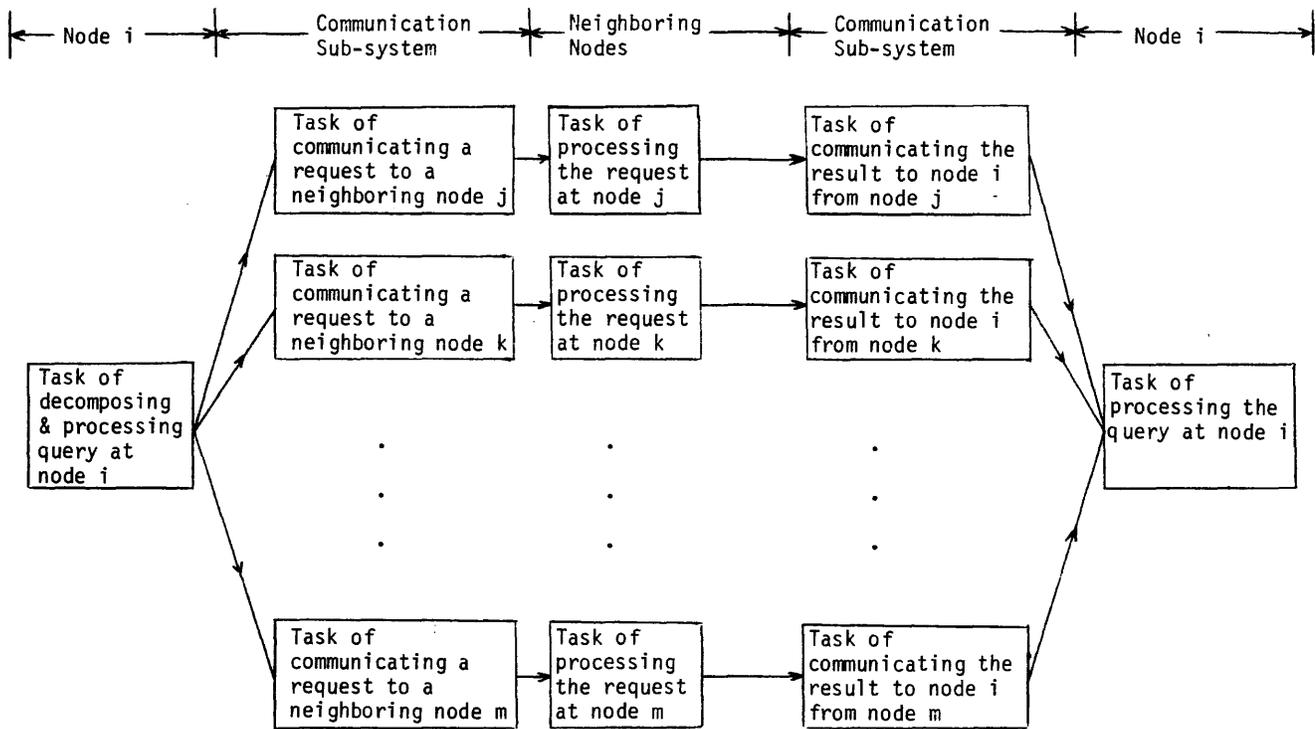


Figure 6—Task precedence graph for the processing of a query in the PQSP which requires the use of geographically distributed files.

scheduling developed now are mostly applicable to the deterministic case.¹⁵ They can be used to approximate the probabilistic case when the average or the worst case processing times are used. Lastly, the difficulty of the scheduling problem can be assessed easily in most cases under the deterministic assumption. NP-completeness of the problem can usually be shown or a polynomial algorithm can be found. The QSP under the independent query assumption, can be shown to be NP-complete. Under this situation, the designer has to look for good heuristics which can be executed within the real time constraints. However, the evaluation of these heuristics is usually difficult. Evaluation methods and techniques are usually of three kinds, analytical techniques, simulations and approximation algorithms. Analytical techniques generally have to make some simplifying assumptions about the system parameters in order for the solution to be tractable and the results obtained are usually not accurate. On the other hand, simulations are almost always expensive to run, and it is difficult to exhaust all the possible cases of the system. A third type of evaluation algorithms are approximation algorithms.⁴⁰ There are two classes of these approximations, one guaranteeing a near-optimal solution always, and the other producing an optimal or near-optimal solution "almost everywhere." These types of algorithms are still in the research stage and a unifying approach in designing algorithms of this type is still lacking. The future trend is in the direction of investigating good approximation algorithms for scheduling queries.

CONCLUSION

In this paper, we have studied in detail the issues of resource management of data on a distributed data base. These issues are divided into three related levels, namely, the query level, the file level and the task level.

On the query level, the major issue is the decomposition of user queries so that parallelism in processing can be maximized and the amount of communications on the system can be minimized. It is shown that the approach using decomposition is deficient when the query is non-decomposable. In this case, the files needed to process the query must be moved to a common location before the query can be processed. An algorithm is proposed in this paper which preanalyzes the type of accesses on the system and introduces redundant information onto the files so that file transfers can be reduced.

On the file level, the issues are the compression of data for efficient storage and communication and the placement and migration of files for efficient accesses. In data compression, the existing techniques has been classified into four areas—run length encoding, differencing, statistical encoding and value set schemes. A multi-level compression scheme is proposed so that data is compressed through a set of cascaded stages. In the area of file placement and migration, a file assignment algorithm has been proposed. In general, this algorithm gives a solution very close to the optimal solution and the computation complexity is very low when compared with that of generating the optimal solution.

On the task level, the problem is to sequence the processing of the sub-queries for a given distribution of the files on the distributed system so that overlap in processing can be maximized. It is shown that the problem of query scheduling on a distributed data base is NP-complete. The future directions of research are therefore in the search of effective approximation algorithms.

The issues we have discussed in this paper encompasses the spectrum from the processing of the query to the scheduling of the requests. However, many other issues may arise in the design of the data base. These include other issues in operational control, such as directory management, concurrency control, security and privacy, etc. and they may affect the strategies used in data management. The study of these issues, however, are beyond the scope of this paper.

APPENDIX—DERIVATION OF THE LOWER BOUND OF A CANDIDATE PROBLEM

We derive in the appendix the lower bound of a candidate problem given the state of it. We can rewrite the objective function (Equation 1) on condition on K_0 , K_1 and K_2 .

$$Z = \sum_{i \in K_1} F_i + \sum_{i \in K_0} Q_i^* \min_{j, Y_j=1} S_{i,j} + \sum_{i \in K_2} Q_i^* \min_{j, Y_j=1} S_{i,j} + \sum_{i \in K_2} F_i Y_i$$

We have

$$\min Z = \sum_{i \in K_1} F_i + \sum_{i \in K_0 \cup K_2} Q_i^* \min_{j, Y_j=1} S_{i,j} + \sum_{i \in K_2} F_i Y_i \quad (A-1)$$

subject to $Y_i=0,1$

where Q_i is defined in Equation 2, and F_i is defined in Equation 3.

Equation A-1 is a non-linear integer program, we can rewrite it in the form of a linear program. Let

$U_{i,j}$ = fraction of accesses made from node i to node j ;

P_i = set of indexes of those nodes that can access node i ;

n_i = cardinality of P_i .

$$\min Z = \sum_{i=1}^n \sum_{j=1}^n Q_i S_{i,j} U_{i,j} + \sum_{i=1}^n F_i Y_i \quad (A-2)$$

$$\text{s.t. } \sum_{j=1}^n U_{i,j} = 1 \quad i = 1, \dots, n \quad (A-3)$$

$$0 \leq \sum_{i \in P_j} U_{i,j} \leq n_j Y_j \quad j = 1, \dots, n \quad (A-4)$$

$$Y_i = 0, 1 \quad (A-5)$$

Equation A-3 is true because the total amount of fractions must be summed to 1. Equation A-4 is derived by summing over all $i \in P_j$, the inequality $0 \leq X_{i,j} \leq Y_i$ which says that only nodes with a copy of the file can supply users' demands. By relaxing constraint A-5, it becomes a linear program and the

value of Z obtained will provide a lower bound to the original integer program. The solution to the linear program (Equation A-2 to Equation A-4) has been solved in Reference 7. The solution is:

$$Y_j = \frac{1}{n_j} \sum_{i \in P_j} U_{i,j} \quad (A-6)$$

$$Z_{i,j} = \begin{cases} 1 & \text{if } S_{i,j} + \frac{g_i}{n_j} = \min_{k \in K_1 \cup K_2} \left(S_{i,k} + \frac{g_k}{n_k} \right) \\ 0 & \text{otherwise} \end{cases} \quad (A-7)$$

$$g_k = \begin{cases} F_k & k \in K_2 \\ 0 & k \in K_1 \end{cases}$$

The complexity of the solution is $O(n^2)$.

REFERENCES

- Casey, R. G., "Allocation of Copies of a File in an Information Network," *AFIPS SJCC*, 1972, pp. 617-625.
- Chen, T. C., and I. T. Ho, "Storage Efficient Representation of Decimal Data," *CACM*, Vol. 18, No. 1, Jan. 1975, pp. 49-52.
- Chu, W. W., "Multiple File Allocation in a Multiple Computer System," *IEEE Trans. on Computers*, Vol. C-18, No. 10, Oct. 1969, pp. 885-889.
- Codd, E. F., "A Relational Model of Data for Large Shared Data Bases," *CACM*, Vol. 13, No. 6, June 1970.
- Coffman, E. G., Jr., and P. J. Denning, *Operating System Theory*, Prentice Hall Inc., N.J., 1973.
- Date, C. J., *An Introduction to Data Base Systems*, 2nd Edition, Addison-Wesley, 1977.
- Efroymsen, M. A., and T. C. Ray, "A Branch and Bound Algorithm for Plant Location," *Operations Research*, May-June 1966, pp. 361-368.
- Epstein, et. al., "Distributed Query Processing in a Relational Data Base System," Report No. UCB/ERL M78/18, Electronics Research Laboratory, University of California, Berkeley, 1978.
- Eswaran, K. P., "Placement of Records in a File and File Allocation in a Computer Network," *Information Processing 74*, IFIPS, North Holland Publishing Co., 1974.
- Fajman, R., and J. Borgelt, "WYLBUR: An Interactive Text Editing and Remote Job Entry System," *CACM*, Vol. 15, No. 5, May 1973, pp. 314-333.
- Foster, D. V., et. al., "File Assignment in Star Network," *Proc. of the 1977 Sigmetrics/CMG VIII Conf. on Comp. Perf.: Modelling, Measurement and Management*, Washington, D.C., Nov. 1977, pp. 247-254.
- Fry, J. P., and E. H. Sibley, "Evolution of Data Base Management Systems," *Computer Surveys*, Vol. 8, No. 1, March 1976, pp. 7-42.
- Ghosh, S. P., "Distributing A Data Base with Logical Associations on a Computer Network for Parallel Searching," *IEEETSE*, Vol. SE-2, No. 2, June 1976, pp. 106-113.
- Goldstein, R. C., and A. J. Strand, "The MacAIMS Data Management System," *Proc. ACM-SIGFIDET Workshop*, Nov. 1970, pp. 201-229.
- Graham, R. L., et. al., "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," *Proc. of Discrete Optimization 1977*, Vancouver, Aug. 1977.
- Grapa, E., and G. G. Belford, "Some Theorems to Aid in Solving the File Allocation Problem," *CACM*, Vol. 20, No. 11, Nov. 1977, pp. 878-882.
- Hofri, M., and C. J. Jenny, "On the Allocation of Processes in Distributed Computing Systems," *IBM Research Report*, RZ905, 1978.
- Hu, T. C., and J. Tucker, "Optimal Computer Search Trees and Variable Length Alphabetic Codes," *SIAM J. of App. Math.*, Vol. 21, 1971, pp. 514-532.
- Huang, T., "An Upper Bound on the Entropy of Run Length Coding," *IEEE Trans. on Info. Theory*, Vol. IT-20, No. 9, Sept. 1974, pp. 675-676.
- Huffman, D. A., "A Method for the Construction of Minimum Redundancy Codes," *Proc. IRE*, Vol. 40, Sept. 1952, pp. 1098-1111.

21. Jenny, C. J., "Process Partitioning in Distributed Systems," *IBM Research Report, RZ873*, 1977.
22. Karp, R. M., "Reducibility among Combinatorial Problems," *Complexity of Computer Computations*, R. E. Miller and J. M. Thatcher (eds.), Plenum Press, New York, 1972, pp. 85-104.
23. Kleinrock, L., *Queueing Systems, Vol. II: Computer Applications*, John Wiley & Sons, 1976.
24. Knuth, D. E., *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.
25. Levin, K. D., and H. L. Morgan, "Optimizing Distributed Data Bases—A Framework for Research," *Proc. NCC*, 1975, pp. 473-478.
26. Loomis, M. E. S., "Data Base Design: Object Distribution and Resource Constrained Task Scheduling," Ph.D. Dissertation, Comp. Sci. Dept., UCLA, 1975.
27. Loomis, M. E. S., and G. J. Popek, "A Model for Data Base Distribution," *Symp. on Trends and Applications*, 1976, Computer Networks, IEEE, 1976, pp. 162-169.
28. Mahmoud, S., and J. S. Riordon, "Optimal Allocation of Resources in Distributed Information Networks," *ACM Trans. on Data Bases*, Vol. 1, No. 1, March, 1976, pp. 66-78.
29. Morgan, H. L., and K. D. Levin, "Optimal Program and Data Locations in Computer Networks," *CACM*, Vol. 20, No. 5, May 1977, pp. 315-322.
30. Ramamoorthy, C. V., and T. Krishnarao, "The Design Issues in Distributed Computer Systems," *Infotech State of the Art Report on Distributed Systems*, 1976, pp. 375-400.
31. Ramamoorthy, C. V., G. S. Ho and B. W. Wah, "Distributed Computer Systems—A Design Methodology and its Application to the Design of Distributed Data Base Systems," to appear *Infotech State of the Art Report on Distributed Systems*, 1979.
32. Schieber, W. D., and G. W. Thomas, "An Algorithm for Compaction of Data," *J. Library Automation*, Vol. 1, No. 4, Dec. 1971, pp. 198-206.
33. Sickle, L. V., and K. M. Chandy, "Computational Complexity of Network Design Algorithms," *Information Processing 77, IFIPS*, North Holland Publishing Co., 1977.
34. Stone, H. S., "Multi-processor Scheduling with the Aid of Network Flows," *IEEE Trans. on Software Engineering*, Vol. SE-3, No. 1, Jan. 1977, pp. 85-93.
35. Stone, H. S., "Critical Load Factors in Distributed Computer Networks," Report ECE-CS-77-2, University of Massachusetts, Amherst, Mass., 1977.
36. Stone, H. S., "Program Assignment in Three-Processor Systems and Tricut Partitioning on Graphs," Report No. ECE-CS-77-7, University of Massachusetts, Amherst, Mass., 1977.
37. Stone, H. S., and S. H. Bokhari, "Control of Distributed Processes," *Computer*, July 1978, pp. 97-106.
38. Titman, P., "An Experimental Data Base System Using Binary Relations," *Proc. IFIP Working Conf. on Data Base Management*, Corsica, 1974.
39. Todd, S. J. P., "The Peterlee Relational Test Vehicle—A System Overview," *IBM Systems Journal*, Vol. 15, No. 4, Dec. 1976, pp. 285-308.
40. Weide, B., "A Survey of Analysis Techniques for Discrete Algorithms," *Computing Surveys*, Vol. 9, No. 4, Dec. 1977.
41. Wong, E., and K. Youssefi, "Decomposition—A Strategy for Query Processing," *ACM Trans. on Data Bases*, Vol. 1, No. 3, Sept. 1976, pp. 223-241.
42. Wong, E., "Restructuring Dispersed Data from SDD-1: A System for Distributed Data Bases," *Comp. Corp. of America Tech. Rep. CCA-77-03*, 1977.

ACKNOWLEDGMENT

We would like to thank Mr. C. R. Vick and Mr. J. E. Scaf for many helpful discussions related to this work.

A unified architecture for data and message management*

by GEORGES GARDARIN*

Institut de Programmation
Paris, France

INTRODUCTION

The ANSI/X3/SPARC study group on data bases has proposed a general architecture for a data base management system.² The keystone of this architecture is the conceptual schema which is an explicit description of the enterprise informations modelled in the data base. It portrays the entities, properties and relationships of interest in the enterprise and constitutes a stable platform in order to map the external schemas which describe the data, as seen by the programmers, onto the internal one which defines the data as seen by the system. Finally the ANSI/X3/SPARC DBMS study group envisions a three-level organization which induces three levels of administration functions, three levels of schema processors and three levels of data manipulation modules. This architecture is partially represented Figure 1. It should be noted that only functions are specified, not their implementation.²⁴

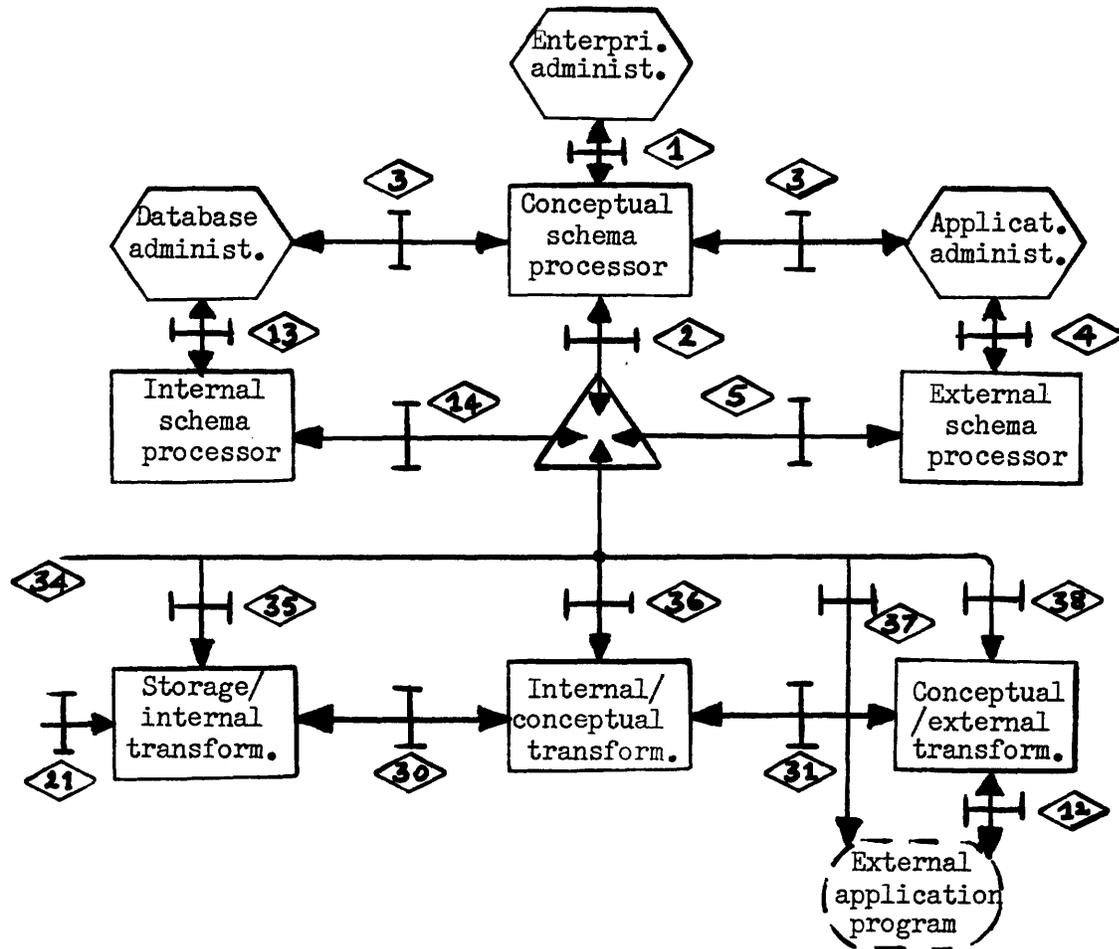
The ANSI/X3/SPARC study group on distributed systems has proposed a reference model.³ This model allows application processes possibly located on different systems to exchange messages through sessions which logically join them. The major contribution is the structuration of the cooperating systems into layers. Each layer can be seen as a distributed sub-system. Cooperation between the distributed parts of a layer is governed by a set of protocols specific to the layer. Six, and now seven,¹¹ layers have been identified. The first three, and now four, layers provide a universal transport service. The next layer supports interaction between cooperating application processes—it performs their binding and unbinding by sessions and controls the exchange of data through these sessions. The presentation layer enables the application processes to interpret the meaning of the data exchanged by transforming them into the desirable representation, format or model. The major concepts of this reference model are represented in Figure 2.

In this paper, the architecture of a distributed data base management system integrating the two proposals is presented. The basic assumption which guarantees the feasibility is that the geographical localization of data has a con-

ceptual meaning. Consequently, the conceptual schema can be distributed, each part representing an enterprise department's description as modeled in a local data base. Obviously, an external schema is entirely situated on a site. The mapping of a user request which refers to an external object into one or several conceptual requests is performed by an external presentation module which can be seen either as an external to conceptual transformer² or as a presentation control service.³ The conceptual requests which result are then taken in charge by a communication kernel which performs at first the session control functions of the ANSI/X3/SPARC distributed system reference model. In addition, it also performs the data base concurrency,^{20,5,9} recovery^{6,13} and security¹² controls. After possible transmission through the transport management layers, the requests are submitted to conceptual data base managers which execute them. The answers follow symmetric paths in the distributed system. Finally, the external presentation module is responsible for constructing the concluding unique user answer.

This paper is organized as follows: In the first section, it is argued that, at least for a large class of applications, the distribution of objects is performed at the conceptual level. Consequently, an implementation of the conceptual schema in a distributed system is suggested. In the second section, the ANSI/X3/SPARC DBMS architecture is distributed in a computer network. That entails the development of a communication kernel allowing the interchange of the conceptual data manipulations through the transport network. Moreover, each local metabase which contains the local part of the conceptual schema must be accessible from remote computers; this feature permits the binding of an external schema with the distributed conceptual schema. In the third section, the communication kernel is presented using several concepts proposed in the ANSI/X3/SPARC distributed system contribution, mainly at the session control level. In the last section, a reference model for a distributed data base system is proposed. Data base and message management are integrated—(a) The external-to-conceptual transformer of the DBMS architecture is coalesced with the presentation controller of the distributed system architecture. (b) The read/write data base commands and the send/receive message commands appear as two different presentations of data interchange basic actions. (c) The integrity controls of

* Sponsored by IRIA, Le Chesnay, France.

Figure 1—DBMS partial schematic.²

the data bases and of the communications between processes are executed by the communication kernel which appears as the heart of the proposed architecture.

CONCEPTUAL DATA DISTRIBUTION

Conceptual realm of data distribution

Several papers^{16,18,8} dealing with distributed data base management system architecture assume that only the internal schema is geographically split and that localization of data concerns the conceptual to internal level mapping. Other papers^{17,1} are quite ambiguous on this topic. Our opinion is that at least for a large family of applications, the distribution of data is connected with the view of the enterprise. This opinion seems to be in agreement with the SDD.1 architecture.²¹

Such an opinion has been verified on real application—first, the ordering, manufacturing and delivery of vehicles at the French motor car company Renault.⁷ For example, the fact that Renault model R30 is manufactured in Flins and that all the entities concerning a car of this model must be stored on the Flins site belongs to the enterprise descrip-

tion and should be explicated at the conceptual level. For this application, the localization criterion of an entity is the place of the modeled object in the real world. This is not directly connected with efficient use of computing facilities. On the other hand, the transfer of R30 entities from Flins to Paris in the distributed system would doubtless be of interest for the life of the enterprise—it would mean that the manufacturing of R30 model is moved from Flins to Paris.

The conceptual meaning of object localization has also been verified on other applications, like the management of spare parts. In the Renault real world, spare parts are distributed to two stores. Consequently, in the Renault computer world, they will be distributed in two local data bases managed by two interconnected computers. The localization criterion of objects is the modeling of the distribution of spare parts in the real world which is a conceptual property.

Conceptual schema design and implementation

The conceptual schema contains the definition of conceptual objects and the expression of their properties. In distributed data bases, the first property of conceptual objects is to be distributed. Therefore, this property can direct the

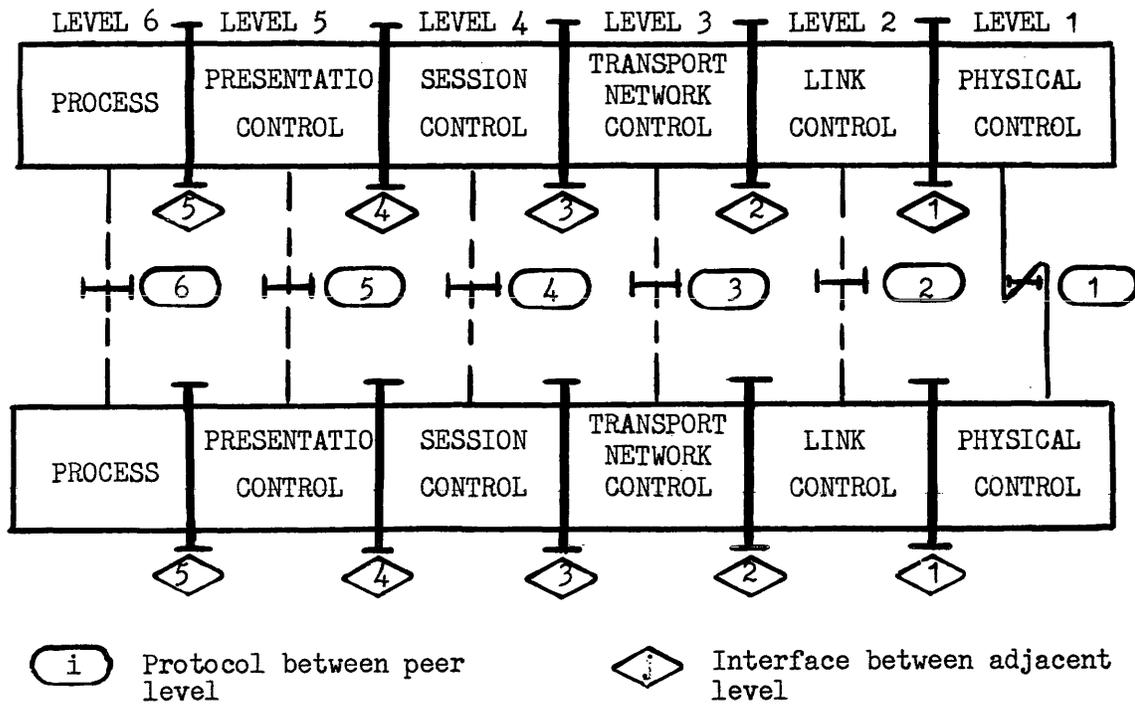


Figure 2—Distributed system levels.³

design of the conceptual schema in agreement with the geographical distribution of sites. Thence, it is proposed to divide the conceptual schema in parts, each part corresponding to a local site. Obviously, it may be argued that some properties involve objects localized on different sites, such as distributed integrity constraints.¹⁹ They must be known in each concerned part; that means that when the conceptual schema is divided, the distributed properties must be integrated to each interested part.

It is emphasized that only one conceptual schema must be designed—to provide a coherent conceptual view of the distributed enterprise couched on a unique booklet using the conceptual model should enhance significantly the interest of the system. The implementation of each part of the unique conceptual schema is performed on each site which must present conceptual objects to other sites in agreement with the conceptual schema. Thus, it should provide a distributed stable platform to which local internal schema and global external ones may be bound (see Figure 3). The design process of the conceptual schema may be distributed (each site designing its part) or centralized (designed by some enterprise administrators), but the existence of such a platform is very important in distributed enterprises because it defines the objects which may be interchanged between the departments.

ANSI/X3/SPARC DBMS ARCHITECTURE DISTRIBUTION

Assuming that the conceptual schema can be divided in local parts, the ANSI/X3/SPARC DBMS architecture can

be easily distributed in a computer network. In order to simplify the presentation, host computers of the network are classified in two types:

- *Data computers*, which contain a set of data stored in a local data base.
- *Processing computers*, which execute external data base application programs.

In the last section, an integration will be presented where

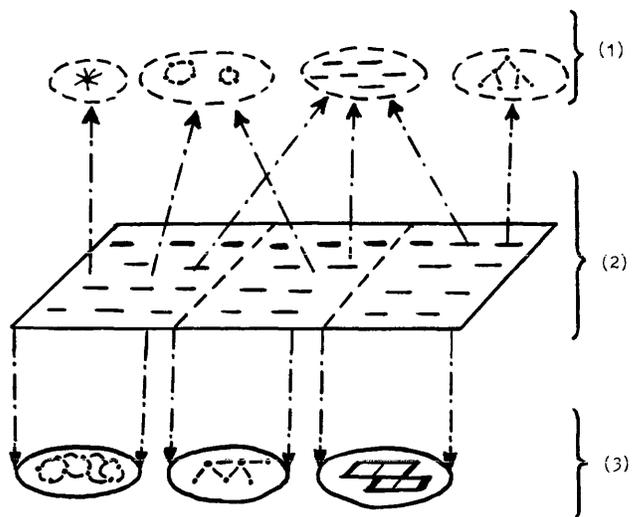


Figure 3—Schema binding and division. (1) External schema (several external models). (2) Conceptual schema divided in local parts (one model). (3) Internal schema (one internal model for each computer type).

each host is simultaneously a data computer and a processing computer.

Data dictionary directory

A key point of the ANSI/X3/SPARC DBMS architecture is the data dictionary directory. It can be seen as a meta-data base where every DBMS processor fetches the parameters required by its execution. In order to implement the ANSI/X3/SPARC DBMS architecture with data computers and processing computers, it is necessary to distribute the elements of this data dictionary directory. The main elements of interest are

- The user program descriptions
- The external data base schema object type descriptions
- The mapping structures relating external and conceptual objects
- The conceptual data base schema object type descriptions
- The mapping structures relating internal data base and conceptual one
- The internal data base schema object type description

A large part of these elements is now easy to distribute. By hypothesis, a user program description and an external schema are located on a processing computer. In the same way, each data computer needs at least an internal schema to describe the internal structure of its local data base. Then, as the conceptual schema is divided in local parts, it is desirable to implement each part on the corresponding data computer, particularly in order to avoid duplication of the whole conceptual schema on each processing computer. Consequently, mapping structures relating internal and conceptual level are situated on data computers.

The only point to discuss is the place of the mapping structures relating external and conceptual objects. As they are strongly dependent on the model seen by external application programs, localization on each computer which uses them improves the independence between computers. Let us point out that an external schema can be mapped on the whole conceptual schema; thence, an external object can be mapped into many conceptual objects of different localization. Consequently, the mapping structures relating external and conceptual objects must include the distribution rules of the external objects.

Functional processors

Once the data dictionary directory is distributed, each DBMS processor can be located using the simplest criterion of setting it on the computer which manages the parameters most frequently required by its executions. Consequently a processing computer is equipped with an external data base schema processor (one for each external model) and a conceptual/external data base transformer (one for each external model). A data computer is endowed with an internal data

base schema processor, a conceptual data base schema processor, an internal data base/conceptual transformer and an internal storage/internal data base transformer. Let us point out that the transformation of external objects into conceptual ones includes the distribution of external objects on data, that is to say, the decomposition of global queries and updates into local ones.^{25,21}

Interfaces

The distribution of ANSI/X3/SPARC DBMS interfaces is straightforward. However, two interfaces must be transformed into a protocol.

- a. The conceptual data manipulation language (system format) must be exchanged on the network between a pair processing computer—data computer. This includes conceptual objects requests and receipts with associated control. For this purpose, it is proposed to develop a standardized *data manipulation protocol*. Such a protocol must be derived from a data manipulation language with a high degree of functionality in order to take care of the network slow rate of communication. A good example of such a language is QUEL²² as used in SDD.1²¹ and in progress in the SIRIUS project.⁹
- b. The external data bases schema processor needs have access to the distributed conceptual schema in order to bind the external objects to objects declared in the conceptual schema. It can be performed by consulting the meta- data base which contains on each data computer the conceptual schema. For this purpose, the previous data manipulation protocol can be utilized. It allows documentation on the conceptual schema from a processing computer if a facility is provided to manipulate the meta- data bases which contain the local parts of the conceptual schema.

Finally, only one data manipulation protocol must be added. This protocol is implemented as the first level of the communication kernel which is described in the next section. Figure 4 gives the schematic of the system on a processing computer and Figure 5 gives the schematic on a data computer. Let us point out that Interface 3 must stay accessible through the network for the application system administrator; this interface can be implemented with the data manipulation protocol.

THE COMMUNICATION KERNEL

An overview

The communication kernel performs and controls the communications of data manipulations and resulting entities. It includes the transport management which carries out the transfer of data between endpoints and which corresponds to Levels 1, 2 and 3 of the ANSI/X3/SPARC distributed

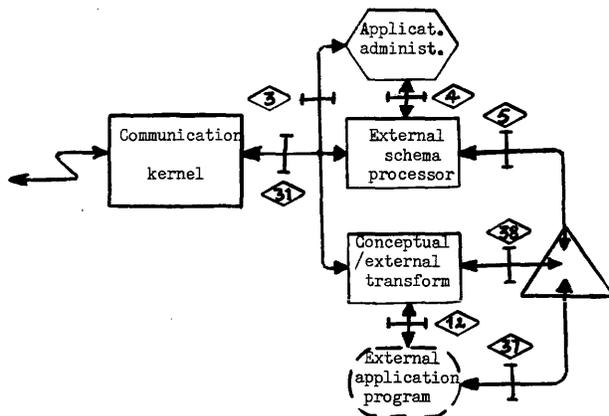


Figure 4—Partial schematic on a processing computer.

reference model (1, 2, 3 and 4 in (ISO 78), see Figure 2) when the endpoints are located on different computers. When they are located on the same computer, the transport management corresponds only to a buffer movement. In addition, the communication kernel includes two layers:

- The session control which controls the correct interaction of processes (transaction and possibly batch processes) with the conceptual data base.

- The data manipulation control which controls requests and receipts of conceptual object occurrences.

The functions of these two levels are specified in the following. Each of them requires a specific protocol between pairs of controllers. The different layers implemented on every computer are summarized in Figure 6. Figure 7 illustrates the different levels of protocols between two sites.

Data manipulation controller functions

The different functions of the data manipulation controller are the following:

- Communication of conceptual data manipulations, i.e. coding/decoding into/from messages and sending/receipt of these messages.
- Communication of status, i.e. coding/decoding into/from messages and sending/receipt of these messages.
- Communication of conceptual object occurrences, i.e. packing/unpacking into/from messages with possibly ciphering/deciphering and sending/receipt of these messages.
- Fatal error control and occasional abortion of conceptual data manipulations.

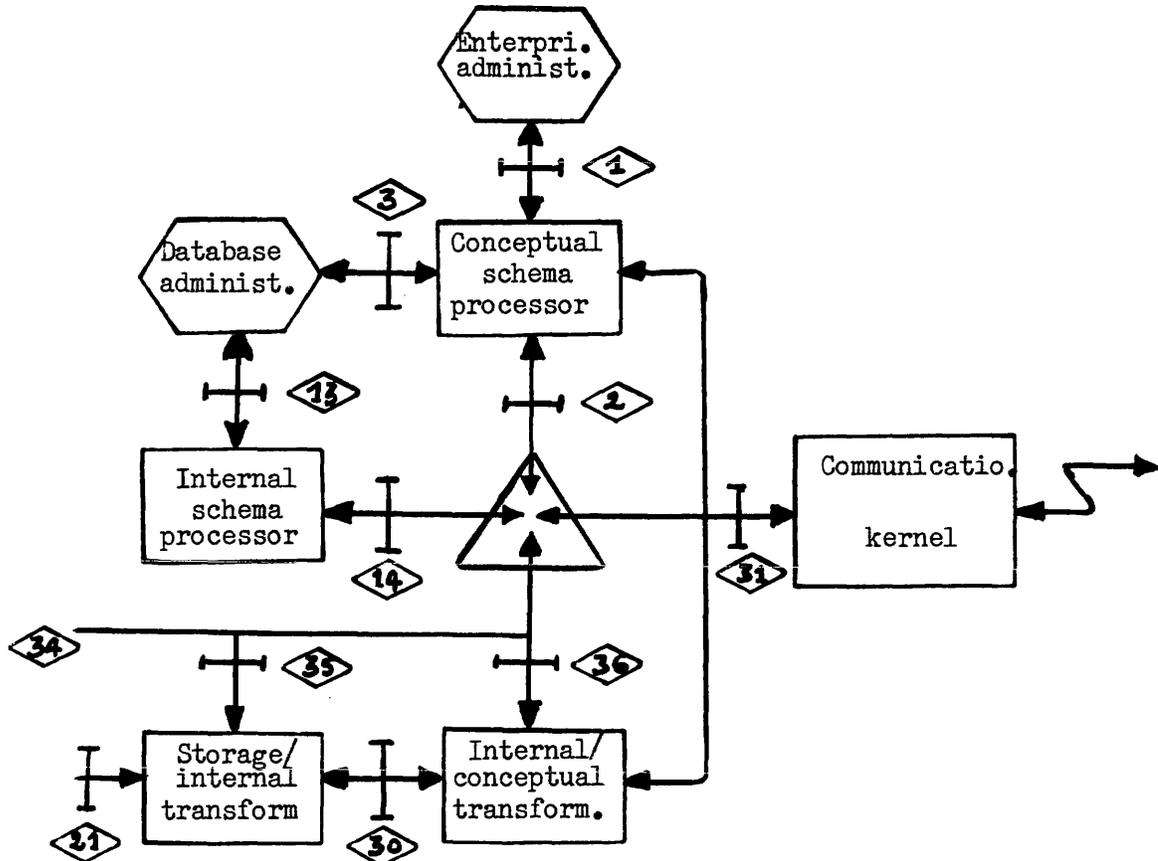


Figure 5—Partial schematic on a data computer.

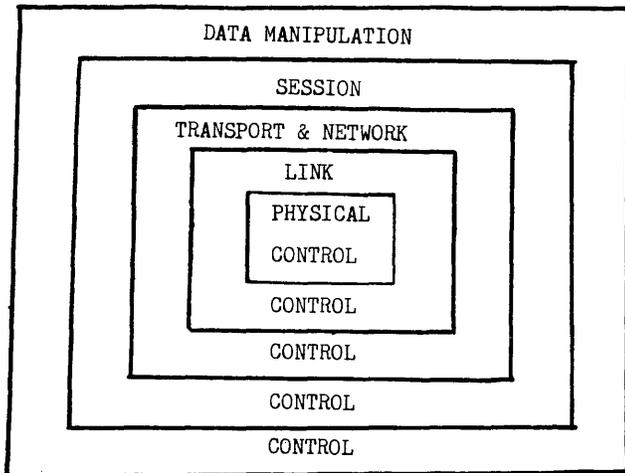


Figure 6—An overview of the communication kernel.

- Global flow control of the number of objects generated by each conceptual data manipulation.

The data manipulation protocol specifications define the format of messages which contain data manipulations, status and conceptual objects.

Session controller functions

The session controllers coordinate the distributed processing. The main functions of this layer are the following:

- Initiation and termination of processes.
- Start/stop of process steps (transaction commitment unit or job step).
- Journalizing of updates.
- Commitment of updates and step back-up and recoveries.
- Resolution of concurrency conflicts.

The session control protocol specifications define the format of messages requesting initiation and termination of processes, start/stop of steps, commitment of updates,⁴ locking and unlocking of objects,¹⁵ collection of locking status for deadlock detection,¹⁰ step back-up and recoveries.¹⁴

THE UNIFIED ARCHITECTURE

The unified architecture is now straightforward. Two general-purpose computers are represented Figure 8.

The external presentation box summarizes the processors and interfaces represented in Figure 4, except the external application programs and the communication kernel. It includes the external schema processors whose roles are the validation of the external schema declarations, their binding to the conceptual schema and the insertion of the resulting structures in the local part of the data dictionary directory.

It also includes the conceptual/external transformers which receive the user primitives and translate them into standardized conceptual data manipulations. For this purpose, the structure describing the external schemas and their binding to the conceptual one are utilized. Of course, several external models and external data manipulation languages should be offered to the users.

The data base management box summarizes the processors and interfaces represented in Figure 5, except the communication kernel. It includes the conceptual schema processor which interacts with the enterprise administrator for the declaration of the local part of the conceptual schema. After compilation, the structures resulting from this local part of the conceptual schema are stored in the local part of the data dictionary directory. This box also includes the internal/conceptual transformer which receives the conceptual data manipulation from the communication kernel and transform them into internal data manipulations using the structures resulting from the conceptual schema and the mapping structures relating conceptual schema and internal one. Of course, the data base management box also includes all the internal and storage facilities.²

Let us point out that there is no direct path to interchange a message between two application programs located on the same computer or on different ones. This can be performed through data bases. However, in order to simplify such a path, objects can be defined in the internal to conceptual mapping description as stored in main memory—only buffering will be performed.

CONCLUSION

The proposed architecture requires the definition of every interchangeable object in the conceptual schema. That is, in the mind of the author, a key-point to ensure the success of distributed systems. Indeed, it is alarming to see in some distributed enterprises the development of distributed applications without control over the application-level communications—each application programmer is allowed to specify his own application protocol.

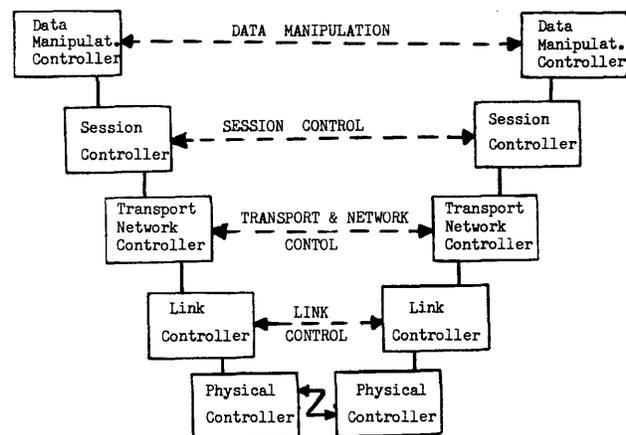


Figure 7—The different levels of protocols.

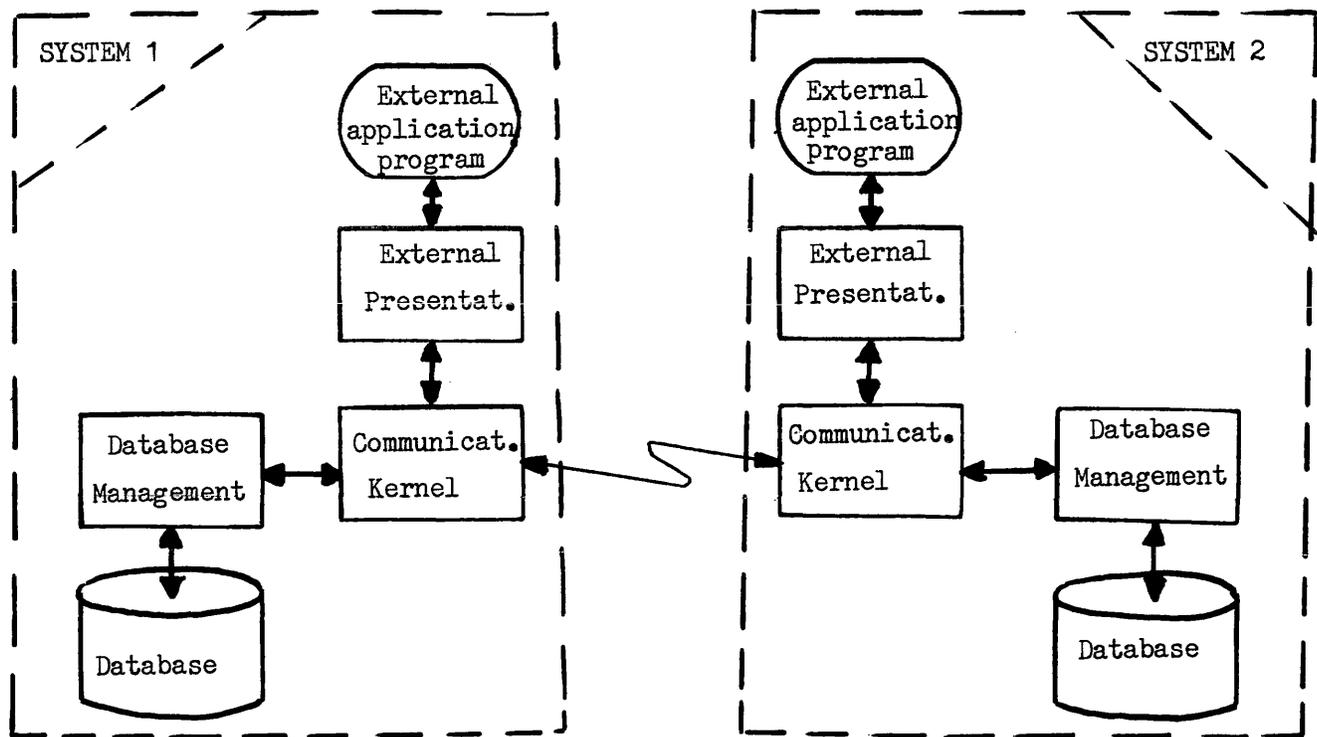


Figure 8—An overview of the unified architecture.

At the present time, the question of great importance is to control and standardize the high-level communications between computerized workstations. The conceptual description of objects which are modeled and consequently can be interchanged is a necessary tool towards the ultimate goals: "To make every process in the world addressable to one another such that they can exchange information when such exchange appears useful. . . ." But also, to make every set of data accessible to every process when such accesses are useful and authorized.

ACKNOWLEDGMENTS

The proposed unified architecture has been improved by helpful discussions with R. Gomez, M. Jouve, with the SIR-IUS project members and with C. W. Bachman.

REFERENCES

1. Adiba, M., J. C. Chupin, R. Demolombe, G. Gardarin and J. Lebihan, "Issues in Distributed Data Base Management System: A Technical Overview," *4th Very Large Data Base*, Berlin, Sept. 1978, pp. 89-110.
2. ANSI/X3/SPARC, "Interim Report on Data Base Management Systems," March 1975.
3. ANSI/X3/SPARC, USA Contribution to ISO/TC97/SC16, Draft Reference Model on Distributed Systems, Jan. 1978.
4. Bachman, C. W., "Advances in Database Technology," *Infotech State of The Art Tutorial*, London, Dec. 1977.
5. Bernstein, P. A., J. B. Rothnie et al., "Analysis of serializability in SDD.1," *CCA Technical Report Number 77-05*, 1977.
6. Blasgen, M. W., "Issues in the design and implementation of DBMS," *1977 Proc. Databases, AICA '77, Pisa*, pp. 3-21.
7. Gardarin, G., and D. Rind, "Specifications Fonctionnelles des Systemes de Bases de Donnees Reparties," *Convention Informatique 1976, Paris*, pp. 84-90.
8. Gardarin, G., and J. Lebihan, "An Approach Towards a Virtual Database Protocol For Computer Networks," *1977 Proc. Databases, AICA '77, Pisa*, pp. 231-241.
9. Gardarin, G., "Resolution des Conflits d'Acces Simultanes a un Ensemble d'Informations Application aux Bases de Donnees Reparties," These D'Etat, Universite de Paris VI, Avril 1978.
10. Goldman, B., "Deadlock Detection in Computer Networks," MIT Thesis, Boston, June 1977, 84 pp.
11. ISO/TC97/SC16 N, Reference Model of Open Systems Architecture, November 1978.
12. Hoffman, L. J., *Modern Methods For Computer Security and Privacy*, Prentice Hall Inc., Englewood Cliffs, New Jersey, 1977.
13. Jouve, M., "Reliability Aspects in a Distributed DBMS," *1977 Proc. Databases, AICA '77, Pisa*, pp. 199-209.
14. Lampson, B., and H. Sturgis, "Crash Recovery in a Distributed Data Storage System," Internal Report, Xerox Research Center, Palo Alto, 1977.
15. Menasce, D. A., and R. R. Muntz, "Locking and Deadlock Detection in Distributed Databases," *3rd Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley, August 1978, pp. 215-234.
16. Nahouraii, E., A. F. Cardenas, and O. Brooks, "An Approach to Data Communication between Different Generalized DBMS," *Second Very Large Databases*, Brussels, Sept. 1976.
17. Neuhold, E. J., and H. Biller, "POREL: A Distributed DBMS on an Inhomogeneous Computer Network," *4th Very Large Data Bases*, Tokyo, Oct. 1977.
18. Nijssen, G. M., "A Gross Architecture For the Next Generation DBMS," *Modelling in Database Management System*, North Holland, 1976.
19. Parent, C., "Integrity in Distributed Databases," *1977 Proc. Databases, AICA '77, Pisa*, pp. 187-198.

20. Rosenkrantz, D. J., R. E. Stearns and P. M. Lewis, "System Level Concurrency Control for Distributed Database Systems," *ACM TODS*, Vol. 3, No. 3, June 1978, pp. 178-198.
21. Rothnie, J. B., and N. Goodman, "An Overview of the Preliminary Design of SDD-1," Technical Report CCA-77-04 March 31, 1977.
22. Stonebraker, M., E. Wong and P. Kreps, "The Design and Implementation of INGRES," *ACM TODS*, Vol. 1, No. 3, Sept. 1976, pp. 189-222.
23. Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," *3rd Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley, Aug. 1978, pp. 235-258.
24. Tsichritzis, D., and A. Klug, "The ANSI/X3/SPARC DBMS Framework," *Technical Note 12*, University of Toronto, July 1977.
25. Wong, E., "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases," *2nd Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley, Aug. 1977.

Design of a prototype ANSI/SPARC three-schema data base system*

by ERIC K. CLEMONS

University of Pennsylvania
Philadelphia, Pennsylvania

INTRODUCTION TO THE THREE-SCHEMA ARCHITECTURE

This paper has three objectives. First, it describes briefly how an ANSI/SPARC three-schema data base system prototype could be constructed, using wherever possible available data models, system software, and research results. Second, it lists data models selected for each of the three levels; it also explains these selections. We find no existing proposal for an external schema facility to be adequate; therefore, our third objective is to develop specifications for the external level. A user interface based upon simple hierarchical user records is proposed, the necessary theory is described and a mapping language for the definition of external schemata is proposed.

The first three sections treat the first and second objective, describing and justifying design decisions for each level. The next three sections introduce specifications for the external level. The final section presents conclusions.

Need for multiple data models

Data structures employed in an integrated data base management system must address three goals: enterprise support, user support, and machine access for retrieval and storage. Enterprise support requires logical completeness: if data have been gathered and maintained at considerable cost, then it is essential that it be possible to use this data to respond to any logically meaningful query. User support requires logical simplicity: regardless of the complexity of the structures needed to support the enterprise's data processing, it is essential that the structures with which an individual user must interact be both simple and well suited to his programming needs. And, for machine access to the stored data, data description must be provided at a level low enough to permit efficient operation by the physical devices.

Unfortunately, these requirements usually appear to be incompatible. A structure that is logically complete enough

for the enterprise is not sufficiently simple for convenient use by most programmers. Likewise, as is shown by example in the following section, a structure that is well designed for one application may not be suitable for another.

A promising mechanism for resolving these difficulties is the three schema model offered by ANSI/SPARC.¹ Rather than attempt to define a single class of data structures of universal applicability, ANSI/SPARC proposes three levels of structures, one each for the enterprise, the users and the machine itself.

Such a model requires not only the ability to declare data structures of different classes, but to define maps between these structures. In this paper we propose choices for data models appropriate to each of the three levels. In particular, we develop an original model for the user level and a language for mapping to it from the enterprise level.

ANSI/SPARC Architecture

In the proposed ANSI/SPARC architecture there are three separate but related levels of data base schema: conceptual schema, external schema, and internal schema. The conceptual schema must be complete; it supports the enterprise and its view of the data required for its operations. The external level includes many external schemata; each external schema supports one or more applications programmers and provides a set of data structures required by and designed for their applications. The internal schema is needed for data access at the device level.

The following terms will prove useful. We define the *stored data base* as the actual data described in the internal schema, and a *user data base* as the collection of *user records* described by the user's external schema. Mappings between the stored data base and a user data base are, at least in theory, composed of maps from the internal to the conceptual level and from the conceptual to the external level. An external schema is often referred to as a user view of a data base or as a *user view*.

* Research supported in part by National Science Foundation Grant MCS77-18108.

Design of a prototype

The following design is proposed for a three-schema data base prototype. Its construction can be facilitated by exploiting existing software. It supports necessary functions for both machine efficiency and applications programmer effectiveness. And the data models used at all three levels are well understood.

For the internal level we propose that a CODASYL data base management system^{9,10} be employed. The conceptual level will be a relational system.^{2,11} And the external level will be based upon a Virtual Information Object (VIO) interface, introduced in an earlier work⁶ and summarized in a later section. These choices are defended in the third section. The remainder of the paper describes the proposed external schema facility.

The three schema prototype will have the structure shown in Figure 1. We shall examine in detail only Interface One.

AN ANSI/SPARC EXAMPLE

We consider a simple university data base including six entity types and the relationships among them: Departments offer courses, employ both students and faculty and have students taking a major concentration in the department's courses. Faculty members teach course sections and advise students. Students enroll in course sections, and for each course taken by each student there is a course grade. A conceptual schema can be presented in several different ways; the data structure diagram in Figure 2 is but one way of displaying this university data base.

Several interesting external schemata can be defined on this conceptual schema; we offer three:

1. A grade report for each student, listing all courses and grades for courses taken.
2. A course roster, listing for each course section the faculty member who taught it and a list of students in the section and their grades.
3. A departmental roster, listing all student majors of each department and their grade-point average.

These data structures are shown in Figure 3.

We note that the structures of Figures 3a and 3b appear to be incompatible; for the first we want the data base

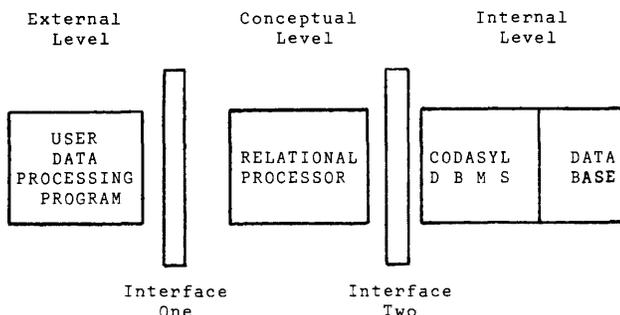


Figure 1—Form of the proposed three-schema data base system prototype.

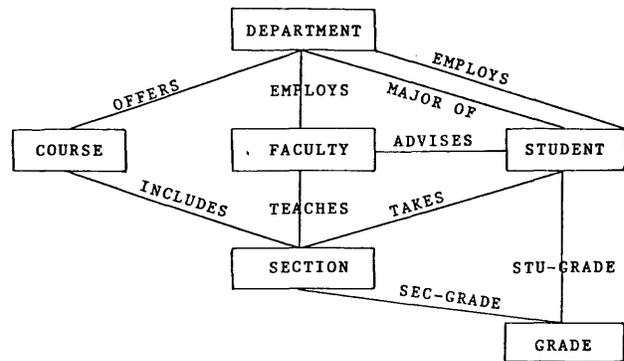


Figure 2—A university data base with six record types and the relationships among them.

organized as a hierarchy with courses listed for each student, while for the second an organization with the students listed for each course seems most appropriate. Also, the grade-point average data in Figure 3c, while derivable from data in Figure 2, is not explicitly stored in the university data base. These observations can be summarized:

1. None of the figures correspond exactly to the data structure of Figure 2 nor to a subset of this figure.
2. Each is a legitimate user view for a data processing application.
3. All are to be supported by the conceptual schema for the academic data base.
4. Each is to be supported by an appropriate external schema.

SELECTION OF DATA MODELS—INTERNAL, CONCEPTUAL AND EXTERNAL SCHEMATA

CODASYL—Choice for internal level

For a variety of reasons relating to efficiency of data access, we select the CODASYL model as the basis for the internal level.

Requirements at this level are control of record placement and definition of access paths for efficient selection of single records, subset queries and complex queries requiring data base navigation. The CODASYL model permits placement of records in areas, and supports indexed and hashed single record access, multi-list processing using pointer chains, inverted access using pointer arrays, and navigation using access through sets.

But we must reject the CODASYL model as too complex for the external level.^{7,13} Likewise, we must reject it as too rigid and inflexible for the conceptual level:⁷ while extensions to the functions to be supported by a data base can generally be encompassed by changes to the CODASYL schema, these changes are likely to result in dramatic restructuring of the schema, rather than mere extensions to it. For example, additional functions require replacing a hierarchical relationship between two record types with a confluency among three.

STUDENT-NAME		
	COURSE	GRADE
	COURSE	GRADE
	
	COURSE	GRADE

(a) A student Grade Report based upon the data base shown in Figure 2

FACULTY-NAME	COURSE	
	STUDENT-NAME	GRADE
	STUDENT-NAME	GRADE
	
	STUDENT-NAME	GRADE

(b) A Course Roster based upon the data base shown in Figure 2

DEPT-NAME		
	STUDENT-NAME	GRADE-POINT-AVE
	STUDENT-NAME	GRADE-POINT-AVE
	
	STUDENT-NAME	GRADE-POINT-AVE

(c) A Departmental Roster based upon the data base shown in Figure 2

Figure 3—Three user views defined upon the university data base of Figure 2.

Relational—Choice for conceptual level

We select the relational model as the basis for the conceptual level.

The relational model is flexible, in that changes in use of the data base will rarely require changes in the structure of the conceptual schema. It is mathematically rigorous, in that its operators are defined in terms of the predicate calculus. And it is complete, in the sense that any retrieval request expressible in the first order predicate calculus may be performed.

But we must reject the relational model as too tedious for the external level:⁶ overcoming normalization to recreate records with structured or repeating data items requires lengthy and unnatural queries. Likewise, we reject the relational model for the internal level, because we require more explicit control of record placement and of the specification of efficient access paths than the relational model provides.

There exist other possibilities for the conceptual level, e.g., Chen's Entity Relationship Model⁵ or recent work by Bachman⁴ or Gerritsen and Lee.¹⁴ We prefer the relational model because it is more proven and more robust. The relational model has been subjected to considerable analysis and several implementations are currently operational.^{2,15} The relational model's syntactic simplicity makes it unlikely that changes to the perceived relationships among entities will require changes to the schema.

Choice for external—No model available

There is no model available that fully meets our requirements for an external schema facility. Requirements are summarized and a proposed model is presented in the following section.

DESIGN OF AN EXTERNAL SCHEMA FACILITY

The external schema provides the basis for convenient use of the data base by individual applications programmers. It supports the definition of user views, that is, virtual data bases constructed from the common stored data base. These virtual data bases may differ dramatically in format and content from the stored data base; ideally this would permit a close or exact match between the cognitive structures employed while analyzing the problem and the data structures employed while writing the program.

Requirements

The external schema facility must support restructuring of data and definition of data items in user records that are not necessarily explicitly present in the stored data base or described in the conceptual schema. Data structures in user views need not support multiple and diverse users; a user view need not be flexible or complete in the sense of a schema designed to support the data processing of the enterprise. Rather, we want the simplest possible use: to obtain the description of a single entity, the user requests a single data access. Thus, to obtain grade-point averages for students majoring in decision sciences, we should be able to write a query of the form:

```
SELECT GRADE-POINT-AVE
FROM STUDENT
WHERE MAJOR = 'DEC SCI'
```

It should not be necessary to traverse a complex data base like the one depicted in Figure 2; it should not be necessary for the user to specify the full details of obtaining a student record, testing the major, then obtaining all grade records for this student for the course grade.

These requirements argue against supporting networks in the external schema. Networks are useful principally because data bases must support multiple users employing different access paths through the data; confluencies, for example, usually appear because different users require different and inconsistent hierarchies, not because any single application requires a network. Networks offer more power than a single user needs, but at a cost: they are too complex to be truly convenient. Similarly, these requirements argue against flat systems like the relational model; because of the simplicity of the supported structures these systems do not permit data structures that closely match the cognitive structures with which the programmer solves his problem, and thus they are too simple to be convenient. We believe that an external schema facility must support the greatest possible variety of virtual hierarchies.

We define a *virtual information object* (VIO) as the program structure corresponding to the programmer's cognitive structure employed during problem analysis and solution. VIOs are constructed as required from the stored data base to meet the needs of individual applications programmers. A *VIO instance* serves as a user record, and a *VIO decla-*

ration provides both the schema definition for a record type in the user view and the definition of the map needed to construct records of this type from the stored data base.

Classification of hierarchies

Hierarchies are either *basic* or *recursive*. A recursive hierarchy is one where subordinate entries are of the same type as their parent node and have all the same properties; thus they can have subordinate entries also of the same type. Such hierarchies can reach arbitrary depths and degrees of complexity. Discussion of recursive hierarchies is deferred until a later section.

In this section we introduce three dichotomies that permit the classification of all basic hierarchies:

1. Extent—"Single" or "all"
2. Entries—"Simple" or "grouped"
3. Content—"Complete" or "summary"

When entries are "simple," descendant nodes correspond to single entities (e.g., single treatments for a patient), and are constructed from single instances of entity descriptions (e.g., a single treatment record). When entries are "grouped" descendant nodes correspond to groups of entities (e.g., groups of all medical treatments or all surgical treatments for a patient), and are constructed from several instances of entity descriptions (e.g., the collection treatment records of the appropriate type).

When extent is "all," then all of a node's descendants are present in a single, wide, tree-structured record; when extent is "single" each node has one descendant and thus relationships between a node and its descendants are represented by a forest of narrow trees.

Combination of options for entries and extent provides power and flexibility in the definition of hierarchies. Grouped entries and single extent would provide a set of records, one for each type of treatment received by a patient. Grouped entries, all extent would provide a single record including data on all treatment types, while simple entries and all extent would provide data on all treatments but without aggregation by treatment type.

Finally, "summary" content indicates that only summary information such as counts, totals or averages are to be included in the hierarchies while "complete" content indicates that both the summaries and the complete data from which they are calculated are to be included.

These three dichotomies can be combined to form seven classes of hierarchies** as shown in Figure 4. By forming hierarchies of greater depth, structures of arbitrary complexity can be formed.

** Generally, three dichotomies would yield eight classifications. Since the combination single, simple, summary is not useful, it is not included. Thus there are only seven types of hierarchies.

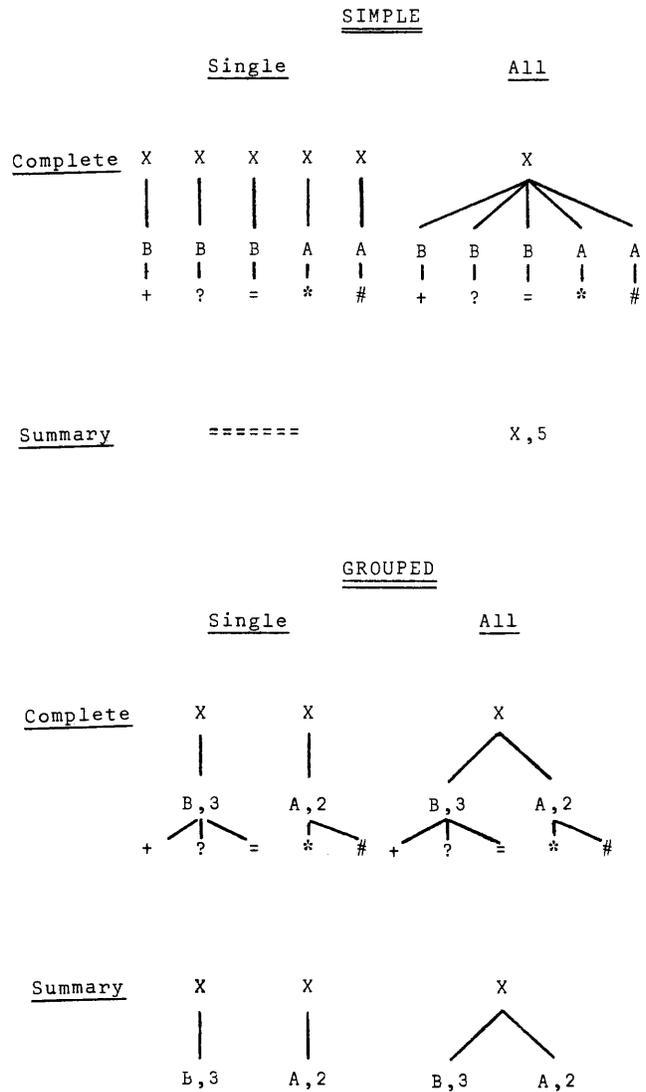


Figure 4—A general taxonomy of hierarchies.

LANGUAGE FOR DECLARATION OF USER VIEWS

Information required by the external schema facility

The external schema facility itself requires information, specifying how data from the physical data base are to be accessed and combined to produce records in the user view. This information is of three types:

1. Access information
2. Restructuring information
3. Data item declaration

Access information specifies which data are to be accessed. It names the relations in the conceptual schema that contain the desired information and gives conditions that determine which tuples are actually to be used.

Restructuring information controls the combination of data from normalized relations to produce the desired general hierarchical records. These hierarchies are of the forms presented in the fourth section; restructuring information must therefore be specified in terms of the three dichotomies that were introduced.

Data item declaration specifies what information is actually to be included in user records; this information may of course be different from the data accessed from the stored data base, as an average of course grades differs from the extensive list of grades and courses taken. Data items may be of three types: real elementary, virtual elementary or structured. A real elementary item is one present in the conceptual schema. A virtual elementary item is unstructured (i.e., a field or COBOL elementary item) and computed from items in the conceptual schema. A structured item itself contains one or more data items; since these in turn may possess structure, hierarchies of arbitrary complexity may be declared.

Introduction to the language

The language we developed for the declaration of user views is based upon SEQUEL.²

Access information is specified using the form:

```
FROM relation-name
WHERE condition
```

We have found that the condition that qualifies the tuples to be accessed is almost always of the same form: use in the descendant items those tuples whose keys are related "in the obvious way" to the keys of the parent items. As an example, in a personnel data base, keys of children contain the keys of a parent; in a customer-order data base, keys identifying details include the keys of orders to which they belong. We call this form of qualification *implication*; e.g., access those children implied by employee tuple, access those details implied by the order tuple. We represent implication by the symbol " \leftarrow ."

Restructuring information is specified by placing an option list within parentheses. The default options are "single" extent, "simple" entries and "complete" content, and only non-standard options need be specified.

There are different formats for declaring each of the types of data items. Virtual elementary items have the form:

```
item-name: defining-expression
```

Real elementary items may be declared with either the form:

```
item-name: conceptual-schema-item-name
```

or, if the data item is to have the same name in both the conceptual schema and the user record, the equivalent form:

```
SELECT item-name
```

may be used. Finally, to declare structured data items, the

form is:

```
STRUCTURE structure-name: (option-list)
item declaration
...
END structure-name.
```

Comments on this language

We offer below an example of a declaration of a map to construct a user record. We present the language statements needed to map from the conceptual schema of the university data base presented in Figure 2 to a departmental roster. In this external schema for each department there is a record containing the department name and a list of all students and their grade-point averages.

```
STRUCTURE ROSTER:
SELECT DEPT-NAME FROM DEPARTMENT;
STRUCTURE STUDENT-ENTRY: (ALL)
SELECT STUDENT-NAME
FROM STUDENT  $\leftarrow$  DEPARTMENT;
STRUCTURE COMPUTE-AVERAGE:
(ALL, SUMMARY)
SELECT GRADE FROM
GRADE-REC  $\leftarrow$  STUDENT;
GRADE-POINT-AVE: (SUMMARY)
AVERAGE (GRADE);
END COMPUTE-AVERAGE.
END STUDENT-ENTRY.
END ROSTER.
```

That is, undeniably, an ugly language for schema declaration. We offer the following observations in its defense:

1. It is not solely a DDL, as the CODASYL subschema facility is. Its mapping function subsumes much that is DML and thus eases the writing of applications programs.
2. It is not solely a subschema facility. Its DML function subsumes many of the data access and restructuring tasks that would otherwise be performed by applications programmers to construct item descriptions from several data base sources.
3. The declaration of a single map, while prepared only once, will be of use in several programs associated with an application.

Thus, while this map may not be easy to code, it permits the easy retrieval of names and averages for students majoring in Decision Sciences:

```
SELECT STUDENT-NAME, GRADE-POINT-AVE
FROM ROSTER
WHERE DEPT-NAME = 'DEC SCI'
```

Likewise, it is now easy to perform several other tasks, e.g.,

comparing averages in the Finance and Marketing Departments:

```
SELECT AVERAGE (GRADE-POINT-AVE)
FROM ROSTER
WHERE DEPT-NAME = 'FIN'
```

```
SELECT AVERAGE (GPA)
FROM ROSTER
WHERE NAME = 'MARKET'
```

In the next section we offer a more advanced example of VIO declaration. Additional details of the language, its use and its implementation, can be found in an earlier work.⁶

A more advanced example of VIO declaration

The following example illustrates use of grouping and multiple levels of summary in VIO declaration. We examine a hospital data base, in which records are maintained on patients, physicians and treatments. In particular, patient records include patient identification number, patient name and identification number of attending physician; treatment records include patient identifier, identifier of the administering physician and associated data.

We wish to construct a patient history that lists all treatments of a single patient, grouped by treatment type and within type grouped by physician. We want only summary information: Counts of the number of treatments by physi-

PATIENT:	(PAT#,	PATIENT-NAME,	PHYS#)
	H018	WELLINGTON	T12
	S127	CROMWELL	P14
	X442	NELSON	T12

TREATMENT:	(TREAT#,	PAT#,	TYPE,	PHYS#,	DATE)
T1	S127	MED	P12	06/01/79	
T2	S127	SURG	P14	06/04/79	
T9	X442	SURG	P14	06/04/79	
T5	S127	MED	P12	06/05/79	
T4	S127	P.T.	T19	06/15/79	
T8	X442	MED	P12	05/10/79	
T6	S127	P.T.	T20	06/18/79	
T3	S127	MED	P11	06/02/79	
T12	H018	MED	P11	05/15/79	
T7	S127	SURG	P14	06/04/79	

(a) Sample data for a hospital data base

NAME	CROMWELL		
TREAT-COUNT	7		
PHYS-COUNT	5		
TYPE-COUNT	3		
TYPE	MED	SURG	PT
TYPE-TREAT-COUNT	3	2	2
TYPE-PHYS-COUNT	2	1	1

(b) A VIO instance constructed from sample data

Figure 5—Sample data and a VIO instance constructed from it.

```
STRUCTURE PATIENT-HISTORY:
SELECT PATIENT-NAME
FROM PATIENT;
```

```
STRUCTURE HISTORY-DATA:
(ALL, SUMMARY, ORDER BY TYPE,
GROUP BY TYPE, GROUP BY PHYS#)
SELECT TYPE, PHYS#, DATE
FROM TREATMENT←PATIENT;
TYPE-PHYS-COUNT: (PHYS# SUMMARY)
COUNT (TREATMENT#);
TYPE-TREAT-COUNT: (TYPE SUMMARY)
COUNT (TREATMENT#);
TREAT-COUNT: (SUMMARY)
COUNT (TREATMENT#);
PHYS-COUNT: (SUMMARY)
COUNT (PHYS#);
TYPE-COUNT: (SUMMARY)
COUNT (TYPE);
```

```
END HISTORY-DATA.
END PATIENT-HISTORY.
```

Figure 6—An example of VIO declaration employing summaries and multiple levels of groupings.

cian within type, by physician, and total, as well as counts of the number of types of treatment and the number of physicians administering treatments to this patient.

An illustrative collection of conceptual level records and an associated VIO instance are depicted in Figure 5. The necessary VIO declaration is shown in Figure 6. We note that in the construction of this user record the interface performed both DML and DDL functions—data are accessed from several data base sources, grouping and ordering is performed and the necessary summaries are prepared. While the associated VIO declaration is somewhat lengthy, it is now possible for simple data processing tasks to be accomplished by the retrieval of the appropriate virtual records.

RECURSIVE HIERARCHIES

Finally, the external schema facility is extended to include declaration and processing of recursive hierarchical structures. Recursive structures are those where lower levels in the hierarchy are of the same type as higher levels, and may themselves have similar lower levels. Recursive hierarchies do arise (e.g., organizational chains of command, parts explosion diagrams) and may readily be encoded in a single relation. Unfortunately, even relationally-complete languages do not support retrieval from such structures.⁸ For example, the ORG relation represents the relationship between a department and its subordinate departments:

```
ORG: (DEPT#, SUB#)
```

The query:

```
SELECT SUB# FROM ORG
WHERE DEPT# = 1
```

retrieves all sub-departments of Department 1.

```
SELECT SUB# FROM ORG
WHERE DEPT# =
  SELECT SUB# FROM ORG
WHERE DEPT# = 1
```

retrieves all sub-departments of sub-departments of Department 1. But there is no single query which will traverse the entire organizational structure, retrieving all subordinate departments regardless of their depth in the tree.

Three concepts are essential in an external schema facility that is to treat recursive structures:

- *Definition—Node specification* is the declaration of the individual nodes of a recursive structure, their format and data content.
- *Definition—Iteration control* is the control of which nodes are to be included in a recursive structure.
- *Definition—Tree traversal* is the processing of the nodes of a recursive structure.

Node specification has much in common with VIO declaration. The structure of the node may be specified using the three dichotomies introduced in the fourth section, data may be accessed from desired tuples of any combination of relations, and the data included in the node may be real or computed virtual items as appropriate. Thus, considerable generality in the format of individual nodes is provided.

Iteration control is used to prune the depth and breadth of the tree. It determines which nodes are to be included, based upon distance from the root, content or contents of subordinate nodes. By combining node specification, which determines the contents of nodes, with iteration control, which determines the shape of the tree, the necessary generality in definition of recursive hierarchies is attained.

Tree traversal is used to process recursive hierarchies. It is used for such things as determining manpower totals over an organization, or cost of components using a parts explosion diagram. A more detailed treatment of recursive structure definition and processing is not possible here, but is available elsewhere.⁸

CONCLUSIONS

Summary of the proposed prototype

For the conceptual schema we want generality, completeness and the ability to support unanticipated users of the data base. For this we employ the relational model.

For the internal schema we want control over storage, record placement and use of indices and access paths. Whenever possible, we want efficient operation; in particular, for common and anticipated requests, we require this efficiency. For this we employ the CODASYL network model.

For the external schema we want user orientation, ease of applications programming, close relationships between the cognitive structures of the programmer and the data structures of the program. For this we employ a hierarchical external schema facility, the details of which have been outlined previously.

Features that must be provided by an external schema facility

A facility to treat adequately the declaration of external schemata must provide the following features:

1. *A set of hierarchies*—These hierarchies are virtual; that is, they are declared in the external schema but need not be stored in the data base in terms of set memberships, repeating groups or related record segments. Hence the set of hierarchies provided may be redundant or inconsistent; we feel this last point eliminates the need for the support of more general networks.^{6,7}
2. *Full support of the classes of hierarchies presented in the fourth section*—Provision must be made for "simple" or "grouped" entries, "single" or "all" extent, "complete" or "summary" content. Additional features have also proved useful; e.g., sorting entries, limiting the number of entries included.
3. *Recursive hierarchies*—Provision must be made for node specification, iteration control, and tree traversal. We feel that node specification is best managed using the features of VIO declaration and that iteration control should be a simple extension to qualification.

Status of implementation

We have completed both an analysis of the requirements for a hierarchical external schema facility and the design of a VIO-based interface to support this facility. A language processed by the interface, used for mapping between conceptual and external levels, has been presented. Two experimental tools have been constructed; a VIO-to-relational processor and a VIO-to-CODASYL processor. No serious conceptual difficulties have been encountered. Implementation of the three-schema prototype described here requires only acquisition of a relational-to-CODASYL mapping facility.

REFERENCES

1. ANSI/X3/SPARC Study Group on Data Base Management Systems Interim Report, 75-02-08, *FDT—Bulletin of the ACM SIGMOD*, Vol. 7, No. 2, 1975.
2. Astrahan, M. M., et al, "System R: A Relational Approach to Database Management," *ACM Transactions on Database Systems*, Vol. 1, No. 2, June 1976, pp. 97-137.
3. Bachman, C. W., "The Programmer as Navigator," *Communications of the ACM*, Vol. 16, No. 11, November 1972, pp. 653-658.
4. Bachman, C. W., and M. Daya, "The Role Concept in Data Models," *Proceedings, Third International Conference on Very Large Data Bases*, Tokyo, Japan, 1977, pp. 464-476.
5. Chen, P. P.-S., "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems*, Vol. 1, No. 1, March 1976, pp. 9-36.
6. Clemons, E. K., "Design of a User Interface for a Relational Data Base," Dissertation, School of Operations Research, Cornell University, 1976.
7. Clemons, E. K., "Rational Data Base Standards," Decision Sciences Working Paper 78-10-02, University of Pennsylvania, 1978.
8. Clemons, E. K., "An External Schema Facility to Define and Process Recursive Structures," Decision Sciences Working Paper 78-12-07, University of Pennsylvania, 1978.

9. CODASYL Data Base Task Group, April 1971, Report, ACM, New York, 1971.
10. CODASYL Data Description Language Committee, *Journal of Development*, January 1978.
11. Codd, E. F., "A Relational Model for Large Shared Data Banks," *Communications of the ACM*, Vol. 13, No. 6, June 1970, pp. 377-387.
12. Codd, E. F., and C. J. Date, "Interactive Support for Non-Programmers," IBM Research RJ 1400, June 1974.
13. Engles, R. W., "An Analysis of the April 1971 Data Base Task Group Report," *Proceedings ACM SIGFIDET Workshop*, 1971, ACM, New York, pp. 69-92.
14. Gerritsen, R., and R. Lee "Extended Semantics for Generalization Hierarchies," *Proceedings ACM SIGMOD Workshop*, Austin, Texas, June 1978, pp. 18-25.
15. "Implementation of Relational Data Base Management Systems," *FDT—Bulletin of the ACM SIGMOD*, Vol. 7, No. 3, 1975, pp. 3-22.

On the implementation of a conceptual schema model within a three-level DBMS architecture*

by SHAMKANT B. NAVATHE

New York University
New York, New York

and

JOHANN LEMKE

Siemens AG
Munich, West Germany

INTRODUCTION

The ANSI/X3/SPARC study group on database management systems² as well as some independent researchers^{17,10} have proposed a three-level "coexistence" architecture to database management systems. Under this approach (see Figure 1) a number of different users can be supported by means of different External Schemas, possibly with different data models and languages. It involves construction of a conceptual schema which completely represents the structure and semantics of a particular database. The underlying internal schema must provide a storage representation for the conceptual schema. Although the composition and scope of the conceptual schema is still a matter of controversy,¹ several data models, semantic models (e.g., References 7, 8, 9, 13, 16, 20, 21) etc. could be considered as candidates for defining conceptual schema. Given a particular model for the conceptual schema, a number of problems arise in its implementation into a three-level DBMS, particularly regarding the design of an internal schema specification language to specify the mapping of the model into storage.

This paper discusses the general issues involved in implementing one specific model due to Falkenberg⁹ termed "the object-role model," as a conceptual schema model. Considerations in the design of a Conceptual Schema Language will be indicated. However, the emphasis will be on the issues of Internal Schema Language design. These languages are currently being developed and implemented at the Research Laboratories, Siemens AG, Munich, West Germany. Detailed analyses of these languages will be presented in forthcoming papers.^{4,15} Rather than present detailed language syntax, mathematical definitions of terms, operations, etc., this paper will highlight the issues and design decisions deemed essential for implementing a conceptual model.

A database administrator (DBA) in an organization is en-

trusted with the task of defining a conceptual schema and an internal schema. In this paper wherever applicable, we point out that the DBA has to choose among alternatives or has to be aware of the implications of his decision.

The object-role (O-R) model

For the sake of completeness we will summarize the concepts from the object-role (O-R) model. This model is an ideal candidate for conceptual schema modeling since it has only a few basic concepts, which make it simple to use. It also has been shown to be evolvable and transformable.^{9,10} The model is used for modeling facts from a particular universe of discourse by means of objects, roles and associations. Objects are atomic, discrete elements in nature; the only information represented by them inherently is their existence. Facts concerning an object correspond to its association with one or more objects. An object performs a role in every association of which it is a part. Thus associations, which are n-ary in general, are composed of object-role pairs. Figure 2 gives an example of a model of a database in which the pairs (Doctor D, performs), (Surgical-procedure S, is-performed), (Patient P, is-operated) define the association "Surgical operation." An association may be "objectified" and may perform a role in another association. The latter is termed a *nested association*. In Figure 3 Miller's-salary is a binary association and (Miller's-salary, has-as-starting-date) is an object-role pair which is a component of the nested association "Miller's-salary-history."

The modeling concepts introduced thus far deal with modeling *instances* of objects, roles and associations. The *type* concept is introduced by which an *association type* refers to all associations with identical object-role pairs. Objects are pooled into *object types* such that objects under one type have at least one role in common. Figure 4 represents a database schema of which Figure 3 is an instance.

Objects and roles may be provided with *significations*.¹¹

* This work was done while the first author was at Siemens AG, Munich, West Germany.

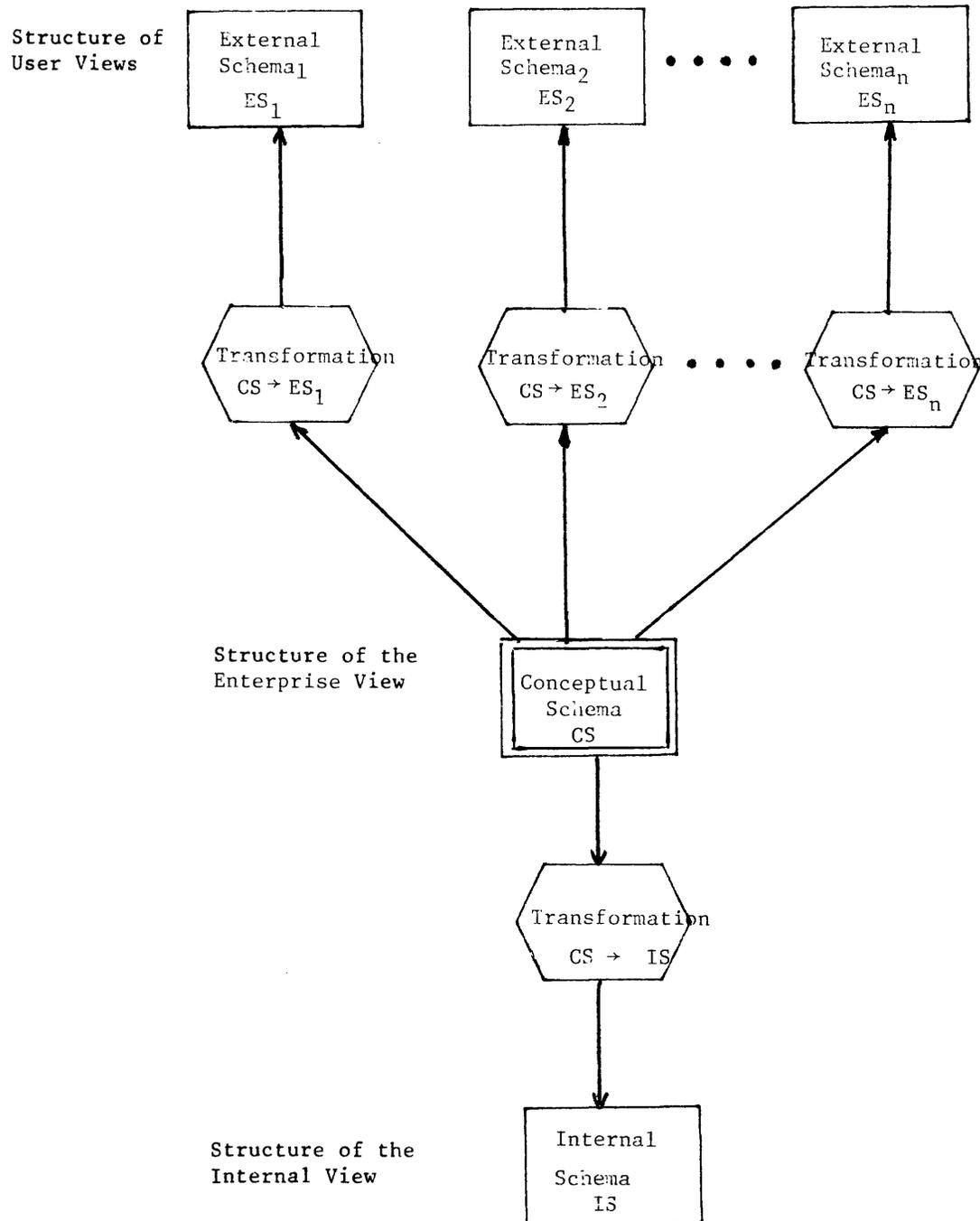


Figure 1—Three-level DBMS architecture.

An object-type named Person may be signified by the person's First-name. A number of significations may be provided, e.g. First-name, Last-name and Date-of-birth for the object-type Person to make the signification unique. Frequency of occurrence of roles may be supplied as an additional information. In Figure 4 they are shown in parentheses. E.g., (0,1000) indicates that a Date may be the starting date in 0 to 1000 Salary-history associations whereas (1,1) indicates that a particular salary association starts on one and only one Date.

A conceptual schema language (CSL)

A CSL is being designed⁴ to define the conceptual schema of a database using the concepts from the object-role model. One of the early design decisions involved the definition of object-types.

As originally defined in the model, an object is atomic and has no information of its own. On principle, *identification* (=unique significance) of an object x must be possible by associating x with objects of a different type(s)—usually

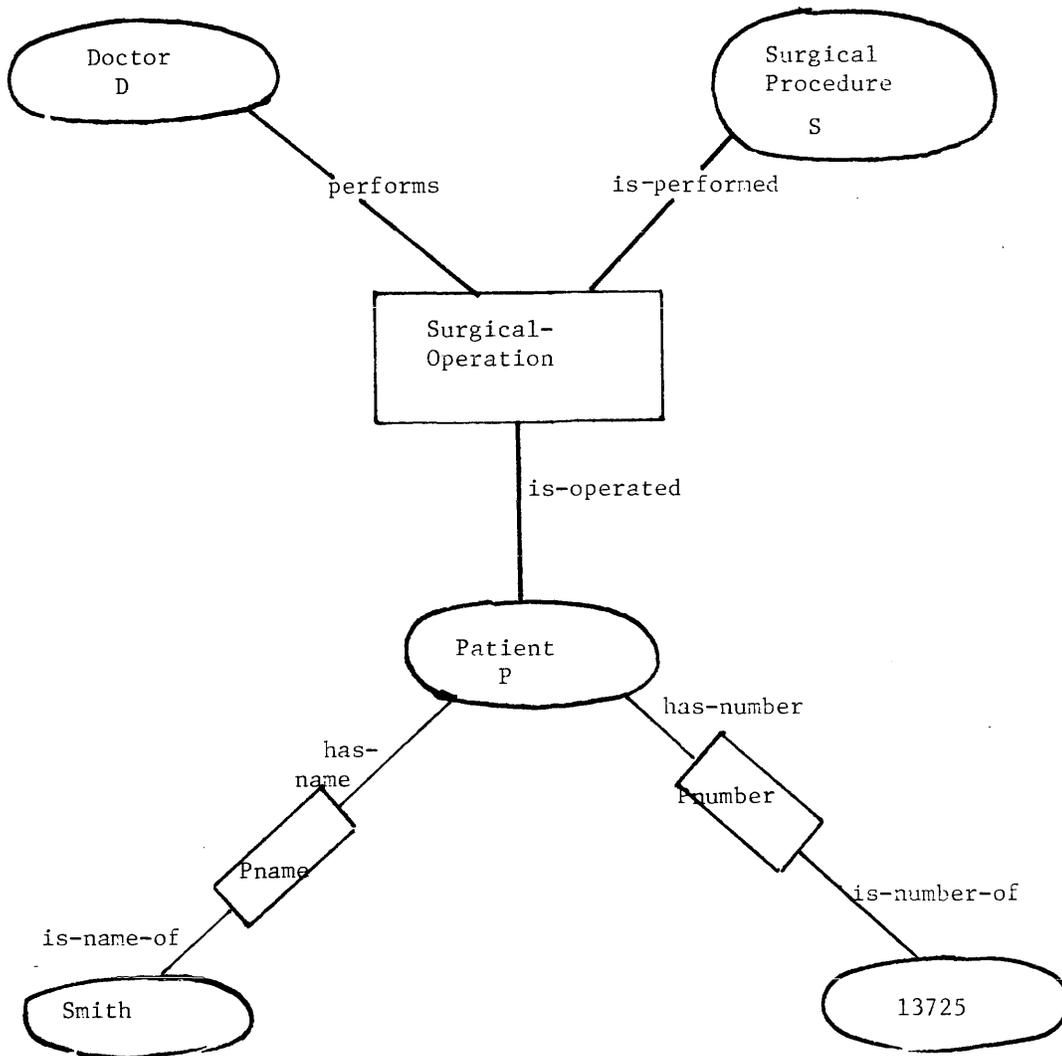


Figure 2—Objects and associations in a conceptual model⁹ (instance diagram).

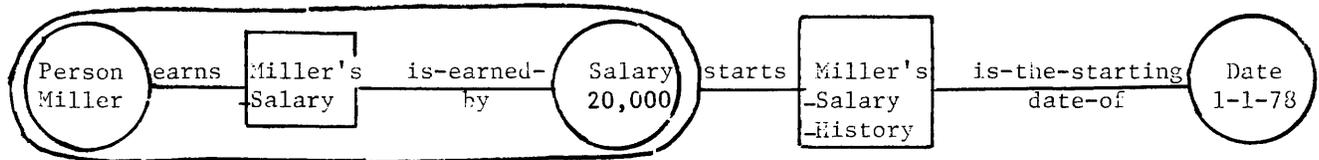


Figure 3—A nested association (instance diagram).

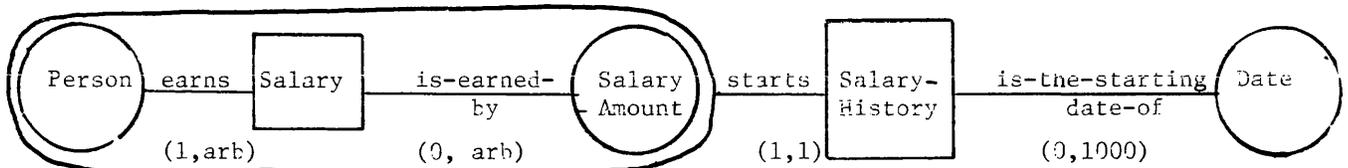


Figure 4—A database schema involving object-types and association-types (schema diagram).

strings over some alphabet. Associations serving this purpose are called *name associations* (also *identifying associations*) and an object ($\neq x$) participating in a name association for x is called a *name object for x* . E.g., in Figure 6 object type Dept is identified by association type Dnum, and Person by Pnum. In general, a hierarchy of associations may identify an object. However, if for each object x of a type X we know a string s_x that can serve as a never-changing identifier for x , CSL allows for defining X as a *self-identified object type*. This simplifies the conceptual schema. CSL deals with object types as follows:

1. An object type has a name (e.g. Person).
2. An object type is devoid of any inner structure. An association participating in a nested association is treated as an association rather than as an object.
3. A non-self-identified object type is distinguished from an object type which stands for the names of the former, (e.g. Person is different from Alpha). It is identified by a number of name associations. (E.g. each Person object is identified by an object of type Empl-number.) This concept is parallel with Navathe's identifying relations.¹⁴
4. For a self-identified object type the distinction mentioned above is not made. The name of an object of such a type is treated as if it were the object it stands for.

Roles, object types and even (object type, role) pairs may occur in more than one association type and also several times within one association type, if this is necessary to describe the semantics of a conceptual model. The latter is a means for expressing symmetric relationships between objects (see Figure 5). However, the DBA must realize that the choice of several identical (object type, role) pairs affects possible manipulations. E.g., a query like "List all persons associated with person x by role is-friend-of" implies investigation of *all* (not only one) roles is-friend-of. This is a typical example of implementing a conceptual model by selecting a proper implementation strategy rather than by changing the model itself.

Semantic rules were postulated⁹ as a means of providing additional constraints for consistency and integrity among data instances. The syntax of CSL can be designed to incorporate such rules to any degree of complexity. E.g., a very high-level semantic rule: no two names in the database can be the same; a very low-level semantic rule:

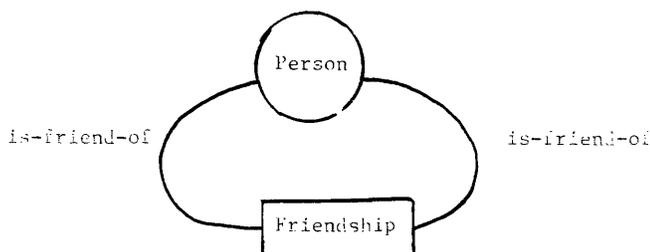


Figure 5—An association-type with identical role names.

$10,000 \leq \text{Salary} \leq 50,000$; a 'complex semantic rule: salary raises do not apply to persons who earn more than their second-level managers. Another design consideration involves the setting up of triggering mechanisms to invoke procedures representing semantic rules. Currently CSL allows semantic rules like the following to be automatically invoked during the execution of corresponding update procedures:

1. Characteristics of association types, e.g. "An employee cannot work on more than two projects."
2. Characteristics of object types, e.g. " $10,000 \leq \text{salary-amount} \leq 50,000$."
3. Dependencies between associations, e.g. "Total salaries of employees on a project cannot exceed its budget."
4. Characteristics of events, e.g., "A person's marital status cannot change from 'divorced' to 'single'."

Most databases have transactions or updates which cause changes in the data instances over a period of time. Time is an important attribute of data and provides valuable information in any dynamic database. Incorporation of time in semantic models has not been addressed sufficiently, barring a few exceptions (References 3, 5 etc.). Objects with a time point as an attribute have been called events, while those with time interval as attributes have been called processes.²⁴ Incorporation of time into the above CSL is being investigated without classifying the object-type into subtypes.⁴

AN APPROACH TO THE DESIGN OF AN INTERNAL SCHEMA LANGUAGE

In the discussion above we highlighted the features of the O-R model and the requirements of a corresponding CSL. After a particular database conceptual schema is expressed using CSL, the database must be populated with instances which are mapped into storage according to an internal schema specification. Any retrieval or updating transactions operating on an external view are mapped into transactions on the conceptual view and further into transactions on the internal view. The internal schema language (ISL) must be capable of expressing all manipulation.

Our ISL design is based on the premise that databases will be stored in electronic cyclic memories using quasi-associative addressing techniques. Existing implementations^{12,18,19} have shown that these storages and accompanying processors can store and manipulate relations very efficiently. The relational data model⁸ has also been shown⁶ to fit the bubble hardware well because of the intrinsic similarities between the two. A relational model-based ISL must allow for specifying the storage of relations corresponding to the conceptual schema in the CSL.

ISL structure

The language is divided into four levels along the logical to physical spectrum.

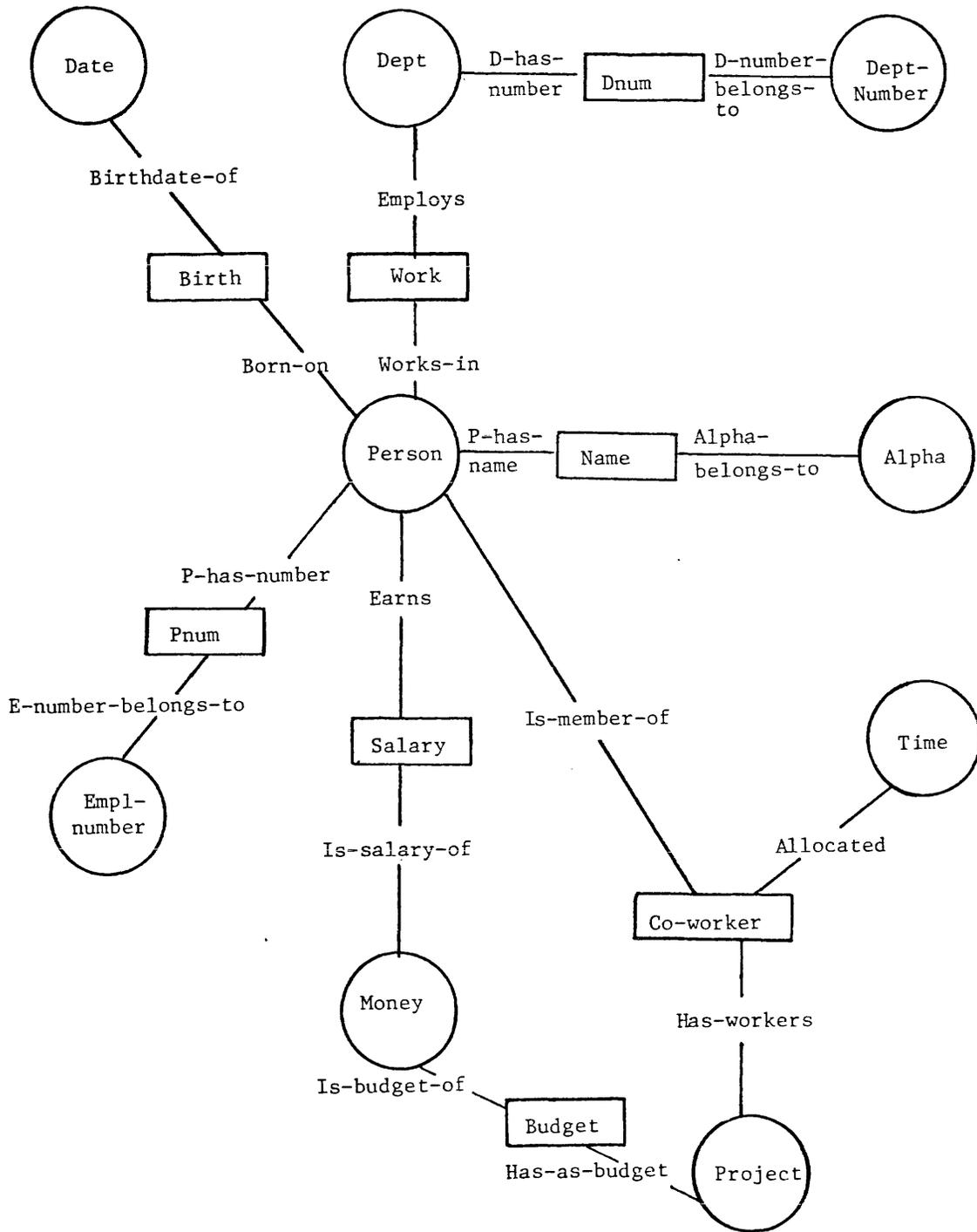


Figure 6—Conceptual model of a personnel database.

Level 1—At this level the logical structure of the stored data is specified. With the assumption that stored data is in the form of relations, one must specify *what* relations need to be stored and *how* they are populated. Two main operations termed aggregation and substitution are defined at this level.

Level 2—Provides for the specification of constraints on

relations whereby they can be organized to support efficient logical access structures. E.g., tuples may be ordered, a relation may be partitioned horizontally by clustering tuples or vertically by reordering and grouping of domains.

Level 3—At this level a lot of information is supplied to define the mapping of relations into storage. It consists

of: the division of the target address space into areas (extents) and the allocation of relations to them; definition of the type of encoding, formatting information at the domain level such as length, type (picture), padding, justification; definition of storage policies dealing with tuple insertion, overflow handling, free space management, space reclamation, relative placement of relations, etc.

Level 4—This level is used to characterize the structure of the storage itself. It defines what a unit of storage is, how these units are grouped into higher-level storage structures, what types of access is supported, whether data compression algorithms are used and so on.

Levels 2 through 4 include the detailed specifications that any Internal Schema Language should be able to express (see Reference 2). If a model other than the relational were used for stored data, relations and tuples would be replaced by corresponding constructs from that model.

In the rest of this paper we propose to focus on the ISL at Level 1 because it is at this level that the conceptual model of a database must be mapped into and specified in terms of a corresponding storage counterpart. The choice of the O-R model for conceptual modeling and the relational model for internal schema modeling presents some interesting problems.

An internal schema "view"

An Internal Schema View defines one particular way that a DBA might choose to map a given conceptual model into storage. Some of the previous work²² on internal schema has dealt with the problem of coming up with optimal solutions to mapping, i.e. defining optimal views when the target storage model is known. We assume that by doing an analysis of the processing requirements and given the knowledge of occurrence frequencies in associations, the DBA would decide to aggregate associations in a particular way; the opposite possibility where the DBA would like to split associations does not exist since in the O-R conceptual model associations represent semantically-irreducible facts.

We first describe the IS structure as it is implied by the CS definition. This is the structure we store in case no structuring specification is given.

Representation of references to objects—Whenever an ob-

ject of a self-identified object type is referred to, we store its name. For each instance y of a non-self identified object type or of an objectified association type we create an internal identifier i_y and store i_y wherever y is referred to.

Representation of associations—An association type $A = ((O_1, r_1), (O_2, r_2), \dots, (O_n, r_n))$, where O_i is an object type or an (objectified) association type and r_i is a role, is represented by an elementary relation

$$R_A \subset O_1 \times O_2 \times \dots \times O_n$$

with domain names $O_1 \cdot r_1, O_2 \cdot r_2, \dots, O_n \cdot r_n$. The term "elementary" was chosen because these relations may be used to build larger relations, but cannot be split. Given an instance of type A that associates object instances x_1, x_2, \dots, x_n such that for $1 \leq i \leq n$, x_i is of type O_i and plays role r_i , we store the tuple (y_1, y_2, \dots, y_n) in relation R_A , where, according to (i), for $1 \leq i \leq n$,

$$y_i = \begin{cases} \text{name of } x_i, & \text{if } O_i \text{ is a self-identified object type} \\ i_{x_i}, & \text{the internal identifier of instance } x_i, \text{ which we} \\ & \text{create, if } O_i \text{ is a non-self-identified object type or} \\ & \text{an objectified association type.} \end{cases}$$

A key of relation R_A comprises a sublist of the domain list of R_A .

E.g. the Birth association is represented by elementary relation $R_{\text{Birth}} \subset \text{Date} \times \text{Person}$ with domain names

$$\text{Date} \cdot \text{Birthdate-of}, \text{ Person} \cdot \text{Born-on}$$

Let us consider Figure 4 as an example of a nested association:

We create elementary relations $R_{\text{Salary-History}}$ and R_{Salary} with domain names $\text{Salary} \cdot \text{Starts}$, $\text{Date} \cdot \text{Is-the-starting-date-of}$, $\text{Person} \cdot \text{Earns}$ and $\text{Salary} \cdot \text{Amount-Is-earned-by}$. The instance of Figure 3 will be treated as follows:

1. Assuming Person , Salary-Amount are self-identified object types, the tuple (Miller, 20000) is stored in R_{Salary} .
2. Since Salary is an objectified association type, we create an internal identifier, e.g. S1, for the tuple (Miller, 20000).
3. Assuming Date is a self-identified object type, the tuple (S1, 1-1-78) is stored in $R_{\text{Salary-History}}$.

Aggregation

An IS View is defined by specifying an aggregation. E.g., to aggregate all associations in Figure 6, the following statements are used (An assumption could be made that each role name occurs in one and only one association type. However, for the sake of generality, we use role names qualified by association type names.):

V1=Name	[(Person · P-has-name)	AGGR	(Person · Born-on)]
Birth	[(Person · Born-on)	AGGR	(Person · Works-in)]
Work	[(Person · Works-in)	AGGR	(Person · Earns)]
Salary	[(Person · Earns)	AGGR	(Person · Is-member-of)]
Co-worker	[(Project · Has-workers)	AGGR	(Project · Has-as-Budget)] Budget

The philosophy of the AGGR statement is to specify a relational join⁸ by selecting a joinable domain from a number of different associations. Two relations are joinable if they are coherent, i.e. if they share at least one object type. In V1, the domains on which join is performed are either Person-ids or Project-ids. Figure 7(a) shows the elementary relations and Figure 7(b) the result of aggregation which is a relation with domains containing identifiers, either internal identifiers or name values.

In aggregating associations we had to use a "modified join" as follows: "A *modified join* takes the union (as opposed to intersection) of values in the join-domain from the relations being joined. Null values are created under appropriate domains corresponding to relations where a domain value may be absent." This is necessary because no elementary facts may be lost in mapping from the conceptual to internal schema. Figure 8 shows a conceptual schema and corresponding elementary relations. Relation BIBLIO is the result of aggregation of the two associations by joining on the Book-ids. The tuples (B3,A2,-) and (B4,-,P1) would be absent under conventional join. However, in relation BIBLIO we must preserve the fact that book B3 has author A2 and book B4 has publisher P1.

Substitution

As shown in Figure 7(b) aggregation results in the definition of an aggregated relation where certain domain values are internal identifiers. A DBA is allowed to define relations by substituting some or all of the internal identifiers by name values

Elementary relations for identifying associations

R_{pnum}	Person • P-has-number	Empl-number • E-number-belongs-to
	P1	100
	P2	110
	P3	200

R_{dnum}	Dept • D-has-number	Dept-number • D-number-belongs-to
	DP1	801
	DP2	802

Elementary relations for associations

R_{Name}	Person • P-has-name	Alpha • Alpha-belongs-to
	P1	Smith
	P2	Jones
	P3	Smith

R_{Salary}	Person • Earns	Money • Is-salary-of
	P1	40K
	P2	60K

R_{Budget}	Project • Has-as-budget	Money • Is-budget-of
	PR1	100K
	PR2	200K
	PR3	1M

R_{Birth}	Person • Born-on	Date • Birthdate-of
	P1	1947 07 01
	P2	1930 01 15

$R_{Co-worker}$	Person • Is-member-of	Time • Allocated	Project • Has-workers
	P1	50%	PR1
	P1	50%	PR2
	P2	20%	PR3
	P2	80%	PR1
	P3	100%	PR2

R_{Work}	Person • Works-in	Dept • Employs
	P1	DP1
	P2	DP2
	P3	DP1

Figure 7a—Elementary relations for the conceptual model in Figure 6.

by using a SUBST specification. E.g., continuing with Figure 7(b),

PERSONNEL=VI where [Person. P-has-name SUBST Pnum]
 [Dept. Employs SUBST Dnum]

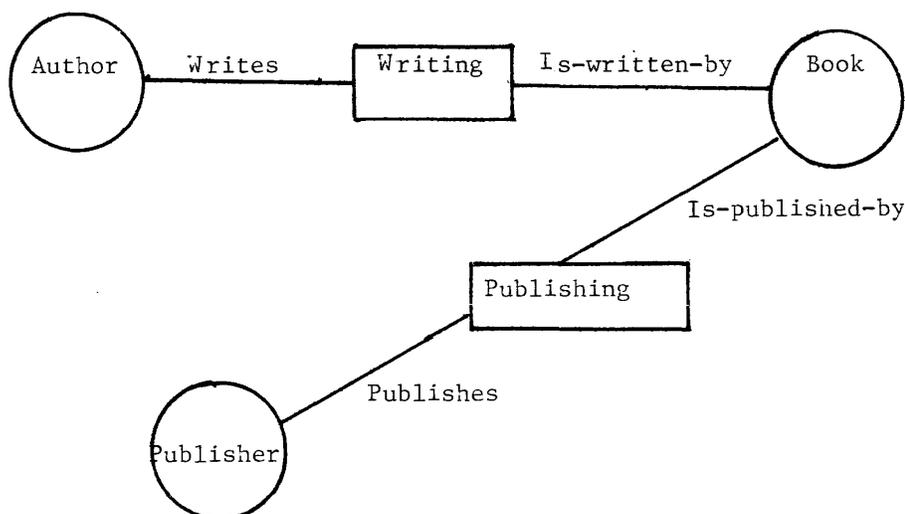
will produce a PERSONNEL relation in which Persons and Departments are represented with name values instead of by internal identifiers. See Figure 7(c).

(Money. Is-budget-of)	(Time, Allocated)	(Project, Has-workers, Has-as-budget)	(Money. Is-salary-of)	(Dept. Employs)	(Date, Birthdate-of)	(Alpha. Alpha-belongs-to)	(Person, P-has-name, Born-on, Works-in, Earns, Is-member-of)
100K	50%	PR1	40K	DP1	1947 07 01	Smith	P1
1M	20%	PR3	60K	DP2	1930 01 15	Jones	P2
200K	100%	PR2	---	DP1	---	Smith	P3
200K	50%	PR2	40K	DP1	1947 07 01	Smith	P1
100K	80%	PR1	60K	DP2	1930 01 15	Jones	P2

Figure 7b—Aggregate relation V1 for the conceptual model in Figure 6.

(Money. Is-budget-of)	(Time, Allocated)	(Project, Has-workers, Has-as-budget)	(Money. Is-salary-of)	(Dept-number, D-number-belongs-to), (Dept. Employs)	(Date, Birthdate-of)	(Alpha. Alpha-belongs-to)	(Empl-number, E-number-belongs-to), (Person, P-has-name, Born-on, Works-in, Earns, Is-member-of)
100K	50%	PR1	40K	801	1947 07 01	Smith	100
1M	20%	PR3	60K	802	1930 01 15	Jones	110
200K	100%	PR2	---	801	---	Smith	200
200K	50%	PR2	40K	801	1947 07 01	Smith	100
100K	80%	PR1	60K	802	1930 01 15	Jones	110

Figure 7c—Aggregate relation PERSONNEL, derived from V1 by substitution.



$R_{Writing}$	<u>Author•</u> <u>Writes</u>	<u>Book•</u> <u>Is-written-by</u>
	A1	B1
	A1	B2
	A2	B1
	A2	B2
	A2	B3

$R_{Publishing}$	<u>Book•</u> <u>Is-published-by</u>	<u>Publisher•</u> <u>Publishes</u>
	B1	P1
	B2	P2
	B4	P1

BIBLIO = Writing [Book.Is-written-by AGGR Book.Is-published-by] Publishing ;

R_{BIBLIO}	<u>Book•</u> <u>Is-written-by,</u> <u>Is-published-by</u>	<u>Author•</u> <u>Writes</u>	<u>Publisher•</u> <u>Publishes</u>
	B1	A1	P1
	B1	A2	P1
	B2	A1	P2
	B2	A2	P2
	B3	A2	--
	B4	--	P1

Figure 8—A modified join operation.

Manipulation of IS

Additional ISL statements have been defined for a further manipulation of the stored relations by the DBA. A DROP statement has the following syntax:

`<dropstatement> ::= DROP <typename> {, <typename>}n
 <typename> ::= <objecttypename> | <associationtypename>`

If the DBA visualizes that all the processing against a conceptual schema can be supported by the defined aggregations, he may proceed to drop certain object types and/or association types. The corresponding elementary relations would be

Given relation R with one:many relationships;

R	<u>A</u>	B	C
	a1	b1	c1
	a1	b1	c2
	a1	b1	c3

	a1	b2	c1
	a1	b2	c3

	a1	b3	c6

	a2	b1	c2
	a2	b1	c4
	a2	b1	c6

	a1	b2	c1
	a1	b2	c2
	a1	b2	c3

R takes up 36 elements of storage

S = R REPEAT C 3 TIMES; gives

S	<u>A</u>	B	C(1)	C(2)	C(3)
	a1	b1	c1	c2	c3
	a1	b2	c1	c3	-
	a1	b3	c6	-	-
	a2	b1	c2	c4	c6
	a1	b2	c1	c2	c3

S takes up 25 elements of storage

T = S REPEAT B 2 TIMES; gives

T	<u>A</u>	B(1)	B(2)	C(1)	C(2)	C(3)
	a1	b1	b2	c1	c2	c3
	a1	b2	-	c1	c3	-
	a1	b3	-	c6	-	-
	a2	b1	-	c2	c4	c6

T takes up 24 elements of storage

Figure 9—Use of REPEAT by DBA to adjust the breadth of relations.

automatically scratched. To scratch a specific relation from internal-schema storage, the DBA uses a SCRATCH statement.

$\langle \text{scratchstatement} \rangle ::= \text{SCRATCH}(\text{relationname})\{\langle \text{relationname} \rangle\}^{\infty}$

When there is an association type which associates an object type with two or more object types in a one:many fashion, multiple values occur in certain domains for a given value in one domain. Figure 9 shows an example relation which results from one:many associations between A:B and A:C. To fit the storage-characteristics a DBA may want to adjust the "breadth" of a relation by allowing domains to be subscripted as shown in Figure 9. A REPEAT statement is used toward that purpose (see Figure 9). This idea is related to multi-valued dependencies.²³

$\langle \text{repeatstatement} \rangle ::= \langle \text{relationname} \rangle \text{REPEAT}(\text{domainlist}) \text{ n TIMES}$

To revert back from the use of SUBSTITUTE, DROP, SCRATCH and REPEAT commands, a DBA can use RESUBST, KEEP, CREATE and COLLAPSE commands.

One of the advantages of choosing a relational structure for the internal schema is that the manipulation operations can be derived from the relational algebra. Queries in external schemas will be mapped into a set of relational operations like Projection, Restriction, Natural join on the elementary and aggregate relations. The implementation will account for repeating domains with subscripts and null values.

Reorganization of IS

According to the procedure outlined previously, a given conceptual schema produces a set of relations in the internal schema. The composition of this set can be controlled to some extent by the DBA by using the manipulation operations. If relations need to be normalized, rearranged, decomposed or synthesized, the relational operators will be available to the DBA. It is conceivable in the future that certain sequences of operations will be made available in the form of IS reorganization verbs.

SUMMARY

This paper has highlighted some of the typical issues that arise during the implementation of a particular semantic model in a three-level DBMS architecture. The model chosen is the object-role model due to Falkenberg.⁹ With a given design decision to organize the internal schema using relations, considerations for the design of an internal schema language were discussed. A systematic approach to defining relations and populating them so that they are equivalent to a conceptual model was outlined. Of particular importance is the concept of aggregation using a modified join operation. IS manipulation verbs for the DBA were also indicated. The Conceptual and Internal Schema Languages are currently being implemented at the Research Laboratories, Siemens AG, Munich, West Germany.

ACKNOWLEDGMENTS

The authors would like to acknowledge the cooperation of and valuable discussions with Mr. Breutmann, Dr. Falkenberg, Mr. Hahne and Mrs. Mauer in the development of the paper. The research contribution of Mr. Breutmann and Mrs. Mauer in the development of the Conceptual Schema Language is particularly recognized.

REFERENCES

1. ACM-SIGMOD 1978 International Conference on Management of Data, Panel: "The Conceptual Schema Controversy," T. William Olle, Chairman.
2. ANSI/X3/SPARC, Study Group on Data Base Management Systems, *Interim Report*, published by FDT Vol. 7, No. 2 February 1975.
3. Bradley, J., "Operations Data Bases," *Proc. Fourth Very Large Database Conference*, Berlin, West Germany, available from ACM, New York, September 1978, pp. 164-176.
4. Breutmann, B., E. Falkenberg and R. Mauer, "CSL: A Language for Defining Conceptual Schemas," Working Paper, Research Laboratory, Siemens AG, Munich, West Germany.
5. Bubenko, J. A., Jr., "The Temporal Dimension in Information Modeling," *Proc. IFIP Conference on Modeling in Database Systems*, 1977.
6. Chang, H., "On Bubble Memories and Relational Data Base," *Proc. Fourth Very Large Database Conference*, Berlin, West Germany, available from ACM, New York, September 1978, pp. 207-228.
7. Chen, P. P. S., "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems* Vol. 1, No. 1, March 1976.
8. Codd, E. F., "A Relational Model of Data For Large Shared Data Banks," *Comm. ACM* Vol. 13, No. 6, June 1970, pp. 377-387.
9. Falkenberg, E., "Concepts for Modelling Information," *Modelling in Data Base Management Systems*, North Holland, Amsterdam, 1976.
10. Falkenberg, E., "Concepts for the Coexistence Approach to Data Base Management," *International Computing Symposium 1977*, North Holland Publishing Co., Amsterdam, 1977.
11. Falkenberg, E., "Significations: The Key to Unify Data Base Management," *Information Systems* Vol. 2, No. 1, Pergamon Press, Great Britain, pp. 19-28.
12. Lin, C. S., D. C. P. Smith and J. M. Smith, "The Design of a Rotating Associative Array Processor for a Relational Data Base Management Application," *ACM TODS* Vol. 1, No. 1, March 1976, pp. 53-65.
13. Mylopoulos, J., P. A. Bernstein and H. K. T. Wong, "A Language Facility For Designing Interactive Database-Intensive Applications," *Proc. ACM-SIGMOD International Conference*, Austin, Texas, May 1978, to appear in *ACM-TODS*.
14. Navathe, S. B., "Schema Analysis for Database Restructuring," presented at the Third VLDB Conference, Tokyo, Japan, October 1977, to appear in *ACM Transactions on Database Systems*.
15. Navathe, S. B. and J. Lemke, "Concepts for the Design of an Internal Schema Using the Relational Model," (in preparation).
16. Navathe, S. B. and M. Schkolnick, "View Representation in Logical Database Design," *Proc. ACM-SIGMOD International Conference on Management of Data*, Austin, Texas, May 1976, pp. 144-156.
17. Nijssen, G. M., "A Gross Architecture For the Next Generation Data Base Management Systems," *Proc. IFIP TC-2 Working Conference on Data Base Management Systems*, Frenndstadt, January 1976.

18. Ozkarahan, E. A., S. A. Schuster and K. C. Sevcik, "Performance Evaluation of a Data Base Machine," *ACM TODS* Vol. 2, No. 4, December 1977, pp. 297-316.
19. Schuster, S. A. et. al., "RAP-2-An Associative Processor for Data Bases," *Proc. Fourth ACM-SIGARCH Workshop on Computer Architecture for Non-numeric Processing*, Blue Mountain Lake, N.Y., August 1978.
20. Senko, M. E., et. al., "Data Structuring and Accessing in Data Base Systems," *IBM Systems Journal* Vol. 12, No. 1, January 1973, pp. 30-93.
21. Smith, J. M. and D. C. P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems* Vol. 2, No. 2, June 1977.
22. Tompa, F. W., "Choosing an Efficient Internal Schema," *Systems for Large Data Bases*, P. C. Lockemann and E. J. Neuhold (editors), North Holland Publishing Co., Amsterdam, 1976, pp. 65-77.
23. Fagin, R., "Multivalued Dependencies and a New Normal Form for Relational Databases," *ACM Transactions on Database Systems* Vol. 2, No. 3, September 1977.
24. Sundgren, B., "Database design in theory and practice, toward an integrated methodology," *Proc. Fourth International Conference on Very Large Databases*, Berlin, W. Germany, September 1978.

The practice of data base administration

by JAY-LOUISE WELDON

New York University
New York, New York

The position or role of data base administrator has been described and discussed since the earliest specifications for data base management systems.^{1,2,4} Most of the literature on DBA is normative—describing in detail what the DBA function *should* include. Different authors have focused on the DBA's role in introducing the data base concept to organizations,⁸ the functions that the DBA should perform,⁷ the specification of tools and DBMS features needed by DBAs,³ and how to establish and perform the DBA's responsibilities within an organization.^{5,6} In practice, however, the organization and content of the DBA function may be quite different from the ideal. Other factors related to the DBA's technical and organizational environment may influence the DBA's role. This paper summarizes the results of an exploratory survey on data base administration⁹ which was undertaken as a first step toward identifying and understanding these factors.

METHODOLOGY

The survey included 25 DBA groups within firms located in the New York metropolitan area. Individual interviews were arranged with the manager of each DBA group. The interviews were unstructured and the questions generally open-ended. However, the interviewer did follow an interview outline that included questions on three topic areas: 1) basic characteristics of the firm, 2) organizational characteristics of the firm and of the DBA group, and 3) the tasks performed by the DBA group. The interviews were summarized immediately after they occurred and the responses were later coded according to previously-defined classification schemes.

Based on existing literature on DBA, seven characteristics were defined as independent variables, i.e. factors that were thought to influence DBA organization and content:

Age of the DBA organization	The number of years that the DBA organization/position had existed.
Origin relative to DBMS	Was DBA instituted before, with, or after the DBMS?
DBMS	Which DBMS package was used?

Industry	The industry most descriptive of the firm's business operations.
Installation size	An index combining the size of the hardware and the number of bytes stored in data bases.
EDP organization	The degree of centralization of the EDP organization.
Data base scope	An index based on the number of different application areas with current or planned data base support.

Five factors descriptive of the DBA function were also defined for use as dependent variables:

Size	Size of DBA staff with respect to total EDP staff.
Organizational position of DBA	Described by the number of levels between the DBA and the head of EDP, the position of DBA with respect to its primary users, and an indicator of organizational change since DBA's inception (higher, lower, no change).
Orientation of DBA staff	Ratings of the DBA staff on a technical-administrative scale and an application-systems scale.
DBA organizational structure	DBA's span of control and the number of organizational levels below DBA.
DBA tasks	A vector of yes-no indicators for 48 representative DBA tasks.

Responses were tabulated for each of the variables above to obtain descriptive statistics (frequency distributions, means or medians, etc.) on each. The task indicators were summed for the sample and used to rank order the DBA tasks with respect to frequency of mention.

To explore possible relationships among the independent and dependent variables, comparative statistics (Chi-square tests, t-tests, and rank-order correlations) were used. The sample was partitioned by each of the seven independent variables and values of the dependent variables for the sub-samples were compared.

RESULTS*

The nature of the survey respondents can be determined from the descriptive statistics on the independent variables. The sample was largely self-selected, and therefore not demonstrably random. However, the characteristics show a reasonably diverse group, representing a spectrum of age (one to five years), industry, installation size, and data base scope. One DBMS (IMS) dominates the sample (used by 65 percent of the respondents) as does one mode of EDP organization (75 percent were centralized).

The characteristics of the DBA groups can be summarized as follows:

- *Size*—The size of the DBA group relative to EDP staff size ranged from 0.4 percent to 12.5 percent with a mean of 3.34 percent and a standard deviation of 3.01 (actual staff sizes ranged from one to 28). If projected staff increases are included, the mean increases to 3.86 percent.
- *Organizational position*—Organizational position was of interest since most authorities claim that DBA requires a high management level for success. Most of the groups surveyed were two or more levels below the top EDP manager. The range for this variable was one to four, with only 25 percent reporting directly to the top EDP manager. Most of the DBA groups had changed their organizational position since their inception, most for the better. This suggests that DBA might start low in the organization and gain in position as the function matures. Over half of the groups, however, were placed on a *lower* organizational level than their primary users (e.g. applications programmers). This suggests that DBA is still considered a support function by most EDP groups. Only 10 percent of the DBA managers were on a higher level than the manager of their primary user group.
- *Staff Orientation*—Each DBA group was assigned a rating on two five-point scales according to the job titles or descriptions provided for their staff members. (Scale 1: 1=All administrative, no technical; 2; 3=Mixed; 4; 5=All technical. Scale 2: 1=Applications-oriented; 2; 3=Mixed; 4; 5=Systems-oriented.) The tabulation of the first scale showed a clear dichotomy, with twice as many groups rated technical as rated administrative (mean rating=3.55, standard deviation=1.5). Responses on Scale 2 showed that most groups were applications-oriented (45 percent) or mixed

(40 percent), and none were entirely systems-oriented. (Mean ratings=2.4, standard deviation=1.1).

- *Organizational Structure*—Almost all of the respondents (89 percent) had either one level (42 percent) or two levels (47 percent) of staff reporting to the DBA manager. For both flat and pyramid structures, the most common span of control was three (the range was zero to six). The most common functional designations associated with these three subgroups were: Support for data base design and maintenance, data standards (including data dictionary), and DBMS support.
- *Tasks*—The tasks performed by each DBA group were recorded on a menu of 48 representative tasks culled from the literature.** By summing across the sample for each task, the proportion of DBA groups performing each task was determined and the tasks were ranked accordingly. Table I shows the tasks most, and least, frequently mentioned.

FACTORS INFLUENCING DBA CHARACTERISTICS

Relationships were detected between each of the independent variables and one or more DBA characteristics. While some of the other cross-classifications suggested possible relationships, due to the small sample size (N=20, five responses being unacceptable for various reasons), they were not statistically significant. Except where noted, the following discussion will be limited to significant relationships ($p < .05$).

The length of time that a DBA had been in existence was found to be related to four of the five DBA characteristics. Only relative staff size was not significantly related. Inspection of the data shows, however, that the younger groups (three years or less) show more variation in size and a higher proportion of small staff sizes than the older groups.

As might be expected, the older groups show more internal structure and have experienced more organizational change than the younger groups. Younger groups also tend to be more applications-oriented, while older groups are more systems-oriented. This might be associated with the development cycle of the data base applications (i.e. younger—design, older—operational).

DBA appears to start as a technical support group for one or more applications areas (and below them in the organization). Over time DBA moves to a higher level (equal to the primary user group) functioning then in a more consultative fashion.

One interesting, though non-significant, result suggests that a change may have occurred in this pattern over the past two years. The youngest DBA groups (two years or less) reported a broad range of applications supported, similar to the oldest groups (greater than four years). This, coupled with the fact that none of the young groups supported only one application area, suggests that DBA may now be avoiding the project approach, i.e. supporting only one application area, which was common in the past.

* An extended discussion of the survey results and supporting data can be found in another paper by the author.⁹

** The full list of tasks is shown in the Appendix.

TABLE I—DBA Tasks Mentioned by Survey Respondents

<i>Most Frequently Performed</i>		<i>Least Frequently Performed</i>	
<i>Task</i>	<i>%</i>	<i>Task</i>	<i>%</i>
Maintain Data Dictionary System	75	Select DB-related hardware	0
Evaluate DB software	60	Enforce DB retention policies	0
Maintain DB descriptions	60	Requirement analysis for DB applications	5
Reorganize DBs	60	Schedule computer time	5
Recover DBs	60	Determine program structure	10
Select DB software	55	Evaluate DB-related hardware	15
Forecast DB growth	55	Design forms/procedures for DB applications	15
Generate DB descriptions	55	Enforce standards for application coding	15
Monitor DB performance	55	Maintain DC monitor	15
Develop standards for data element names	55	Enforce standards for documentation	20
Determine physical structures	50	Educate DB users	20
Specify DB access policies	50	Set application priorities	20
Load DBs	50		
Set policies on DB backup and recovery	50		

DBA groups established before a DBMS was installed tend to be larger (in relative size) than those instituted with or after the DBMS. This, however, may be indirectly related to age, since groups started before the DBMS are older than most of the groups started with or after the DBMS.

Installation size was found to be related to the level of the DBA with respect to the top EDP manager and also to the amount of organization change experienced by the DBA. DBAs in organizations with large installations (large model hardware and large data bases) tend to be lower in the EDP organization than those in smaller installations. These DBAs also reported more organizational change (both positive and negative) in the DBA group than did DBAs at smaller installations.

The relationship between the DBMS package used and DBA characteristics was significant for only two variables: DBA's span of control and DBA tasks. Both of these results compared IMS DBAs with DBAs using other DBMS packages. IMS DBAs were found to have larger span of control, i.e. more persons reporting to them, than the other DBAs. Further, the ranks assigned to DBA tasks by the IMS DBAs were found to be inversely correlated with those assigned by the other DBAs. The IMS DBAs emphasize tasks related to planning and control (forecast growth, data dictionary, data name standards) while the others emphasize operational support tasks (e.g. data base load, troubleshooting).

In contrast with the sparse relationships detected between DBA organizational characteristics and the independent variables, the content of the DBA function was related to all but one. Differences in the rank order of tasks performed were detected among groups partitioned by every independent variable, except installation size. For a fuller discussion, see Reference 9.

CONCLUSIONS

The results of this survey support two major conclusions. First, the organizational aspects of data base administration groups are affected primarily by the length of time that the

group has existed. This suggests a maturation process for DBA, perhaps similar to Nolan's stages of EDP growth. To formulate such a hypothesis, criteria defining each stage in the process must be specified and characteristics of DBA organization and structure related to each stage.

Second, it appears that the tasks performed by DBA groups vary independently from organizational structure or position. The composition of a DBA's job and the rank ordering of tasks within it are influenced by several factors, including the DBMS package used, EDP organization type, and the scope of data base applications. Additional task data from stratified random samples for each of the related variables could be used to explore these relationships in greater detail. It may then be possible to develop task profiles for the several different types of data base administration functions that exist in practice.

APPENDIX—CLASSIFICATION OF DBA TASKS

Planning and Management

- Evaluate data base software (e.g. DBMS, DDDS)
- Select data base software
- Evaluate data base-related hardware (e.g. disks, terminals)
- Select data base-related hardware
- Define implementation strategy for data bases
- Forecast data base growth
- Set operational goals (performance, downtime)
- Set application priorities
- Hire, fire, promote data base personnel

Data Base Design

- Requirement analysis for data base applications (data identification)
- Develop data definitions
- Determine data structures (views)
- Determine physical structures
- Generate data base descriptions (DDL)
- Design integrity controls for data base applications
- Design forms and procedures for data base applications

Application Programming/Testing

Determine program structures (processes)
 Develop standards for application coding
 Enforce standards for application coding
 Application system testing

Controls

Generate data base descriptions (DDL)
 Maintain data base descriptions
 Design integrity controls for data base applications
 Specify data base access policies
 Record data base usage
 Monitor data base controls
 Develop standards for data names
 Develop standards for application coding
 Develop standards for documentation
 Maintain DDDS
 Set data base retention policies
 Select/design security techniques (passwords, locks, etc.)

Operational Support

Monitor data base controls
 Load data bases
 Reorganize data bases
 Recover data bases
 Monitor data base performance
 Tune data base to meet operational goals
 Troubleshooting (i.e. track down problems)
 Enforce standards for data element naming
 Enforce standards for application coding
 Enforce standards for documentation
 Set policies on data base backup and recovery
 Schedule computer time (e.g. when data base is up)
 Develop data base utilities (data compression, encryption, query languages)
 Enforce data base retention policies

DBMS Support

Install new DBMS features
 Modify or fix DBMS
 Maintain data communications monitor
 Develop data base utilities (data compression, encryption, query languages)

User Support

Prepare data base documentation
 Disseminate data base documentation
 Educate data base users
 Maintain DDDS
 Maintain query language(s)

REFERENCES

1. Canning, R. G., "The 'data administrator' function," *EDP Analyzer*, Vol. 10, No. 11, November 1972, pp. 1-14.
2. CODASYL Systems Committee, "Data Administration Functions," Chapter 8 of *Feature Analysis of Generalized Data Base Management Systems*, May, 1971 (available from ACM, 1133 Avenue of the Americas, New York, New York 10026).
3. CODASYL/D.B.C., D.B.A. Working Group, *June 1975 Report*, The British Computer Society.
4. GUIDE Data Base Administration Project, "The Data Base Administrator," 1973 (available from GUIDE International, 111 East Wacker Drive, Chicago, Illinois 60601).
5. GUIDE, "Establishing the Data Administration Function," report of the EDAF Project, June 1977 (available from GUIDE International).
6. GUIDE, "Data Administration Methodology," report of the Data Administration Methodology Project, January, 1978 (available from GUIDE International).
7. Lyon, John K., *The Database Administrator*, John Wiley & Sons, 1976.
8. Nolan, Richard L., "Data Base—An emerging organizational function," *National Computer Conference*, 1974, pp. 897-901.
9. Weldon, J. L., "Data Base Administration—Organization and Tasks," New York University, Graduate School of Business Administration, Working Paper #78-143(CA), December 1978.

An approach to automatic maintenance of semantic integrity in large design data bases

by GILLES M. E. LAFUE

Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

One way a data base can cease retaining a meaningful relationship with the "real world" situation that it models is through violations of its *semantic integrity*, i.e., transgressions of the defining constraints of its data. These constraints state the legality of the data values and are defined by the creators of the data, e.g., a checking account cannot be negative, or the salary of an assistant professor cannot be greater than that of a full professor.

The variety of integrity constraints to be maintained can be extremely wide. It can range from general low-level management such as dangling references to complex knowledge specific to the "real world" situation, e.g., in the case of a building, detection of spatial conflicts or even statics and mechanics.

Many of the debates about the different data models proposed in the past few years center around the extent of the semantics that these various models hold. Whatever data model one can use nowadays, there will always be semantics which escape it. In daily practice, these semantics are often maintained by the application programs or the manual users. This forces them to maintain information about the data base in a form external to it (a data base about the data base) when in fact it is conceptually part of the data base itself. Incorporating this external information in the data base would be an extension of the effort which led in the first place to modeling a "real world" situation with a computer data base. Moreover, interaction with the data base would be improved by automating the maintenance of these semantics.

This need exists for any kind of data base but it is especially pressing for what I call *design data bases*.⁴ A design data base not only stores a model of some artifact and provides primitive accesses to it (e.g., query processing) but also supports the design of this model with the hesitations and backtracking usually involved, e.g., the *schema* (the set of data types) is continuously modified. The design of, and the interaction with, such a system are greatly helped by the data and control abstraction features of a high-level programming language. GLIDE² is an example of a design data base. The integration of language and data base concepts for

design applications and its relationships with the principles presented here are discussed in Reference 4.

Unary integrity constraints apply to a single record and *n-ary* constraints apply to several records. The proposed approach starts by considering an *n-ary* integrity constraint as composed of one set of *dependent* variables and one set of *independent* variables. Variables can be both dependent and independent. The value of a dependent variable is partially or entirely determined by the values of the independent ones. Maintaining an *n-ary* integrity constraint consists in recomputing or checking the values of its dependent variables when the independent ones change.

The variables of an *intra-record* integrity constraint are *attributes* of the same record. A record is a collection of data logically related and stored together. The variables of an *inter-record* constraint are attributes of several records. These records can be instances of the same type or of different types.

The distinction of constraints according to the number of records they involve is important from two viewpoints. First, the record types and the operations performed on them determine the modularity of the data base schema and of the "contexts" in which the applications execute. Secondly, records belonging to the same integrity constraint are rarely in core at the same time and, in a traditional computer architecture, the cost of disk access is high. The emphasis of this approach is on inter-record constraints.

This paper presents three basic principles of an approach to automatic maintenance of semantic integrity in large design data bases. First, the maintenance of integrity is delayed until strictly necessary. Second, integrity violations are temporarily tolerated. Third, integrity constraints are procedures included in the record definitions and automatically activated by the system. These principles are generally not supported by the data base systems attempting elaborated automatic maintenance of integrity, in particular INGRES⁷ and System R.¹ A mechanism based on these principles is currently under investigation.

DELAYED MAINTENANCE OF INTEGRITY

An integrity constraint is to be maintained when one or more of its independents has been updated. This mainte-

nance consists of propagating the updates from the independents to the dependents. Scheduling this propagation requires knowing which records are the dependents, and also something of the ways these dependents use the independents in order to decide when to notify them and in what order. This propagation can be scheduled either by the records themselves or by the system.

The records are not the best qualified to schedule this propagation. As noted by Hammer,³ the dependents of a record are often defined long after the record itself. Records should not have to anticipate their users and their uses. They should not even have to maintain the knowledge of their existing uses in order to confine their view of the world to themselves, thus reinforcing the modularity of the schema.

Making the system schedule the propagation of updates has important costs, too. The propagation must be scheduled relative to other tasks. The scheduling policy must be general enough to fit all applications or, at least, to fit most applications and offer the others a way to override it.

A simple and general policy is immediate propagation of updates, which consists of giving priority to propagation over everything else. It is the policy of INGRES. However, it often means that control leaves the context which introduced the update to wander through the data base. This happens when a record is used in different contexts. For instance, in a building, a post can be used for structural purposes as well as for running pipes and wires. When updating the post for structural reasons, it would be long and difficult to consider simultaneously the effects of this update on the piping and the wiring systems. Often, accesses of overlapping contexts must be serialized. Furthermore, the disk accesses to the remote dependent records can drastically slow down the operation of writing in records.

Other scheduling policies consisting of alternating propagation and other tasks would require well defined priorities between these tasks and propagation. They would require spreading some knowledge of record dependencies and uses out of the records. This knowledge should remain associated with the records, even if not necessarily with the independent ones.

The proposition, then, is to delay the propagation of updates until strictly necessary, i.e., when the dependents are accessed, for whatever reason, including checking their external integrity. The integrity of a record is more important when the record is about to be used than when it resides on disk.

TOLERANCE OF INTEGRITY VIOLATIONS

A consequence of delayed maintenance of integrity is that the data base must be able to tolerate violations of its integrity, at least temporarily. Between the time an independent record is updated and the time a record depending on this record is accessed, the constraint that links these two records is left unenforced. Furthermore, the independent update may imply a dependent value which is illegal because of another constraint applying on the dependent. Violations

are tolerable only as long as they, or their causes, i.e., the updates, are recorded. In contrast, in System R and INGRES, updates leading to violations are immediately rejected.

Sometimes, tolerance of violations is not only an acceptable consequence but is desirable. Integrity constraints may apply at some moments and not at others. This is particularly true for design data bases. One advantage of using a model to design a complex artifact is that the constraints for making the model are different, and hopefully more manageable, than those for making the artifact. Some constraints pertaining to the artifact may be ignored for convenience, until the end of a design phase. Such a phase defines an *integrity transaction* for the constraints temporarily dropped and the records they concern.

In the current approach, integrity transactions are delimited by (dependent) record openings. Opening a record ends the transaction for the record and the constraints in which it is a dependent. If it is desired to open a record without maintaining one of its constraints, this constraint can be associated with another record. This other record is opened when the constraint needs to be checked. The purpose of some records may be to implement the external integrity of other records. Such records are typically *aggregation abstractions* as defined by Smith and Smith.⁶ They are the dependents of the aggregations and the records whose integrity they implement are the independents. For instance, aggregations are useful for centralizing the management of circular dependencies.

In the cases described so far, violations are tolerated until they are detected. Now, tolerating and recording a violation may be the action to take when the violation is detected, in order to postpone its resolution. For instance, in a transaction during which an integrity constraint does not apply, it may be useful to record the violations of this constraint on the fly, so that they can be readily taken care of at the end of the transaction, instead of being recomputed. Recording violations is also useful when a constraint possesses several independent variables and is violated by the update of one of them. This violation can be notified to the other independent variables. Often, at least one can change its value in order to accommodate the requested update. For instance, if several persons share a checking account, one person can overdraw the account and another one compensate the overdraw.

Simultaneous alternative values for variables can be assimilated to violations of the data base integrity. They indicate some indecision as to what the unique values of the variables are. They are often useful to tolerate temporarily, especially in design activities, due to the hesitant nature of these activities.

Alternatives may be generated by the same process or by different ones. Regarding the latter case, the principles of resource protection and concurrency control, traditionally used in operating systems, are necessary but not sufficient for data bases. Locking mechanisms serialize concurrent accesses. However, since writing in a data base is to leave a more or less permanent mark, the question of whether the authorized processes write in a record serially or in parallel,

is immaterial. What is important is that overwriting each other may lead the processes into conflicts which cannot necessarily be solved as soon as they occur.

INTEGRITY CONSTRAINTS INCLUDED IN THE RECORDS AND AUTOMATICALLY MAINTAINED

Since all the semantics usually desired in a data base cannot be incorporated in the data structures, one has to resort to executable code. This code should be of a high level in order to maximize the power of integrity constraints.

The code implementing the integrity constraints of a record can be included in the record definition in the manner of abstract data types.⁵ It participates in defining the record semantics, particularly, its behavior, as much as data structures. This code should be pre-compiled in order to avoid recompiling it every time it is executed. Integrity constraints can then be implemented by pre-compiled procedures called *integrity procedures*.

Clearly, including integrity constraints in the records is acceptable for intra-record constraints since it confines them to the concerned records. It also holds for inter-record constraints. While the records should not know their uses, they must know the records on which they depend or, rather, abstractions of these records. Consequently, integrity procedures are included in the dependent records. This inclusion respects abstraction boundaries. Integrity procedures write in the dependent records or check their values, and they simply read the independent ones.

While the usefulness of the notion of abstract record type in data bases has been acknowledged by several authors, e.g. Reference 6, others have objected to it. In particular, Hammer³ rightly points out that the behavior and the uses of a record type evolve over time. The proposal to exclude the uses of a record from its definition reduces this evolution significantly.

This procedural approach is different from one which guarantees integrity by providing procedures for accessing the data base. Such procedures guarantee that the data base transits from one valid state to another. They implement *state transition* integrity. State transition procedures enforce immediate maintenance of integrity and do not tolerate violations.

The inclusion of integrity constraints not only in the data base but in the records, contrasts with other approaches. The separation of integrity constraints from the records, as in INGRES and System R, makes insertions and deletions of integrity constraints easier and it centralizes the detection

of conflicts between them. The cost of this separation, however, is to spread the definition of the records.

The mode of activating integrity checks, i.e., integrity procedures, remains to be examined. There are two kinds of integrity checks. Some are implicit. The user does not want to have to activate them explicitly all the time. They can be activated automatically whenever necessary, i.e., when the records are accessed. The others are explicit. They take place at the end of a user-defined transaction which can span several record accesses. A unified scheme is proposed which satisfies both sorts of integrity checks.

Integrity procedures should be written by the user and compiled in relation with record declarations (either types or instances), but automatically invoked by the system upon well defined conditions. These conditions are data base operations, e.g., record reads and writes. System R possesses a similar triggering mechanism which activates the execution of statements of a query language.

For implicit checks, integrity procedures are automatically activated every time their records, supposedly the dependent records of the constraints, are accessed. As for explicit checks, opening a record which implements integrity constraints of other records marks the end of a user-programmed transaction.

ACKNOWLEDGMENTS

Thanks are due to Charles Eastman, Kevin Weiler, Molly Meigs and to the NCC referees for their helpful comments.

REFERENCES

1. Astrahan, M. M., et al., "System R: Relational Approach to Database Management" *ACM Transactions on Database Management* Vol. 1, No. 2, June 1976.
2. Birnbaum, M., C. M. Eastman, M. Henrion, G. M. E. Lafue, R. W. Thornton and K. J. Weiler, "GLIDE Reference Manual," Institute of Physical Planning Research Report, Carnegie-Mellon University, Oct. 1978.
3. Hammer, M. M., "Data Abstractions for Databases," *SIGPLAN Notices*, Vol. 8, No. 2, March 1976.
4. Lafue, G. M. E., "Integrating Language and Database for CAD Applications," to be published in the *Computer-Aided Design* journal.
5. Liskov, B., S. Zilles, "Programming with Abstract Data Types," *SIGPLAN Notices*, Vol. 9, No. 4, April 1974.
6. Smith, J. M., and D. C. P. Smith, "Database Abstractions: Aggregation," *Communications of the A.C.M.* Vol. 20, No. 6, June 1977.
7. Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification," *Proc. ACM/SIGMOD Conf. on Management of Data*, San Jose, Ca., May 1975.

On query-answering in relational data bases

by E. L. LOZINSKII*

Rutgers University
New Brunswick, New Jersey

INTRODUCTION

In recent years the relational model has been widely adopted for data base description. According to this model a data base, DB, describes certain objects of the world having certain attributes, and the relationships among them. Thus, DB is characterized by a set of *attributes*, D , a set of *domains* associated with the attributes, and a set of *dependencies*, F , corresponding to the relationships among the attributes (all the terms and concepts not defined here are those of References 1-3). A data base is a collection of *relations*, $R = \{R_i\}$. Each relation R_i is characterized by a set of attributes $S_i = \{D_j | D_j \in D\}$ called its *scheme*, and consists of a set of *tuples*. Each tuple is a map from the attributes of the relation scheme to their domains that satisfies all the dependencies of F (we shall consider functional dependencies).

We shall consider the following model of a data base operation. A data base represented by a collection of relations R is accessed by a set of jobs submitted by users in order to retrieve, delete, insert or modify any subset of the data. In general, retrieval is an essential part of these processes. Thus, each job accessing a data base requires retrieval of some information from it. Users submit their requests for information in the form of queries, specifying the data which must be retrieved and the conditions that must be satisfied by the desired data. The task of query processing is to determine the set of data to be checked and retrieved from the data base, the proper order in which the data should be accessed and the types of manipulations that must be performed on the data.⁴ This processing is referred to by different authors as query translation⁵ or access path finding.^{4,6,7}

Let a query require a retrieval of data which belong to a set of attributes D' . It can be shown that such a query may be answered in a certain relational data base only if from the same data base a relation can be derived which contains all the attributes of D' specified by the query. Let us call *R-query* a query that requires a creation of relation with a given scheme. *R-queries* originate not only from users' requests but can also be initiated by a data base management system (a data base administrator) as a means of relations

transformation in the process of data base evolution and restructuring.

Let a query be answerable by means of a procedure which contains a sequence of relational operations project, join, divide and restrict.⁸ Among these operations, the join operation is subject to a number of strong constraints. Specifically, an execution of a join operation on two relations R_1 and R_2 with the purpose of creating a new relation $R_3 = R_1 * R_2$ can produce invalid data in the sense that R_3 can contain a tuple which does not satisfy the original set of dependencies F . In order to avoid the appearance of invalid data, the condition of a *lossless join*⁹⁻¹² must be satisfied. Thus, we shall concentrate upon join operations of a query-answering procedure, assuming that all the relations produced by joins are properly projected and restricted.

So, considering a collection of relations, R , and a R -query that requires a relation R_k with a scheme S_k , the following questions arise:

1. Can R_k be derived from R ?
2. Which relations should be joined to produce R_k from R ?
3. What is the sequence of join operations (optimal according to the accepted criteria) that produces R_k ?

The algorithm described in the fourth section answers these questions.

LOSSLESS JOINS

As was shown by J. Rissanen,¹⁰ a relation R_k with a scheme S_k can be produced by means of join and project operations from a set of relations $R = \{R_i\}$ (such that $S_k \subseteq \cup S_i | R_i \in R$) if these relations satisfy the following *conditions for lossless join* given in References 9-12.

Let us say that a set of attributes B functionally depends on a set of attributes A (abbr. $A \rightarrow B$) if there is a functional dependency $f \in F$ such that $f: A \rightarrow B$. We shall use the concept of *closure*, $CL(X)$, of a set of attributes X defined in Reference 11 as follows:

1. $X \subseteq CL(X)$
2. If $Y \subseteq CL(X)$ and there is $f \in F$ such that $f: Y \rightarrow Z$, then $Z \subseteq CL(X)$

* On leave from the Department of Computer Science, Hebrew University, Jerusalem, Israel.

- No attribute is in $CL(X)$ unless it so follows from (1) and (2). $X \xrightarrow{*} Y$ denotes the fact that $Y \subseteq CL(X)$.

It can be shown that if $X \xrightarrow{*} Y$, then the functional dependency $X \rightarrow Y$ is in F or can be derived from F using a set of inference rules based on the axioms given by W. Armstrong.¹³ So, the closure of X is the union of all the sets of attributes dependent on X according to the adopted inference rules.

A join of relations R_i and R_j is *lossless* (that is, it does not produce any invalid information) iff $S_i \cap S_j \xrightarrow{*} S_i$ or $S_i \cap S_j \xrightarrow{*} S_j$.

As an example (taken from Reference 11), let us consider a data base characterized by the attributes NAME (N), ADDRESS (A), PHONE NUMBER (P), DISTRICT (D), under the given set of dependencies $F = \{N \rightarrow A, N \rightarrow P, A \rightarrow D, P \rightarrow D\}$, and represented by the following set of relations (the primary keys of relations are underlined): $R = \{R_1 = (\underline{NA}), R_2 = (\underline{NP}), R_3 = (\underline{AD}), R_4 = (\underline{PD})\}$.

By the definition of join operation,^{1,2,8} two relations, R_i and R_j , may be joined if they contain common or comparable sets of attributes, A and B , such that $A \subseteq S_i$ and $B \subseteq S_j$. So, four different joins can be executed on the sample relations of R (common attributes are indicated in parentheses): $R_{12} = R_1 * R_2 (N)$, $R_{13} = R_1 * R_3 (A)$, $R_{24} = R_2 * R_4 (P)$, $R_{34} = R_3 * R_4 (D)$. The three former joins satisfy the condition for losslessness, but the latter one does not, because $CL(N) = \{N, A, P, D\}$, $CL(A) = \{A, D\}$, $CL(P) = \{P, D\}$, $CL(D) = \{D\}$, so neither $\{A, D\}$ nor $\{P, D\}$ belongs to $CL(D)$. Relation R_{34} , indeed, can contain invalid data, because within a district it relates each phone number to arbitrary addresses not considering that the related phone number and address must belong to the same person name.

Suppose that a relation R' with a scheme S' required by a R -query can be produced losslessly by a sequence of join operations on a set of relations $\{R_i\}$. Let us denote the operands of a join operation R_l (*left relation*) and R_r (*right relation*), such that if the join is lossless, then without any loss of generality $S_l \cap S_r \xrightarrow{*} S_r$. A join $R_l * R_r$ produces a relation R_{lr} with a scheme $S_{lr} = S_l \cup S_r$. There is a subset of attributes $K_l \subseteq S_l$ (called a key of R_l), such that

$$K_l \xrightarrow{*} S_l \tag{1}$$

By Armstrong's axioms,¹³ $K_l \xrightarrow{*} S_{lr}$, i.e. a key of the left relation is also a key of the product. This implies the following property of sequences of lossless join operations:

If a relation R' with a scheme S' is produced losslessly by a sequence of join operations from a set of initial relations $\{R_i\}$ such that $S' \subseteq \cup S_i$, then there is a relation $R_j \in \{R_i\}$, called a *source relation* of R' , such that S' functionally depends on each key, K , of R_j . (Proofs are omitted here for brevity).

GRAPH REPRESENTATION

A given set of functional dependencies, F , can be represented by an AND/OR graph¹⁴ in the following way. Let $G = (V, W, E_1, E_2)$ denote an AND/OR graph, where V is

a set of AND-nodes and terminal nodes, W is a set of OR-nodes, E_1 is a set of AND-arcs (the arcs going out of AND-nodes), E_2 is a set of OR-arcs (the arcs going out of OR-nodes). We shall say that an AND/OR graph G , called an F -graph, displays a given set of dependencies $F = \{f_i\}$ iff for each $f_i \in F$ such that $f_i: A_i \rightarrow B_i$ (A_i and B_i are sets of attributes) all of the following hold:

- $V \supset \{v_j \mid D_j \in (A_i \cup B_i)\}$, where v_j denotes the node displaying D_j
- $w_i \in W$
- $E_1 \supset \{(v_k, w_i) \mid D_k \in A_i\}$
- $E_2 \supset \{(w_i, v_m) \mid D_m \in B_i\}$
- There are neither other nodes nor other arcs in G .

In words, V contains nodes, called v -nodes, corresponding to all the attributes appearing in F ($D(v_i)$ denotes the attribute displayed by v_i , $D(v_i) = D_i$); W contains nodes, called w -nodes, corresponding to all the dependencies appearing in F ($f(w_j)$ denotes the dependency displayed by w_j , $f(w_j) = f_j$); for each distinct left-side set of attributes A_i (of a dependency f_i) there is a w -node, $w_i \in W$, which accepts incoming arcs from all the v -nodes corresponding to A_i and emits outgoing arcs to all the v -nodes corresponding to B_i .

For example, Figure 1 shows a F -graph which displays the following set of dependencies: $F = \{D_1 D_2 D_3 \rightarrow D_6 D_7, D_4 D_5 \rightarrow D_7, D_6 \rightarrow D_9, D_7 \rightarrow D_9, D_7 D_8 \rightarrow D_{10}\}$. The AND-arcs going to the same w -node are linked by a bow, w -nodes are marked by a double circle.

Let a collection of relations $R = \{R_i\}$ represent a DB with a given F . Because our aim is to construct a requested relation from $\{R_i\}$ we supplement the F -graph $G = (V, W, E_1, E_2)$ with the following information corresponding to the relations of $\{R_i\}$: for each R_i add to G a node u_i , called a u -node, such that if K is one of the keys of R_i , then u_i

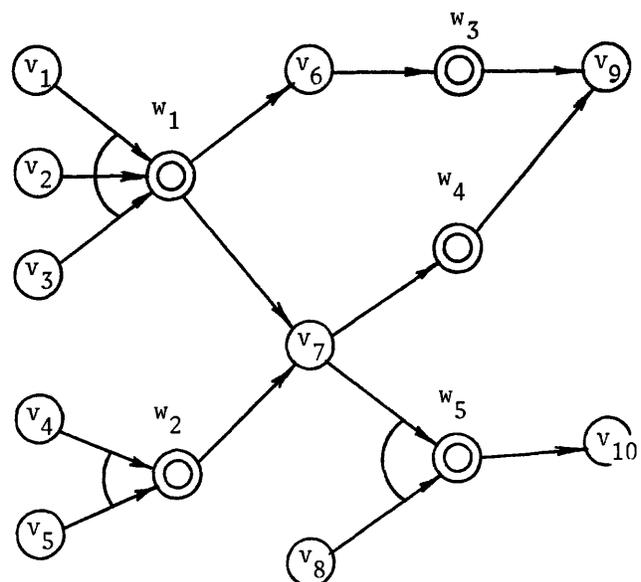


Figure 1

emits outgoing arcs to all the nodes of G displaying attributes of K . R_i is displayed by u_i ; K is said to be tied by u_i .

We shall say that an AND/OR graph $H=(U, V, W, E_1, E_2, E_3)$, called an R -graph, displays a given representation R of a DB with a given F , iff all of the following hold:

1. V, W, E_1 and E_2 are those of the F -graph G
2. $U=\{u_i | R_i \in R\}$, u_i denotes the node displaying R_i
3. $E_3 \supset \{(u_i, v_k) | D_k \in K_i\}$, K_i denotes a key of R_i
4. There are neither other nodes nor other arcs in H .

As an example, Figure 2 shows a R -graph displaying the following representation of a DB having the F -graph of Figure 1: $R_1=(D_1D_2D_3D_6D_7D_9)$, $R_2=(D_4D_5D_7)$, $R_3=(D_7D_8D_9D_{10})$.

Let us say that a node $v_i \in V$ in $H=(U, V, W, E_1, E_2, E_3)$ is reachable from a set of nodes $\tilde{V} \subseteq V$ (abbr. $\tilde{V} \xrightarrow{*} v_i$) if the following holds:

- a. $v_i \in \tilde{V}$, or
- b. There is a node w_j preceding v_i , $(w_j, v_i) \in E_2$, s.t. all the nodes preceding w_j , $\{v_k | (v_k, w_j) \in E_1\}$, are reachable from \tilde{V} .

Because each dependency of F is displayed by the corresponding w -node of H , reachability is equivalent to belonging to a closure in the same sense as H is equivalent to F . This implies the following:

- (i) A node $v_i \in V$ is reachable from a set of nodes $\tilde{V} \subseteq V$ iff $D(\tilde{V}) \xrightarrow{*} D(v_i)$.
- (ii) A node $v_i \in V$ is reachable from a node $u_j \in U$ if u_j ties a set of nodes $\tilde{V} \subseteq V$ such that $\tilde{V} \xrightarrow{*} v_i$.
- (iii) If a relation R' with a scheme S' can be produced losslessly from a set of relations $\{R_i\}$, then a R -graph H which displays $\{R_i\}$ contains a u -node, u_s , called a source node of S' , from which all nodes $\{v_i | D_i \in S'\}$ displaying the attributes of S' are reachable. Indeed, let R_m be a source relation of R' , then the node u_m displaying R_m in H is a source node of S' .

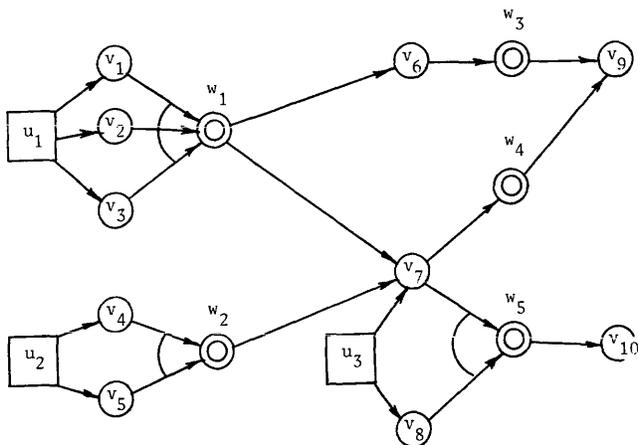


Figure 2

JOIN PROCEDURE FOR R-QUERY

If a relation R' can be produced from a set of relations $R=\{R_i\}$ there is, in general, a number of different sequences of join operations producing R' . For example, if a data base having the F -graph shown in Figure 3 is represented by the relations $R_1=(D_1D_2)$, $R_2=(D_1D_3)$, $R_3=(D_2D_4)$, then a relation $R'=(D_1D_3D_4)$ can be produced by the following join sequences:

- a. $J_1: R'_1=R_1 * R_2(D_1), R'_1=(D_1D_2D_3);$
 $J_2: R'=R'_1 * R_3(D_2), R'=(D_1D_3D_4).$
- b. $J_1: R'_1=R_1 * R_3(D_2), R'_1=(D_1D_2D_4);$
 $J_2: R'=R'_1 * R_2(D_1), R'=(D_1D_3D_4).$

In these sequences we do not care about project operations which should appear in practical procedures. For instance, in Sequence b the relation R'_1 probably should be projected in (D_1D_4) before J_2 . We assume that relations R'_i produced by joins are properly projected.

One of the join sequences producing R' should be preferred to others according to certain criteria, one of which, in particular, is the total processing cost, $CJ=\sum c(J_i)$, where $c(J_i)$ is the processing cost of J_i . The processing cost of a join operation depends on the structure, composition and location of the joined relations and of the product as well. We assume that each relation R_i is assigned an access cost $c(R_i)$ (say, proportional to its physical size).

Our goal is to construct an efficient sequence of joins $J(R')=\{J_1, J_2, \dots\}$ which produces a relation R' (with a scheme S' , required by a R -query) from a given set of relations R representing a data base with a R -graph H . The features of lossless joins of relations and the graph representation described in the second and third sections provide a basis for an algorithm which performs this task in a number of stages as follows.

- A. Let X be a target set, that is a set of nodes displaying the attributes required by a given R -query, such that $X=\{x | D(x) \in S'\}$. At this stage all the source nodes of X (from which all the nodes of X are reachable) are found in H , constituting a set U_s . If U_s is empty then, by (iii), the requested relation cannot be constructed according to the given data. The algorithm used at this stage is a graph equivalent of the Membership algorithm presented in Reference 15.

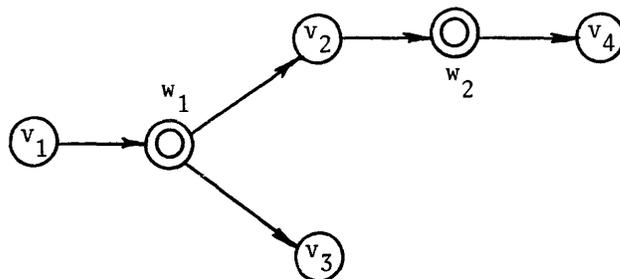


Figure 3

- B. Let us consider a u -node u_i , a v -node y and a set of relations $P(u_i, y)$ such that a relation R' with a scheme $S' = D(K_i) \cup \{D(y)\}$ can be produced from $P(u_i, y)$ but cannot be produced from a subset of $P(u_i, y)$. We shall call $P(u_i, y)$ a *join path* from u_i to y . $CP(u_i, y)$ stands for the cost of $P(u_i, y)$ such that $CP(u_i, y) = \sum c(R_j) | R_j \in P(u_i, y)$. $P(u_i, Y)$ denotes a join path from u_i to a set of nodes Y , such that $P(u_i, Y) = \cup P(u_i, y) | y \in Y$. Stage *B* finds a join path from $u_i \in U_s$ to a given target set X which has the minimum join path cost among all the join paths from u_i to X found by the algorithm. This stage is based on Dijkstra's algorithm for the shortest path in a graph.¹⁶
- C. As was mentioned previously, there is, in general, a number of different *join procedures* (i.e. sequences of join operations), $J(u_i, X)$ producing R' from $P(u_i, X)$. Let $J^o(u_i, X)$ denote the *optimal join procedure* that has the minimum total processing cost, $CJ^o(u_i, X) = \min CJ(u_i, X)$. At Stage *C* for all the source nodes $u_i \in U_s$, an optimal join procedure $J^o(u_i, X)$ is built. This stage is based on Huffman's algorithm¹⁷ for finding a tree with minimum weighted path length. Then among all these join procedures the algorithm chooses the one, $J(R')$, with the lowest processing cost, such that $CJ(R') = \min CJ^o(u_i, X) | u_i \in U_s$. If R' cannot be produced from the given data, the algorithm returns $J(R') = \phi$.

COMPLEXITY ESTIMATION

Let τ stand for a *measure of complexity* (say, the running time) of an algorithm, and let \bar{F} denote the number of all appearances of attributes in all the dependencies of F (if all the dependencies belonging to F are written down as a sequence of names of attributes appearing in them, then \bar{F} is the length of this sequence). Let $|R|$ denote the cardinality of R .

The analysis of the algorithm of the fourth section gives the following estimation of the complexity of its stages:

$$\tau(A) = 0(|R| \cdot \bar{F}),$$

$$\tau(B) = 0(|R|^3 \bar{F}),$$

$$\tau(C) = 0(|R|^3).$$

Thus, the complexity of the algorithm finding an efficient

join procedure for a given query is polynomial,

$$\tau = 0(|R|^3 \bar{F})$$

(cf. an exponential-time algorithm given in Reference 12).

ACKNOWLEDGMENTS

The author is grateful to his anonymous referees for helpful comments on a draft of this paper.

REFERENCES

1. Codd, E. F., "A relational model of data for large shared data banks," *CACM*, Vol. 13, No. 6, 1970, pp. 377-387.
2. Codd, E. F., "Further normalization of the data base relational model," in *Data Base Systems, Courant Computer Science Symposium 6*, R. Rustin (ed.), Prentice-Hall, 1972, pp. 33-64.
3. Codd, E. F., "Recent investigations in relational data base systems," *Proc. IFIP 74*, North-Holland, 1974, pp. 1017-1021.
4. Ghosh, S. P., *Data Base Organization for Data Management*, Academic Press, 1977.
5. Chang, S. K., and J. S. Ke, "Database skeleton and its application to fuzzy query translation," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 1, Jan. 1978, pp. 31-44.
6. Senko, M. E., "Data Structures and Data Accessing in Data Base Systems: Past, Present, Future," *IBM System J.*, Vol. 16, No. 3, 1977, pp. 208-257.
7. Carlson, C. R., and R. S. Kaplan, "A generalized access path model and its application to a relational data base system," *Proc. of ACM-SIGMOD International Conference on Management of Data*, J. B. Rothnie (ed.), June 1976, pp. 143-154.
8. Codd, E. F., "Relational completeness of data base sublanguages," in *Data Base Systems*, R. Rustin (ed.), *Courant Computer Science Symposium 6*, Prentice-Hall, 1972, pp. 65-98.
9. Delobel, C., and R. G. Casey, "Decomposition of a data base and the theory of Boolean switching functions," *IBM J. of Res. and Dev.*, Vol. 17, No. 3, Sept. 1973.
10. Rissanen, J., "Independent components of relations," *ACM Trans. on Data Base Systems*, Vol. 2, No. 4, Dec. 1977, pp. 317-325.
11. Aho, A. V., C. Beeri and J. D. Ullman, "The theory of joins in relational data bases," *Proc. 18th IEEE Symp. on Foundations of Computer Science*, 1977.
12. Schenk, K. L., and J. R. Pinkert, "An algorithm for servicing multi-relational queries," *Proc. of ACM-SIGMOD International Conference on Management of Data*, Toronto, 1977, pp. 10-20.
13. Armstrong, W. W., "Dependency structures of data base relationships," *Information Processing 74*, North-Holland, Amsterdam, 1974, pp. 580-583.
14. Nilsson, N. J., *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, 1971.
15. Bernstein, P. A., and C. Beeri, "An algorithmic approach to normalization of relational data base schemes," Computer Systems Research Group, University of Toronto, Technical Report CSRG-73, 1976.
16. Dijkstra, E. W., "A note on two problems in connection with graphs," *Numer. Math.*, Vol. 1, 1959, pp. 269-271.
17. Huffman, D. A., *Proc. IRE*, Vol. 40, 1952, pp. 1098-1101.

ASTROL—An associative structure-oriented language

by JAMES F. WIRTH

East Carolina University
Greenville, North Carolina

INTRODUCTION

The language ASTROL resulted from the search for a "small" language somewhat like LISP 1.5 which could be easily implemented on a minicomputer with about 32K bytes of store. The LISP cell was considered to be an example of the record—an object whose structure is specified by a set of field descriptors. However, the set of descriptors is often allocated at compile time, whereas it would be useful to be able to add descriptors to a record dynamically.

The essential information requirement for a record is the ability to associate values to certain combinations of record and descriptor—a task for an associative memory. Consequently, the author determined to create a conversational language whose structure was implicitly bound to the facilities provided by an associative memory—an Associative Structure-Oriented Language. This was a conceptual experiment to discover what ideas would arise in the attempt to adapt traditional programming constructs to such an environment.

FUNDAMENTAL CONSTRUCTS

The kind of memory envisioned for ASTROL was one that could associate a value z to a pair x, y where x, y and z were any members of some set A . Perhaps a content addressable memory (CAM) could be used—where each word of the memory is divided into three fields x, y and z . Instead of being addressed, a word (x, y, z) would be accessed by specifying its x and y field contents. The storage of the word (x, y, z) would represent the association of z to the pair (x, y) . In ASTROL the instruction $xy \leftarrow z$ expresses the storage of such an association in the memory while the expression (xy) denotes the retrieval of the value z last associated to x and y . The association is deterministic in that a subsequent storage operation $xy \leftarrow w$ will erase the previous association. The operation (xy) can be thought of as meaning "the y th element of array x ," "field y of record x ," or "subject x 's y -attribute." It is left associative with maximum precedence so that xyz is the same as $(xy)z$ while $xy \leftarrow w$ means $(xy)z \leftarrow w$.

The objects x, y and z from the set A are called atoms and include the signed integers, symbols à la LISP such as BIRZWP, and two other categories—the code atom and

anonymous atom—to be described later. A special anonymous atom, *nil*, is included as the default value for an undefined association (xy) .

Beyond the simple notations given above, the evolution of ASTROL was guided by the principle of simplicity, and so the language has only one data type—the atom from A . The subcategories of A are mostly irrelevant to the language so that no expression is restricted to using a special kind of atom.

Simplicity also required that there be a single statement type and so ASTROL is an "expression" language. Each instruction is an expression with a value and so can appear elsewhere within a larger expression. The value of the association $xy \leftarrow z$ is z , whereas the expression $x(y \leftarrow z)$ causes the same association but has the value x . The latter form is useful in assigning multiple attributes to the same subject. Thus

$$x(y_1 \leftarrow z_1)(y_2 \leftarrow z_2) \dots (y_n \leftarrow z_n)$$

can be thought of as defining a "record" x whose y_i th field has the value z_i . As in ALGOL 60, the operation \leftarrow is right associative so that $xy \leftarrow zw \leftarrow u$ means $xy \leftarrow (zw \leftarrow u)$ and results in two associations $zw \leftarrow u$ and $xy \leftarrow u$. For a general example suppose that the memory has already stored the associations:

MARY SON ← BILLY JOHN WIFE ← MARY
JOHN DOG ← SPOT REL 1 ← WIFE

Then the expression:

JOHN (REL 1) SON (DOG ← MARY DOG
← JOHN DOG) BUDDY ← JOE,

has the value *JOE* and causes the associations *MARY DOG ← SPOT*, *BILLY DOG ← SPOT*, and *BILLY BUDDY ← JOE* to be placed in memory.

In addition to the (xy) and $xy \leftarrow z$ operations, there are seven binary infix operators $+, -, *, /, <, <=,$ and $=$ with the usual meanings. These all have the same precedence as \leftarrow and are all right-associative. The relational operators yield 1 for true and the nil atom for false; and a missing left operand for $-$ is presumed to be 0.

CONTROL STRUCTURES

Formally speaking ASTROL does not support the concept of a program. However, a series of expression e_i can be

combined into a new expression ($e_1; e_2; \dots e_n$) by the operator ; which is left-associative, has minimum precedence and operates by deleting its left operand after evaluating both. Clearly this is a thinly-disguised program block.

Conversational languages such as BASIC provide editing flexibility by labeling every instruction in a program. This process can be seen as the association of an instruction code segment to each label. It is natural to adapt the associative mechanism in ASTROL to this purpose, and to that end code segments are allowed to be atoms in A . The series of instructions which evaluate an expression ($e_1; e_2; \dots e_n$) can be organized into a single object called a code atom and will then be a member of A . This data object is denoted by $[e_1; e_2; \dots e_n]$. The operator DO is used to evaluate the instructions in such a code atom. For example the value of $[2+3]$ is the code atom which can add 2 and 3, but the value of $DO [2+3]$ is 5. The operator DO has a precedence just above that of the semicolon. When DO is applied to a non-code atom, it leaves that atom unchanged. Consequently, the value of $DO MARY$ is just $MARY$. However if the association:

$STEP\ 1 \leftarrow [MARY\ DOG \leftarrow SPOT; JOHN\ HAIR \leftarrow BROWN]$

were placed in memory, then the value of $(STEP\ 1)$ would be a single code atom while the expression $DO\ STEP\ 1$ would have the value $BROWN$ and would cause the two indicated associations.

The code atom is the only device for controlling program flow. It is unnecessary to have a "case" construction, because what might have been written:

```
case X(1) of begin
A: MARY DOG ← SPOT;
B: JOHN DOG ← SPOT;
Y: MARY HAIR ← BLOND
end case
```

can instead be rendered as $DO\ CS\ (X\ 1)$, provided that the associations:

```
CS A ← [MARY DOG ← SPOT]
CS B ← [JOHN DOG ← SPOT]
CS Y ← [MARY HAIR ← BLOND]
```

are present in memory.

Because the "if-then-else" construction is a special version of the "case" statement, it too becomes unnecessary. However, the resulting ASTROL formulation is awkward and so an $IF-ELSE$ operator was included in the language:

$$\text{value}\{x\ IF\ y\ ELSE\ z\} = \begin{cases} \text{value}\{x\} & \text{when value}\{y\} \text{ is not nil} \\ \text{value}\{z\} & \text{when value}\{y\} \text{ is nil} \end{cases}$$

This operator does not control program flow as in ALGOL because each operand of the $IF-ELSE$ is evaluated before the result is selected. Thus the code atom remains the sole vehicle for program control and the ALGOL version of "if x then y else z " must be expressed as $DO\ ([y]\ IF\ x\ ELSE\ [z])$, where the parentheses could be omitted since the DO and $IF-ELSE$ operators have the same precedence and are associated to the right.

The traditional loop construction can be implemented by recursion, but it was felt that this would be an expensive solution in practice. Consequently the language supports an operator, $REPEAT$, with the same precedence as DO . Its operand will be repeatedly activated by DO until it yields a non-nil value, which will then be the result of the $REPEAT$ operator. Several loop flowcharts are given in Figure 1 along with equivalent expressions in ASTROL.

It should be emphasized that $REPEAT\ x$ is an expression and does have a value. For example if A were an array of 20 elements, then the instruction:

```
STEP FIND ←
[POS I ← 0; REPEAT [DO NONE IF (20 < POS I ← POS
I + 1) ELSE [POS I IF A(POS I) = 3 ELSE NIL]]];
```

will create a code atom $STEP\ FIND$ whose activation gives the value of the first position in A whose element is 3, or the value $NONE$ in case none are.

SYSTEM SYMBOLS

Although generally speaking there are no variables in ASTROL, it does support six operandless operators #, \$, @, ?, ??, and NIL which are like read-only variables. The result of executing each of these system symbols is described below:

- # Results in a new anonymous atom for each execution.
- \$ Results in an anonymous atom, called the procedural context, which represents the procedure call depth.
- @ Results in the atom representing the function procedure currently executing.
- ? Results in the next atom from the input data stream each time that it is executed.
- ?? Results in the atom previously read by the ? operator.
- NIL Results in the nil atom.

The operator # introduces the fourth category of atom—the anonymous atom—which is a data object not in one of the other three categories: integer, spelled symbol or code segment. An anonymous atom is an internal object created by the system when the user needs a new object but does not want to name it. Each occurrence of the operator # in an expression will represent a new such object. As an example consider the expression:

```
THE ROOT ← # (OP ← ADD)
(LEFT ← # (OP ← ADD)(LEFT ← 3)(RIGHT ← 7))
(RIGHT ← # (OP ← SQRT)(RIGHT ← 9))
```

which creates the labeled ordered tree shown in Figure 2. The first occurrence of # creates the root node of the tree, while the second and third occurrences create the left and right daughters of the root. The root node is associated to the pair $THE\ ROOT$. The node labels are considered to be OP-attributes of the anonymous nodes.

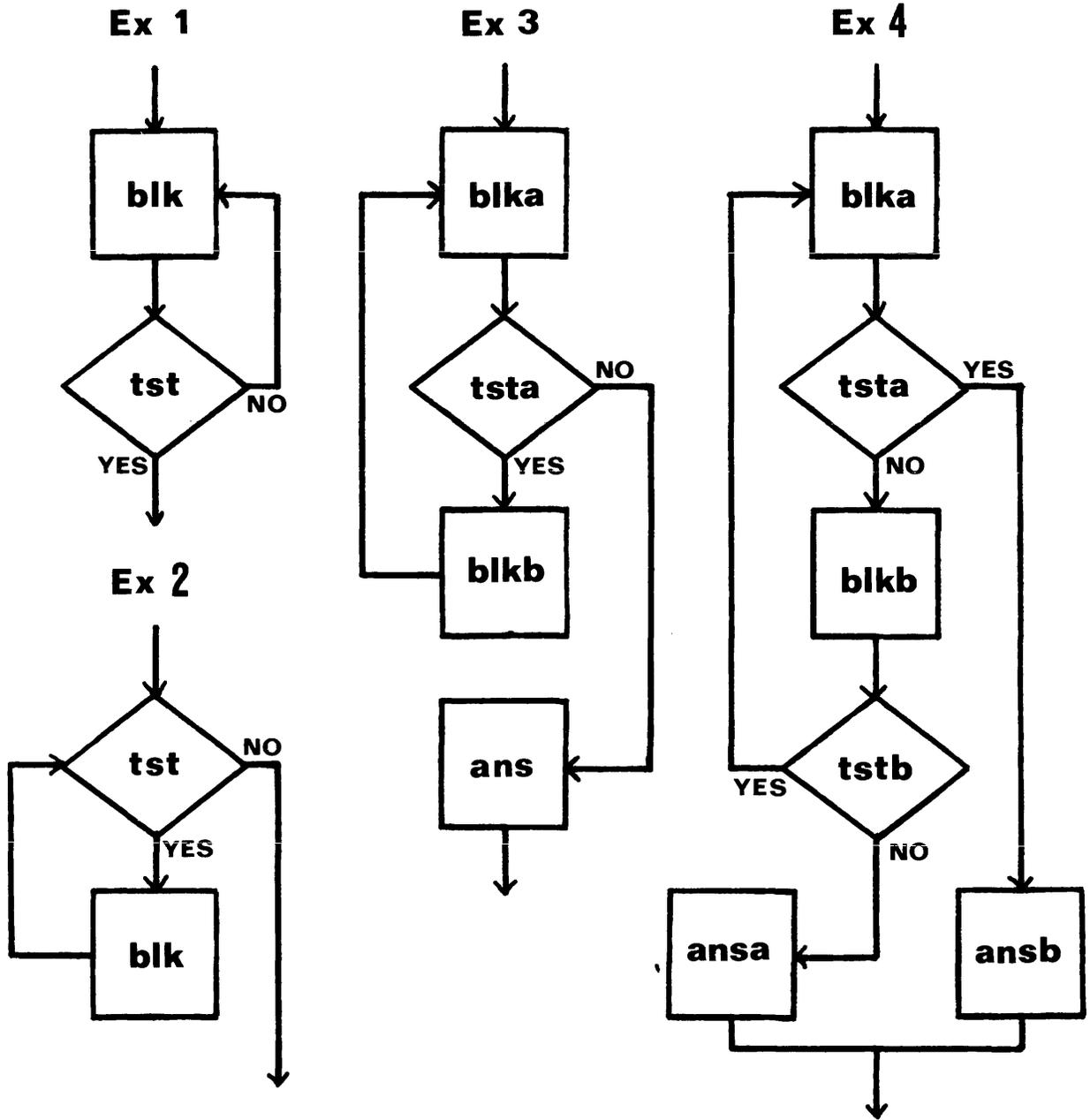


Figure 1

The operator \$ is related to the procedural treatment of identifiers. In a sense no procedural language has explicit variables but rather uses an identifier to represent a variable in that specific context which is the scope of the variable. Hence the value of the variable is associated with the context/scope and the identifier. This association is explicit in ASTROL so that the context/scope is a concrete atom in A represented by \$ and not an abstraction of the form of a program. Hence \$x represents the value associated to the identifier x in the current context \$ without need of a special mechanism. When a procedure is invoked, \$ acquires a new value to represent the context of that copy of the procedure,

while return from the procedure reinstates the previous value of \$. Hence the action of \$ simulates the pushing and popping of procedure stack "frames." In ASTROL the only means of changing context is the procedure call and so contexts can be stacked dynamically (even recursively) but cannot be statically nested as in ALGOL. The use of \$ is discussed further in the next section.

FUNCTIONS

Since ASTROL is an expression language, the only procedures are functions. There is no special syntax for the

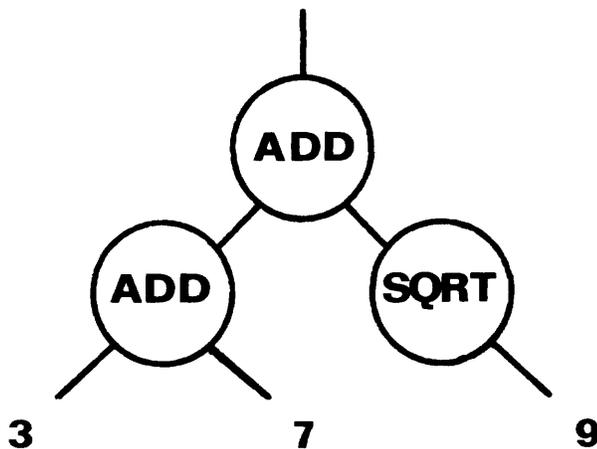


Figure 2

definition of a function and *any* atom can represent a function. A function invocation has the form:

$$x[y_1, y_2, \dots, y_n]$$

where x is the function name and the y_i are the actual parameters and are passed by value. The $[]$ operation has the same precedence as the elided operator (xy) so that $xy[z, w]p(q \leftarrow r)$ means the same as $((xy)[z, w])p(q \leftarrow r)$.

The actual parameter y_i is passed to the function x as follows. The expression (xi) should be the corresponding parameter name and so the association $\$(xi) \leftarrow y_i$ is used to pass y_i under the name (xi) . After the parameters have been passed, the system performs a $DO(x \text{ VAL})$ to execute the code body $(x \text{ VAL})$ associated to the function. As mentioned in the last section, the value of $\$$ is changed just before initiating the function call and is restored when the code atom $(x \text{ VAL})$ has finished execution. Even if x is not a function, or $(x \text{ VAL})$ is nil, all these operations are still legal, although the function value may be nil.

Since a local variable n used in a function can be referred to as $(\$n)$, the associative mechanism and the explicit context atom $\$$ make it unnecessary to allocate local variables to the function in any special way.

As one example, consider the expression:

$$F(1 \leftarrow X)(2 \leftarrow Y) \text{VAL} \leftarrow [(2 * \$X) - \$Y],$$

which creates the function $F(x, y) = 2x - y$. A more elaborate example is given by the definition of the function composition operator *COMPOS* which creates a function dynamically. This created function is represented by an anonymous atom and its composition factors are stored as the F and G attributes of that atom. Since the created function is anonymous, its name is accessed by the operator $@$ within its code body:

$$\text{COMPOS}(1 \leftarrow F)(2 \leftarrow G) \text{VAL} \leftarrow \\ \#[(1 \leftarrow X)(F \leftarrow \$F)(G \leftarrow \$G) \text{VAL} \leftarrow [@ F [@ G [\$X]]]]]$$

Although parameters are passed by value in ASTROL, the existence of the code atom allows a parameter to be effectively passed by name.

USER INTERACTION

User input is handled by the $?$ and $??$ operators covered earlier, while output is done with the operators $!$ and $:$ which have a precedence slightly higher than the semicolon and are left-associative. Both operators tabulate to the position given in the left operand. The $!$ then outputs the print record and begins a new one. Finally both operators enter the right operand into the print record and yield the next available record position as their value. For example the expression $!A:B:C!D$ would produce:

```

ABC
D {no carriage return},

```

and have 3 as its value. Output of anonymous atoms and *NIL* is omitted, while output of a code atom will print it out in source form. This eliminates the need for a program listing command.

There are no special editing or command instructions. The user types an expression followed by $a .$ and the system evaluates it. For example the instruction $: 1+2.$ would print out 3.

Liberal use of the *DO* operator will allow simple instruction editing via the \leftarrow operator. This also facilitates debugging since individual steps in a task can be tested in isolation.

Creating user program files presents a problem, since a task in ASTROL is performed by a loosely organized system of code segments and there is no program as such for the user to save. Dumping *all* the associations in memory would be awkward because of the anonymous atoms. Consequently, the primitive function $\text{SAVE}[file-name, n]$ includes a parameter n to determine the class of associations $xy \leftarrow z$ to save. When n is 0, then z must be a code atom and x and y must be integers or symbols. When n is 1, then z may also be an integer or symbol. The associations are dumped in source-readable format so that the primitive function $\text{READ}[file-name]$ can simply substitute the specified text file for the normal teletype input stream.

Actual implementation of the language required simulation of the associative memory. This was done by hash-coding the subject-attribute pairs. For efficiency in execution, code atoms were translated into a postfix polish notation, although this requires de-translation in order to list such an atom. The only garbage collection is to delete unreferenced code atoms. Currently, ASTROL runs on a PDP11/20 with 32K bytes of store under the RT11 operating system.

CONCLUSION

A potential technological advance such as the associative memory might be expected to have some impact on programming concepts. ASTROL was created in anticipation of this advance and illustrates some of its consequences for programming. Although it is goto-less and programs should be carefully organized in a top-down fashion—the code segmentation and automatic data structure allocation impart an unstructured flavor to the language. The use of data associations to control program flow allows easy extension of

conversational programs—often without requiring *any* alteration of existing code segments.

Currently the language is being used to implement CAI programs for mathematics and computer science and to investigate an alternative to transformational English grammar.

REFERENCES

1. Berkeley, E. C. and D. G. Bobrow, *The Programming Language LISP, Its Operation and Applications*. Cambridge: M.I.T. Press, 1966.
2. Ekman, T. and C. Froberg, *Introduction to ALGOL Programming*. London: Oxford University Press, 1967.
3. Kohonen, T. *Associative Memory*. Berlin: Springer-Verlag, 1977.

APPENDIX

BNF grammar for ASTROL

```

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter> ::= A | B | C | D . . . X | Y | Z
<binary operator> ::= + | - | * | < | <= | = | /
<io operator> ::= : | !
<system symbol> ::= $ | # | ? | ?? | @ | NIL

```

```

<integer> ::= <digit> | <integer><digit>
<name> ::= <letter> | <name><letter> | <name><digit>
<instruction> ::= <block>.
<block> ::= <io> | <block>; <io>
<io> ::= <expression> | <io operator><expression> | <io><io operator><expression>
<expression> ::= <simple expression> | -<expression> | DO<expression> | REPEAT<expression> | <simple expression>IF<expression>ELSE<expression>
<simple expression> ::= <term> | <term><primary> <-> <expression> | <term><binary operator><expression>
<term> ::= <primary> | <term><primary> | <term>(<primary> <-> <expression>) | <term>[ ] | <term>[<argument list>] | [<block>]
<primary> ::= <integer> | <name> | <system symbol> | (<block>) | RND[<expression>] | READ[<expression>] | SAVE[<expression>, <expression>]
<argument list> ::= <expression> | <argument list>, <expression>

```


An associative search language for data management

by AMAR MUKHOPADHYAY and ALIREZA HURSON

*The University of Iowa
Iowa City, Iowa*

INTRODUCTION

Recent years have witnessed a widespread and intensive effort to develop systems to store, maintain and access data bases of varied size. Such systems are referred to as DBMS—Data Base Management Systems. In different areas, such as artificial intelligence, management information systems, military and corporate logistics and medical diagnosis, a wide variety of DBMS exist. All these systems have generally been implemented on conventional computers, which are based on the von Neumann design. In this design, operations will be performed on the information in the memory by means of their addresses. Because of the size of typical data bases and costs of memory, we cannot hold all information in the main memory and swapping converts the search problem to a transportation problem. Present-day systems have to transfer large sets of data from their mass storage to the CPU, where simple compare-functions are performed in order to separate relevant data from irrelevant data. The transfer channels with their limited capacity form the main bottleneck of this system and as a result, great efforts have been made to reduce the necessary data flow by means of sophisticated software systems and additional redundancy such as index tables and inverted files. By these techniques, address of information will be obtained from a directory. Although directory partially solved the bottleneck problem, it nevertheless created some problems. The directory should logically be kept in the main memory. If we are dealing with large data bases, naturally we are dealing also with large directories, and large directories occupy a large portion of the memory. Also, the use of directories will create some complexity in the search, update and delete algorithms. Conventional computers are all based on numerical operations. The necessity of designing new hardware based on non-numerical operations has been discussed in detail by one of the authors.¹ In contrast, use of associative or content addressable memories and hardware design based on non-numerical operations as well as numerical operations causes information stored at unknown locations to be processed efficiently on the basis of some knowledge of its content.

In developing new architectures for future machines some of the most important trends in hardware and software technology must be brought into focus. On the hardware side,

the significant trends are the development of LSI and VLSI technology which allow increased functionality of hardware components coupled with a drastic reduction in cost per function. This implies that new architectures must exploit this trend to incorporate more software functions by hardware. The second important trend in hardware is the development of serial access secondary storage devices such as CCD or magnetic bubble memory (MBM). It is forecast that by 1980/81 a 1Mbit memory will be available for both CCD and MBM, which will emerge as powerful competitors for disks and tapes.² We could make the assumption that a major part of future data base systems will reside in such shift-register type memories. The proposed architectures in this paper will, of course, work for other kinds of rotating secondary devices such as disks. A common feature of all these secondary devices is that they are block-oriented, that is, access to a block is slow but data flow rate is high. If the processing of information has to be done by CPU, blocks of information need to be transferred back and forth from the CPU to secondary devices. The new machine architectures, therefore, should attempt to provide processing capabilities outside of CPU and along with secondary devices in the form of associative hardware which will operate on information on the fly. This idea was originally suggested by Slotnik,³ and in recent years a significant amount of effort has been expended in the development of similar architectures in data base applications.⁸⁻¹⁷

There are, however, several disadvantages to an associative approach—first, hardware for associative processing still has to prove its cost-effectiveness compared to the existing software implementations; second, future innovations in computer systems design and architecture have to confront inertia and large investments on the existing systems. Although the principles of associative processing are as old as second generation computers (and there is an extensive body of literature dealing with many aspects of this phenomenon in computer design), the major bottleneck for the development of associative systems has been the degradation of speed (the memory cycle time) for the required size of the memory. The demand on the size of the memory can be reduced by intelligent design of the architecture. For example, the design of the Lee machine is impractical because the design assumes the main memory system to be associative. In an earlier paper one of the authors¹ showed that a small associative memory could em-

ulate the functions of the Lee machine if information could be passed over it. In the data base context, we are looking for similar architecture and therefore the size of the associative memory will be reduced. This, coupled with the development of VSLI technologies, shows good promise for the use of associative hardware for future machines. A relatively recent associative processor ALAP¹⁹ demonstrates the feasibility of making large associative processors practical.

This paper will introduce a high-level data base language, ASL—Associative Search Language—which is suitable for direct hardware implementation by associative hardware. The architecture for ASL and its proposed hardware implementation will be described in future papers. This paper is concerned with a preliminary definition and specification of ASL.

On the software side, as far as the data base management systems are concerned, an overall recent trend is characterized by the phrase *data independence*. Data independence means that the user is able to perform retrieval and storage operations on the data base depending on the *information content* without having to deal with representational details. Codd^{4,5} proposed an important data model, the now-famous relational model, which provides the user with a model of data based on content or data value only. Based on Codd's model, a number of families of data sub-languages and query languages have been proposed—one class based on relational calculus (viz., DSL ALPHA, INGRES, DAMAS), another class based on relational algebra (PRTV, SQUIRAL), and a third class which adopts a "mapping" approach based on the relational algebra (SQUARE, SEQUEL, Query-by-example). The reader is referred to the text by Date⁶ or the review article by Chamberlin⁷ for more details. In the late 60s, an associative language, ASP,²⁰ was developed in the context of list and language processing.

The conceptual framework of ASL is very similar to that of SQUARE,¹⁸ a data sub-language which formed the basis for the development of SEQUEL. The similarities of these two languages are reflected in the definition of the basic operations which brings forth explicitly the associative nature of processing, and also in avoiding the use of quantifiers required by languages based on relational calculus. Both ASL and SQUARE are relationally complete, provide facilities for query, insertion, deletion and update operations, and are meant for non-professional programmers who do not possess a high degree of mathematical sophistication. However, the originators of SQUARE did not emphasize the associative nature of the primitive operations since the language was not considered for hardware implementation. The authors are of the opinion that for hardware implementations of data base operators, development of a language like SQUARE or ASL is essential. The relationship between a high-level user language such as SEQUEL or Query-by-example with SQUARE or ASL is similar to the relationship between FORTRAN or PL/1 and assembly language except that the definition of ASL is independent of actual hardware implementation. We also claim that ASL has several advantages over SQUARE, as follows: the structure of the language is more precise and the expressions in the language

can be handled by parsing algorithms that are available for well formed arithmetic expressions in programming languages which, in the hardware context, define a precise sequence of controlling operations by the hardware; ASL allows parallel computations of a set of relations and is based on variable size information fields with complete independence; ASL is also easily extendable to multi-dimensional as well as non-relational data bases; and finally, ASL provides potential for the development of a FORTRAN-like language for data base applications.

ASL—AN ASSOCIATIVE SEARCH LANGUAGE FOR DATA BASE MANAGEMENT

The language ASL—Associative Search Language—is a high-level data base language designed for information retrieval and storage operation on data bases using associative principles for basic operations. The language has been defined based on the relational model of data presented by Codd, although extensions to non-relational models are possible. A partially complete syntax for ASL is included in the Appendix.

The fundamental operation in ASL is a search on a data base with respect to criteria (search arguments). The result is a relation yielding the retrieved information. This operation can be thought of as an assignment statement $W:=X$, where W is a relation and X is composed of three parts,

HOW WHERE WHAT,

where "HOW" specifies the search arguments (search criteria), "WHERE" specifies the relation name and "WHAT" specifies the output domains of the relation. If R , C , D stand for *WHERE*, *HOW* and *WHAT* respectively, we can say R is a binary transformation operation whose operands are C and D , and can be expressed as

$$C \textcircled{R} D.$$

C is a set of sets ($C=(c_0, \dots, c_k)$) where each element in this set ($c_i, 0 \leq i \leq k$) is an unordered list of search arguments; for $i=0$, $c_i=\phi$, denotes an empty set of search. D is also a set of sets ($D=(d_0, \dots, d_n)$) where each $d_i(0 \leq i \leq n)$ is a non-empty unordered list of domains over R . When d_i includes all the domains of R , it will be denoted by δ . The result of operation \textcircled{R} on C and D is a set of relations (W_0, \dots, W_n) where the domains of $W_i(0 \leq i \leq n)$ are the same as those of $d_i(0 \leq i \leq n)$. In our implementation we assume C and D as well as W are singleton sets.

The informal presentation of ASL will be with respect to the employee-department data base as follows:

```
E(ENO, ENAME, DEGREE, LOCATION)
D(DNO, DNAME, DHEAD, LOCATION)
ED(ENO, DNO, HOUR)
```

where *ENO*, *DNO* stand for Employee and Department number, *ENAME*, *DNAME* stand for Employee and Department name, and *HOUR* shows how many hours each

employee works in each department. Each query will be expressed in two forms, first as assignment statements $W:=X$, where X is specified using conventional notations in data base literature,⁶ and then in the form of ASL statements.

RETRIEVAL OPERATIONS

Simple retrieval

- Get department number of all departments

$$W:=\phi \textcircled{D} DNO$$

According to our current implementation of ASL, this query will be presented as below in ASL language:

$$D, \\ W=[D]DNO;$$

where D . declares the name of the relation. Note the set of rules in the appendix shows that each program will be started with the declaration of all relations used in that program. Henceforth, in all the examples, we will omit the declarative statement.

- Get all the information of all employees:

$$W:=\phi \textcircled{E} \delta$$

which in fact is equivalent to $W:=E$. This query in ASL will be written as

$$W=[E];$$

Qualified retrieval

- Get employee numbers for all employees in LOCATION L1 with DEGREE>3:

$$W:=(LOCATION='L1'\wedge DEGREE>'3')\textcircled{E} ENO$$

In ASL it is $W=((LOCATION \text{ EQ } 'L1')\wedge DEGREE \text{ GT } '3')[E]ENO;$.

This example illustrates that C could be specified by a predicate. The output W is a relation whose tuples are ENO of relation E .

Complex retrieval

Up to now, retrieval operations with simple search arguments and over one relation which yields a sub-relation of the original relation have been discussed. A sub-relation is a relation derived from a given relation by the selection of a row, the projection on columns, and then removing the redundant tuples. We will now consider more complex retrieval operations.

Retrievals that yield sub-relations by using more than one

relation:

- Get employee names for employees who work in department D1:

$$W:=((DNO='D1')\textcircled{ED} ENO)\textcircled{E} ENAME$$

The nested nature of operations has been illustrated by this example—first, the relation ED will be searched for all department numbers equal to $D1$ yielding a subrelation of ENO . This sub-relation will then be used as an argument to obtain $ENAME$. In fact, this sub-relation will be searched in associative fashion. The operation can be seen as:

$$X:=(DNO='D1')\textcircled{ED} ENO$$

$$W:=\forall_{(ENO\in X)}\textcircled{E} ENAME$$

In ASL we have $W=((DNO \text{ EQ } 'D1')[ED]ENO)[E]ENAME;$.

The right-hand side of an assignment statement will be called a *set expression*. A set expression, like arithmetic expressions, can be written in infix notation, as previously, or in postfix notation as

$$W:=(DNO='D1')ENO \textcircled{ED} ENAME E$$

defining the same relation W . The use of postfix notation avoids redundant use of parentheses and also has the advantage with respect to compilation and the design of hardware controller that implements the language. We will write the set expressions in infix notation. The predicate expressions will always be parenthesized.

- Get employee numbers for employees who work in department managed by 'SMITH':

$$W:=(DHEAD='SMITH')\textcircled{D} DNO \textcircled{ED} ENO$$

and in ASL:

$$W:=(DHEAD \text{ EQ } 'SMITH')[D]DNO[ED]ENO;$$

- Get employee names for employees who work in department managed by 'SMITH':

$$W:=(((DHEAD='SMITH')\textcircled{D} DNO)\textcircled{ED} ENO)\textcircled{E} ENAME$$

and in ASL:

$$W:= \\ (((DHEAD \text{ EQ } 'SMITH')[D]DNO)[ED]ENO)[E]ENAME;$$

A complicated set expression like the previous one could be parsed starting from the postfix equivalent expression using dynamic relation variables such as A and B :

$$W:=(DHEAD='SMITH')DNO \textcircled{D} ENO \textcircled{ED} ENAME \textcircled{E}$$

The parsing will proceed from left to right and the computation will proceed as

$$A:=(DHEAD='SMITH')\textcircled{D} DNO$$

$$B:=A \textcircled{ED} ENO$$

$$W:=B \textcircled{E} ENAME$$

We will now consider retrievals that do not necessarily

yield sub-relations and use several construction operators like tuple-compatible union (\cup), intersection (\cap), complementation ($\bar{}$), cartesian product (\otimes), join, division (\div) and restriction (\downarrow), along with Boolean operations AND (\wedge), OR (\vee), NOT (\neg), and predicates using these and relational binary operators $<$, $>$, $=$, \leq , \geq , $< >$. The normal set of arithmetic functions and 'library' functions of counting and averaging, etc. could be super-imposed on these. In many of the expressions written that follow, more than one statement, sometimes with implicit loops, will be used. Although our language is relationally complete, it does not satisfy the single statement requirement. The data sub-languages using relational calculus that satisfy the single sentence constraint but use existential and universal quantifiers do not seem to be so easily understood by even reasonably sophisticated users. Furthermore, our motivation also has roots in the hardware implementation and provides us with a simple interface design for translating queries into hardware control signals. Only the first forms of the queries (viz., $W:=X$) are specified; ASL statements are omitted.

- Get a list of all employees with the amount of hours they work for each department:

$$X := \phi ED$$

$$W := \bigcup_{x \in X} [x \otimes ((ENO = x(ENO)) \textcircled{E} ENAME)]$$

where $x(ENO)$ denotes the ENO field of $x \in X$.

- Get employee names for employees who do not work in department $D1$:

$$W := [(\phi \textcircled{ED} DNO)$$

$$-((DNO = 'D1') \textcircled{ED} DNO)] \textcircled{E} ENAME$$

- Get all employee and department names. Pair so that the indicated employee and department are located in the same place:

$$X := \phi E \text{ LOCATION}$$

$$W := \bigcup_{x \in X} [(LOCATION = x) \textcircled{E} ENAME]$$

$$\otimes (LOCATION = x) \textcircled{D} DNAME]$$

- Get employee names for employees who work in all departments:

$$X := \phi \textcircled{ED} ENO$$

$$Y := \phi \textcircled{D} DNO$$

$$W := ((X \textcircled{ED} DNO) = Y) \textcircled{E} ENAME$$

JOIN OPERATION

Let θ be any relation $=$, $<$, $>$, $< \leq$, $>$, \geq . Then the θ join of a relation R on domain D_r with relation S on domain

D_s is defined (using Codd's notation) as:

$$R[D_r \theta D_s] S = \{(rs) | r \in R \wedge s \in S \wedge (r(D_r) \theta s(D_s))\}$$

where $r(D_r)$ and $s(D_s)$ are assumed to be θ -comparable. The quantity (rs) denotes a concatenation of tuples. It is possible to specify a subset of domains to be concatenated without repeating the comparand domains. Let us denote these domains by D_r^* and D_s^* . Then we can define our set expression equivalent of θ -join as:

$$Y := \phi \textcircled{R} D_r$$

$$Z := \phi \textcircled{S} D_s$$

$$X := \{(y,z) | y \in Y, z \in Z \text{ and } y \theta z\}$$

$$W := \bigcup_{a \in X} \{y \textcircled{R} D_r^*\} \otimes t \otimes \{z \textcircled{S} D_s^*\}$$

where t could be either y or z or (y,z) .

Example

Let

R	A	B	C
a	1	1	
a	2	1	
b	1	2	
c	2	5	
c	3	3	

S	C	D
2	u	
3	u	
4	u	

and let $D_r = D_s = C$, $D_r^* = (A,B)$, $D_s^* = (D)$, $t = y$ and θ be equality ($=$) so that we are computing equijoin. Then

$$Y := \phi \textcircled{R} C = (1,2,3,5)$$

$$Z := \phi \textcircled{S} C = (2,3,4)$$

$$X := ((2,2), (3,3))$$

$$W := \bigcup_{a \in X} [y \textcircled{R} (A,B)] \otimes y \otimes [z \textcircled{S} (D)] = \begin{bmatrix} b & 1 & 2 & u \\ c & 3 & 3 & u \end{bmatrix}.$$

The computations of the set X do not follow the format of a set expression although it is a well defined set. Similar well defined set computation procedures must be part of ASL facilities.

DIVISION OPERATION

Let D_r , D_0 denote domains in relation R such that $D_r \cap D_0 = \phi$. Let D_s be a domain in relation S such that it is comparable with D_r . Then the division operator

$$R[D_r \div D_s] \textcircled{S} (D_0)$$

is defined by the set expressions

$$X := \phi \textcircled{S} D_s$$

$$W := \bigcap_{x \in X} x \textcircled{R} D_0$$

Example

	A	B	C
R	1	11	x
	2	11	y
	3	11	x
	4	12	x

	D	F
S	x	1
	x	2
	y	1

Let $D_r=C$ and $D_s=D$. Let $D_0=(B)$. Then

$$X := \phi(\textcircled{S})D = (x, y)$$

$$W := (x \textcircled{R}(B)) \cap (y \textcircled{R}(B)) = (11, 12) \cap (11) = (11)$$

RESTRICTION OPERATION

The restriction operation is simple to model in our scheme. The search arguments are defined in terms of a binary relation θ between comparable subsets of domains D_1 and D_2 of the relation R , and an output domain D_3 is specified. Thus

$$W := ((D_1 \theta D_2) \textcircled{R}) D_3$$

Example

	A	B	C
R	p	2	1
	q	2	3
	q	5	4
	r	3	3

Let $D_1=B$, $D_2=C$, $D_3=(A,B,C)$, $\theta=>$. Then

$$W := ((B > C) \textcircled{R})(A, B, C) = \begin{bmatrix} p & 2 & 1 \\ q & 5 & 4 \end{bmatrix}$$

STORAGE OPERATIONS

Storage operations such as "update," "insert," and "delete" can be performed easily by accessing appropriate tuple(s) from the appropriate relation followed by modification of information. They can be specified in two forms:

1. Set Expression := new-data.

This is a simple store operation that changes the existing information to new values without regard to previous value and does not alter the relation otherwise.

2. $W := a$ modified relation expressed by appropriate transformation of the set expression.

We give some examples below:

- Update change the location of department D1 to "NEW YORK"

$$(DNO = 'D1') \textcircled{D} LOCATION := 'NEW YORK'$$

- Delete all employees whose degree is less than 4.

$$E := E - (DEGREE < 4) \textcircled{E} \delta$$

In ASL it is $E = E - (DEGREE LE '4')[E]$.

- Delete relation E.

$$E := E - E$$

In ASL it is simply $E -$.

- Add a new employee to EMPLOYEE relation.

$$E := EU(ENO = '123'; ENAME = 'JOE', DEGREE = 'S', LOCATION = 'ROME')$$

and in ASL we have

$$EU(ENO = '123'; ENAME = 'JOE'; DEGREE = 'S'; LOCATION = 'ROME');$$

- Add 5 to Hour for employee number N:

$$x := N \textcircled{ED} HOUR$$

$$x := x + 5$$

$$(ENO = N) \textcircled{ED} HOUR := x$$

In ASL it is given by

$$(ENO = N)[ED]HOUR = HOUR + 5$$

CONCLUSION

A preliminary definition of a data base management language based on associative primitive operations along with its BNF specification has been presented in this paper. Detailed implementation of ASL as a language processor and hardware architecture based on ASL will be reported in future papers. Because of the upsurge of recent interest in nonnumeric computation, development of specialized language processors will result in simplification of software systems and will lead to the development of specialized hardware processors which will exploit the current trends in technology.

ACKNOWLEDGMENT

The work is supported by National Science Foundation grant MCS76-04763.

REFERENCES

1. Mukhopadhyay, Amar, "Hardware Algorithms for Nonnumeric Computation," *Proc. 5th Annual Symposium on Computer Architecture*, April 3-5, 1978, Palo Alto, California.
2. Juliussen, J. Egil, "Bubbles and CCD Memories—Solid State Mass Storage," *Proc. National Computer Conference*, Anaheim, California, June 5-8, 1978. (*AFIPS Proc.*, Vol. 47, Part C—"Systems," 1978, p. 1067)

3. Slotnik, D. L., "Logic Per Track Devices," *Advances in Computers*, Academic Press, 1970.
4. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, Vol. 13, June 1970, pp. 377-387.
5. Codd, E. F., "A Data Base Sub-language Founded on the Relational Calculus," in *Data Base Systems*, Courant Computer Science Symposium Series, Vol. 6, Englewood Cliffs, N.J., Prentice-Hall, 1972.
6. Date, C. J., "An Introduction to Data Base Systems," Addison-Wesley, Second Edition, 1977.
7. Chamberlin, D. D., "Relational Data Base Management Systems," *Computing Surveys*, Vol. 8, No. 1, March 1976, p. 43.
8. Defiore, C. R., and P. B. Berra, "A Data Management System Utilizing an Associative Memory," *Proc. AFIPS*, 1973, NCC, Vol. 42, AFIPS Press, Montvale, N.J., p. 121.
9. Copeland, G. P., G. J. Lipovski and S. Y. W. Su, "The Architectures of CASSM: A Cellular System for Nonnumeric Processing," *First Annual Symposium on Computer Architecture*, 1973.
10. Berra, P. Bruce, "Some Problems in Associative Processor Applications to Data Base Management," *National Computer Conference*, 1974.
11. Moulder, Richard, "An Implementation of a Data Management System on an Associative Processor," *National Computer Conference*, 1973.
12. Healy, L. D., G. J. Lipovski and K. L. Doty, "The Architecture of a Context Addressed Segment-Sequential Storage," *Fall Joint Computer Conference*, 1972.
13. Ozkarahan, K. A., S. A. Schuster and K. C. Smith, "RAP—An Associative Processor for Data Base Management," *Proc. National Computer Conference*, 1975.
14. Karlowsky, I., H. O. Leilich and G. Steige, "A Search Processor Proposed for Data Base Applications," abstract in *Computer Abstracts*, Vol. 19, No. 9, Sept. 1975, p. 194.
15. Copeland, G. P., and S. Y. W. Su, "A High Level Data Sublanguage for Context-Addressed Segment-Sequential Memory," *Proc. ACM SIGFIDET Workshop on Data Description, Access and Control*, May 1974.
16. Hsiao, D. K., et al., "The Architecture of a Database Computer," Ohio State University, CISC Technical Reports: 0S4-CISRC-TR-76-1/2/3, Columbus, Ohio, 1976.
17. Bird, R. M., J. C. Tu and R. M. Worthy, "Associative/Parallel Processors for Searching Very Large Textual Data Bases," *Proc. 3rd Workshop on Computer Architecture and Nonnumeric Processing*, Syracuse University, May 17-18, 1977.
18. Boyce, R. F., et al., "Specifying Queries as Relational Expressions, the SQUARE Data Sublanguage," *Communications of ACM*, Vol. 18, Nov. 1975.
19. Finnila, C. A., and H. H. Love, Jr., "The Associative Linear Array Processor," *IEEE Transactions on Computers*, Vol. C-26, No. 2, Feb. 1977, pp. 112-125.
20. Savitt, D. A., and H. H. Love, "An Iterative-Cell Processor for the ASP Language," in *Associative Information Techniques*, Edwin L. Jacks (ed.), Elsevier, 1971. (Also, see *Proc. AFIPS*, Vol. 30, 1967, p. 87.)

APPENDIX

The following syntax, which is based on BNF form, gives an intuitive understanding about the structure of ASL. This is not a complete set of rules—the syntax has been simplified for ease of understanding.

```

<Program> ::= <Dec list> <Re list>
<Dec list> ::= empty |
              <Dec list> <Relation Id>.
<Re list> ::= <Re list> <Relation> |
              <Relation>:
<Relation> ::= <Up Relation> |
              <Q Relation> |
              <T Relation>
<Up Relation> ::= <Ch Relation> |
                <Add Relation> |
                <D Relation>
<Add Relation> ::= <Relation Id> ∪ <Domain-value list> |
                 <Relation Id> ∪ <Domain-type list>
<D Relation> ::= <Relation Id> − <Set expression> |
                <Relation Id> −
<Ch Relation> ::= <Search set> <Relation Op> <Domain-value list> |
                 (<Set expression>) <Relation Op> <Domain-value list>
<Q Relation> ::= <Set expression> |
                 <Relation Id> ⊗ <Relation Id> |
                 <Relation Id> <Join> <Relation Id> |
                 <Relation Id> <Divide> <Relation Id> |
                 <Relation Id> <Restriction> <Output Set> |
                 <Relation Id> <Log-set Op> <Relation Id>
<T Relation> ::= <Relation Id> = <Q Relation> |
                 <Relation Id> = <Up Relation>
<Set Expression> ::= <Search Set> <Relation Op> <Output Set> |
                   (<Set expression>) <Relation Op> <Output Set> |
                   <Relation Id>
<Search Set> ::= empty |
               <expression> |
               (<Search Set>) <Log-set Op> <expression>*
  
```

* <Expression> is any usual arithmetic expression.

Updating defined relations

by I. M. OSMAN
University of Khartoum
Khartoum, Sudan

INTRODUCTION

Some relational data base systems¹ such as PRTV² and SE-QUEL allow the user to create his own relations (logical files) as subsets of the main data. With such a facility the user may express his own view of the data in terms of the relations he creates. If a large number of such relations are created, disc storage space problems and other maintenance difficulties can arise.

A relation which is expressed in terms of other relations is called a defined (derived or implied) relation² or view.³ The user submits to the system a definition of these relations expressed in terms of data base relations using relational operators. For example, consider relations MALE, FEMALE whose domains (fields) are name, age and income. The user may define the relation PEOPLE as the union of relations MALE and FEMALE. He may also define the relation OLD as a subset of relation PEOPLE where (age>50).

The data base system decodes the definition and stores it in a retrievable form. The defined relation is then available to the user at the same logical level as the other data base relations. The defined relation remains merely a stored definition (i.e. implicit) until it is requested by a query. The implicit form takes negligible disc space.

When the defined relation in its implicit form is referenced by a query, e.g. list the people whose income is greater than 5,000, it is then created, i.e. made explicit, by carrying out on the stored data the operations indicated in the definition. For example, to create relation OLD relation PEOPLE is assembled first. The tuples (records) of relation PEOPLE are then accessed and those matching the selection criterion are written back to disc as tuples of the relation OLD.

The explicit forms of relations PEOPLE and OLD may stay in the data base and they will not have to be recreated if they are requested by other queries. In this way defined relations may substantially improve the response time of recurring queries.

We may consider the data base storage space as made up of two parts. One part is for storing the base relations (e.g. MALE and FEMALE) and another part whose content changes dynamically with the needs of the users. The defined relations facility enables the system to make efficient use of the dynamic storage space. At some stage in the

process of creating and querying relations the dynamic storage space may be consumed. One or more explicit relations will then have to be made implicit in order to free space for other requested relations. The relations in the dynamic storage switch between being explicit and implicit. The replacement algorithm which picks the relations to be made implicit has been discussed in a previous paper.⁴

Notation

In this paper the set notation is adopted for relations. This is explained in the following table:

SYMBOL	MEANING
\cup	union
\cap	intersection
:(filter)	selection, the filter is a boolean expression
%(list)	projection, the list is a string of selectors
*	join (cartesian product or equijoin).
-	difference
\leftarrow	is defined as
:=	assigned to

e.g. the definitions of relations PEOPLE and OLD are expressed as follows:

```
PEOPLE $\leftarrow$ MALE $\cup$ FEMALE  
OLD $\leftarrow$ PEOPLE: (age>50)
```

The hierarchy of defined relations

In Figure 1 a data base having a hierarchy of defined relations is shown. In such a hierarchy the user may not be aware of the relationships and the dependencies that exist among the data base relations. It is therefore advantageous to provide the facilities by means of which the user can perform all the operations on a defined relation without restrictions. Ideally, the user will treat the defined relations in exactly the same way as any other base relations, without restrictions forcing him to be aware of the dependencies between relations. That is, the system should possess a higher degree of data independence. It is therefore desirable to have the user activities (queries, application programs,

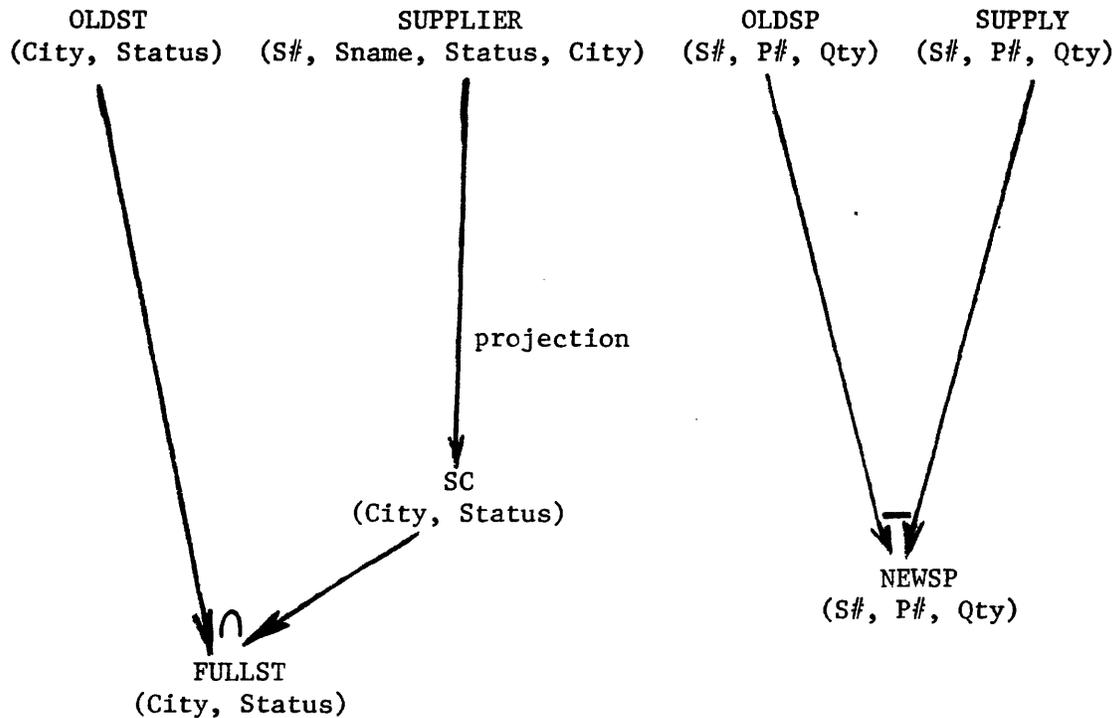


Figure 1—Supply data base. In these hierarchies of defined relations, OLDST, SUPPLIER, OLDSP and SUPPLY are base relations. SC is defined on SUPPLIER:

FULLST is defined on SC and OLDST.
 Definitions: $SC \leftarrow \text{SUPPLIER} \% (\text{City}, \text{Status})$
 $FULLST \leftarrow \text{OLDST} \cap SC$
 $NEWSP \leftarrow \text{OLDSP} \bowtie \text{SUPPLY}$

updates, etc.) independent of the logical representation, the access path and the level in the hierarchy of the data. The user need not know where a relation is a base or a defined relation, is implicit or explicit, nor will he need to know whether it is stored as a file or a mere collection of pointers to some other files. However, giving all this freedom to the user is desirable only as long as the consistency of the definitions and the defined relations is preserved.

The problems of the management of defined relations

In a data base with a defined relations capability such as the one shown in Figure 1, some of the system operators (e.g. updates, removal of relations from the data base and the redefinition of relations) may lead to logical problems. These operators must be adapted to take care of the hierarchical structure and the dependencies that exist among the relations. This paper is concerned only with the update problems.

Examples of these problems are:

- In Figure 1, if relation SUPPLIER is updated and relation SC is implicit, how can we update an explicit copy of relation FULLST? This is an efficiency problem.
- If relation FULLST is updated, how can we update SUPPLIER and OLDST? This is a logic problem.

Objectives

This paper explains the update problems and offers solutions for some of these problems. Examples are given to illustrate these solutions. The paper also identifies the unsolved problems which require further research and investigation.

Some update algorithms are suggested. They aim at preserving the consistency of the information in a data base having a high degree of data independence. These update algorithms deal with the logical part of the problem and hence pave the way for other algorithms to care for the particular details in particular implementations.

UPDATES

Updates may be divided into:

- a. Insertions
- b. Deletions
- c. Changes in object values.

In the following discussions, the tuples which will update a relation are termed the updating relation. The insertion is seen as a union of the updating relation and the relation to be updated. The deletion is seen as the difference of the relation to be updated and the updating relation. The

changes in object values are taken conceptually as deletions followed by insertions.

The update algorithms

Let us divide the updates into two:

1. Update at higher levels of the hierarchy.
2. Update at lower levels of the hierarchy.

Referring to Figure 1, updating relation SUPPLY or SUPPLIER is an update at a higher level of the hierarchy with respect to NEWSP and SC, FULLST respectively. Updating FULLST or NEWSP is an update at a lower level. Updating SC is an update at a higher level with respect to FULLST and an update at a lower level with respect to SUPPLIER.

Update at a higher level

In this type the update is filtered down the hierarchy and reflected on all the relations defined on the updated relation. The corresponding definition is applied to the updating relation successively at each level of the hierarchy. The problem here is how to pass updates down efficiently and in particular when intermediate levels are implicit.

Insertions

Given relations A and R, let RUI be the updated value of R. Let $X=f(A,R)$ be a defined relation. Relation IN is sought such that $X \cup IN$ is the updated value for X which corresponds to the update of R. Such IN can be found for all operators (f) in case of insertions at high levels. This is shown in table CHANGE (Figure 2).

When f is union, intersection, projection or selection $IN=f(A,I)$ i.e. X need not be present in order to evaluate IN.

The mechanism of the insertion is essentially carried out at each level as follows:

- a. Given the updating relation (I) and a definition, relation IN is found from table CHANGE. Relation IN is now the updating relation which will be passed to the following levels down the hierarchy. (i.e. IN will be I of the lower level).
- b. If the relation to be updated (X) is explicit, the updated value will be $X \cup IN$. This applies to all relations at lower levels. If X is implicit, updates are passed down the hierarchy without materializing X.

Examples 1 and 3 in the Appendix illustrate this type of insertion.

The exception to Rules a and b is the definition containing the difference operator when the second relation is to be

	DEFINITION	PROOF	IN
INSERT	$X \leftarrow A \cup R$	$A \cup (R \cup I) = (A \cup R) \cup I = (A \cup R) \cup (A \cup I)$	I or $A \cup I$
	$X \leftarrow A \cap R$	$A \cap (R \cup I) = (A \cap R) \cup (A \cap I)$	$A \cap I$
	$X \leftarrow R - A$	$(R \cup I) - A = (R - A) \cup (I - A)$	$I - A$
	$X \leftarrow A - R$	$A - (R \cup I) = (A - R) - I = (A - R) - (I \cap A)$	I or $I \cap A$
	$X \leftarrow R\%(list)$	$(R \cup I)\%(list) = R\%(list) \cup I\%(list)$	$I\%(list)$
	$X \leftarrow A * R$	$A * (R \cup I) = (A * R) \cup (A * I)$	$A * I$
	$X \leftarrow R:(filter)$	$(R \cup I) : (filter) = (R:(filter)) \cup I:(filter)$	$I : (filter)$
DELETE	$X \leftarrow A \cup R$	$A \cup (R - I) = (A \cup R) - (I - A)$	$I - A$
	$X \leftarrow A \cap R$	$A \cap (R - I) = (A \cap R) - I = (A \cap R) - (A \cap I)$	I or $A \cap I$
	$X \leftarrow R - A$	$(R - I) - A = (R - A) - I = (R - A) - (I - A)$	I or $I - A$
	$X \leftarrow A - R$	$A - (R - I) = (A - R) \cup (I \cap A)$	$I \cap A$
	$X \leftarrow R\%(list)$	$(R - I)\%(list) \neq R\%(list) - I\%(list)$	R must be made explicit
	$X \leftarrow A * R$	$A * (R - I) = (A * R) - (A * I)$	$A * I$
	$X \leftarrow R:(filter)$	$(R - I) : (filter) = R:(filter) - I:(filter)$ $= R:(filter) - I$	I or $I : (filter)$

Figure 2—Table CHANGE for procedure alter. A is a relation; R is the relation to be updated; I is the relation of tuples to be inserted or deleted; RUI is the process of update by insertion; R-I is the process of update by deletion; IN is the resulting updating relation to be passed to lower levels. Given the definition, the updating relation IN is found. If a complex expression defines X in terms of R, the expression is broken down into simple forms.

updated i.e. definitions of the type $X \leftarrow A - R$, where R is the relation to be updated. In this type the insertion is carried out as follows:

1. Either:
 $IN = I$ is passed to lower levels.
 Or:
 Alternatively, $IN = I \cap A$ will be passed to lower levels (as in table CHANGE).
 The updating relation (IN) is passed to the lower levels as a deletion and the process continues as a deletion operation.
2. If the relation to be updated (X) is explicit, the updated value will be $X - IN$. This applies to all the relations at lower levels. If X is implicit, updates are passed down the hierarchy without materializing X .

Example 2 in the Appendix illustrates this type of definition.

Deletions

In this case the update may be specified by providing either a relation (I) containing the tuples to be deleted or boolean filter which selects the tuples to be deleted from the relation to be updated. These extracted tuples constitute the updating relation (I).

In a fashion similar to the previous section's, given relations A and R , let $R - I$ be the updated value of R . Let $X = f(A, R)$ be a defined relation. Relation IN is sought such that $X - IN$ is the updated value for X which corresponds to the update of R . Such IN can be found for all operators, f , as in table CHANGE (Figure 2).

When f is intersection, join or projection $IN = f(A, I)$, i.e. X need not be present in order to evaluate IN . The mechanism of the deletion is the same as that of the insertion (described in the previous section) except for the following:

- In (b) the union is replaced by the difference.
- In (1) only the alternative method is applicable. The updating relation is passed to lower levels as an insertion.
- In (2) the difference is replaced by the union.

Change in object value

A change in object value (modification) can be conceptually considered as deletion followed by insertions.

The algorithm

The algorithm for insertions and deletions from higher levels follows:

```
procedure alter (R,I, DELT) recursive;
  boolean DELT, DELN: relations R,I,IN;
  comment R is the relation to be altered
```

```
I is the relation of tuples to be inserted or
  deleted from R
n is the number of relations defined on R
Dj is the jth definition on R
Xj is the subject of Dj, i.e. the jth relation
  defined.
DELT, DELN true if deletions are to be
  made from R false if insertions
  are to be made to R.
DELj is true if an insertion to R
  implies a deletion from Xj
CHANGE is a procedure which calls
  table CHANGE. Given the
  definition Dj, the updating
  relation I and DELT, the
  procedure returns the tuples to
  be inserted or deleted;
```

```
if R is explicit then begin
  if DELT then R: =R-I
  else R: =RUI
end;
for j:=1 step 1 until n do
begin
  DELN: =DELT#DELj
  IN: =CHANGE (Dj,I,DELT);
  if IN#null
  then alter (Xj,IN, DELN)
end
end alter;
```

The algorithm for changes in object value follows from the previous algorithm.

Discussion

The great advantage of the approach described above is that only the explicit forms are updated. Implicit relations will not be recreated just for the purpose of having them updated. This saves the cost of creating all the relations down the hierarchy. Hence, in very large data bases the advantages of defined relations will not be outweighed by the substantial cost of recreating implicit relations whenever an update occurs. In such an environment, the overhead of storing table CHANGE and executing the update algorithm is negligible.

It is noteworthy that table CHANGE will have entries for all the user-defined operators. One problem, however, is to find rules for dealing with the complex definitions. No attempt has been made to explore this subject in the present paper.

Update at a lower level

Here the problem is how to pass the update up the hierarchy without introducing inconsistency. It is a logic problem. In addition to the conditions imposed on ordinary updates, e.g. the compatibility of the updating relation and the

relation to be updated, the following conditions should be satisfied for updating relations at lower levels:

1. The update should not result in incomplete information at upper levels.^{6,7}

Consider the following example:

Given relations X, Y and R.

	Y	X	R
2	C 5	3 A 8	3 A
3	A 8	6 B 8	6 B
6	B 8		

Definitions:

$X \leftarrow Y: (\text{field}(1) > 2)$
 $R \leftarrow X\% (\text{fields } 1 \text{ and } 2)$

To update R by I

4 M
5 T

and X by I'

9 K 3

When relation R is updated, the union of the updating tuples (I) and relation X cannot be formed because these tuples supply information for only two of the components of X. In such cases the update will provide incomplete information and should therefore be prohibited. Updating X, however, does not lead to missing information in upper levels. The system should prompt the user to update the relation at the lowest level whose update does not violate Conditions 1 and 2.

2. No ambiguity should result at higher levels due to the update.

S	Q	Z	I
1	4	1	9
6	5	6	3
2	2	2	
		4	
		5	

Definition: $Z \leftarrow \text{SUQ}$

e.g. if a relation Z is defined as the union of relation S and Q.

When Z is updated the data base system cannot readily know which of the updating tuples should update each of S and Q and which should update both relations. Whenever such an ambiguity exists, the update operation must be prohibited.

3. The updating tuples must satisfy the definition of the relation to be updated. In the example in (1) above, if

relation X is updated by relation P,

P		
9	K	3
1	T	6

the system must reject the last tuple because it contradicts the definition of the relation to which it will belong. Similarly, if a tuple is to be deleted from such a relation, an equivalent tuple must exist in the relation; otherwise the deletion is meaningless.

With these constraints, more weight is attached to the consistency of the data base information at the expense of the user convenience. Indeed, inconsistency in itself, regardless of any other repercussions, may perhaps lead to more inconvenience to the user. With updating at a lower level, almost every operator in the definition requires different updating algorithms as will be explained in Figure 4. In all cases the update can be passed up without explicit values of the relations involved. For joins, the upper level relations must be implicit if side effects of updates are to be avoided.

A more formal but less complete discussion of these problems, based in part on an earlier version of this paper, is given by Todd.⁵ The relational operators may be divided into regular and irregular operators according to their performance when relations having these operators in their definitions are updated. Regularity is a property of the type of update as well as of the form of operation.

The regular and irregular operators

Regular operators have two major properties when they appear in the definition of the relation to be updated:

- a. They do not lead to ambiguity.
- b. The updating information can be passed to higher levels.

When the operator is regular we will have the advantages of the updates at higher levels. When a definition contains an irregular operator, the problem of ambiguity arises. It is interesting to note (Figure 3) that with the exception of the selection operator, those operators that are irregular with insertion are regular with deletion and vice versa.

Validity of updates

In certain cases of join and selection, even regular updates can make the data base invalid or have undesirable side effects. These can only be checked given particular data values.

Consider a relation I inserted into relation X, $X \leftarrow A: (\text{filter})$. If the tuples of I do not satisfy the filter an inconsistency will be introduced. If such a set I is deleted

Insert:		
DEFINITION	PROOF	
$X \leftarrow A \cup B$	$X \cup I = (A \cup I) \cup B = A \cup (B \cup I)$	irregular
$X \leftarrow A \cap B$	$X \cup I = (A \cup I) \cap (B \cup I)$	regular
$X \leftarrow A - B$	$X \cup I = (A \cup I) - (B - I)$	regular
$X \leftarrow A\%(list)$	$X \cup I = A \cup (Q\%(list)), \text{ many } Q$	irregular
$X \leftarrow A * B$	$X \cup I \subseteq (A \cup I\%(fields(A))) * (B \cup I\%(fields(B)))$	regular
$X \leftarrow A:(filter)$	$X \cup I \supseteq A \cup I:(filter)$	regular
Delete:		
$X \leftarrow A \cup B$	$X - I = (A - I) \cup (B - I)$	regular
$X \leftarrow A \cap B$	$X - I = (A - I) \cap (B - I)$	irregular
$X \leftarrow A - B$	$X - I = (A - I) - B = A - (B \cup I)$	irregular
$X \leftarrow A\%(list)$	$X - I = (A - I)\%(list)^\dagger$	regular
$X \leftarrow A * B$	$X - I \subseteq (A - I\%(fields(A))) * B$	irregular
	$X - I \subseteq A * (B - I\%(fields(B)))$	
$X \leftarrow A:(filter)$	$X - I = (A - I):(filter)$	regular

Figure 3—Regular and irregular operators for updates at a lower level.

† A-I, generalized difference, those tuples of A that do not project into I.

from X, only tuples satisfying the filter must be deleted from A (the remainder cannot be deleted from X in any case, as they could not have been X to start with).

If a relation I is inserted into a relation X, $X \leftarrow A * B$, tuples are inserted into A and B. This may have the side effect of inserting tuples (E) into X. ($E = I\%(fields(A)) * B \cup A * I\%(fields(B))$.) Whether or not this is desirable depends on the application. It can be checked by reference to A and B without materializing X.

Other cases of regular updates cannot directly cause validity or side effect problems.

Discussion

The table in Figure 4 shows the update form of the defining relations. Each defining relation has its own form of update. Updates from lower levels of the hierarchy should be care-

DEFINITION	NEW FORM AFTER UPDATE
Insertion:	
1. $X \leftarrow A \cap B$	$\begin{cases} A \cup I \\ B \cup I \end{cases}$
2. $X \leftarrow A - B$	$\begin{cases} A \cup I \\ B - I \end{cases}$
3. $X \leftarrow A * B$	$\begin{cases} A \cup I\%(fields(A)) \\ B \cup I\%(fields(B)) \end{cases}$
4. $X \leftarrow A:(filter)$	A \cup I
Deletion:	
1. $X \leftarrow A \cup B$	$\begin{cases} A - I \\ B - I \end{cases}$
2. $X \leftarrow A\%(list)$	A - I (a generalized difference; those tuples of A which do not project into I).
3. $X \leftarrow A:(filter)$	A - I

Figure 4—CHANGE table for updates at a lower level. Only the regular operators are shown.

fully checked to ensure that the operators in the definitions involved are regular and that the updating information is logically valid. With large data bases such overheads are tolerable because of the probable avoidance of recreating intermediate implicit relations. However, with small sized data bases it may be sensible to prohibit updates from lower levels.

CONCLUSIONS

This paper has attempted to investigate the major aspects of the update problem for data bases with defined relations capabilities. It has been shown that in the majority of the cases implicit relations can be updated at higher levels of the hierarchy without the need to create their explicit forms. No ambiguity or inconsistency will arise from updates at higher levels.

However, updates at lower levels may lead to ambiguity in some situations. Checks are also needed to establish the validity of the updating information. In very large data bases updating the defined relations in the manner described in this paper will lead to substantial saving in the computer system resources. It is hoped that this paper will initiate some research in this interesting and practical problem. Thorough investigation is needed for the whole problem of lower level updates. Algorithms for handling complex definitions are also required.

ACKNOWLEDGMENTS

The author is indebted to Barry Aldred of IBM Scientific Research Center (UK), to the referees for their valuable suggestions and to Stephen Todd for editing this paper.

REFERENCES

- Codd, E. F., "A relational model of data for large shared data banks," *Communication of ACM*, Vol. 13, No. 6, June 1970.
- Todd, S. J. P., "The Peterless Relational Test Vehicle—a system overview," *IBM Systems Journal*, Vol. 15, No. 4, 1976.
- Boyce, R. F., and D. D. Chamberlin, "Using a structured English query language as a data definition facility," IBM, San Jose, California, RJ-1318, December 1973.
- Casey, R. G., and I. M. Osman, "Replacement algorithms for storage management in relational data bases," *The Computer Journal*, Vol. 19, No. 4, November 1976.
- Todd, S. J. P., "Automatic Constraint Maintenance and Updating Defined Relations," *Proceedings of IFIP Congress 77*, B. Gilchrist (ed.), North Holland, 1977.
- Chamberlin, Gray and Traiger, "Views, authorization and locking in a relational data base system," IBM, San Jose, RJ 1486, December 1974 and *Proceedings of NCC Conference*, May 1975.
- Paolini, P., and G. Pelagatti, "Formal definition of mappings in a data base," *SIGMOD International Conference on Management of Data*, Toronto, Canada, 1977.

APPENDIX

Relation *SUPPLIER*

S#	SNAME	STATUS	CITY
S1	SMITH	20	LONDON
S2	JONES	10	PARIS
S3	BLAKE	10	PARIS
S4	CLARK	20	LONDON
S5	ADAMS	30	ATHENS

Relation *SUPPLY*

S#	P#	QTY
S1	P1	3
S1	P2	2
S1	P3	4
S1	P4	2
S1	P5	1
S2	P1	3
S2	P2	4
S3	P3	4
S3	P3	4
S3	P5	2
S4	P2	2
S4	P4	3
S4	P5	4

The hierarchy of the defined relations mentioned in the following examples has been shown in Figure 1.

Relation *OLDST*

CITY	STATUS
LONDON	20
KHARTOUM	15

Relation *OLDSP*

S#	P#	QTY
S1	P1	3
S2	P2	4
S6	P1	3
S7	P4	1

Example 1—(Insertion at a higher level)

Definitions:

- (i) $SC \leftarrow SUPPLIER\%(CITY, STATUS)$
- (ii) $FULLST \leftarrow OLDST \cap SC$

Before Update:

Relation *SC*

CITY	STATUS
LONDON	20
PARIS	10
ATHENS	30

Relation *FULLST*

CITY	STATUS
LONDON	20

Relation *I* (The updating tuples to be inserted in relation *SUPPLIER*)

S#	SNAME	STATUS	CITY
S6	AHMED	15	KHARTOUM
S7	KIM	35	TOKYO

After Update:

- (1) Update relation *SUPPLIER*
 $SUPPLIER := SUPPLIER \cup I$

- (2) Compute new *I* by substituting *I* in the definition for the relation to be updated.

$I := I\%(CITY, STATUS)$	<i>I</i>
	KHARTOUM 15
Update SC. $SC := SC \cup I$	TOKYO 35
	<i>SC</i>
	LONDON 20
	PARIS 10
	ATHENS 30
	KHARTOUM 15
	TOKYO 35

- (3) $I := OLDST \cap I$

<i>I</i>	<i>I</i>
	KHARTOUM 15
Update FULLST $FULLST := FULLST \cup I$	
	<i>FULLST</i>
	LONDON 20
	KHARTOUM 15

Example 2—(Insertion and deletion at a higher level)

Definitions:

- (i) $NEWSP \leftarrow OLDST - SUPPLY$
- (ii) $R \leftarrow NEWSP\%(1)$

Before Update:

Relations *SUPPLY* and *OLDSP* are as above

NEWSP

S#	P#	QTY
S6	P1	3
S7	P4	1

R

S6
S7

After Update:

update *SUPPLY* by *I*

<i>I</i>	S#	P#	QTY
	S6	P1	3
	S8	P2	1

- (1) $SUPPLY := SUPPLY \cup I$
- (2) following the proposed method, *I* either remains unchanged, or $I := OLDSP \cap I$.

<i>I</i>	<i>I</i>	<i>I</i>
	S6 P1 3	<i>NEWSP</i>
	S8 P2 1	S7 P4 1
	$NEWSP := NEWSP - I$	

- (3) the update continues as deletion.

$I := D := I\%(1)$	<i>D</i>
	S6
	S8

$R := R - I$

<i>R</i>
S7

Example 3—(Insertion at a higher level, with implicit relations)

Definitions—Same as Example 1.
 Assume that relation SC is implicit.

Before Update—As in Example 1
 (1) $SUPPLIER := SUPPLIER \cup I$
 (2) $I := I \%(CITY, STATUS)$

I
 KHARTOUM 15
 TOKYO 35

SC is implicit. We do not create it for the update. Continue to the next level.

(3) $I := OLDST \cap I$

FULLST := FULLST \cup I

I
 KHARTOUM 15

FULLST
 LONDON 20
 KHARTOUM 15

Performance enhancement for relational systems through query compilation*

by RANDY H. KATZ

University of California, Berkeley
Berkeley, California

INTRODUCTION

In recent years, considerable research has been directed toward the relational model of data first proposed in Reference 5. The advantages of this approach have been discussed elsewhere, however they can be summarized as follows: (1) the user is presented with a very simple view of his data (i.e., organized as tables of information), (2) the user is freed from explicitly knowing the underlying implementation structure of his data (i.e., data independence) and (3) very powerful non-procedural, set-oriented query languages can be defined for the relational model because of its simple conceptual structure.

While commercial versions of data base systems based upon hierarchical¹⁸ and network¹⁷ models have existed for several years and are used widely,^{9,8} at this time there exist few commercially available relational systems. Two relational prototype systems currently in operation are the INGRES system⁷ and System-R.³

A major criticism of the relational approach is that the model does not lend itself to efficient processing because there is no provision for the user to explicitly "navigate" through access paths in his data. Until recently, the experimental systems previously mentioned were not overly concerned with efficiency. However, efficiency has now become an important goal.

Hence, from direction of relational data base efficiency, motivation is provided for compilation of data base queries. The goal is to move as much query processing to compile-time as possible, thereby reducing the overhead of query execution at run-time. However, there are several problems. Some query processing algorithms are data-dependent. For example, the decomposition algorithm of Reference 19 makes use of the cardinalities of relations, rather than statistical information, in selecting among alternative plans to implement a query. Such algorithms may actually suffer in performance if the processing strategy is determined at compile-time rather than with the more perfect information available at run-time. Further, any change in the data base schema will invalidate some compiled queries. These include

changes to the authorization rights of a given user, changes to views of the data base one is allowed to see and changes to the physical structure of the data. Note that in many data base applications, the structure of the data is relatively static over time. Compilation can be most effective in these cases. Even given these problems, compilation techniques should improve relational system performance.

The goal of this paper is to analyze the problems of query compilation in the specific environment of the INGRES data base system. A proposal was implemented and its space and time efficiency is analyzed.

PREVIOUS WORK

In the early work on relational systems, high level non-procedural query languages were provided to process ad-hoc queries presented interactively by non-programmers.^{5,6} Current work is directed towards applications-oriented languages for programmers, which integrate relational query capabilities into programming languages. These languages are meant to complement conversational query languages. In this situation, it is possible to take advantage of the more static non-interactive environment to improve the efficiency of query processing.

In order to classify and to compare the work which will be summarized, it is useful to define "levels of compilation." Query processing consists of the following steps:

1. The query string is analyzed (i.e., parsing).
2. Object names are converted to an internal form via System Catalogs (i.e., lookup).
3. A processing plan is built.
4. The processing plan is executed by calls to the access methods.

Using this, five levels of compilation can be defined:

- 0—The query is completely interpreted at run-time (i.e., Steps 1-4 are performed at run-time).
- 1—The internal form of the query is built at compile-time (i.e., the query is converted from a string to a parse tree at compile-time).

* Research supported by the Army Research Office under Grant DAAG29-76-G-0245.

- 2—The internal form is built at compile-time, with name resolution taking place if possible. Otherwise, Step 2 will have to be executed when the information is available at run-time.
- 3—The internal processing plan is determined at compile-time and translated into an internal form suitable for run-time interpretation.
- 4—The internal processing plan is represented within the user program by calls to the underlying access methods.

Level 0 represents the least amount work done at compile-time, namely complete interpretive execution of the query at run-time. Level 4, on the other hand, represents the movement of the most amount of processing to compile-time.

In the remainder of this section, we describe several existing or proposed data base programming languages and describe what level of compilation they support or propose to support.

ASTRA¹²

ASTRA is a relational system built on a hierarchical system at the University of Trondheim, in Norway. It supports a high-level applications-oriented language called ASTRAL (A STRUCTURED Relational Applications Language), based upon SIMULA. The compiler maps an ASTRAL program into a SIMULA program augmented by subroutine calls to the underlying data base system. It is the job of the compiler to choose the best access paths when translating relational expressions into sequences of host language statements and subroutine calls. The functions of query parsing and of determining a query processing plan are moved to compile-time. The plan, represented by calls to the underlying system, is interpretively executed at run-time. Hence, the ASTRAL compiler supports Level 3 compilation. No mention is made of the recompilation problems associated with changes to the data base schema.

Extended PL/1^{16,10}

Reference 16 describes extensions to PL/1 which allow an applications programmer to access data from a relational data base. The language is extended by introducing the notion of a template, which is a PL/1 structure containing data base relation names and attribute names, along with a SELECT statement, that qualifies those tuples which are to be retrieved. In Reference 10 the problems of recompilation under a changing schema are explored. The authors point out that the performance advantages for a compilation approach are somewhat offset by the increased sensitivity of a compiled program to schema changes. They propose that the program be separated into three modules—one for applications-oriented processing, one for data base interactions and one for authorization and integrity enforcement. If there is a change in some aspect of the schema, only the relatively

small module of code which is effected by that change will have to be recompiled. This approach has actually been used in a system to support Extended PL/1. An architecture similar to this proposal will be necessary for any system which attempts to provide Level 3 or Level 4 compilation, because the internal processing plan will become invalid if the schema changes. Note, however, that the proposed architecture implies many inter-module communications. This overhead may outweigh the advantages of compilation.

System-R¹⁻⁴

System-R is an interesting system from the standpoint of compilation because it originally supported only interpretive query execution, but now supports both compiled and interpretive queries. The origin of System-R can be found in SEQUEL-XRM,^{1,2} a single-user relational system designed to support the SEQUEL query language.⁴ The system interpretively mapped the non-procedural statements of the query language into tuple at a time commands to the underlying relational memory system, XRM (extended n-ary relational memory). The interpreter was organized into three modules—parser, optimizer and scanner. The parser translated a SEQUEL query into a corresponding parse tree. The optimizer, when given a query tree, selected a processing plan which would minimize the number of tuples retrieved based upon the available access paths. The scanner actually executed this plan by judiciously scanning the underlying relations.

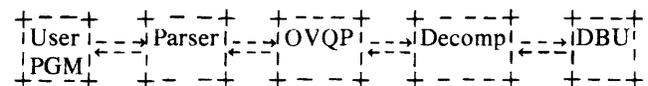
Note that the parsing step is independent of whether the query is being interpreted or compiled. Furthermore, if the optimization step is independent of the actual data in the data base then the optimized processing plan can be determined at compile-time as well. It then becomes a simple matter to compile the scanner into a query specific data access routine. This approach was taken in System-R.¹¹ Because the data access routines consist essentially of access method calls, System-R supports Level 4 compilation.

All data access routines are separated from the user program to facilitate recompilation when the data base schema changes. A routine is marked invalid by the system if it depends on the schema change and is recompiled the next time it is called. The recompilation process is simplified if the query tree is saved along with the data access routine. This is because only the optimization and code generation phases need to be re-executed.

COMPILATION FOR INGRES

In this section, a scheme for the compilation of QUEL, the data sub-language of the INGRES Data Base System,⁷ is presented. Complications due to the compilation approach are then examined.

The INGRES system consists of five concurrently executing processes:



These are the user program, the parser, the one-variable query processor, decomposition and the data base utilities. The parser converts query strings into trees. OVQP interpretively executes one-variable queries, i.e., those involving a single relation. Decomp controls the decomposition of multivariable queries into a sequence of one-variable queries. The DBUs provide utility support. Communication between the processes is accomplished by "pipes," which are essentially message buffers.

Borrowing from System-R terminology, there are three phases in the lifetime of a program—preprocess-time, compile-time and run-time. Preprocess-time refers to the time when data manipulation statements of a program are first analyzed and translated. Compile-time refers to the time when the actual host language program, with associated data base calls, is compiled. Run-time refers to the time when the program is in execution. Compilation of data base queries is most advantageous if it can be performed at preprocess-time. However, flexibility considerations may make it necessary to defer compilation of the data base portion of a program until run-time, when complete information is specified.

It is possible to describe the various levels of compilation presented in the previous section in terms of the INGRES system:

Level 0—INGRES interpretively executes all queries. The QUEL preprocessor translates a user program into a sequence of host language statements and calls on the underlying data base system. These calls pass the queries in the form of character strings to INGRES for run-time execution.

Level 1—INGRES translates the query string into a tree at run-time. The structure of the query tree can always be determined from the string. Thus, it is possible to parse the query and build the tree at preprocess-time. If the data base, relation, and attribute names are known then information associated with those names can be placed into the tree. If we make the restriction that this information must be known at preprocess-time, Level 1 compilation can be achieved.

Level 2—If the restriction that all names must be known at preprocess-time is lifted, then it is necessary to fill in missing information at run-time when all names must become known. Thus Level 2 compilation can be accomplished. Note that the form of the tree is unaffected by the lack of information at preprocess-time. The information in the nodes of the tree, on the other hand, depends upon information associated with unresolved names. These names must be looked up in the system catalogs at run-time.

Level 3—INGRES constructs an execution plan by decomposing a complex query into a sequence of one-variable queries. This is accomplished by alternatively applying tuple substitution and reduction.¹⁹ Once a collection of decomposed query trees is available, it is possible to generate code at preprocess-time which will perform the decomposition at run-time. Because tuple substitution can not be performed until run-time, the generated code will have to be parameterized in order to allow actual values from the data base to be substituted. It is useful to think of the generated code as query procedures which can be invoked by other query procedures, with the lowest level of invocation representing

parameterized one-variable queries. These queries can then be passed to the OVQP for interpretive execution. Note that the invocation of the compiler to build these query procedures will have to be postponed until run-time unless complete information is available at preprocess-time.

Level 4—Based upon the physical structure of the underlying relations, OVQP determines the efficient access paths for the processing of a one-variable query. Once the query procedures are constructed, it is possible to generate access method calls for the query procedures. Thus, Level 4 compilation is easily achieved once Level 3 compilation is operative.

At this point, we will give an example of Level 4 compilation. Consider the following QUEL query:

```
range of E,M is employee
range of D is department
retrieve (E.name) where E.salary>M.salary
                        and E.manager=M.name
                        and E.dept=D.dept
                        and D.floor#=1
                        and E.age>40
```

The query requests the names of all employees over 40 years old who make more than their managers and work in a department which is situated on the first floor. The following two queries can be detached from the original:

```
range of D is department
retrieve into T1 (D.dept) where D.floor#=1
range of E is employee
retrieve into T2 (E.name, E.salary, E.manager, E.dept)
                        where E.age>40
```

The original query becomes:

```
range of D is T1
range of E is T2
range of M is employee
retrieve (E.name) where E.salary>M.salary
                        and E.manager=M.name
                        and E.dept=D.dept
```

The decomposition algorithm chooses a variable for substitution. If D is chosen, D.dept is replaced by actual values from the data base. A parameterized query of the above form is decomposed further, with the value of D.dept being the parameter. Continuing, the following query is detached:

```
range of E is T2
retrieve into T3 (E.name, E.salary, E.manager)
                        where E.dept=value
```

The above query now becomes:

```
range of E is T3
range of M is employee
retrieve (E.name) where E.salary>M.salary
                        and E.manager=M.name
```


TABLE II.—Relative Sizes of INGRES Processes

Process	Size (K=1024 bytes)
Parser	54K
OVQP	60K
Decomp	57K
DBUs	58K

relative sizes of the major INGRES processes. Using this measure of program size, the QUEL programs required 224K bytes, the C-QUEL programs required 181K bytes, and AM programs required 24K bytes.

INGRES processes are actually shared among concurrently executing INGRES programs, since the processes are re-entrant. Thus, the storage requirements for QUEL and C-QUEL programs should be distributed among the concurrent users of the system. This would require approximately ten concurrent users of INGRES for an QUEL program to match the storage requirements of the AM program. The results indicate that for certain kinds of programs, the compilation approach may actually decrease the overall storage requirements for data base programs. This will depend upon the complexity of the program as well as the number of concurrent users. In any case, the storage requirements for compiled programs do not appear to be prohibitive.

The second comparison made was in speed of query processing. For each of the four systems, five different tests were run. These tests differed mainly on the test program used and the number of tuples scanned during the test. The first three tests consisted of the three test programs run on a relation with a single tuple. The remaining tests consisted of Test Programs 1 and 2 run with the same relation, but this time with 25 tuples. The timing results are listed in Table III.

TABLE III.—Test Run Elapsed Times

Program # (tuples scanned)	Compiler Used	Mean (sec)	Standard Deviation
1 (1 tuple)	QUEL	235.0	.7
	C-QUEL (5)	201.9	.9
	C-QUEL (4)	183.0	.5
	AM Code	4.5	.5
2 (1 tuple)	QUEL½	310.8	1.2
	C-QUEL (5)	224.6	.5
	C-QUEL (4)	205.1	.3
	AM Code	4.4	.5
3 (1 tuple)	QUEL	308.4	.9
	C-QUEL (5)	223.1	.6
	C-QUEL (4)	203.4	2.0
	AM Code	4.4	.5
1 (25 tuples)	QUEL	489.8	.5
	C-QUEL (5)	459.2	.7
	C-QUEL (4)	440.5	.7
	AM Code	28.8	.4
2 (25 tuples)	QUEL	641.3	1.4
	C-QUEL (5)	553.8	1.1
	C-QUEL (4)	532.9	1.8
	AM Code	28.8	.5

For each experiment, the four versions of the same program were run 20 times while there was no activity on the system. Elapsed time was used for the measurements because it is relatively easy to measure and because it is a good indicator of combined CPU and I/O time when there is little other activity on the system. The mean and standard deviation of the readings were computed and are reproduced in Table III. For easy comparison, bar graphs of these results can be found in Figure 1. Because the standard deviations are typically quite small (i.e. less than one percent), the mean run-times are a good basis for comparing the benchmarks.

Table IV is a compilation of the percentage improvements between two given configurations. The improvement due to parsing at compile-time is presented in Line 1. The improvement due to eliminating the parser process is listed in Line 2. The total improvement of C-QUEL over QUEL is available as Line 3. The improvement of complete compilation over interpretation is recorded in Line 4.

The advantages of parsing at compile-time are obvious from Table IV. The more complex the query is, in terms of the time it takes to parse the query string, the greater the improvement possible with C-QUEL. This is illustrated in the percentage improvement for Run 1 and Runs 2 and 3. However, because parsing represents a fixed overhead, as the total time to process the query increases, the improvement due to compile-time parsing decreases. This is evident from the improvements for Run 1 and Run 4.

The major difference between C-QUEL (5) and C-QUEL (4) is the inclusion of a dummy parser process that copies data from input to output. The improvement represents the benefits gained by reducing the size of the system by one process. As can be seen from the results of Runs 1, 2 and 3 and Runs 4 and 5, the overhead associated with the existence of an extra process is essentially fixed. As the

TABLE IV.—Percent Improvements

Program # (tuples scanned)	Compiler 1	Compiler 2	Percent Change
1 (1 tuple)	QUEL	C-QUEL (5)	14
	C-QUEL (5)	C-QUEL (4)	9
	QUEL	C-QUEL (4)	22
	QUEL	AM Code	98
2 (1 tuple)	QUEL	C-QUEL (5)	28
	C-QUEL (5)	C-QUEL (4)	9
	QUEL	C-QUEL (4)	34
	QUEL	AM Code	98
3 (1 tuple)	QUEL	C-QUEL (5)	28
	C-QUEL (5)	C-QUEL (4)	9
	QUEL	C-QUEL (4)	34
	QUEL	AM Code	98
1 (25 tuples)	QUEL	C-QUEL (5)	6
	C-QUEL (5)	C-QUEL (4)	4
	QUEL	C-QUEL (4)	10
	QUEL	AM Code	94
2 (25 tuples)	QUEL	C-QUEL (5)	14
	C-QUEL (5)	C-QUEL (4)	3
	QUEL	C-QUEL (4)	17
	QUEL	AM Code	96

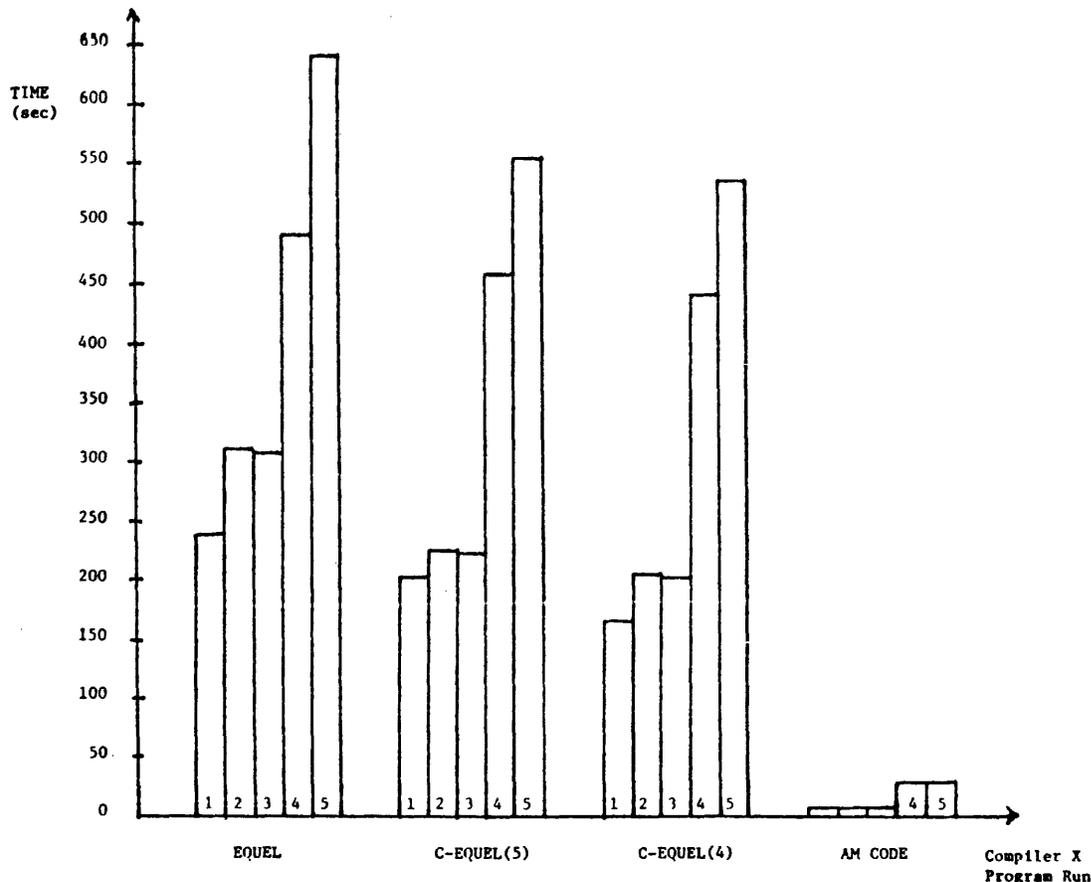


Figure 1—Comparison of program execution times.

total processing time increases, the associated overhead becomes less significant, as shown in Runs 4 and 5.

The improvement of C-EQUEL over EQUEL is represented by the percentage improvement of C-EQUEL (4) over EQUEL. In the case of a complex query with little associated processing, the improvement is over one-third. In the case of simple queries which require much processing, however, this improvement is considerably less.

The most striking result is the improvement of compiled programs over interpreted ones. The result that compiled programs will run faster than interpreted ones is not surprising; the magnitude of this improvement is. It indicates that there is considerable overhead associated with a multiple process interpretive system that has little to do with performing the actual functions necessary for query processing.

CONCLUSIONS

The compilation approach to query processing has the potential of greatly reducing the overhead associated with execution of queries. An actual parse at compile-time system was implemented and was shown empirically to reduce

query processing time, at a modest increase in storage requirements. This is only one possible point in a continuum of possible levels of compilation, each with its own time/space tradeoffs. Complete compilation was shown to produce substantial time savings, with possible space savings as well.

A general notion of compilation level was presented and it was shown how to describe an interpretation-based system such as INGRES in terms of different levels of compilation. On the basis of experience with C-EQUEL, some conclusions about the ease of implementing other proposed levels of compilation are possible. The extension of Level 1 to Level 2 would require little additional work. All that is needed is to look up names in the INGRES system catalogs. The extension of Level 2 to Level 3 would entail a considerable implementation effort. Programs would become sensitive to changes in the physical schema. Decomposition would have to be executed at preprocess-time in order to generate the query procedures. Further, the compiler would have to be invocable at run-time, because data base, relation and attribute names may not be known until then. Once these implementation obstacles have been surmounted, Level 4 is relatively easy to implement. All necessary information is known when the Level 3 program is created.

Essentially, code is generated directly, as opposed to being generated with conditional statements to control execution.

REFERENCES

1. Astrahan, M. M., and D. D. Chamberlin, "Implementation of a Structured English Query Language," *CACM*, Vol. 18, No. 10, October 1975, pp. 580-587.
2. Astrahan, M. M., and R. A. Lorie, "SEQUEL-XRM, A Relational System," *ACM Pacific Conf.*, 1975, pp. 34-38.
3. Astrahan, M. M., et. al., "System R: A Relational Approach to Database Management," *TODS V1 N2*, June 1976 (pp. 97-137).
4. Chamberlain, D. D., et. al., "SEQUEL2: A Unified Approach to Data Definition, Manipulation, and Control," *IBM Research Report RJ 1978*, June 1976.
5. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *CACM*, Vol. 13, No. 6, June 1970, pp. 377-387.
6. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proc. 1971 ACM SIGFIDET Workshop*, 1971.
7. Held, G. D., M. R. Stonebraker and E. Wong, "INGRES—A Relational Data Base System," *NCC 75*, 1975, pp. 409-416.
8. IDMS Sales Literature, Cullinane Corporation.
9. International Business Machines, Inc., "Information Management System/Virtual Storage (IMS/VS) General Information Manual," Form # GH20-1260-3, 1975.
10. Lang, T., E. B. Fernandez and R. C. Summers, "A System for Compile-Time Actions in Databases," *Proc. ACM 77 Conf.*, October 1977, pp. 11-15.
11. Lorie, R. A., and B. W. Wade, "The Compilation of a Very High Data Language," *IBM Research Report RJ2008(28098)*, May 1977.
12. Risnes, O., and K. Bratbergensengen, "ASTRA—A DBMS Based on a High Level, Relational DML with Data Access via a Hierarchical DDL," *Proc. SIMULA Users Conf.*, September 1977.
13. Stonebraker, M. R., and E. Wong, "Access Control in a Relational Data Base Management System by Query Modification," *Proc. 1974 ACM Conf.*, November 1974.
14. Stonebraker, M. R., "Implementation of Integrity Constraints and Views by Query Modification," *Proc. 1975 SIGMOD Workshop*, May 1975.
15. Stonebraker, M. R., et. al., "The Design and Implementation of INGRES," *TODS Vol. 1, No. 3*, September 1976, pp. 189-222.
16. Summers, R. C., C. D. Coleman and E. B. Fernandez, "A Programming Language Extension for Access to a Shared Data Base," *ACM Pacific Conf.*, April 1975, pp. 114-118.
17. Taylor, R. W., and R. L. Frank, "CODASYL Data-Base Management Systems," *ACM Computing Surveys*, Vol. 8, No. 1, March 1976.
18. Tsichritizis, D. C., and F. H. Lochofsky, "Hierarchical Data-Base Management," *ACM Computing Surveys*, Vol. 8, No. 1, March 1976, pp. 105-124.
19. Wong, E., and K. Youseffi, "Decomposition—A Strategy for Query Processing," *TODS*, Vol. 1, No. 3, September 1976, pp. 223-241.

System considerations for predicting mass storage subsystem behavior

by E. J. McBRIDE, A. B. TONIK and G. R. FINNIN

Sperry Univac
Blue Bell, Pennsylvania

INTRODUCTION

We are always interested in predicting the performance of systems and/or subsystems. A "typical" question is: "Is the most important component of system performance the way mass storage subsystems behave?" The answer is yes, most of the time.

Assume a solitary program is running on a system. The running of the program is broken into two major parts: the loading and the running. Loading usually takes between $\frac{1}{4}$ to $\frac{1}{10}$ of the total run time. The exceptions are programs that run for an hour or more during which the load time is insignificant. During load time there is little CPU-time and during the total run time there is usually little overlap between CPU-time and I/O-time. The amount of CPU-time will usually range anywhere from $\frac{1}{6}$ to slightly more than $\frac{1}{2}$ of the I/O-time. The exceptions are programs that will load the data and just sit there and grind away on it. These time periods, shown in Figure 1, are actually pieces of unequal duration that are intermixed. This is true of a transaction program as well as a batch program.

Another consideration of performance is multiprogramming, i.e., the way one program multiplexes with many others. In this case, when a program wants to use one of the facilities of the system, that program may find it busy. In fact there may be several of the programs waiting for the use of that facility. The waiting time until a program gets to use that facility has to be added to the run-time of that program. The average queue waiting for service depends upon the average utilization of that facility and the mix of programs. The higher the utilization, the longer the queue. Since mass storage already takes a majority of the running time, we do not want to multiply the I/O-time by more than three or four because of long queues. On the other hand since CPU-time is relatively short, we do not mind multiplying that time by a large factor. Therefore, we can afford multiprogramming up to the point of using the CPU for 80 to 90 percent of the time while we try to keep disk utilization down to 50 percent.

Disk performance is important to system performance, but trying to predict its performance is very complicated. The control unit is used for a short period of time to get a

disk started toward a location. During disk seek time, the control unit can start seeks to many other disks or perform data transfers to/from them. At the end of a seek, the disk enters latency time. At this point, the disk will either attempt to interrupt the control unit (non-RPS case) or wait until it arrives at a rotational position. At RPS time (rotational positioning sensing), the disk tries to request service from the control unit. If the control unit is busy and the disk is using RPS, the disk may have to wait a revolution or more. This is caused by the disk not gaining access to the control unit within a given RPS time. The number of missed revolutions depends on the number of disks on a subsystem, the request rate of I/O commands to the subsystem, the RPS timing constraints and the amount of the data to be transferred. These are related in very complex ways.

This paper is written to document some of the findings from our studies of Mass Storage Subsystem behavior. It is not intended to be an analytical treatise of Mass Storage behavior. It is written more as an attempt to describe some of the techniques we are using. GPSS models and analytical models were constructed. The study of these models yielded some interesting observations into subsystem behavior. A typical question is how many missed revolutions will an "average" disk have when attached to a system. Since each missed revolution costs 16.6 milliseconds on conventional disks, the answer to such a question can have a significant impact on system performance.

Figure 2 shows an example of the complex set of events which occur during normal disk accessing. The times shown are not necessarily to scale. Note that access completions need not be in the same order as initiations. Some disks may complete without missing revolutions while others may miss many. The remainder of this paper describes a method of studying this type of subsystem plus the effects of some of the critical parameters.

From a systems level the interesting parameters of a subsystem are its response time and its throughput. Response time as measured from the time an I/O is requested at the user level (not the control unit level) until the time the data is in memory. Throughput is the maximum number of requests per second which can be sent to a subsystem without having the queue build up beyond bounds.

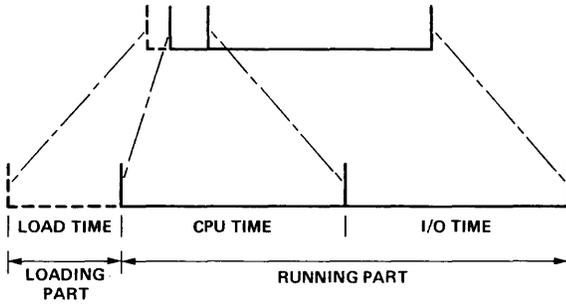


Figure 1—Typical distribution of time for a program.

MASS STORAGE SUBSYSTEMS

The term Mass Storage is used to refer to input-output (I/O) subsystems consisting of the rotating magnetic disk media or the new solid state technologies such as Bubble and Charge Coupled devices. The disk drives may have movable data recording and read back mechanisms (R/W Head) or fixed head mechanisms. The solid state devices are equivalent to the fixed head disks in that no physical positioning (seek) of the R/W Head is required prior to a data read or write operation.

The data are stored on these Mass Storage devices in what are called tracks. Figure 3a illustrates, in a simplified manner, the concept of data tracks on a movable head disk device. Each track contains a set of data blocks (four shown

in the figure) which are units of information read from or written to the device per operation. Each data block may contain one or more file records as specified by the application software. The number of blocks per track and the number of tracks per device are determined by the storage density of the device. The tracks are in continuous motion at a speed dictated by the design of the device. Now, in order to read or write a data block, the read/write head must be positioned over the desired track and the intended block must move under this head.

In Figure 3a, a seek (head positioning) has occurred from Track 1 to Track 200. Since this is mechanical motion, it tends to be very time-consuming; the time element is represented as S_{sk} (seek time). When the seek completes, the read/write head must wait for the particular data blocks. This also involves physical motion and is represented as S_l (latency time). In Figure 3a, the head is waiting for data block 3. When the data block reaches the head, time must be spent transferring the data to or from the block (provided the control facilities that connect the disk to system main storage are available for use). The data transfer time (S_d) is determined by the track speed, the density of storage on the track (i.e. bits per inch) and the size of the block. When the block transfer is completed, the operation is finished and the disk is free to accept a new operation.

In Figure 3b, the track concept is illustrated for the solid state technologies. There is no seek time, instead, the track is switched to a single read/write head. However, there is a latency time in moving the desired data block under the

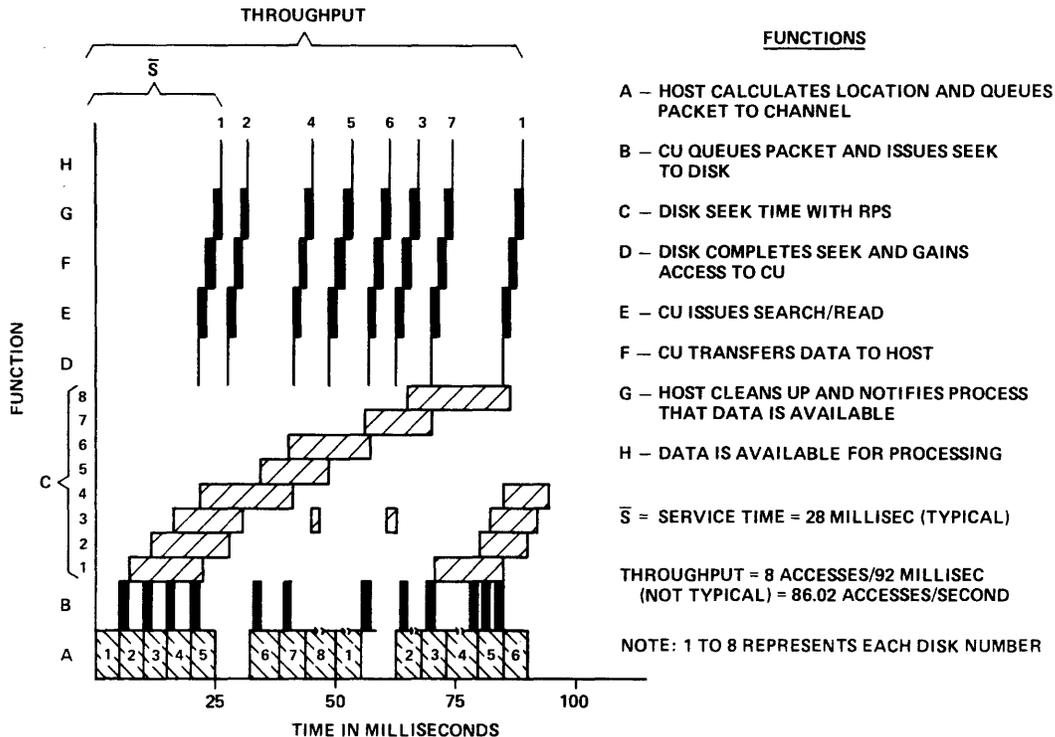


Figure 2—Performance factors.

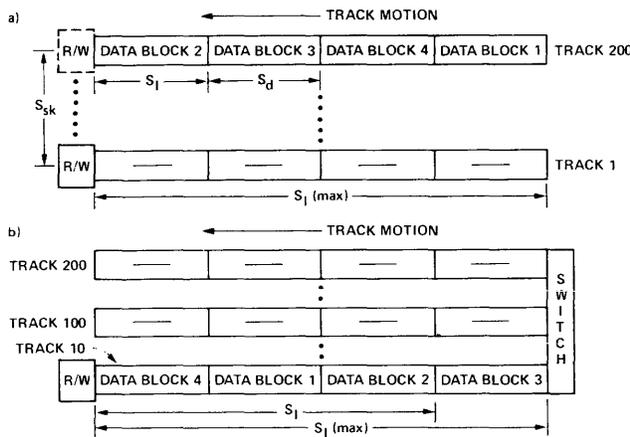


Figure 3—Logical representation of latency.

head. The maximum latency time ($S_1 \text{ max}$) for these devices ranges in the one to five milli-second time interval, whereas disks are typically in the 16 milli-second range. This shorter latency time, zero seek time and higher track speed and density give the solid state devices a definite performance advantage.

Storage devices are only one constituent of the subsystem. Each subsystem has at least one control unit. The control unit usually services more than one storage device and also connects to the system facility called a channel. The channel services the software queues requesting input or output (I/O), issues commands to the subsystem control unit to start I/O operations and controls the flow of data blocks to or from main storage. Channels may also share facilities among more than one subsystem. However, the real burden is placed on the control unit. It must service requests from the channel to start I/O and it must service requests from the storage devices to transfer data blocks. To obtain efficiency, control units are designed to overlap certain I/O operations (i.e. time multiplex its resource) but once it connects to a storage device for data block transfer, it remains connected until the transfer completes. The utilization of the control and channel facilities plays an important role in determining the performance of the subsystem. This aspect will be discussed later in the paper.

ROTATIONAL POSITION SENSING (RPS)

A storage device after completing its seek or track select operation must then attempt to gain use of the control unit and channel facilities. A device without RPS will attempt to obtain the control unit immediately at the end of its seek or track select. On the average, the read/write head will be one-half the maximum latency time from the desired data block. Nothing can happen until the data block moves under the head. In the non-RPS case the control unit like the device is used during this waiting period. This tends to increase the control unit utilization during each data transfer operation and tends to limit the number of requests the subsystem can handle.

To keep from wasting control unit time by forcing it to wait out latency, RPS can be used. Figure 4 illustrates the concept of RPS. The tracks are divided into sectors (the figure shows eight sectors, typically there are 128) and the storage device does not become a candidate for control unit service until a specific sector is reached. This allows the control unit to service other requests from the channel or other storage devices during RPS time. In the illustration, the read/write head is shown within Sector 3 when the seek or track select completes.

The storage device will not become a candidate for control unit use until Sector 3 is sensed. Since Sector 3 is under the read/write head the device attempts to obtain the control and channel facilities. If these facilities are free and have set up for the data transfer before Sector 4 is sensed, the device and control facilities stay connected through Sector 4 and transfer at data block 14 in Sector 5. Otherwise, the device must wait until its track rotates almost one full latency time (to Sector 3) and try again to obtain the control facilities.

CONFIGURATIONS

The term configuration refers to how the subsystem connects to the system. This is very important to performance evaluation. This paper will deal to two basic configurations: 1) the single access and 2) the dual access. Figure 5a illustrates single access. Shown is one channel facility servicing two single access mass storage subsystems. Single access means that the storage devices connect to only one control unit. It will be shown that this configuration significantly limits the number I/O requests a subsystem can handle per unit time.

In the illustration, the channel is shown interfacing a software facility and main storage. The software is shown servicing the subsystem queues which hold requests for I/O. The software processes the termination of each I/O operation, removes requests from queues and starts I/O operations on the channel facility. This software process is not considered

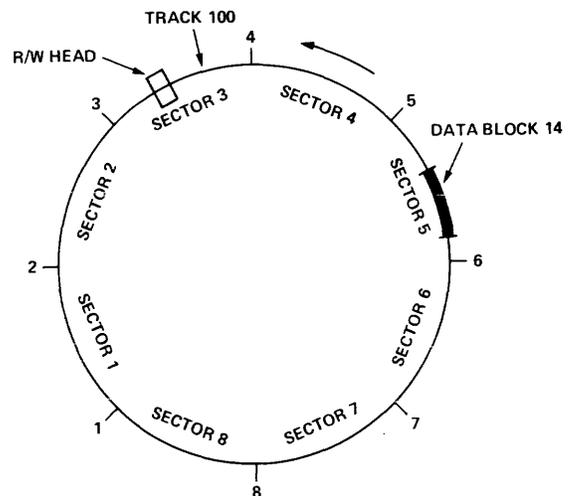


Figure 4—Sector positioning.

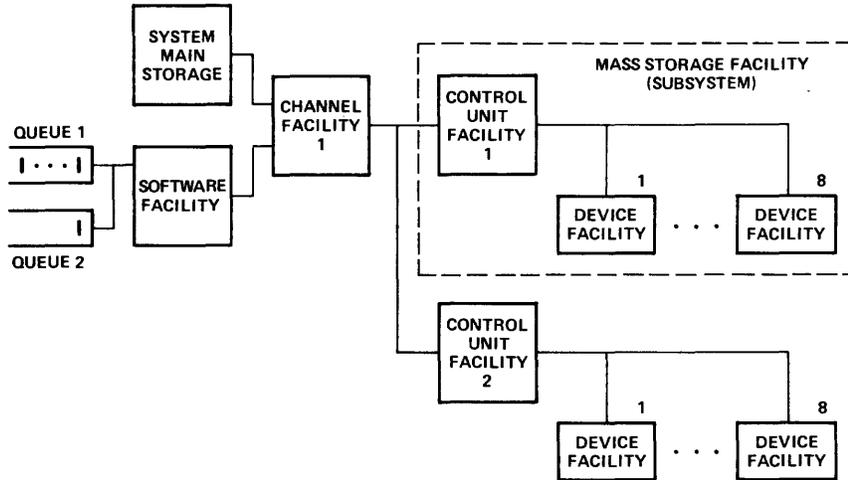


Figure 5a—Single-access subsystems configurations.

in this paper. The channel to main storage interface is shown to indicate the path for data block transfers. It too affects performance but will not be considered in this paper.

The control units are shown connecting up to eight device facilities. This value, typically, represents an efficient load on the control unit. However, cost-effective configurations with 16 or more devices per control unit are possible.

The second type of configuration (dual access) is illustrated in Figure 5b. Two channel facilities are shown connecting to two control units which share the eight storage devices. This connection, at the expense of hardware, enhances performance by effectively lessening the load on a single control unit. In general, the performance limitation is now predominately shifted to the service capabilities of the storage devices.

SUBSYSTEM ANALYTIC MODEL

When viewing subsystems performance, the concern is with individual storage devices and their software device queues. That is: 1) How long, on the average, must a request

sit on the device queue before being serviced by its device? 2) What is the average time for a request to pass through the subsystem? 3) What is the average size of the device queue? 4) How do the service time elements of the channel, control unit and device affect performance?

It is desirable to structure a model that is simple but takes account of the important parameters affecting performance. The model described in this paper views a single device queue and its storage device. The interference of other storage devices sharing the control unit and other subsystems sharing the channel are also included in the model.

Figure 6 represents a queuing model flow diagram for a subsystem storage device using RPS. The device queue resides in the system and holds requests for I/Os issued by the software. The queue is nothing more than a waiting line and obviously we would like the number of requests waiting in line to be small. Average time spent waiting on the queue for service is designated \bar{t}_w , and is dependent upon the rate at which requests enter the queue and the average time the subsystem takes to service the queue. The arrival rate to the group of device queues associated with a specific control unit is λ_r . This value is a random arrival and the device queues are not skewed. Each device queue receives on the average an equal fraction of λ_r . The arrival rate per device

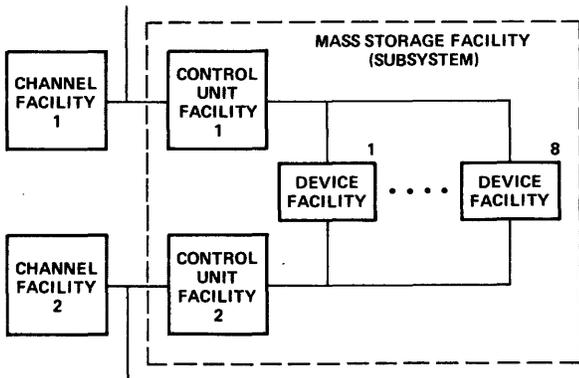


Figure 5b—Dual-access subsystems configurations.

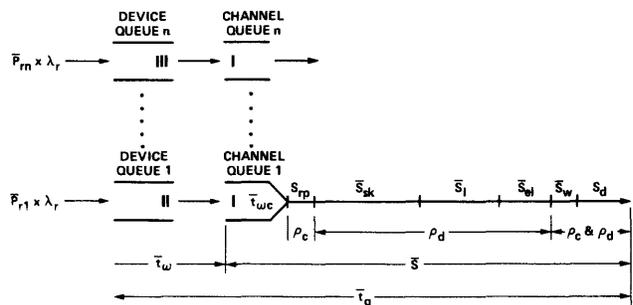


Figure 6—Time sequence for RPS model.

is $\lambda_r P_n$, where:

$$P_n = \frac{1}{N_d} \text{ where } N_d = \# \text{ of devices per control unit.} \quad (1)$$

The wait time (\bar{t}_w) can, under heavy load, exceed the time to service a request (\bar{S}). Service time (\bar{S}), as Figure 6 indicates, is comprised of seven time elements. The first time element is the mean wait time that a request, taken from the device queue, must experience before gaining access to the control unit (\bar{t}_{wc}). That is, when a request leaves the device queue, it enters a queue in the channel. This queue can hold only one request; however, the channel has a large number of such queues to handle many devices on one or more subsystems. \bar{t}_{wc} depends on the effective control unit utilization and the mean time the control takes to service requests. The effective control unit utilization (ρ_{ce}) is a measure of the average time the control unit appears to be busy to a request on the channel queue. If only one control unit is connected to the channel, ρ_{ce} equals ρ_c (the real control unit utilization). However, if other control units are also using the channel, ρ_{ce} is larger than ρ_c . This is a method used to factor in interference due to other subsystems sharing the channel facility. The service time of the control unit is the average of the time to process a command from the channel (\bar{S}_{rp}) and to transfer a data block to or from the device ($\bar{S}_w + S_d$).

When the control unit becomes available it processes the request (\bar{S}_{rn}). This action consumes some control unit time as indicated in Figure 6 by ρ_c under the \bar{S}_{rn} increment. The control unit then signals the storage device to seek to the track specified in the channel command. At this point, the control unit is free to service other requests and the storage device is now busy in its seek time interval. When the seek completes, the read/write head must wait for the track sector to position. This is the mean latency time (\bar{S}_l) and is equal to one-half the maximum latency time. The device now attempts to gain use of the control unit. If the control unit is busy and if the track sector slips past a given point, the track must reposition again to the specified sector. This requires one full latency time interval and is sometimes referred to as missed revolutions. The average of these missed revolutions serves to increase or extend the mean latency by an additional time interval depicted as \bar{S}_{el} . It is desirable to keep this time as small as possible.

When the device finally obtains the control unit, both the control and device must wait out any sector time preceding the data block to be transferred. This time is determined by the sector window time and has a mean value depicted by \bar{S}_w . In Figure 4 the maximum window time would be Sector 3 and 4 plus Sector 5 up to the start of Data Block 14. Window time should be small if RPS is to provide a performance advantage over non-RPS.

When the data block is reached, both control unit and device take part in the block transfer during period S_d . At block transfer completion, the I/O request leaves the model and the next request on the device queue can enter the channel queue. Figure 7 is a flow diagram for a non-RPS device. The model is the same as RPS up to the end of \bar{S}_{sk} . The device then attempts to gain use of the control unit and

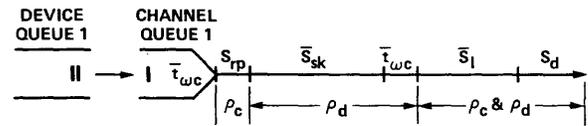


Figure 7—Time sequence for non-RPS model.

like the channel request, a mean wait time for the control unit is encountered (\bar{t}_{wc}).

Finally, when the control unit is obtained, both the control unit and device must wait out the mean latency time (\bar{S}_l). Here lies the disadvantage, the value of ρ_c increases because of device latency. This increases \bar{t}_{wc} and thus the service time. The increased \bar{S} and subsystem utilization causes an increase in the device queue waiting time.

MATHEMATICAL EXERCISE

Now that we have walked through the flow diagrams and have defined the terms relating performance, we are ready to handle the equations of the model. The main time element of interest is the average response time (\bar{t}_q) in performing an I/O request to a specific device. This measures the time waiting on the device queue and time being serviced by the subsystem device:

$$\bar{t}_q = \bar{S} + \bar{t}_w, \quad (2)$$

\bar{t}_q is referenced to the independent variable λ_r . That is, we are interested in examining how \bar{t}_q varies as the load to the subsystem is changed.

Service time (\bar{S}) as indicated in the flow diagrams is composed of subsystem-dependent parameters. The degree of control that can be exercised over these time elements will dictate the control over subsystem performance. Ultimately, performance relates to throughput which is λ_r times the size of the data blocks transferred per unit time. This throughput may be constrained by response time requirements and buffer requirements for the queues in main storage. If a system can tolerate large response times in I/O and the resulting variance in \bar{t}_q , then high throughput can be obtained. Otherwise, response time must be restricted which results in lower throughput:

$$\bar{S} = \bar{t}_{wc} + S_{rp} + \bar{S}_{sk} + \bar{S}_l + \bar{S}_{el} + \bar{S}_w + S_{dl} \quad \text{RPS} \quad (3)$$

$$\bar{S} = 2\bar{t}_{wc} + S_{rp} + \bar{S}_{sk} + \bar{S}_l + S_{dl} \quad \text{non-RPS} \quad (4)$$

- \bar{t}_{wc} (Mean control unit wait time)—See Equation 12
- S_{rp} (Control unit processing time)—Fixed by control unit design. Values may range from 0.2 milliseconds to 1.0 millisecond.
- \bar{S}_{sk} (Mean seek time)—Determined, to some degree, by device design and system factors. Refer to Appendix I and Reference 1. For the model, specific values are assumed and the variance from the mean is assumed to be uniformly distributed.
- \bar{S}_l (Mean latency time)—Fixed by device design. \bar{S}_l equals one half time maximum latency time which assumes a variance that is uniformly distributed.

\bar{S}_{el} (Mean extended latency)—See Equation 20

\bar{S}_w (Mean window time)—See Equation 11

S_d (Data block transfer time)—Determined by the design of the device and the software application's need for certain data block sizes. S_d is the block size in bytes divided by the data transfer rate of the device.

Each time interval with a bar has a variance. Variance is a measure of the spread or dispersion from the mean value. Our intent is to obtain a reasonable estimate of variance which will be used in evaluating mean wait time on the device queue:

$$\sigma^2(\bar{S}) = \sigma^2(\bar{t}_{oc}) + \sigma^2(\bar{S}_{sk}) + \sigma^2(\bar{S}_1) + \sigma^2(\bar{S}_{el}) + \sigma^2(\bar{S}_w) \quad (5)$$

$$\sigma^2(\bar{t}_{oc}) = (\bar{t}_{oc})^2 \quad \text{exponential distribution}$$

$$\sigma^2(\bar{S}_{sk}) = (S_{sk} \text{ max})^2/12 \quad \text{uniform distribution}$$

$$\sigma^2(\bar{S}_1) = (S_1 \text{ max})^2/12 \quad \text{uniform distribution}$$

$$\sigma^2(\bar{S}_{el}) = \text{See Equation 21}$$

$$\sigma^2(\bar{S}_w) = (S_w \text{ max})^2/12 \quad \text{uniform distribution}$$

Before device queue wait time (\bar{t}_o) can be evaluated, the subsystem utilization (ρ_s) must be specified:

$$\rho_s = \bar{P}_{rn} \times \lambda_r \times \bar{S} \quad (6)$$

Now, \bar{t}_o can be determined and when added to \bar{S} , we have the mean response time. The equation for \bar{t}_o is a basic equation from the field of queueing theory (see Reference 1):

$$\bar{t}_o = [\bar{S} \times \rho_s / 2(1 - \rho_s)] \times [1 + \sigma^2(\bar{S})/\bar{S}^2] \quad (7)$$

The mean wait time for the control unit (t_{oc}), the extended latency time (\bar{S}_{el}), and the sector window time (\bar{S}_w) must still be evaluated before service time (\bar{S}) can be known.

In the section on RPS, the concept of sectors was discussed. The data block to be transferred starts in a specific sector. The starting point can be located anywhere within the sector (i.e. between beginning and end). Now, it takes time for the control unit and channel to prepare for the transfer. Therefore, all resources must be ready at least one sector prior to the sector containing the data. We will call this portion of the sector window the fixed portion (\bar{S}_{wf}). A specific number of sectors prior to \bar{S}_{wf} is allotted to the device to obtain use of the control and channel facility. This is called the window connect time (S_{wc}). Obviously for the sector RPS device to have a performance advantage over non-RPS devices, the window connect time must be small in comparison to the maximum latency time. Only a few sectors per track are normally required. Typical disk devices have 128 sectors per track at about 0.130 milli-seconds per sector.

T_{sec} equals the time interval of one sector:

$$T_{sec} = S_1 \text{ max} / \# \text{ of sectors per track} \quad (8)$$

\bar{S}_{wc} equals mean window connect time:

$$\bar{S}_{wc} = N \times T_{sec} / 2 \quad N = 0, 1, 2, \dots \text{ etc.} \quad (9)$$

\bar{S}_{wf} equals mean fixed window time:

$$\bar{S}_{wf} = 1.5 \times T_{sec} \quad (10)$$

The mean window time is now available:

$$\bar{S}_w = \bar{S}_{wc} + \bar{S}_{wf} \quad (11)$$

The next step is to determine the control unit wait (\bar{t}_{oc}). This time element is dependent on how busy the control unit and channel are and on the average time the control unit takes to service a channel command or transfer data. A modification of the Equation 7 used to evaluate the wait time on the device queue is used to calculate \bar{t}_{oc} . The adjustment to the equation is necessary because the queues in the subsystem are finite. That is the number of requests at any given time for control unit service can not exceed the number of devices connected to the control unit. A very good discussion on this subject can be found in Reference 1, p. 451. The adjustment factor (A) used in this model was developed by trial and error to force fit the wait time results to the graph on p. 453 of Reference 1.

$$\bar{t}_{oc} = \frac{\bar{S}_c \times \rho_{ce} \times A}{(1 - \rho_{ce})} \quad (12)$$

\bar{S}_c is the control unit service time:

$$\bar{S}_c = [S_{rp} + (\bar{S}_w + S_d)]/2 \quad \text{for RPS} \quad (13)$$

$$\bar{S}_c = [S_{rp} + (\bar{S}_1 + S_d)]/2 \quad \text{for non-RPS} \quad (14)$$

ρ_{ce} is the effective control unit utilization. It factors in the real control unit utilization ρ_c with the channel utilization (ρ_{ch}) that may result due to other subsystems sharing the channel:

$$\rho_{ce} = 1 - (1 - \rho_{ch}) \times (1 - \rho_c) \quad (15)$$

If only one subsystem is connected to the channel, ρ_{ch} equals zero. Therefore, ρ_{ce} equals ρ_c . Under this condition the channel and control unit can be viewed as one single facility:

$$\rho_c = 2\lambda_r \times \bar{S}_c \quad (16)$$

A is the adjustment factor to compensate for a limited queue:

$$A = (1 - \rho_{ce}^{(N_d - 1)}) \times \exp [(-10) \times (N_d - 1)] \times \rho_{ce} / [(N_d \times [N_d + 1])] \quad (17)$$

N_d equals the number of devices connected to the control unit.

The last equation to be evaluated is for the extended latency time (\bar{S}_{el}). To do this, we need to determine what the probability is that the control unit will be available during the sector window connect time. The time the control unit is free is $1 - \rho_{ce}$. If the control unit is busy when the connect sector is sensed, the probability that it will become free within S_{wc} (window connect time) is:

$$P_{wc} = \rho_{ce} \times [1 - \exp(-\bar{S}_{wc}/\bar{t}_{oc})] \quad (18)$$

Therefore, the probability the control unit is available is:

$$P_{ca} = 1 - \rho_{ce} + \rho_{ce} \times [1 - \exp(-\bar{S}_{wc}/\bar{t}_{oc})] \quad (19)$$

The extended latency (\bar{S}_{el}) is:

$$\bar{S}_{el} = \bar{S}_1 \max \times (1/P_{ca} - 1) \tag{20}$$

The variance of \bar{S}_{el} is:

$$\sigma^2(\bar{S}_{el}) = \frac{\bar{S}_{el}^2(1-P_{ca})}{P_{ca}^2} \tag{21}$$

The set of equations are now complete and response time as a function of request rate can be evaluated. The supplementary equations above also provide additional insight into what is going on in the subsystem.

SUBSYSTEM EVALUATION

The first configuration that will be looked at is the single-access, non-RPS disk subsystem with one control unit connected to the channel. The components of service time assumed for this configuration are:

- $S_d = 0.5$ milli-seconds
- $\bar{S}_1 = 8.33$ milli-seconds
- $S_{rp} = 1$ milli-second
- $\bar{S}_{sk} = 15$ milli-seconds
- \bar{t}_{wc} = varies with the load
- $\bar{S} = 24.8$ milli-seconds + $2\bar{t}_{wc}$

Figure 8 is a plot of response time (\bar{t}_q) as a function of requests (λ_r). Figure 8a shows three curves: (1) two disks per control unit; (2) four disks per control unit; and (3) eight disks per control unit. Each disk services an equal percentage of λ_r . The arrows indicate the point on the curves where the probability that a request to the subsystem must wait because a previous request has not been completed is 0.5.

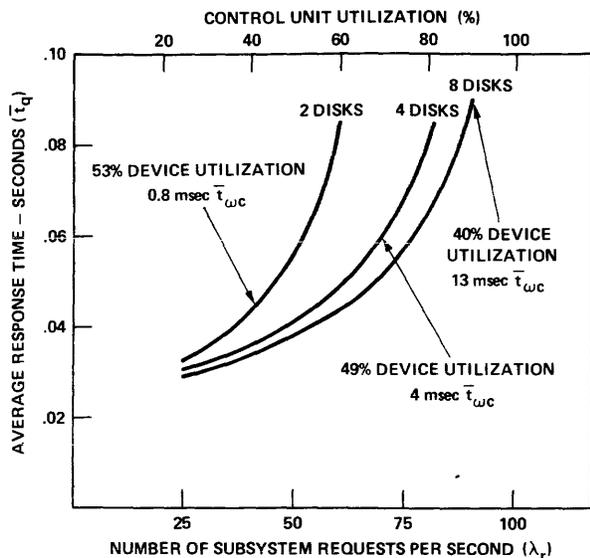


Figure 8a—Single control unit with non-RPS.

This will be called subsystem queue=1. If the service time is assumed to be exponentially distributed (worse case), the variance of response time when subsystem queue=1 can be very large. This means that it is possible for approximately 10 percent of the requests to the device to exceed response times that are three to five times the mean. Note that as the device queue approaches one, the wait time on the queue (\bar{t}_w) almost equals the service time. This is due to the increase in control unit and device utilization. The two-disk curves show that the disk devices are the limiting resource at about 20 requests per disk per second. The four-disk curve indicates that both control unit and devices limit the performance at about 17 requests per disk per second. The eight-disk curves show the control unit limiting the performance at about 11 requests per disk per second. It appears that in the area of the four-disk curve, the subsystem is most cost effectively utilized. The reason that control unit utilization increases rapidly is due to waiting out the disk latency time since the devices are not using RPS. A point of concern is how the subsystem performance behaves as λ_r is increased beyond a device queue of one. For the eight-disk curve, the subsystem would bog down with large queues and long response times.

The next configuration is a single-access disk subsystem using RPS. The service time elements assumed are those specified for the non-RPS configuration. However, two additional elements are required for RPS:

- $\bar{S}_c = 0.5$ milli-seconds (~4 sector times)
- $\bar{S}_{wr} = 0.19$ milli-seconds (~1.5 sector times)

A plot for a four-, eight-, and 16-disk configuration is shown in Figure 8b. It's easy to see that a definite performance increase has resulted by not tying up the control unit during latency time. The disk devices are now the major limiting resources in performance. The gain in performance over the

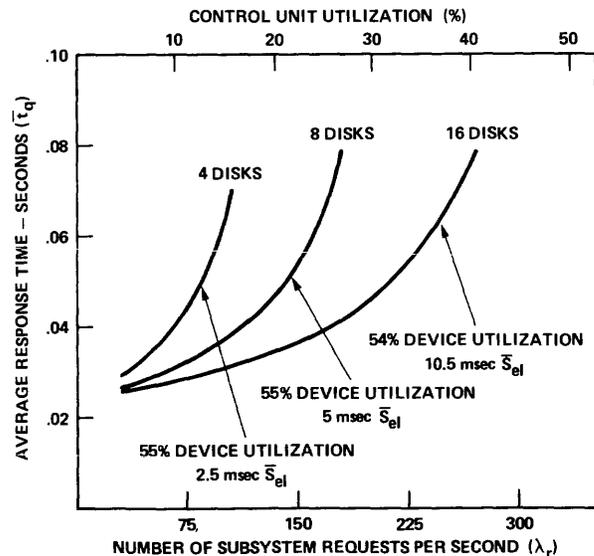


Figure 8b—Single control unit with RPS.

non-RPS for the four-disk configuration isn't too drastic (i.e. 20 requests per disk per second with respect to 17). However, the control unit utilization for RPS is very low and this implies that additional devices can be cost-effectively added to the subsystem. Note the gain in performance for the eight-disk curve over the non-RPS. The devices are capable of servicing 19 requests per disk per second as opposed to 11 for the non-RPS and with a response time in the area of 50 milli-seconds instead of 90 milli-seconds. Additionally, the curve beyond the subsystem queue=1 is more gradual, implying a greater tolerance to variation in the request rate. The RPS control unit is even capable of handling up to 16 disk devices in an efficient manner.

Beyond this point cost-effective gains in performance become questionable. The magnitude of the extended latency time at a subsystem queue=1 increases as disks are added and λ_r is increased. This is due to the increase in control unit utilization and reduces the device's capability to handle requests. The situation isn't too bad because the time to transfer data chosen for this configuration was small. This implies that the device either has a fast data transfer rate or a small data block size (e.g. 500-1000 bytes per block). If a larger S_{el} is assumed the control unit utilization would increase and with it the extended latency. The only way to reduce extended latency is to reduce control unit utilization. A good way to do this is to add a second control unit and channel and configure it as a dual access subsystem. This cuts the control unit utilization because the two units share the request load.

Figure 8c illustrates a dual-access configuration using RPS. Again, four-, eight-, and 16-disk curves are shown. The service time elements are specified for the single access RPS configuration. For the reason stated above, the gain in performance is not dramatic for the four- and eight-disk curve. For the 16-disk curve, a respectable improvement in λ_r is indicated. In addition, the low control unit utilization

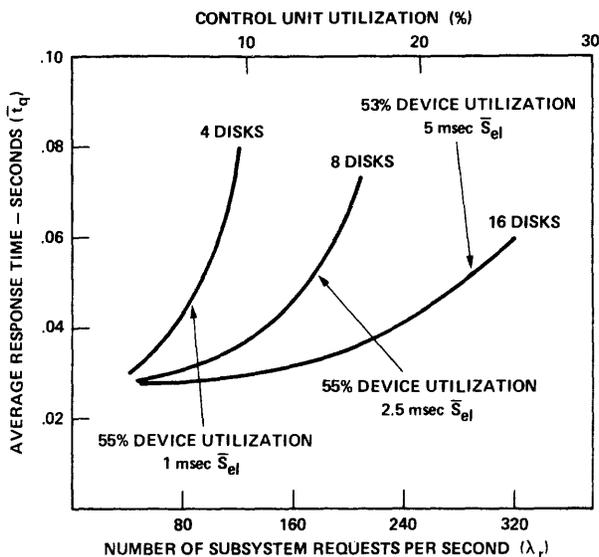


Figure 8c—Dual control unit with RPS.

indicates that disks in excess of 16 can provide a cost-effective increase in performance.

SUPPLEMENTAL EVALUATION

Channel utilization impact

If the channel is being used by more than one subsystem, each subsystem interferes with the other. The effect of this interference is felt in increased response times at a given load on the subsystem. This in turn forces the system to drive the subsystem at fewer requests per second to obtain reasonable response times and more stable queues. A reasonable approximation of the degree of impact channel utilization has on a subsystem is listed in Table I.

Latency time

The role that maximum latency time plays in limiting the throughput of a subsystem depends on the magnitude of the maximum latency time with respect to the service time (\bar{S}) of the subsystem. Reducing latency will improve performance. Table II lists the gain in performance when maximum latency time is reduced by 25 percent, 50 percent and 100 percent. Column 1 indicates that if maximum latency time has a value in the area of 80 percent of the service time, a significant increase in throughput is gained if it can be reduced by 50 to 100 percent. For disk units there is little chance to reduce latency but for bubble and charge coupled devices some latitude of control exists. Furthermore, these solid state technologies have performance characteristics such that latency time may be in the area of 80 percent \bar{S} .

Seek time

Seek time, like latency time, has a major affect on throughput. For disk subsystems, some control can be exercised over seek time. The degree of performance improvement, again, depends on the initial value of seek time with respect to the service time (\bar{S}). Table III lists the percentage of increase in throughput by reducing seek time by 25 percent, 50 percent and 100 percent. Column 1 shows that significant gains are realized if the initial value of S_{sk} is 80 percent of \bar{S} .

TABLE I—Performance Impact Due to Channel Utilization

CHANNEL UTILIZATION	APPROXIMATE LOSS IN THROUGHPUT
0.1	6%
0.2	15%
0.3	25%
0.4	36%
0.5	48%

TABLE II—Predicted Improvement Due to Reduced Latency

	$S_l \text{ max} = .8 \bar{S}$ PERFORMANCE INCREASE	$S_l \text{ max} = .5 \bar{S}$ PERFORMANCE INCREASE	$S_l \text{ max} = .2 \bar{S}$ PERFORMANCE INCREASE
25%	12%	6%	3%
50%	25%	14%	5%
100%	75%	34%	12%

TABLE IV—Predicted Improvement Due to Reduced Data Transfer Time

	$S_d \text{ max} = .1 \bar{S}$ PERFORMANCE INCREASE	$S_d \text{ max} = .05 \bar{S}$ PERFORMANCE INCREASE	$S_d \text{ max} = .02 \bar{S}$ PERFORMANCE INCREASE
25%	14%	3%	1%
50%	33%	7%	2%
100%	90%	13%	4%

Data transfer time

Typically, S_d is small compared to service time. However, it not only consumes device time, it also consumes control and channel time. These latter two facilities are shared and time consumption must be well managed. Table IV shows the impact of reducing S_d when it is 10 percent, 5 percent, and 2 percent of the service time (\bar{S}). Disks typically have S_d in the 5 percent, to 2 percent of \bar{S} range. A change in S_d can be the result of a change in data block size or data transfer rate.

Control unit processing time

This time is small compared to \bar{S} for disk subsystems, but may become significant for solid-state devices. It impacts throughput in a manner similar to data transfer time. Like S_d , it consumes control unit time and if the control unit is to service a large number of requests for high-performance solid-state devices, it must be held to a small percentage of \bar{S} . Table V lists the impact on performance of S_{rp} when its value is 10 percent, 5 percent and 2 percent of \bar{S} .

CONCLUSION

The mathematical model defined in this paper can be used to give a system designer a good deal of insight into what parameters can be most effectively changed to yield the desired performance. Given parameters such as seek time, latency time, control unit processing time, sector time, and channel loading, these parameters can be used to predict subsystem performance. The model described here was checked against an internal GPSS model simulating overlapped seeks on disks. The results of our mathematical model agreed within 10 percent of the GPSS model.

APPENDIX I

Seek time

There are a number of ways to compute seek time. Some of them are analytic, some are heuristic. There are a number of ways to shorten seek time. This appendix will talk about shortening seek time by restricting the number of cylinders that contain data.

On the assumption that the requests to a disk are to random locations, then the average movement of the head mechanism is across $\frac{1}{3}$ of the cylinders. Many people have suggested that the average seek time could be shortened by taking the contents of the queue of requests for this disk and moving to one of the closer positions.

Two techniques have emerged called Scan and C-Scan. In both, the list of commands are arranged in cylinder position sequence. Any new command is inserted in the list in the proper numerical position. Scan says to start at one end of the list and always go to the next position as you go through the list in one direction. When there are no more in the list for that direction, turn around and go through the list in the opposite direction. The head mechanism will take a number of steps in one direction and then a number in the other. The C-Scan (circular scan) will do the same, except that at the end of the list, it will jump back to the beginning of the list and start over.

Intuitively, C-Scan seems more efficient. In Scan, when a new position is deposited in the list after the scan has just passed that point, the command has to sit there until the scan reaches the end and comes back through the list once again. In C-Scan, such a command only has to wait through one scan (not two). Also, in Scan, when the end of the list is reached and it turns around, there is small probability that there is anything in the list at a close cylinder position. Therefore, at the beginning of each scan, the distances to

TABLE III—Predicted Improvement Due to Reduced Seek Time

	$S_{sk} \text{ max} = .8 \bar{S}$ PERFORMANCE INCREASE	$S_{sk} \text{ max} = .5 \bar{S}$ PERFORMANCE INCREASE	$S_{sk} \text{ max} = .2 \bar{S}$ PERFORMANCE INCREASE
25%	24%	12%	5%
50%	59%	28%	10%
100%	278%	67%	12%

TABLE V—Predicted Improvement Due to Reduced Processing Time

	$S_{rp} \text{ max} = .1 \bar{S}$ PERFORMANCE INCREASE	$S_{rp} \text{ max} = .05 \bar{S}$ PERFORMANCE INCREASE	$S_{rp} \text{ max} = .02 \bar{S}$ PERFORMANCE INCREASE
25%	13%	3%	0.5%
50%	30%	5%	1%
100%	69%	9%	2%

the next cylinder will be relatively large. Surprisingly, intuition does not work. Simulations² show that for small queues, Scan is better than C-Scan. The large seek distance to the beginning of the list seems to hurt. For example, if there are two commands in the list, C-Scan has one large seek and one small seek, while Scan has two small seeks. Teorey³ says that C-Scan does not begin to pay off until the queue length goes over 100. But we cannot allow the queue length to ever reach such lengths because of the prohibitively long response times.

Croteau states that whenever the average queue length exceeds 0.2 per disk, Scan begins to give better results than random. She also states that when the average queue length is two or three (response time is two or three times service time), then the average positioning time has been reduced about 10 percent.

Restriction to part of disk

The only way to radically shorten the seek time on a disk is to keep the head from moving long distances. To do this, take the group of files or databases that are accessed by a set of programs. Record part of these files on all the non-removable disk packs. The part of any disk should be confined to contiguous cylinders. Then when running those programs, seek time is confined and shortened and as many disks as possible are operated in an overlapped way. Multiprogramming has to be restricted to one set of programs at a time. If multiprogramming cannot be restricted, then the only way to shorten seek time is to use only a fraction of the cylinders to record data. This costs more, because more disks are needed. In addition, it could mean more throughput because more disks are operating in parallel.

The average positioning distance, if requests are random, is one-third of the occupied cylinders. The average positioning time is not the time to move one-third of the cylinder positions. You have to add all the times to move from any possible position to any other position (including the zero time to stay on the same cylinder position) and then divide by the number of moves. The average positioning time is usually less than the positioning time to move an average number of cylinder positions. Waters⁴ gives the formula for computing average seek time.

The average seek time is reduced if the requests are not randomly distributed over all cylinder positions, but are bunched. Waters shows how to compute the average positioning distance if the high activity records are all grouped together. He shows that the minimum seek distance occurs when the high activity records are recorded in the middle group of cylinders while the maximum seek distance occurs when they are in the lowest numbered cylinders. Waters then gives two examples. In one case 80 percent of the requests go to 20 percent of the disk. When the 20 percent are in the beginning cylinders, then the average seek distance is about one-fifth the number of cylinder positions. When the 20 percent are in the middle cylinders, the average

seek distance is about one-seventh of the cylinder positions. Both are better than the one-third for random requests.

The other case is for 60 percent of the requests to go to 5 percent of the disk. When the cylinders are in the beginning, the average number of positions moved is about 0.3 of the cylinder positions. When the 5 percent are in the middle cylinders, the average number of cylinder positions moved is less than 0.2 of them.

If you confine 100 percent of the requests to a smaller number of cylinders, then you get even better average seek times. In fact, if you only used 40 percent of the cylinder positions for data, then you get a small average number of cylinders moved than any of the previous cases.

APPENDIX II

Wait times with RPS

The following derivation is due to John Marsden of our Sperry Univac group in London sometime in the middle or early 1970s. Usually, wait time is a function of queue length. He has come up with an ingenious scheme where wait time is a function of the utilization of the control unit. Ordinarily, with only one command waiting for service, the wait time is latency time with an average of $R/2$ (half a rotation time). With more than one command waiting for service, some of the commands will have extra rotation times added to their wait time because the control unit is busy when it is time to acquire it. The average wait time is:

$$\text{Avg. wait time} = \frac{R}{2} + \frac{R \cdot \rho_{ce}}{1 - \rho_{ce}}$$

where ρ_{ce} = utilization of the control unit caused by all other disks and channel contention.

R = Time for 1 revolution of the data or media.

The extra wait time begins to be noticeable when the control unit is busy more than 20 percent of the time.

The derivation goes like this. The average latency on the first rotation is $R/2$. This will be true if the control unit is not busy at the RPS time. The other disks keep the control unit busy with a utilization of ρ_{ce} . The probability that it will not be busy is $(1 - \rho_{ce})$. The probability it will be busy on the first revolution is ρ_{ce} , and not busy on the second is $(1 - \rho_{ce})$. The average wait time in this case is one-and-a-half rotations.

$$\text{Prob. of first rev.} = 1 - \rho_{ce}; \quad \text{avg. time} = \frac{R}{2}$$

$$\text{Prob. of second rev.} = \rho_{ce}(1 - \rho_{ce}); \quad \text{avg. time} = \frac{3R}{2}$$

$$\text{Prob. of third rev.} = \rho_{ce}^2(1 - \rho_{ce}); \quad \text{avg. time} = \frac{5R}{2}$$

Weighted Avg. Time: W_{at}

$$\begin{aligned} &= \frac{R}{2}(1-\rho_{ce}) + \frac{3R}{2}\rho_{ce}(1-\rho_{ce}) + \frac{5R}{2}\rho_{ce}^2(1-\rho_{ce}) + \dots \\ &= \frac{R}{2}1 + 3\rho_{ce} + 5\rho_{ce}^2 + \dots(1-\rho_{ce}) \\ &= \frac{R}{2}(1-\rho_{ce})\frac{1}{1-\rho_{ce}} + \frac{2\rho_{ce}}{(1-\rho_{ce})}2 \\ &= \frac{R}{2} + \frac{R \cdot \rho_{ce}}{1-\rho_{ce}} \end{aligned}$$

If $P_{ca} = 1 - \rho_{ce}$

Then

$$W_{at} = \frac{R}{2} + R \left(\frac{1}{P} - 1 \right)$$

Variance of latency

The variance of the extra missed revolutions has been computed by Antony Jenkins. It is derived as follows:

$$\begin{aligned} \sigma^2 &= (1-\rho_{ce}) \left(0 - \frac{R \cdot \rho_{ce}}{1-\rho_{ce}} \right)^2 + \rho_{ce}(1-\rho_{ce}) \left(R - \frac{R \cdot \rho_{ce}}{1-\rho_{ce}} \right)^2 \\ &\quad + \rho_{ce}^2(1-\rho_{ce}) \left(2R - \frac{R \cdot \rho_{ce}}{1-\rho_{ce}} \right)^2 + \dots \\ &= (1-\rho_{ce}) \frac{R^2}{(1-\rho_{ce})^2} [\rho_{ce}^2 + \rho_{ce}(1-2\rho_{ce})^2 + \rho_{ce}^2(2-3\rho_{ce})^2 + \dots] \\ &= \frac{R^2}{1-\rho_{ce}} (\rho_{ce} + \rho_{ce}^2 + \rho_{ce}^3 + \rho_{ce}^4 + \dots) \\ &= \frac{R^2 \cdot \rho_{ce}}{(1-\rho_{ce})^2} \end{aligned}$$

REFERENCES

1. Martin, James, *Design of Real-Time Computer Systems*, Prentice Hall, 1967.
2. Croteau, Stephanie, "A Study of Different Service Policies for the SPERRY UNIVAC 8414 and 8440 Disks," Tech. Memo 6834-05, June 10, 1971.
3. Teorey, Toby J., and Tad B. Pinkerton, "A Comparative Analysis of Disk Scheduling Policies," *CACM*, March, 1972.
4. Waters, S. J., "Estimating Magnetic Disk Seeks," *British Computer Journal*, Vol. 18, No. 1, 1975.

A software reliability study using a complexity measure

by THOMAS J. WALSH

Control Data Corporation
Shrewsbury, New Jersey

INTRODUCTION

Software engineers face a real problem in guaranteeing that their computer programming systems under development will be able to function in a reliable manner and be easily understood, maintained and extended. A major impediment of this problem is coping with the inherent complexity of the software system in an effective way. The complexity of the computer system will defeat the designer's efforts unless a relatively simple way is found to break the problem down in order that the resulting programs are testable and maintainable. Complex problems must be factored into smaller units to be treated by the human intelligence because man's capacity for logically precise invention is limited. The consequence of ignoring these bounds to man's cognitive and creative capacity was well stated by Harlan Mills of IBM:¹

We often ignore the complexity of a planned program or sub-program. But when the complexity exceeds certain unknown limits, frustration ensues. Computer programs capsize under their own logical weight, or become so crippled that maintenance is impossible.

Complexity is an attribute of a computer program much like storage and speed of execution, the difference being a measurement stick of a program's complexity has not been available. Thus, this important quantitative attribute has been generally ignored.

To appreciate the impact of ignoring a program complexity measure, let us take a brief look at the phases constituting the life-cycle of a typical software system. These phases encompass the process of design, implementation, testing and maintenance.

Scientific studies have validated the facts that at least half of the systems development time is spent in testing² and most dollars are spent on maintaining systems.³ Figure 1 illustrates the typical break-down of software costs.⁴

The high cost of software is primarily due to software not being reliable enough. The key to software reliability is in the degree of precision and accuracy achieved during the design process, which has the greatest effect on the overall efficiency of the system and consequently the overall cost. This point cannot be over-emphasized. Frequently, programming systems have not been designed to facilitate testing efforts nor to constrain the impact of change. What is

needed is a methodology for software system development which makes a concern for reliability an integral part of the development process, especially in the design phase. Such a methodology would consist of various tools and techniques distributed over the four previously-mentioned phases.

In recent years, what I will refer to here as "structured programming ideas" have emerged as a foundation for programming to become a science. This set of ideas has been referred to as "improved programming technologies" by IBM and as "programmer productivity techniques" by Yourdon.⁵

Regardless of the label, they represent several interrelated disciplines which have a natural affinity and yield noticeable benefits to the software system builder when used together in a systematic fashion.⁵⁻⁸

Each time a new idea is added to this evolving programming methodology, the opportunity for greater precision and reliability in programming practices increases. The analytical complexity measurement of McCabe is a new idea and deserves both recognition and membership in the promising methodology.⁹ The complexity measurement represents an attractive and powerful concept because of its relatively simple applicability, its direct impact on the process of design, and its mathematical origin.

Complexity can usually be calculated for computer programs by simply counting the number of decision statements and adding one. The complexity measurement focuses attention on the process of design so the function specified for the software can be intellectually gripped and performed in the simplest possible manner. This advanced design technique allows for comparing alternate designs in search of the true order that the best solution really called for. It strives to eliminate obscure structures, cumbersome decision-making processes, and overly complicated control paths. This design capability is language-independent. The mathematical origin accounts for the measure being highly correlated with the expected amount of testing work. It facilitates a more thorough and methodical testing process by yielding a minimum number of paths through a program that must be exercised in order to make testing meaningful (Structured Testing). Furthermore, it simplifies maintenance activities by the strengthening of testing and limiting the complexity of the program to be fixed. It identifies software programs that will be difficult to test and maintain and encourages the creation of a more testable and maintainable

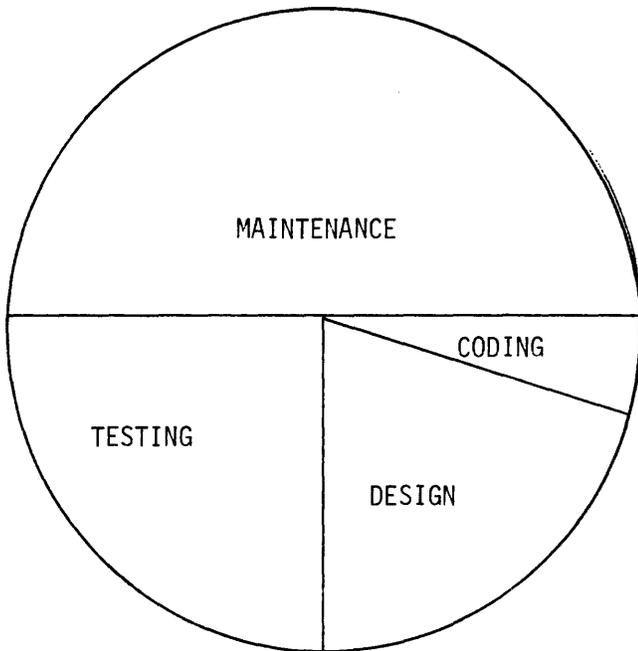


Figure 1—Typical breakdown of software costs.

system. Overall, this mathematically-based measurement permits management and control of program complexity via a quantitative basis.

Figure 2 illustrates the hierarchical relationship and utilization order between top-down design, the complexity measure and structured programming. Top-down design is concerned with the "divide-and-rule" principle. It recognizes that complex problems must be factored into a combination of many small solvable problems. The complexity measure then weighs the individual module's complexity to facilitate proper design, testing and maintenance. This may indicate the need for further top-down design. Finally, structured programming tackles the problem of program logic design. Figure 3 represents a survey of structured programming ideas distributed over the four phases of software development. This chart is not complete, but it is intended to be illustrative. Definitions for these techniques and related documentation techniques may be found in the Glossary.

BACKGROUND

This section's purpose is to sketch some significant historical developments in the evolution of structured programming ideas in order to properly view McCabe's complexity measure. The first major result was a paper by Bohm and Jacopini.¹⁰ This classic manuscript introduced organization and discipline by showing that any program, no matter how complex, can be composed in a structured manner with only three relatively simple control structures which are popularly known as "SEQUENCE," "IF THEN ELSE" and "DOWHILE" (Figure 4). An analogy of this powerful development is often made to engineering where any logic

circuit can be constructed from "AND," "OR" and "NOT" gates. The importance of this result was that it mathematically dealt with the problem of complexity in control logic. The proof for the "structure theorem" is grounded firmly in mathematics. Solidly grounding viable techniques in mathematics not only increases the confidence level of present users of these techniques, but it facilitates the future development of even more powerful techniques. Although this paper was published in English in 1966, the proper recognition it deserved did not materialize until the 1970s. A major driving force in its recognition was Edgar W. Dijkstra who strongly endorsed structured programming by a famous letter in the *Communications of the ACM* and numerous articles.^{11,12}

Professor Dijkstra, historically, is a man ahead of his time whose clear thinking and proper design of programs have earned him a most influential and respected position in the computer programming profession.¹³ An underlying theme to much of his work has been the view of software as a creative branch of mathematics. Therefore, he sees the mathematical method as the most effective way for the human mind to come to grips with complexity.¹⁴

Many individuals have contributed to the methodology referred to here as structured programming ideas—especially Harlan Mills, who also was an early advocate of structured programming.¹⁵ He provided mathematical assurance for structured programming ideas in his "Mathematical Foundations for Structured Programming."¹¹ It should be noted that Mills' paper also contains the mathematical seeds that McCabe will use to simplify his theory of cyclomatic complexity. Mills has been very concerned about the problem of complexity. He views¹⁶ complexity as the "principal barrier to the application of computers to intelligent problem-solving." In a more recent article,¹⁷ he reiterates the call of Dijkstra for a mathematical basis for the practical control of computers in complex applications.

Recently, endorsements of McCabe's method as being both reasonable and intuitive have come from various sources.^{18,19} Glenford J. Myers of the IBM Systems Research Institute concludes his manuscript by stating:

Although it is an extremely simple concept, $V(G)$ appears to be a practical complexity measure because it is easy to calculate, it confirms subjective opinions about complexity, and it is consistent with studies showing a high correlation between the number of decisions in a module and the modules' complexity and error proneness.

McCABE'S COMPLEXITY MEASURE

McCabe's complexity measure is a mathematical technique for calculating the logical complexity of a computer program. The complexity of a computer program is an attribute which may be assigned a number representing its logical weight. The quantitative complexity number generated is independent of the program's size, but dependent on a program's decision structure or the number of basic paths

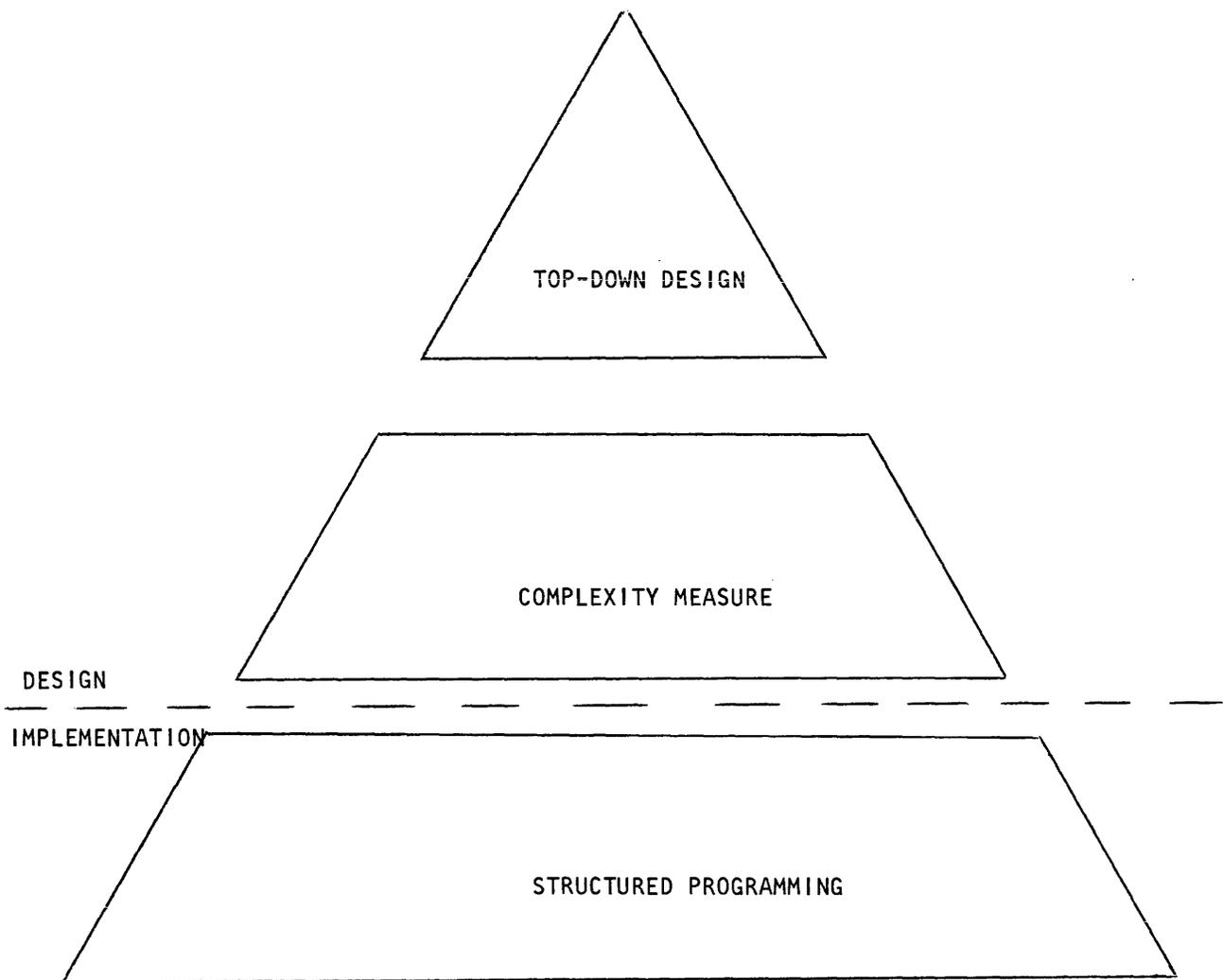


Figure 2—Hierarchies of system development reliability tools.

through a program. It provides a quantitative means for modularization and allows for identification of programs that will be difficult to test and maintain. Although complexity can assume a value of one to infinity, a reasonable upper limit of intellectual manageability has been placed by McCabe at ten. This number is only slightly higher than the upper bound that psychological studies confirm as the number of issues man can consider simultaneously.²⁰

McCabe recommends that designers be required to calculate complexity as they create software programs and when complexity exceeds ten, sub-functions should be given their own procedure or the software should be redone.

In the interests of communication, this section has suppressed much of the technical mathematics which forms a solid foundation for this advanced methodology to come to grips with complexity. This would include the works of many people, including Bohm, Jacopini, and Mills.^{10,1} It would require a highly technical type of discussion outside the scope of this paper. I am very much aware of these

shortcomings and can only trust this approach will not diminish the serious respect this body of knowledge so well deserves.

The following material represents some of the highlights. The theoretical basis for McCabe's complexity measure is graph theory. The following connection exists between graph theory and computer programs. Each node in the graph corresponds to a block of code in the program where the flow is sequential and the arcs correspond to branches taken in the program. Thus, all computer programs may be expressed as graphs or to be precise "program control graphs." This is an important concept because it represents a gateway through which the power of mathematical analysis may be applied to computer programs. Graph theory allows for such a graph to yield a quantitative cyclomatic complexity number via the formula:

$$v(g) = e - n + 2 \quad (1)$$

For example, in Figure 5 e (arcs) has a value of nine, n

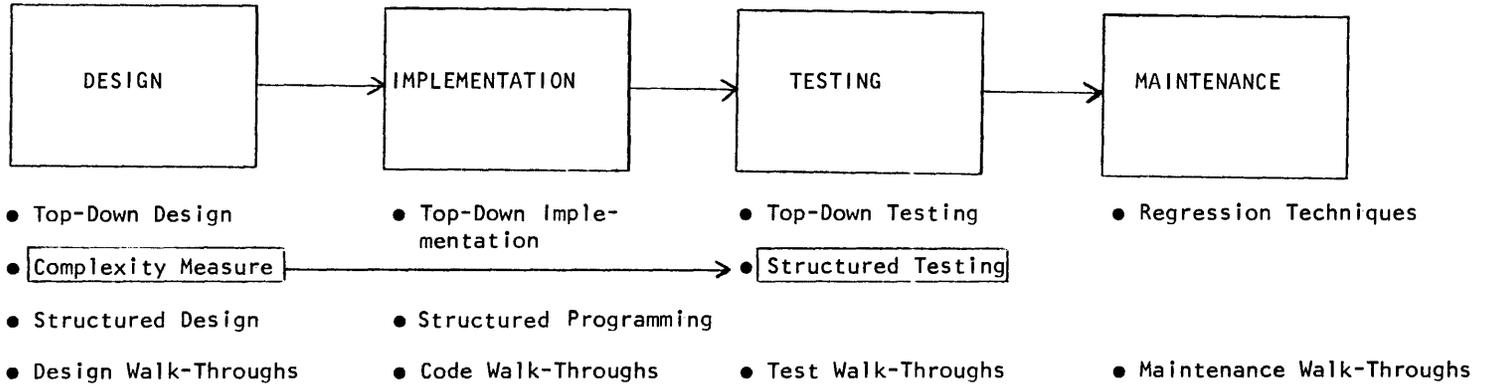


Figure 3—Partial list of structured programming ideas.

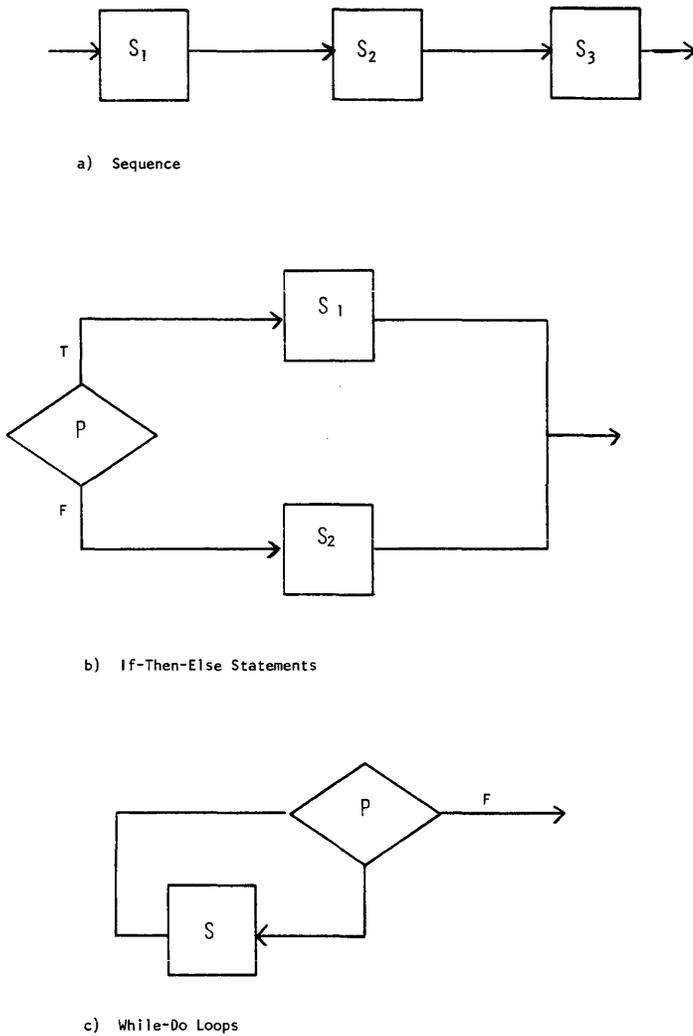


Figure 4—Control structures of structured programming.

(nodes) has a value of eight, and the complexity of the program is three.

A simple but powerful key to McCabe's proposal is brought about by a mathematical simplification that shows the complexity for a structured program to be equal to the number of decision statements plus one. Mills¹ showed that with the number of function, predicate and collecting nodes represented by θ , π , and γ , respectively, and the number of control lines (edges) represented by e in a structured program, that the following holds true:

$$e = 1 + \theta + 3\pi. \tag{2}$$

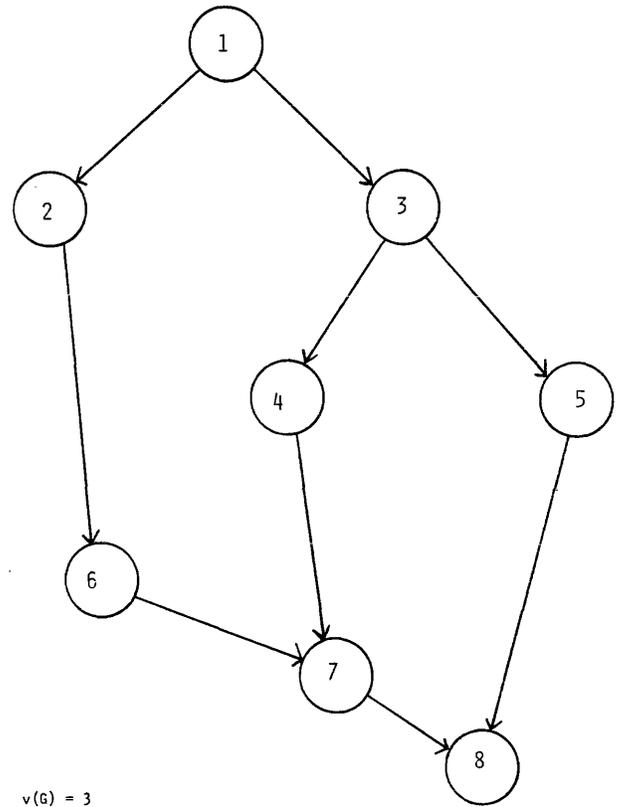
Furthermore, the number of collecting nodes is always equal to the number of predicates as in:

$$\pi = \gamma. \tag{3}$$

Therefore, it follows that (1) can be transformed,

$$V(G) = (1 + \theta = 3\pi) - (\theta + 2\pi + 2) + 2 = \tag{4}$$

$$\pi + 1. \tag{5}$$



$v(6) = 3$

Figure 5—Example of complexity graph.

It should be noted that McCabe has recently proved that the structure of all programs (structured and unstructured) is equal to the number of conditions plus one. The preceding grants us the mathematical assurance to calculate the complexity of a given program either by counting the predicate nodes in the flowchart or by inspecting the source code. This allows for simplicity in the application of the complexity measure by suspending the task of drawing time consuming graphs. In addition, the measurement process can be easily automated. In fact, McCabe has built a control structure complexity tool to run on a PDP-10 that analyzes the structure of FORTRAN programs.

AEGIS CASE STUDY

The AEGIS Naval Weapon System is an advanced ship-board combat system which is tasked with shielding the U.S. fleet.²¹ System control is governed by three high-speed general purpose AN/UYK-7 computers. An individual computer is assigned to the Radar system, the Weapons and Control System and the Command and Decision system. The heart of this large computerized system is the AN/SPY-1A three-dimensional phased array radar. The reliability study focused on eight functionally-related computer modules which constituted what is known as the software control loop and display processing of the radar. Each module was assigned with a primary function of the radar control soft-

ware. This included radar scheduling, search management, track processing, radar return processing, etc. A module itself is further divided into procedures to perform specific tasks within the module. These eight modules were composed of 276 procedures or programs which were the actual subjects of the complexity study. These procedures were written in a high-level language (CMS-2Y) and run on a 4-bay AN/UYK-7 high-speed computer. The AEGIS project utilized the software engineering techniques of top-down design and structured programming.

The methodology called for the calculation of a number of quantitative parameters for each procedure. These parameters included the number of software errors experienced, the number of source statements, the number of machine words, and a complexity number. The first parameter called for the collection and analysis of software errors detected during the development phase of these modules. The number of software errors were distilled from an existing program trouble report system. A serious effort was made to obtain only those problem reports related to software errors rather than design changes. Interviews with the programmers responsible for the code facilitated this end. The procurement of a complexity number was through McCabe's complexity measurement recommendation.

A correlation was found between a high complexity value and the occurrence of bugs for a procedure. Those procedures with a complexity greater than or equal to ten accounted for a disproportionate share of the bugs. Overall, 23 percent of the procedures accounted for 53 percent of the bugs. This fact alone is not conclusive. In general, the procedures with a complexity measurement greater than or equal to ten were also the largest users of source statements.

This meant that a correlation also existed between a high number of source statements and the occurrence of bugs for a procedure. Recognition of this phenomenon, most notably described in the *New York Times* project by IBM,⁷ has led to attempts to limit the physical size of procedures, e.g., 50 lines of source code. An obvious flaw with limiting programs solely by physical size is that it ignores the density of control structures in those 50 lines.

The principal result of the study into logical complexity occurs when the 276 procedures are divided into two groups and their respective error rates are compared. These groups are defined as those procedures with a complexity measurement less than ten and those procedures with a complexity measurement greater than or equal to ten (Figure 6). Approximately half of the actual source code is in each of these groups. Yet, those procedures with complexity greater than or equal to ten experienced over 21 percent more errors. The error rate for the group of procedures with complexity measured below ten was 4.59 errors per 100 source statements. The error rate for the group of procedures with complexity greater than or equal to ten was 5.60 errors per 100 source statements. Clearly, the effect of these error rate differences in a large software system is significant.

Now, Figure 7 illustrates what empirical studies have shown concerning the relationship between detected and undetected errors. As the number of detected errors in a piece of software increases, the probability of the existence of more undetected errors also increases.⁴ Put simply, errors come in clusters. Thus, it can be confidently predicted that when the procedures in the study enter the maintenance phase of their existence, the procedures with a complexity greater than or equal to ten will continue to experience

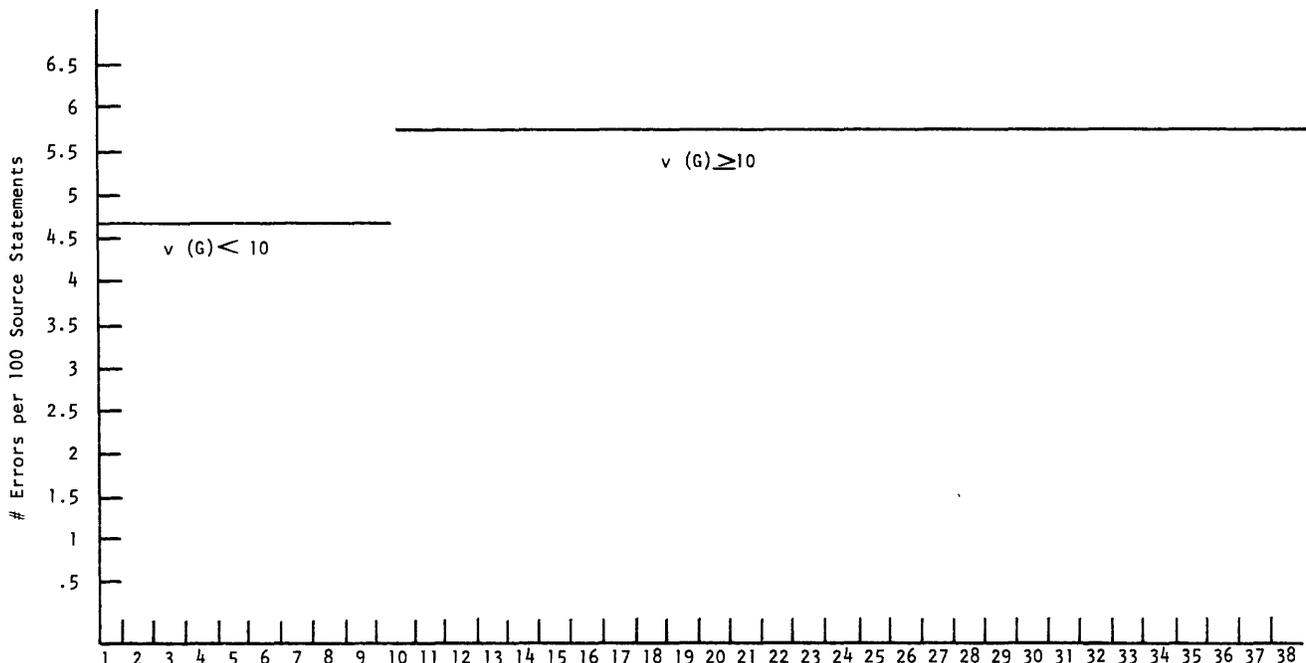


Figure 6—Complexity measurement.

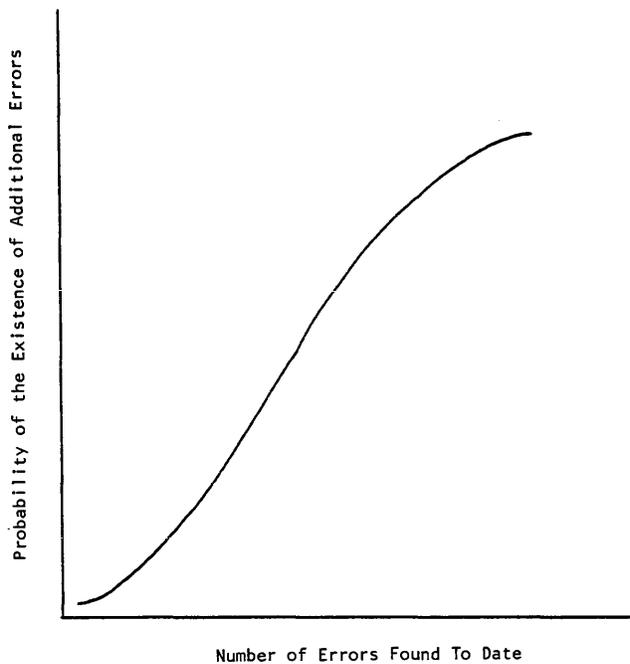


Figure 7—Relationship between discovered and undiscovered errors.

higher error rates than those procedures with complexity below ten. A far more reliable radar computer system could be achieved by reducing the software logical complexity of all procedures below ten. In fact, this suspicion can be extended to all large systems that were developed without a complexity measure.

SUMMARY AND RECOMMENDATIONS

Reliable software is no accident. It is the residue of a collection of software engineering techniques which have a natural affinity and are distributed over the four phases of software development. The key phase is design because its effects propagate through all the other phases. A new software engineering design technique is McCabe's quantitative complexity measure which is mathematically linked to the works of Bohm, Jacopini and Mills. The following recommendations are made as a result of my study utilizing McCabe's complexity measure to software system builders:

1. The complexity measure should be viewed as a structured programming technique and employed with the other structured programming techniques to enhance software reliability.
2. The complexity measure should be used to create a more testable and maintainable system by warning designers when a program has become too complex.
3. The complexity measure should be used to evaluate alternate designs with the goal of finding the simplest possible solution to the problem specifications.
4. The complexity measure should be used as a more

thorough and methodical testing process which quantifies the amount of work necessary for reliable testing.

5. The complexity measure should be viewed as an aid to the maintenance process via its strengthening of testing and the limiting of the complexity of the program to be fixed.
6. The complexity measure should be used on existing software to identify programs that will be difficult to maintain and extend. These programs are prime candidates for redesign.

REFERENCES

1. Mills, H. D., "Mathematical Foundations for Structured Programming," Federal Systems Division, IBM Corporation, Gaithersburg, MD, FSC 72-6012, 1972.
2. Bohm, B. W., "Software and Its Impact: A Quantitative Assessment," *Datamation*, Vol. 19, May 1973, pp. 48-59.
3. "That Maintenance Iceberg," *EDP Analyzer*, Vol. 10, No. 10, October 1972.
4. Myers, G. J., *Software Reliability: Principles and Practices*, John Wiley & Sons, 1976.
5. Yourdon, E., *How to Manage Structured Programming*, Yourdon, Inc., 1976.
6. Baker, F. T., "Organizing for Structured Programming," IBM Federal Systems Division, Gaithersburg, MD.
7. Baker, F. T., "Chief Programmer Team Management of Production Programming," *IBM Systems Journal*, Vol. II, No. 1, pp. 56-73, Jan 1972.
8. Baker, F. T., "System Quality Through Structured Programming," *Fall Joint Computer Conference*, Vol. 41, Part 1, 1972.
9. McCabe, T. J., "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, December 1976.
10. Bohm, C., and G. Jacopini, "Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules," *CACM*, Vol. 9, 1966, pp. 366-371.
11. Dijkstra, E. W., "GO TO Statement Considered Harmful," *CACM*, Vol. II, No. 3, March 1968.
12. Dijkstra, E. W., "Notes on Structured Programming," *Technische Hogeschool*, Eindhoven, The Netherlands, August 1969.
13. Dijkstra, E. W., "The Humble Programmer," *CACM*, Vol. 15, No. 10, October 1972, pp. 859-866.
14. Dijkstra, E. W., "On a Methodology of Design," *MC-25 Informatica Symposium*, Mathematical Centre Tracts, Amsterdam 1971.
15. Mills, H. D., "Structured Programming in Large Systems," *Debugging Techniques in Large Systems*, R. Rustin (ed.), Prentice Hall, Inc., Englewood Cliffs, New Jersey, pp. 41-55.
16. Mills, H. D., "The Complexity of Programs," *Program Test Methods* W. D. Hetzell (ed.), Prentice Hall, Inc., pp. 225-239, 1973.
17. Mills, H. D., "Software Engineering," *Science*, Vol. 195, March 1977, pp. 1199-1205.
18. Myers, G. J., "An Extension to the Cyclomatic Measure of Program Complexity," *SIGPLAN Notices*, October 1977, pp. 61-64.
19. Elshoff, J. L., and M. Marcotty, "On the Use of the Cyclomatic Number to Measure Program Complexity," *SIGPLAN Notices*, December 1978.
20. Miller, G. A., "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychological Review*, Vol. 63, 1956, pp. 81-97.
21. DeVore, C., "AEGIS: Shield of the Fleet of the 80's," *Signal Magazine*, October 1978.

STRUCTURED PROGRAMMING IDEAS GLOSSARY

1. *Chief Programmer Team*—A group of computer specialists organized into a team much like a surgical team.
2. *Code Walk-Throughs*—A walk-through of the actual code to guarantee that it reflects the design document.

3. *Complexity Measure*—A quantitative attribute of a computer program which measures its logical weight. Mathematically expressed via formula $v(g) = e - n + 2$.
4. *Hipo Hierarchy Chart*—A documentation technique based on function.
5. *Design Walk-Throughs*—A walk-through of the design document checking for errors or omissions in the architecture of the design.
6. *Maintenance Walk-Throughs*—A walk-through of the proposed changes to guarantee the "fix" is not going to unintentionally impact other parts of the system.
7. *Nassi-Shneiderman Charts (Chapin Charts)*—A chart detailing the internal logic of a module.
8. *Programmer Librarian*—A designated person who handles the clerical activities of programming.
9. *Pseudocode*—An informal documentation method representing structured programming logic.
10. *Regression Techniques*—Analytical techniques which are capable of comparing different versions of the system and indicating differences.
11. *Structured Design*—A set of design guidelines to be used at the modular level.
12. *Structured Programming*—A programming technique based on the combination of three basic forms resulting in programs which can be easily read, modified and maintained.
13. *Structured Testing*—Testing guidelines which require that the number of tests not be less than the cyclomatic complexity.
14. *Top-Down Design*—A design strategy which constantly seeks to factor a problem into its smallest parts.
15. *Top-Down Implementation*—Coding in a top-down fashion or the higher level modules first.
16. *Top-Down Testing*—Testing the higher-level modules of a system before the lower-level modules.

A Markovian model for reliability and other performance measures of software systems*

by AMRIT L. GOEL and KAZU OKUMOTO

Syracuse University
Syracuse, New York

INTRODUCTION

Several studies have been undertaken in recent years to investigate the software failure phenomenon, with the objective of developing analytical models for quantitative assessment of software performance. Most of these studies assume that the times between software failures follow an exponential distribution with a failure rate that depends on the number of errors in the system (see, for example, Jelinski and Moranda,⁵ Littlewood and Verrall⁶ and Shooman¹¹). A key assumption made in most of these studies is that the errors are removed with certainty when detected. However, as pointed out in Miyamoto⁷ and Thayer *et al.*,¹² errors are not always corrected when detected. The existing models do not provide a solution for such situations.

In this paper we develop a Markovian model, which we call an Imperfect Debugging Model (IDM), for studying software failures when errors are not removed/corrected with certainty, i.e., for the case of imperfect debugging. Also, expressions are derived for the following probabilistic performance measures:

- Time to a completely debugged system.
- Time to a specified number of remaining errors.
- Number of remaining errors at time t .
- Number of errors detected by time t .
- Reliability function.

Actual failure data from a large Department of Defense (DoD) software project are analyzed and the results compared with the observed values.

MODEL DEVELOPMENT

The following assumptions are made for developing the model:

1. An error causing a software failure, when detected, is corrected with probability p , while with probability

$q(p+q=1)$ we fail to completely remove it. Thus, q is the probability of imperfect debugging.

2. The time, T_i to a software failure, when i errors remain in the system, follows an exponential distribution with parameter $i\lambda$. The parameter λ represents the mean error occurrence rate per unit time.
3. The time to remove an error will be neglected in this model.
4. No new errors are introduced during the debugging process.
5. At most, one error is removed at correction time.

Let $X(t)$ denote the number of errors remaining in a software system at time t . We will use this random variable to describe the behavior of the error removal process as a function of time. Further, let N be the number of errors at the beginning of the debugging phase, i.e., $X(0)=N$.

Suppose that there are i errors in the package at some time. Then from Assumption 1, we note that after the occurrence of the next failure

$$X(t) = \begin{cases} i-1 & \text{with probability } p \\ i & \text{with probability } q \end{cases} \quad (1)$$

In other words, if we were to observe the $X(t)$ process at times of software failures, then its behavior is governed by Equation 1. The transition probabilities P_{ij} from state i to state j are given by

$$P_{ij} = \begin{cases} p & j=i-1 \\ q & j=i \\ 1 & i=j=0 \\ 0 & \text{otherwise} \end{cases} \quad i, j=0, 1, 2, \dots, N. \quad (2)$$

A diagrammatic representation of transitions between states corresponding to Equation 2 is given in Figure 1.

Assumption 2 implies that the Probability Density Function (PDF) and the Cumulative Distribution Function (CDF) of the random variable T_i are, respectively given by

$$f_i(t) = i\lambda \cdot e^{-i\lambda t} \quad (3)$$

and

$$F_i(t) = 1 - e^{-i\lambda t}. \quad (4)$$

We note that even though the stochastic process $X(t)$

* This work was supported by the Air Force Systems Command's Rome Air Development Center, Griffiss Air Force Base, NY.

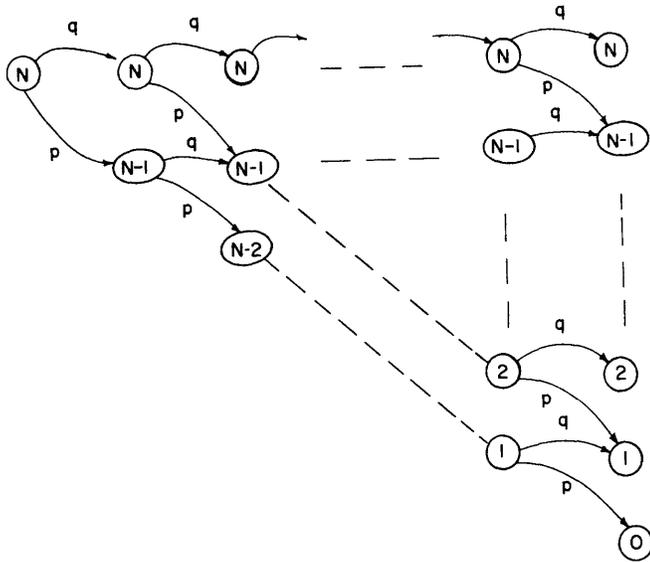


Figure 1—A diagrammatic representation of transitions between states of $X(t)$.

makes transitions from state to state in accordance with Equation 2, the times spent in various states are random and are given by Equation 3. Hence, the process $\{X(t), t \geq 0\}$ forms a semi-Markov process (see, for example, Ross⁹). A typical realization of this process is shown in Figure 2. It should be pointed out that in our formulation the process $X(t)$ undergoes both real and virtual transitions. This means that after an attempt to remove an error the state of $X(t)$ may change or may remain unchanged. In Figure 2, real transitions occur at states $N, N-2$ and i while a virtual transition occurs at state $N-1$.

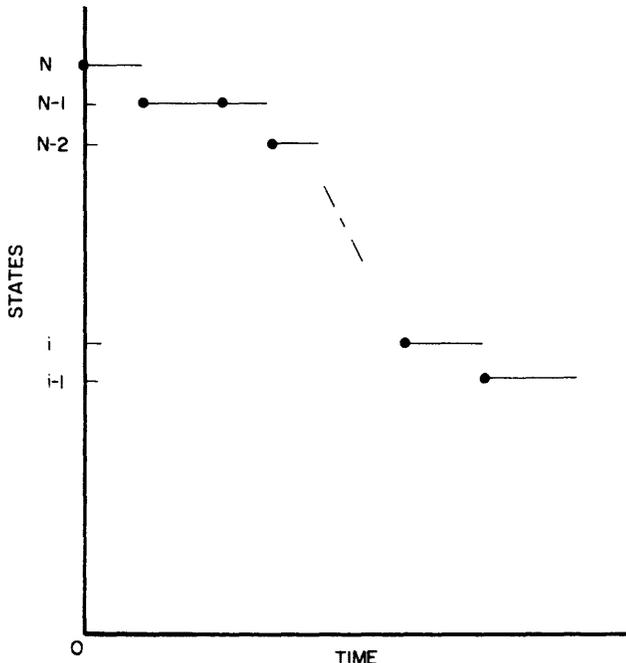


Figure 2—A typical realization of the $X(t)$ process.

Let $Q_{ij}(t)$ denote the one-step transition probability that, after making a transition into state i , the process $X(t)$ next makes a transition into state j by time t . In other words, if a software package has i remaining errors at time zero, then $Q_{ij}(t)$ represents the probability that the next failure, resulting in j remaining errors, will be by time t . Then, we can show that

$$Q_{ij}(t) = P_{ij} \cdot F_i(t) \tag{5}$$

for $i, j=0, 1, 2, \dots, N$. Substituting (2) in (5) yields

$$Q_{ij}(t) = \begin{cases} pF_i(t) & j=i-1 \\ qF_i(t) & j=i \\ 1 & j=i=0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

For known parameters N, p and λ , the probabilities $Q_{ij}(t)$ are obtained from Equation 6. This equation represents the basic model that will be used in the following sections for obtaining the various performance measures for software systems.

EXPRESSIONS FOR PERFORMANCE MEASURES

Distribution of time to a completely debugged software system

Suppose i is the number of errors remaining in a software system at some time during the debugging process and let $G_{i,0}(t)$ represent the probability that the time required to go from i to 0 errors is less than or equal to t . In other words, $G_{i,0}(t)$ represents the CDF of the time required to get a completely debugged system when the current number of remaining errors is i .

Recall that at time zero, $X(0)=N$ and at the time of the first failure

$$X(t) = \begin{cases} N-1 & \text{with probability } p \\ N & \text{with probability } q \end{cases} \tag{7}$$

as shown in Figure 1. Suppose that the debugging at the first error occurrence is perfect. Then the probability of going from N to $N-1$ errors in time $[u, u+du]$ is given by $dQ_{N,N-1}(u)$. If the system clears an error at time u , then the process $X(t)$ restarts with $(N-1)$ remaining errors at time u and the probability of going from $N-1$ to 0 in time $t-u$ is $G_{N-1,0}(t-u)$. Hence the probability of going from N to 0 in time t is

$$\int_0^t G_{N-1,0}(t-u) \cdot dQ_{N,N-1}(u) = Q_{N,N-1} * G_{N-1,0}(t), \tag{8}$$

where $*$ denotes convolution.

Similarly, if the debugging at the first error occurrence is imperfect, the probability of going from N to 0 in time t is

$$\int_0^t G_{N,0}(t-u) \cdot dQ_{N,N}(u) = Q_{N,N} * G_{N,0}(t). \tag{9}$$

Since the events depicted in Equations 8 and 9 are mutually exclusive, we get the renewal equation

$$G_{N,0}(t) = Q_{N,N-1} * G_{N-1,0}(t) + Q_{N,N} * G_{N,0}(t). \tag{10}$$

In general, we get the renewal equation

$$G_{i,0}(t) = Q_{i,i-1} * G_{i-1,0}(t) + Q_{i,i} * G_{i,0}(t) \tag{11}$$

for $i=1, 2, \dots, N$

where $G_{0,0}(t)=1$.

Using the Laplace-Stieltjes (L-S) transform technique (see Ross,⁹ for example) to solve the renewal Equation 11, we obtain the probability that the software system will be completely debugged by time t as

$$G_{N,0}(t) = \sum_{j=1}^N C_{N,j} (1 - e^{-jp\lambda t}). \tag{12}$$

where

$$C_{N,j} = \binom{N}{j} (-1)^{j-1}.$$

Distribution of time to a specified number of remaining errors

In many instances a completely debugged software is not cost-effective and we may be willing to tolerate a certain number of remaining errors, say n_0 , which will ensure some desired reliability. The distribution of time to n_0 is then of interest.

Using an approach similar to the one above, we get the renewal equation

$$G_{i,n_0}(t) = Q_{i,i-1} * G_{i-1,n_0}(t) + Q_{i,i} * G_{i,n_0}(t), \tag{13}$$

for $i=n_0+1, \dots, N$

where $G_{n_0,n_0}(t)=1$. Then the probability that the software system will have n_0 errors by time t , starting with N errors at time 0, is obtained by

$$G_{N,n_0}(t) = \sum_{j=1}^{N-n_0} B_{N,j,n_0} \{1 - e^{-(n_0+j)p\lambda t}\}, \tag{14}$$

where

$$B_{N,j,n_0} = \frac{N!}{n_0! j! (N-n_0-j)!} (-1)^{j-1} \frac{j}{n_0+j}.$$

Distribution of number of remaining errors

Let $P_{N,n_0}(t)$ represent the probability that there are n_0 errors remaining in a software package at time t , given that there are N errors at the beginning of debugging, i.e.,

$$P_{N,n_0}(t) = P\{X(t) = n_0 | X(0) = N\}$$

which is the so-called state occupancy probability. Conditioning on the next failure and following an approach similar to the above, we get the following renewal equations:

$$P_{n_0,n_0}(t) = e^{-n_0\lambda t} + Q_{n_0,n_0} * P_{n_0,n_0}(t), \quad N \leq n_0, \tag{15}$$

$$P_{N,n_0}(t) = P_{n_0,n_0} * G_{N,n_0}(t), \quad n_0 < N. \tag{16}$$

The distribution of the number of remaining errors at time t is

$$P_{N,n_0}(t) = G_{N,n_0}(t) - G_{N,n_0-1}(t), \quad n_0=0, 1, 2, \dots, N \tag{17}$$

where

$$G_{N,N}(t) = 1,$$

$$G_{N,-1}(t) = 0.$$

Finally, the expected number of remaining errors in the software at time t is

$$E[X(t) | X(0) = N] = \sum_{n_0=0}^N n_0 P_{N,n_0}(t) = N e^{-p\lambda t}. \tag{18}$$

From Equation 18, we note that the number of errors remaining in the software system is expected to decrease exponentially over the debugging time.

Expected number of errors detected by time t

We introduce a new random variable $N(t)$ which denotes the total number of errors detected by time t . The process $\{N(t), t \geq 0\}$ is called a counting process. We are interested in obtaining the expression for the expected number of errors detected, $M_N(t)$, during the debugging period, t , when the initial number of errors is N , i.e.

$$M_N(t) = E[N(t) | X(0) = N]$$

which is called a Markov renewal function. By conditioning on the next software failure, we obtain the renewal equations

$$M_j(t) = F_j(t) + p M_{j-1} * F_j(t) + q M_j * F_j(t), \quad j=1, 2, \dots, N \tag{19}$$

where $M_0(t)=0$. From (19) we obtain

$$M_N(t) = \frac{N}{p} (1 - e^{-p\lambda t}). \tag{20}$$

By taking the derivative of (20) we can get the error detection rate at time t ,

$$M'_N(t) = N\lambda e^{-p\lambda t} \tag{21}$$

which is exponentially decreasing over time. This implies that more errors are detected during the early stages of debugging. Note that if we let $t \rightarrow \infty$ we have

$$M_N(\infty) = \frac{N}{p}$$

which is the expected number of software errors detected by the end of debugging.

Let us now consider the case when the detected errors are separated as new errors and errors which were not corrected due to imperfect debugging. Let $N_I(t)$ be a random variable which denotes the total number of imperfect debugging errors by time t . Then we can show that

$$D_N(t) = q M_N(t), \tag{22}$$

where

$$D_N(t) = E[N_I(t) | X(0) = N].$$

Note that $D_N(\infty) = q \frac{N}{p}$. Equation 22 implies that 100q percent of the software failures during debugging will be due to imperfect debugging.

Software system reliability

So far we have studied the stochastic behavior of the number of errors in the software system during the debugging period. Now we investigate the distribution of the time between software failures and study the problem of reliability growth. From the second section recall that the random variable T_i denotes the time to next failure when the number of remaining errors is i and $F_i(t)$ is the CDF of T_i . Let X_k denote the time between the $(k-1)$ st and k th software failures and $\Phi_k(x)$ be the CDF of X_k . Note that X_k does depend on the number of remaining errors at the $(k-1)$ st failure but this number is not explicitly known. Further, let η_k be a random variable which denotes the number of remaining errors between the $(k-1)$ st and k th software failures. Then, from the second section we have

$$\eta_1 = N$$

$$\Phi_1(x) = F_N(x),$$

and

$$\Phi_2(x) = pF_{N-1}(x) + qF_N(x).$$

In general, we have

$$\Phi_k(x) = p(X_k \leq x) = \sum_{i=N-(k-1)}^N p(X_k \leq x | \eta_k = i) p(\eta_k = i)$$

or

$$\Phi_k(x) = \sum_{j=0}^{k-1} p[X_k \leq x | \eta_k = N - k + j + 1] p(\eta_k = N - k + j + 1)$$

or

$$\Phi_k(x) = \sum_{j=0}^{k-1} \binom{k-1}{j} p^{k-j-1} q^j F_{N-(k-j-1)}(x). \tag{23}$$

This is called a mixture of exponential distributions with binomial mixing portions. It can be shown that $\Phi_n(x)$ is a Decreasing Failure Rate (DFR) distribution. The reliability function at the k th stage, i.e., between $(k-1)$ st and k th failure, is given by

$$R_k(x) = p\{X_k > x\}$$

$$= 1 - \Phi_k(x)$$

$$= \sum_{j=0}^{k-1} \binom{k-1}{j} p^{k-j-1} q^j \bar{F}_{N-(k-j-1)}(x) \tag{24}$$

where in general

$$\bar{F}_N(x) = 1 - F_N(x) = e^{-\lambda x}.$$

From a practical point of view the reliability obtained in (24) is not easy to work with. For computational purposes we use the following approximation.

$$R_k(x) \sim e^{-[N - p(k-1)]\lambda x}, \quad k=1, 2, \dots \tag{25}$$

For details of this approximation, see Goel and Okumoto.²

APPLICATION TO A LARGE SOFTWARE PROJECT

In this section we analyse the error data from a large software project and compute various performance measures using the results of the two preceding sections. The error data is taken from Fries,¹ and represents software errors from a large DoD systems development project. The project consisted of approximately 320,000 assembly language instructions. It included the operational software and the simulation software necessary to develop and test the former. Software Problem Reports (SPRs) were written in the time period from the beginning of configuration management (approximately start of integration testing) to delivery of the software. A total of 2036 SPRs were encountered during this period and they were categorized into 20 major groups. Data are also included about the source of errors, the type of correction made, and the time to find and fix the error.

For purposes of this analysis, we consider 1612 errors reported during the last 12 months of the software testing phase. For estimating the parameters N , p and λ we use the method given in Goel and Okumoto.³ The estimated values of these parameters are

$$\hat{N} = 2108, \quad \hat{p} = 0.936 \quad \text{and} \quad \hat{\lambda} = 0.1127.$$

Substituting \hat{N} , \hat{p} and $\hat{\lambda}$ for N , p and λ , respectively, in Equation 18, the estimated value of the expected number of remaining errors over the testing period t is obtained as

$$\hat{E}[X(t) | X(0) = \hat{N}] = 2108 e^{-(.936)(.1127)t}.$$

A plot of this quantity is shown in Figure 3. Also shown in this figure are the observed values by month. The fitted curves for $M_N(t)$ and $D_N(t)$ from Equations 20 and 22 are shown in Figure 4 along with the actual values for these quantities. From Figures 3 and 4 we see that the model seems to describe the behavior of the software error phenomenon very well.

The reliability function for the system is obtained from Equation 25 as

$$\hat{R}_k(x) = e^{-[2108 - .936(k-1)](0.1127)x}.$$

Plots of reliability for $k=1613(50)1863$, i.e., for the cases when the number of errors removed is 1612(50)1862, are shown in Figure 5. From these plots the extent of reliability growth with k can be easily evaluated. For example, $R_{1613}(0.1) = 0.24$ while $R_{1863}(0.1) = 0.74$, i.e., the value of $R(0.1)$ increases by 200 percent when k goes from 1613 to 1863.

The plots in Figure 5 can also be used to determine the expected time required to achieve a desired reliability. Suppose our objective is to have a reliability of 0.3 at 0.2 weeks.

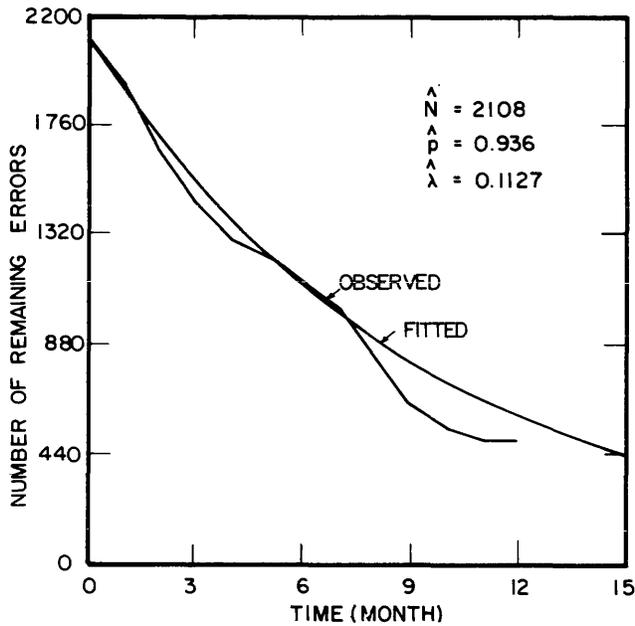


Figure 3—Plots of the number of remaining errors and the fitted curve.

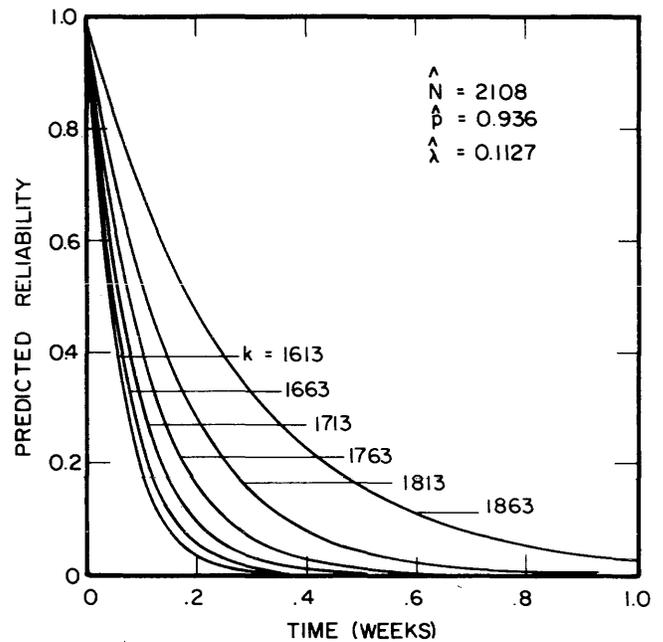


Figure 5—Plots of the reliability function for various values of k .

We want to know the number of errors that must be removed to achieve this reliability. In other words we want to know the value of k that yields $R_k(0.2)=0.3$. From Figure 5, we see that $R_{1813}(0.2)=0.3$ so that $k=1813$ errors to achieve desired reliability for the software system. Then the number of remaining errors will be

$$\hat{n}_0 = 2108 - .936(1813 - 1)$$

or

$$\hat{n}_0 = 412.$$

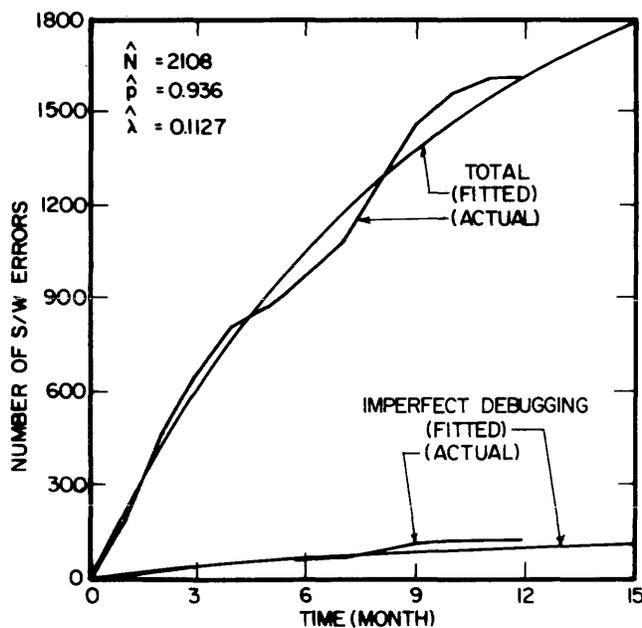


Figure 4—Total and imperfect debugging errors by month.

The expected time required to remove $(N - n_0)$ errors is

$$\mu_{N, n_0} = \frac{1}{p\lambda} \sum_{j=n_0+1}^N \frac{1}{j} = \frac{1}{p\lambda} \ln \frac{N-1}{n_0+1}.$$

For our case, we get

$$\hat{\mu}_{2108, 412} = \frac{1}{(0.936)(0.1127)} \cdot \ln \frac{2108+1}{412+1} = 15.46 \text{ months.}$$

In other words, to achieve the desired reliability, we will need to continue testing for $15.46 - 12 = 3.46$ additional months.

CONCLUSION

We have developed an imperfect debugging model (IDM) for software systems and derived expressions for various performance measures in terms of the first passage time distribution of the underlying semi-Markov process. The failure data from a large software project were analyzed using the model developed in this paper. A comparison of the fitted and the observed values indicates that the model provides a good description of the underlying failure phenomenon. Also, reliability curves were used to determine the debugging time required to achieve a desired level of software system performance.

REFERENCES

1. Fries, M. J., "Software Error Data Acquisition," Boeing Aerospace Co., *Final Technical Report*, RADC-TR-77-130, 1977.
2. Goel, A. L., and K. Okumoto, "An Imperfect Debugging Model for Reliability and Other Quantitative Measures of Software Systems," RADC-TR-155, Vol. 1, 1978.

3. Goel, A. L., and K. Okumoto, "An Analysis of Recurrent Software Errors in a Real-Time Control System," *Proc. of ACM*, 1978, pp. 496-501.
4. Goel, A. L., and K. Okumoto, "A Time-Dependent Error Detection Rate Model for Software Reliability and Other Performance Measures," (to appear) *IEEE Trans. Rel.* (Special issue on Software Reliability), 1979.
5. Jelinski, Z., and P. Moranda, "Software Reliability Research," *Statistical Computer Performance Evaluation*, W. Freiberger (ed.), Academic Press, 1972, pp. 465-484.
6. Littlewood, B., and J. L. Verrall, "A Bayesian Reliability Growth Model for Computer Software," *Applied Statistics*, Vol. 22, 1973, pp. 332-346.
7. Miyamoto, I., "Software Reliability in On-Line Real Time Environment," *Proc. 1975 International Conference on Reliable Software*, pp. 194-203.
8. Okumoto, K., and A. L. Goel, "Availability and Other Performance Measures of Software Systems Under Imperfect Maintenance," *Proc. of Computer Software & Applications Conference*, 1978, pp. 35-40.
9. Ross, S. M., *Applied Probability Models with Optimization Applications*, Holden-Day.
10. Schneidewind, N. J., "Analysis of Error Processes in Computer Software," *Proc. 1975 International Conference on Reliable Software*, pp. 337-346.
11. Shooman, M. L., "Operational Testing and Software Reliability Estimation During Program Development," *Record 1973 IEEE Symposium on Computer Software Reliability*, pp. 51-57.
12. Thayer, T. A., M. Lipow and E. C. Nelson, "Software Reliability Study," TRW Defense & Space Systems Group, *Final Technical Report*, RADC-TR-76-238, 1976.

Partial match retrieval for non-uniform query distributions*

by V. S. ALAGAR and C. SOOCHAN

Concordia University
Montreal, Canada

INTRODUCTION

A good file design and investigation of reasonably fast and efficient algorithms to perform a required task on the information stored in the file are important in data base studies. In a reasonably large class of problems one can devise and evaluate algorithms with respect to measures independent of the storage and structuring of data. But when the question is the investigation of a suitable data structure for retrieval of a specific kind, algorithms and their measures depend heavily on the proper file design. This paper addresses both file design and algorithm design when searches are to be made in a data base for queries of special type.

We are concerned with information retrieval based on secondary keys. In general the secondary keys form a subset of the attributes of a record and, thus, they can not uniquely identify a record. We view a secondary key as a k -tuple (i_1, \dots, i_k) , where i_1, \dots, i_k is a subset of the attributes. The standard methods of whole key comparisons cannot be done and hence the complexity measures for search problems involving secondary keys are different.

Several authors, notably Schkolnick,¹¹ have given methods to select an optimal or near optimal subset of attributes for indexing under suitable assumptions on the probability distributions of transactions to be done on the data base. Another approach to handle multi-attributed files is to design tries, originally proposed by de la Briandais⁷ and Fredkin.⁸

A trie is essentially a k -ary tree whose leaves correspond to records and each internal node is a vector with components corresponding to attribute values. Each node on Level j specifies the set of all secondary keys that begin with a sequence of j characters and the node specifies a k -way branch depending on the $(j+1)$ st attribute.

Recently, considerable interest has been shown in the design of good tries for handling partially-specified queries. Partial match retrieval is concerned with searching and accessing those records that satisfy the given query, although only fewer than k attribute positions are specified in the query. A survey of the software and hardware requirements for such associate retrieval problems is given in Minker.⁹

Rivest¹⁰ is the first to propose and use tries for partial match searching on binary attributed files. He discusses the

average case behavior of tries for uniform data and uniform query patterns. One of his conclusions is the conjecture that if s out of k attributes are specified in a query and there are N buckets, the average number A of buckets examined must be at least $N^{1-s/k}$. This analysis for tries shows that the average A is close to $N^{\log_2(2-s/k)}$.

Burkhard³⁻⁵ has investigated partial match file designs and in particular gives methods to construct tries with good-worst case performance. Bentley and Burkhard² have suggested several strong heuristics for the design of tries to handle partial match retrieval on real life files.

We report our preliminary results on the construction and performance of tries for non-uniform data and non-uniform query patterns. Rivest suggested this as an open problem. The methods to be discussed here have a strong bearing on the characteristics of the records in a file and assume a known probability distribution on the query patterns. The concept of non-uniformity is thus different from the one used in Reference 5. Our empirical results and statistics gathered on exhaustive testing are reported in the fifth section. These results convincingly indicate that the tries constructed handle partially specified queries better both in the average and in the worst case; the average case performance is better than the one reported by Rivest¹⁰ and the worst case results are better than the one reported by Burkhard.⁴

DEFINITIONS AND PRELIMINARY RESULTS

In this section we define the basic notions of record, file, query and trie. Some examples are also given. Let A_1, \dots, A_k be a finite set of attributes, where A_i takes values from a domain D_i , $1 \leq i \leq k$. A record R is defined to be an ordered k -tuple (r_1, \dots, r_k) , where r_i is in D_i . A file F is a collection of records and thus a subset of $D_1 \times D_2 \times \dots \times D_k$. Thus every record has the same number of attributes and we do not consider records with variable number of attributes. A key is an ordered k -tuple and is an element of F . A fully specified query is an element of $D_1 \times \dots \times D_k$. A partially specified query is an element of $(\prod_1^k (D_i U \{*\}))$, where the $*$ indicates the unspecified of a component. One can regard this special symbol to mean a 'don't care' condition. Any record which matches the specified q values is relevant for this query. We restrict ourselves to the special case when each $D_i = \{0, 1\}$ and we say that F is a binary attributed file.

* This research is supported by National Research Council Grant A 3552.

We let Q denote the set of all queries, i.e. the set of all sequences of length k over the alphabet $(0, 1, *)$. Thus, the cardinality $|Q|=3^k$. Let Q_s denote the set of all queries in each of which exactly s positions are specified (thus there are $u=k-s$'s). It is easy to see that $|Q_s|=\binom{k}{s}2^s$. For any query q_s in Q_s , we let $q_s(f)$ denote the set of records each matching q_s in the s specified components. *Example 1*—Let $k=3$ and

$$F=\{001, 011, 100, 101, 111\}$$

The query $q_1=(0, *, 1)$ requests all records with $r_1=0$ and $r_3=1$ without any regard to r_2 value. Thus $q_1(F)=\{001, 011\}$. The set of all queries for $s=2$ is

$$Q_2: \begin{aligned} & (*, 0, 0), (*, 0, 1), (*, 1, 0), (*, 1, 1) \\ & (0, *, 0), (0, *, 1), (1, *, 0), (1, *, 1) \\ & (0, 0, *), (0, 1, *), (1, 0, *), (1, 1, *) \end{aligned}$$

One can verify that the total number of records which are retrieved from F to answer all queries in Q_2 is 15. If we assume that the queries from Q_2 occur with uniform distribution then the average number of records examined to answer a query from Q_2 is 1.25.

We will define a binary search trie (to be referred to simply as a trie). Each leaf node of a trie stores one record or bucket. Each internal node at level i (the root is at level 1) specifies a 2-way branch based on the j th component of the record being stored. We insist that

1. Each internal node specifies an attribute position j , $1 \leq j \leq k$, such that no other node from the root to that position specifies j .
2. If attribute j is tested in a node, then all nodes in the left subtree of that node have a 0 in attribute position j and all nodes in the right subtree of that node have a 1 in attribute position j .

The records in F of Example 1 may be stored as shown in Figure 1. We have stored a record in a leaf node as soon as we find that this is the only record in its subtree. Obviously this is not the only possible trie structure for F . The trie in Figure 2 is another mode of storage for F .

A trie such as the one in Figure 1 or Figure 2 is known as a pruned trie. More formally we have the following defini-

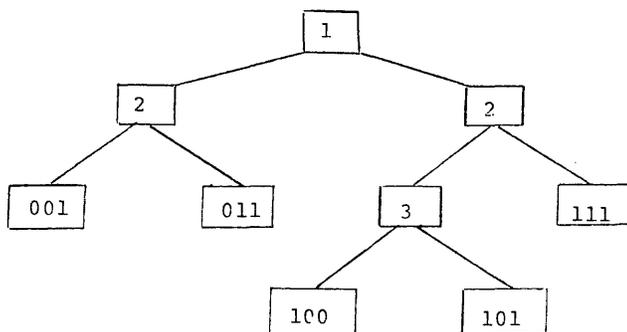


Figure 1

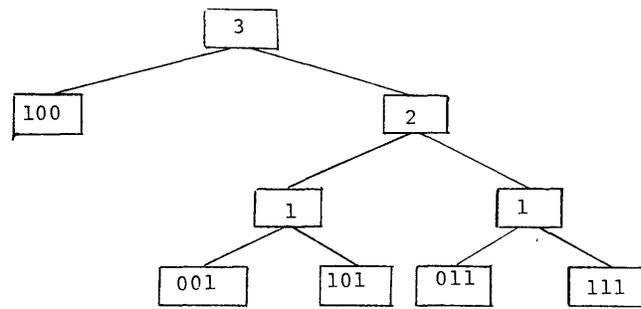


Figure 2

tions:⁶ A full binary trie for a file F is a binary tree with all leaves at level $(k+1)$, where k is the number of attributes in a record. The skeleton of a full trie is completely specified by the order of testing of the attributes on the trie. A pruned or collapsed trie is obtained from a full trie by deleting all empty leaves and pruning upwards until each leaf node stores one record. The decision regarding the order of testing the attributes can be made either globally (in which case all paths have the same relative order of attributes) or such a decision can be localized at each node. In the latter case the attributes might be tested in different relative orders along different paths from the root to a leaf node. A trie in which the order of testing is explicit in each node is called an O-trie. The tries shown in Figures 1 and 2 are collapsed O-tries; they have the same depth but have different shapes. If queries from the set $\{(1, *, 0), (*, *, 0), (*, 1, 0)\}$ come more often than queries from the set $\{(0, *, 1), (*, *, 1), (*, 0, *)\}$ then fewer comparisons need be made in Trie 2 than in Trie 1. Moreover, the number of records examined in Trie 2 is smaller than the number of records examined in Trie 1 to answer a single query from the first set.

The methods that we describe in the fourth section will construct collapsed order containing tries. Thus a record is stored as a leaf as soon as it is the only record in the subtree; this reduces the average path length from the root to a record in a random binary trie from k to about $\log_2 N$, where N is the size of the file. We remark that the partial match file tries considered by Burkhard⁴ differ slightly from ours in the sense that he considers full (complete) binary tries and we get full binary tries only for the case $N=2^k$.

Without any regard to specific implementations, we assume that the space taken for a trie is simply the number of internal nodes in the trie. If the size of the file is small then the entire trie can be maintained in the main memory and in this case the cost of a search (or search time) is the number of internal and leaf nodes visited during the search. If the file is large, which is usually the case in real-life situations, the internal nodes of the trie can be maintained in the main memory and the leaf nodes are stored in external storage such as disk. In this case each leaf node contains more than one record; we refer to this as a bucket or a page. The cost of a search in this situation is the number of distinct accesses to the secondary storage. Since the time to search the trie itself is relatively small compared to the time needed for several accesses to the secondary storage, the cost measured

in number of buckets examined is justified. In our analysis the following is true: All buckets examined by a search contain all of the records in the data base satisfying the given partially-specified query. However, all the records in the examined buckets are not necessarily relevant to the given query. This would happen especially when a certain unspecified attribute in the given query is not chosen for testing anywhere along the search path in the trie. For example, the query 01* will examine two buckets in the trie shown in Figure 2, although the record 100 in one of the buckets is not relevant to this query; however the other bucket examined has the record 011 which is the only relevant record for this query in the data base.

RECORD SPACE AND QUERY SPACE

All the previous researchers in this area assumed uniform distribution for record space as well as query space. This approach presupposes that all queries from the query space are equally likely to be queried when the data base contains a set $N(\leq 2^k)$ of records and these N -subsets are once again equally likely. Such an assumption leads to closed form results for the average case and worst case behavior of search algorithm for a trie.

But in real-life situations it is more natural to encounter non-random data. For example, consider a data base consisting of records over non-binary alphabets. Each record can be suitably coded to obtain a binary coded data base. Trie methods can be applied to this coded data base and with suitable decoding procedures querying in the original data base can be handled. In such cases the strong characteristics of the original data base and the properties must be taken into account in trie designs. Clustering, sparsity and coding influence the distribution of the binary-coded data base.

As pointed out by Rivest,¹⁰ for non-random data an interesting problem is the choice of bits in order to split the file in the best possible manner. Thus, it is natural to assume that certain attributes are more often queried than others. Such an assumption must clearly reflect the relative frequency of instances of query patterns in the query space. The sample of query instances actually appearing over a period of time must be estimated in an unbiased way and must be incorporated in the construction and performance of the tries.

It is also necessary for us to remark here that the concept of randomness or non-randomness must be clearly defined so that the results can be evaluated properly. Burkhard⁵ has discussed non-uniform partial match file designs; he achieves non-uniformity of labels within the tries. One clear accomplishment of this non-uniform distribution of labelling has been to obtain inversions of certain recurrence relations and establishing bounds for the worst case behavior. Our concept of non-uniformity is different, although our methods of trie construction based on this concept might cause non-uniform distribution of labels within the constructed tries.

In the next section we discuss three methods for construction of tries. These methods seem immensely suited to the

following problem instances—non-uniform data and uniform query pattern; uniform data and non-uniform query pattern and non-uniform data and non-uniform query pattern. The basic approach in each method is to make use of the characteristics of the records and the distribution of queries to decide the order of testing the attributes and obtain a collapsed O-trie.

TRIE DESIGNS

We discuss three methods for constructing tries; the first one is the one suggested by Bentley and Burkhard;² the second method is appropriate to non-uniform query distributions and the last method is suitable for non-uniform record and query spaces.

Method A—Rivest¹⁰ observes that very unbalanced tries have good average case performance. Assuming that all partial match queries are equally likely (uniform), the probability that a query will examine a node at level l in the trie is $(2/3)^l$; this is because there are 2^{3^k-l} queries out of 3^k queries that will visit this node. Hence, a query will less frequently visit the node that is farthest from the root. This suggests that one should try to maximize the unbalance at every level in the trie, hoping to achieve global unbalance. As suggested by Bentley and Burkhard,² it seems advantageous to look several levels down in the trie in choosing a bit position to be tested at a certain level; this look-ahead scheme tries to avoid those choices of bits that would optimize locally but not globally.

The notion of unbalance with respect to a bit position can be defined in more than one way. For example, consider the file as a table of N rows and k columns. Compute c_1, \dots, c_k , where c_j is the j th column sum. Then the ratio $c_i/(N-c_i)$ is a measure of unbalance for bit position i ; another measure for the same bit position would be the absolute value of $(N-2c_i)$. Whatever the choice of measure, the main idea is to choose that bit position for a level in the trie that maximizes the measure and also maximizes the unbalance of the sub-trie at that level.

Thus, a brief description of the method will be as follows: For definiteness assume the measure $c_i/(N-c_i)$ for every bit position that is not yet selected for testing in the trie. For the full file of N records, we compute the k ratios $c_i/(N-c_i)$ and pick j , $1 \leq j \leq k$ for which the ratio is maximum. The attribute j is chosen to be tested at the root. Subdivide the file into two parts so that the left sub-trie will have records that have 0 in attribute position j and the right subtrie will have records that have 1 in attribute position j . We recursively construct the sub-tries for the partitioned files.

Example 2—Consider the file $F=\{a, b, c, d, e, f\}$, $a=1011$, $b=0001$, $c=0110$, $d=1101$, $e=1010$ and $f=1111$. The trie constructed by Method A is shown in Figure 3.

Method B—We assume that P is a given probability matrix having k rows (one for each attribute) and three columns (for the values *, 0, 1). The entry $P_{i,j}$ is the probability that the i th attribute will have the value j , $1 \leq i \leq k$, $j=*, 0, 1$. The probability associated with any query can be computed

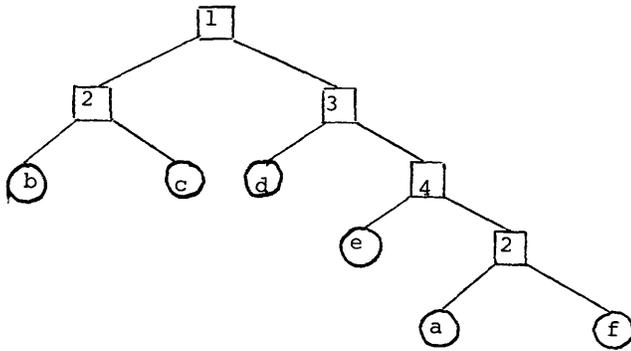


Figure 3

from P . In real situations the most probable queries are concentrated in a small subset of the query space and the matrix is assumed to reflect such a situation.

This method is mainly concerned with P and one may assume that the record space is uniform; i.e., the N records are equally likely to be chosen from 2^k records. Since a search on the trie will visit both sub-tries of a node at Level l if the attribute tested there is unspecified in the query, and the probability of unspecified of that attribute is available in the matrix P , it is important to test that attribute in the lowest possible level in the trie; for this would decrease the number of buckets examined. Towards this we choose those attributes with high probability of unspecified and test them lower in the trie and choose those with low probability of being unspecified to be tested higher in the trie. Hence, we order the probabilities that the attributes may be unspecified. If (j_1, \dots, j_k) is the ordering of the attributes corresponding to the increasing order of the probabilities of unspecified, then this is the order in which the attributes would be tested along any path from the root to a leaf node. Without any regard to the file, the previous ordering establishes a full binary trie skeleton. At the root the attribute j_1 is tested; in Level 2 the attribute j_2 is tested and so on. The records are stored as leaf nodes in this trie and finally the trie is collapsed to minimize the internal path length. This collapsed order-containing and order-preserving trie will retain the same relative order of testing of the attributes, i.e., j_i is tested higher in the trie than j_j if and only if j_i precedes j_j in the ordering obtained from P . Note that not all attributes would be tested along every path from the root to a leaf node. We have reason to believe from our results that the branching decisions made on the untested bit positions are favorable both in the average and worst case analysis. The trie constructed by this method for Example 2 is shown in Figure 4.

Method C—Here we propose to exploit both the query space and record space and record space non-uniformity. In some sense we correlate the structure of most probable queries with record structures to choose bit positions to be tested at various levels.

From the file F considered as a $(0, 1)$ matrix and the probability distribution matrix P , we form the product $F.P$, a matrix with N rows and three columns. The entries in this matrix can be looked upon as 'similarity measure' between

the record space and query space. The selection of attributes and their order of testing is done as follows:

Find the minimum in the first column of $F.P$ and the maximums in Columns 2 and 3. Consider the subset of F indicated by those row numbers wherein these extremums occur. Note that these extremums may not be unique. Apply Method A to this subset of F and select that attribute number that maximizes the unbalance at the root. Partition the file so that all records with 0 in the selected attribute position will be in the left sub-trie and those with 1 in that position will be in the right sub-trie. The partitioned files and the remaining attributes are used to compute the similarity measure matrix for the sub-tries and the procedure is repeated.

Some remarks are in order regarding the methods discussed here. Method A is one of the heuristics suggested in Reference 2, but we have tried several modifications. This method is easy to implement but requires several passes over the data. If the measure is properly normalized and the measure of skewness or unbalance of every node lies between p and q , $0 < p < q < 1$, $q = 1 - p$, it can be shown that the amount of work necessary to construct the trie with N leaf nodes is of the order $O(kN \log_\alpha N)$, $\alpha = p^{-p}q^{-q}$. For very large files (large k and N close to 2^k), we have tried to reduce the amount of work by considering statistical sampling of subsets of records and estimating the attribute position that maximizes the unbalance. Our empirical results on the sampling technique are not conclusive; yet in several problem instances produce a good ordering of attributes in a small amount of time. We also remark that this method is very sensitive for update (insertion, deletion) operations; i.e., requires a total reorganization to produce a good trie for dynamically changing files.

The other two methods are suited for non-uniform query and record sets. The order of testing the attributes in Method

	*	0	1
1	1/3	1/2	1/6
2	1/4	1/3	5/12
3	1/5	3/10	1/2
4	1/2	1/5	3/10

P=

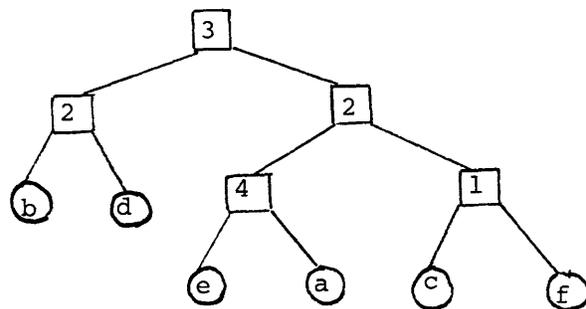


Figure 4

B is totally independent of the record space. Thus, it is well suited for all kinds of transactions i.e., retrieval, insertion and deletion. The cost of constructing the skeleton trie includes the cost of ordering the k probabilities of unspecifications and the insertion of N records; thus, it is of the order $O(k \log_2 k) + O(hN)$, where h is the average internal path length of the trie. These remarks do not apply to Method C.

CONCLUDING REMARKS AND TEST RESULTS

For a given trie, one can always find a partially specified query for which the cost of search is maximum. If this query occurs more often than others in practice, the constructed trie is far from optimal even in the average case. Keeping this in mind we have given methods, wherein near optimal number of buckets are examined for "almost all" query instances. We shall clarify this notion of "almost all."

Suppose B_s , A_s and W_s denote the optimal, average and worst case costs for answering a query with s specified attributes. Let Q_1 contain the set of queries for each of which the cost of searching the trie is either close to B_s or A_s and Q_2 contain those queries for each of which the cost of searching the trie lie between A_s and W_s . We say that the performance of the trie is near optimal almost everywhere (in probabilistic sense) over the query space if the following conditions are met:

1. $|\lceil A_s \rceil - B_s|$ remains small for all $s=0, \dots, k$. Here $\lceil \cdot \rceil$ is the ceiling function.
2. $\sum_{q \in Q_2} P_r[q]$ remains small.
3. $\sum_{q \in Q_1} P_r[q]$ is large.

TABLE I.—Uniform Query Distribution (Method A)

	K=4	u	B	T _B	W	T _W	A	T _A	N ^{u/k}
N=9	0	1.0	16.0	1.0	16.0	1.0	0.0	1.0	1.0
	1	1.0	7.81	2.0	24.19	1.77	0.0	1.73	1.73
	2	1.33	2.72	4.0	7.78	3.06	7.78	3.0	3.0
	3	3.39	1.41	6.81	2.29	5.28	3.54	5.2	5.2
N=12	4	9.0	1.0	9.0	1.0	9.0	0.0	9.0	9.0
	0	1.0	16.0	1.0	16.0	1.0	0.0	1.0	1.0
	1	1.0	4.16	2.0	27.84	1.87	0.0	1.86	1.86
	2	1.87	2.08	4.0	13.68	3.48	13.68	3.46	3.46
N=17	3	4.9	1.63	7.94	1.5	6.48	4.08	6.45	6.45
	4	12.0	1.0	12.0	1.0	12.0	0.0	12.0	12.0
	K=5								
	0	1.0	32.0	1.0	32.0	1.0	0.0	1.0	1.0
N=17	1	1.0	17.14	2.0	62.86	1.78	0.0	1.76	1.76
	2	1.07	3.19	4.0	31.16	3.17	31.16	3.11	3.11
	3	2.57	1.94	7.9	3.25	5.59	21.6	5.47	5.47
	4	6.85	1.30	12.54	1.46	9.79	3.42	9.64	9.64
N=24	5	17.0	1.0	17.0	1.0	17.0	0.0	17.0	17.0
	0	1.0	32.0	1.0	32.0	1.0	0.0	1.0	1.0
	1	1.0	8.3	2.0	71.7	1.90	0.0	1.89	1.89
	2	1.74	3.31	4.0	51.79	3.59	0.0	3.57	3.57
N=24	3	4.31	2.40	8.0	10.71	6.78	10.71	6.73	6.73
	4	10.5	1.8	15.08	1.65	12.77	3.09	12.71	12.71
	5	24.0	1.0	24.0	1.0	24.0	0.0	24.0	24.0

TABLE II.—Non-uniform Query Distribution (Method B)

	K=4	U	B	T _B	W	T _W	A	T _A
N=9	0	1.0	16.0	1.0	16.0	1.0	1.0	0.0
	1	1.0	7.74	2.0	24.26	1.54	0.0	0.0
	2	1.32	2.96	4.0	8.52	2.51	8.52	2.51
	3	3.5	1.68	7.0	2.08	4.6	3.18	3.18
N=12	4	9.0	1.0	9.0	1.0	9.0	0.0	0.0
	0	1.0	16.0	1.0	16.0	1.0	0.0	0.0
	1	1.0	4.1	2.0	27.9	1.75	0.0	0.0
	2	1.9	2.12	4.0	14.08	3.19	14.08	3.19
N=17	3	4.92	1.7	7.8	2.5	6.13	4.2	4.2
	4	12.0	1.0	12.0	1.0	12.0	0.0	0.0
	K=5							
	0	1.0	32.0	1.0	32.0	1.0	0.0	0.0
N=17	1	1.0	17.28	2.0	62.72	1.63	0.0	0.0
	2	1.02	2.68	4.0	33.88	2.74	33.88	2.74
	3	2.5	2.04	7.9	4.94	4.78	21.18	4.78
	4	6.8	1.68	12.7	2.08	8.82	3.52	8.82
N=24	5	17.0	1.0	17.0	1.0	17.0	0.0	0.0
	0	1.0	32.0	1.0	32.0	1.0	0.0	0.0
	1	1.0	8.26	2.0	71.74	1.80	0.0	0.0
	2	1.74	4.78	4.0	53.44	3.33	0.0	0.0
N=24	3	4.18	2.74	8.0	13.4	6.27	13.4	6.27
	4	10.38	1.62	15.22	2.44	12.14	3.16	3.16
	5	24.0	1.0	24.0	1.0	24.0	0.0	0.0

Our extensive simulation for values $k=3, 4$ and 5 and the collected statistics have shown that tries constructed by Methods B and C conform to this standard. See Tables I and II. The following explanation applies to the tables:

U—Number of unspecified attributes.

B—Best Case cost (averaged over all tries with N leaf nodes).

T_B—Total number of queries that achieve B.

W—Worst case cost.

T_W—Total number of queries that achieve W.

A—Average case cost.

T_A—Total number of queries that exceed [A].

For Method A and uniform querying, the computed average A_s satisfies the inequality,

$$N^{u/k} < A_s < N^{\log_2(1+u/k)}, \quad u = k - s$$

$$u \neq 0, k$$

The right side of the previous inequality is the best average cost reported and the left side is the conjectured minimum cost. See Reference 10. So one conclusion is that Method A improves the average case performance. Our empirical studies show that for large values of k , and N close to 2^k , A_s approaches $N^{u/k}$. This only reaffirms the earlier conjecture on the lower bound. In the case of non-uniform querying, we have observed in several cases A_s lower than $N^{u/k}$; in general A_s oscillates on either side of $N^{u/k}$. However, according to the 'almost everywhere' concept, the average case behavior prevails almost everywhere. This is due to the fact that 'bad' queries are taken care of by

these methods. Thus, our overall conclusion is that the probability of worst case behavior is very low, the average retrieval cost prevails almost everywhere over the query space and hence the constructed tries are either optimal or near optimal.

ACKNOWLEDGMENTS

We wish to thank one of the referees for suggesting two important references as well as for his critical remarks. We also thank the National Research Council of Canada for financial support in carrying out this research.

REFERENCES

1. Aho, A., and J. D. Ullman, "Optimal Partial Match Retrieval When Fields Are Independently Specified." (unpublished).
2. Bentley, J. L., and W. A. Burkhard, "Heuristics for Partial Match Retrieval Data Base Design," *IPL*, Vol. 4, No. 5, 1976, pp. 132-135.
3. Burkhard, W. A., "Partial Match Retrieval," *BIT*, Vol. 16, 1976, pp. 13-31.
4. Burkhard, W. A., "Hashing and Trie Algorithms for Partial Match Retrieval," *ACM Trans. Data Base Systems*, Vol. 2, 1976, pp. 175-187.
5. Burkhard, W. A., "Non-uniform Partial Match Retrieval File Designs," *Theoretical Computer Science*, Vol. 5, 1977, pp. 1-23.
6. Comer, D., and R. Sethi, "The Complexity of Trie Index Construction," *JACM*, Vol. 3, 1977, pp. 428-440.
7. de la Briandais, "File Searching Using Variable Length Keys," *Proc. Western Joint Comp. Conference*, IRE, New York, 1959, pp. 295-298.
8. Fredkin, E., "Trie Memory," *CACM*, Vol. 9, 1960, pp. 490-499.
9. Minker, J., "An Overview of Associative or Content Addressable Memory Systems and a KWIC Index to the Literature," *Technical Report TR-157*, University of Maryland Computer Center, College Park, 1971.
10. Rivest, R. L., "Partial Match Retrieval Algorithms," *SIAM J. Compt.*, Vol. 1, 1976, pp. 19-50.
11. Schkolnick, M., "Secondary Index Optimization," *ACM-SIGMOD International Conference on Management of Data*, San Jose, California, 1975, pp. 186-192.

Comparing interactive computer services—Theoretical, technical and economic feasibility

by S. A. MAMRAK and P. D. AMER

National Bureau of Standards
Washington, D.C.

and

The Ohio State University
Columbus, Ohio

INTRODUCTION—COMPUTER COMPARISON MEASUREMENT PROBLEMS

The comparison of interactive computer services is a frequent, important and essential activity for those involved in selecting from among alternative remote access services available through a computer network. The comparison and selection process can be very complicated, relying on both nonmeasurable and measurable comparison criteria. Nonmeasurable criteria, such as the availability of a particular compiler, are typically easy and inexpensive to evaluate. In contrast, for those criteria which are measurable, comparison calls for an expensive process of collecting and analyzing relevant performance measurements from the network computer services under consideration. (The term "network computer services" is used to refer to interactive computer services available via a computer network). Further, if measurement is not carefully planned to accommodate certain inherent theoretical, technical and economic problems, the measurements which result may be misleading or invalid. A legitimate question arises as to whether comparison measurements should be done at all. This paper addresses the feasibility of including measurement phases in the process of comparing and selecting interactive network computer services.

Two essential problems arise when measuring interactive services. The first is developing a test workload or "scenario" that is truly representative of a real workload. "Representativeness" is a critical property since measurement experiments are designed so that the performance of an interactive service while processing the test workload is assumed to be close to what its performance would be while processing the real workload. This representativeness problem is not addressed here, but will be discussed in the accompanying paper. The second problem in measuring interactive services arises while collecting and analyzing data about the performance of a test workload. Three aspects of the data collection and analysis problem must be considered

in order to determine the feasibility of comparison measurement:

1. *Theoretical*—Do statistical techniques which provide confidence statements about the probability of having made a correct selection exist for comparing interactive services?
2. *Technical*—Does the technology exist to apply sound theoretical techniques to the measurement phases of a selection process?
3. *Economic*—Is it economically feasible to perform measurement experiments?

These aspects of the data collection and analysis problem will be discussed in turn.

A COMPUTER SELECTION MODEL

In order to discuss the feasibility of performing computer service comparison measurements, a model of the computer selection process is presented in Figure 1. The selection process is divided into three phases, leading progressively from the set of all computer services offered by those systems on a network to the isolation of the best one. Each phase involves evaluation of the services with respect to different classes of performance criteria. At each phase, those services which fail to satisfy the respective performance criteria are eliminated. The kinds of performance criteria that are evaluated are described briefly in this section. The measurement phases are discussed in detail in the following three sections.

Classification of performance criteria

In choosing the best service from several alternative network computer services, performance criteria which describe what is meant by best must be determined. These

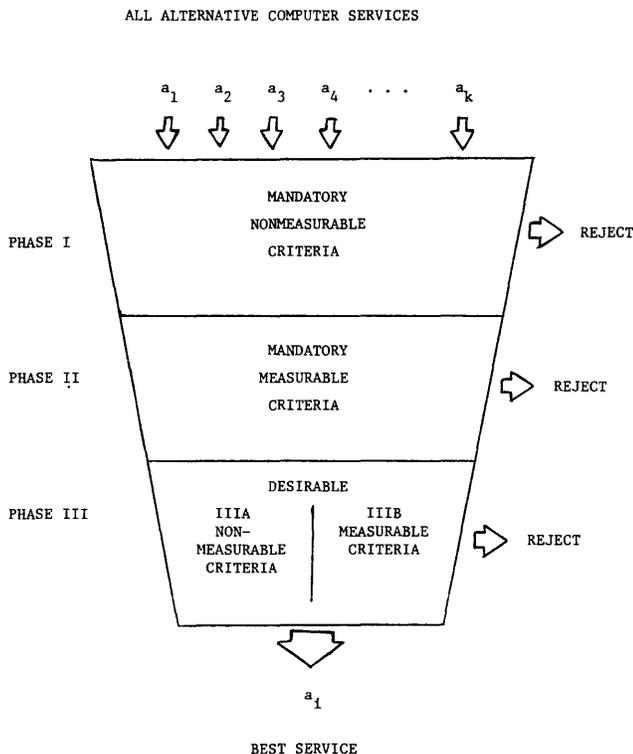


Figure 1—Computer selection model.³

criteria can be divided into those for which no empirical measurement is necessary and those whose values are derived from actual system measurements. For example, evaluating system documentation and amount of main memory does not require measurement collecting activity, whereas evaluating system turnaround time and response time for particular instructions clearly requires measurement.

Performance criteria can be divided further into those which are mandatory and those which are desirable.¹⁰ Mandatory criteria are defined to be any performance requirements which must be satisfied by the computer services being considered for selection. Desirable criteria, on the other hand, are those which are not absolute requirements for system selection, but which would make the implementation of the user's work easier. Therefore, if a given network computer service does not include some desirable features, it would continue to be considered for selection, but the lack of each desirable feature would invoke some penalty on the respective computer service.

Based on the two characteristics described above, performance criteria can be classified as: Mandatory Nonmeasurable (MN), Mandatory Measurable (MM), Desirable Nonmeasurable (DN), and Desirable Measurable (DM). Examples of each class of criteria are provided in Table I.

Application of performance criteria

Figure 1 illustrates a sequence in which the classes of performance criteria should be applied in the process of

choosing the best network computer service. This sequence is composed of three phases. Phase I involves the application of MN criteria. This phase is managed easily since each computer service either does or does not have each required feature. All of those network computer services which do not satisfy the MN criteria are eliminated.

Phase II involves the application of MM criteria. In general, for each MM criterion, performance measurements are gathered from each network computer service and a decision is made as to whether or not the criterion is satisfied. Failure to satisfy a single MM criterion results in a service's elimination.

Finally, determination of the best alternative is made in Phase III. This phase is separated into two parts, Phase IIIA for the application of DN criteria, and Phase IIIB for the application of DM criteria. (Note: It is not implied that DN and DM criteria necessarily can be applied separately or in any particular order.) Phase IIIA requires an analyst to determine whether an alternative does or does not provide desired features. Provision of a particular desirable feature results in an alternative being credited with the "value" of that feature. This is done for all alternatives and all DN criteria.

The information required in Phase IIIB is similar to the information required in Phase II in that it can only be obtained by system management. Data are collected from each alternative network computer service being evaluated and compared, and on the basis of relative performance, one service is selected as the best. In both of these phases, comparison requires collecting and analyzing relevant performance measurements for the various network computer service alternatives under consideration—in Phase II, to select those satisfying certain mandatory performance requirements, and in Phase IIIB, to select the best remaining one. It is specifically these two measurement phases that are the topic of further discussion.

DATA COLLECTION AND ANALYSIS— THEORETICAL ASPECTS

A collection of measurements can often give an illusion of fairness and objectivity to a network computer service

TABLE I—Examples of Performance Criteria

TYPE	EXAMPLE
Mandatory Nonmeasurable	1. The system must be fully delivered and operational no later than September 1, 1979. 2. Timesharing service must include FORTRAN, Basic, Lisp, SNOBOL and editing facilities.
Mandatory Measurable	1. The mean-time-to-failure for a specific one month period must be greater than 4 hours. 2. 95% of all trivial command response times must be less than 1 second.
Desirable Nonmeasurable	1. It is desirable that the system include Pascal and COBOL facilities. 2. It is desired that the system provide a text editing capability.
Desirable Measurable	1. It is desired that the system provide a mean turnaround time for the benchmark run of 5 minutes or less. 2. It is desired that 95% of all trivial command response times be 0.5 seconds or less.

selection when in reality the measurements contain little information of value. This can be the case when a selection is based on methods and techniques which are not statistically sound. Suppose, for example, that an analyst executes 30 script runs on a particular network computer service, intending to use the measurements which result as a fair and objective evaluation of that service. As was demonstrated in a full-scale case study discussed in Reference 12, if the measurements are not collected properly, it is possible that as few as two or three of 30 such measurements are statistically independent. Hence, 30 measurements may be quoted as the basis for an evaluation, but in reality the actual information content is equivalent to that provided by a much smaller set of measurements. Thus, only an illusion of fairness and objectivity exists.

The above example illustrates that ignoring correlation among consecutive measurements can create a false sense of objectivity. Another problem arises if test conditions are such that the measurement error is relatively large compared to the difference between the services being compared. If appropriate statistical methods are not used to analyze the data, then the chance of selecting the best service by comparing measurement data may be no better than the chance of selecting the best service by random drawing.¹⁶

The comparison of computer services provided by a network, therefore, requires a methodology designed to lead to the selection of the best service, and to provide control of the probability of having made a correct choice. Methodologies dictated by classical statistical designs (which are inappropriate for network computer service selection) lead to regression analyses of the data, employing either analysis of variance or curve-fitting techniques. The questions that can be answered using such methodologies are of the type, "Is the performance of several alternative services the same?" (i.e. "Are the distributions of the performance measurements identical from a statistical point of view?") or "How does one particular service performance parameter depend upon the other service parameters?" In most network computer service comparison efforts, however, these questions are not appropriate. The question of real interest is, "Which service is the best?" or "How do the services rank from best to worst?" It is for problems of this type that statistical ranking and selection procedures were developed.⁸ These procedures provide the theoretical foundation for valid measurement phases in a network computer service selection effort.

Statistical ranking and selection procedures are appropriate for three types of computer service comparison problems—ranking services by 1) comparing sample means, 2) comparing sample percentiles and 3) comparing sample proportions. In all three cases, the procedures specify the number of data points which must be collected from each service in a comparison study (or they specify a selection rule) in order to guarantee that the probability of a correct selection be at least a predetermined minimum value.

The use of a mean, percentile or proportion statistic for network computer service comparison is an analyst choice based on considerations about the comparison study's objectives, the statistical properties of the data, and the statis-

tical requirements of the selection methodology. Means often are used for comparisons when criteria such as script turnaround time or script cost are of interest because these performance measures tend to have approximately normal distributions. For comparison criteria such as response times for various types of interactive commands, which tend to have exponential-like distributions,^{7,15} the mean is not as informative a statistic. Percentiles or proportions are more appropriate.⁴⁻⁶

A detailed description of how the three classes of statistical ranking and selection procedures can be applied to computer comparison studies has been presented in several recently published reports.^{2,3,11,12} In general, the procedures provide information regarding the natural ordering of a set of k distributions, where each distribution is summarized by some (unknown) parameter like its mean or a specific percentile value. The procedures determine how the k parameters rank with respect to a prespecified standard (Phase II in Figure 1) or with respect to each other (Phase IIIB in Figure 1). The ranking is accomplished by collecting sample observations and computing an appropriate statistical estimate for each of the parameters. These estimates are then numerically ordered and, depending on the goal of a selection, inferences are drawn about the true population parameters (i.e., their true ranking). A variety of assumptions concerning the populations and data collection process can be made. These include assumptions regarding the underlying form of the distributions (e.g. normal, gamma) and the level of dependence between obtained observations.

A procedure for making a selection based on proportions is presented below as an example. Suppose a representative test workload (script) has been prepared to be run on several computer services available on a network, and it is desired to select those computer services which can run the script in 10 minutes or less at least 80% of the time. (Turnaround time is variable due to uncontrolled external influences such as other users of the service.) The procedure for making the selection is given in four steps. Assume the following definitions:

- k —Number of alternative computer systems in a study
- P^* —Desired level of confidence in the correctness of the selection results
- n —Number of measurements collected from each computer system
- $C(thld)$ —Criterion threshold value; i.e., mandatory value for a particular performance measure (10 minutes for the above criterion)
- $X(i)$ —Number of measurements from the i th computer system which are less than $C(thld)$
- $p(min)$ —Minimum proportion of measurements which must be less than $C(thld)$ (80% for the above criterion)

This procedure for selection assumes that measurements are independent and that measurements from the same computer service have a common probability of being less than $C(thld)$. The procedure makes no assumption about the underlying form of the distribution of the data.

Step 1—Choose appropriate P^* , $C(thld)$ and $p(min)$ values according to nonstatistical considerations.

Step 2—Collect n independent measurements from each of the k computer systems. The analyst chooses n based on nonstatistical considerations, bearing in mind that as n increases, more accurate estimates of each alternative's proportion will be attained.

Step 3—Let $X(i)$ =number of measurements from system i which are $<C(thld)$. Note that since n , the total number of measurements made, is identical for each computer system, the $X(i)$ values can be used to estimate a ranking of the true proportions.

Step 4—For each system, compare $X(i)$ to a constant M , which is derived from ranking and selection theory. If $X(i) \geq M$, then include the computer system in the selected subset; otherwise eliminate it. M is determined by table lookup based on the parameters k , n , P^* and $p(min)$. Extensive values for M are tabulated in Reference 3 and also appear in Reference 12.

The conclusion to be drawn from the development of statistical ranking and selection techniques which are appropriate for computer comparison is that *theoretically*, statistically sound techniques are available for using mean, percentile or proportion statistics in either Phase II (evaluation of mandatory measurable criteria), or Phase IIIB (evaluation of desirable measurable criteria) of the selection process. (It should be noted that ranking and selection theory does not meet fully the needs of computer comparison experiments. Some extensions to the theory have been made⁹ and several more are currently under investigation.) The next section discusses the technical feasibility of applying these techniques in actual network computer service comparison studies.

DATA COLLECTION AND ANALYSIS—TECHNICAL ASPECTS

A large-scale feasibility case study was conducted at the National Bureau of Standards to evaluate the time and cost required to apply statistical ranking and selection techniques in a computer service comparison study. Four large-scale heterogeneous remote-access time sharing systems were compared: a DEC System-10 running a TOPS-10 Monitor, a Honeywell 6180 running MULTICS, an IBM 360/65 running MVT/TSO and a UNIVAC 1108 running Exec 8. The major goal of the study was to determine if it was technically feasible, given modern measurement technology (such as automatic measurement machines and automatic interactive script drivers), to collect the amount of data generally required for analysis when using ranking and selection procedures. The specifications for the case study and the experimental results are presented here.

Scenario and scripts

A scenario for the case study, presented in Table II, was designed to be reasonably typical of the functional requirements of a real workload. It is not intended to be, nor is it, representative of any specific user's workload. The scenario

TABLE II—Scenario for the Case Study

1.	Logon to the system
2.	Execute a COBOL search of a bibliographic database
3.	Execute a FORTRAN version of a synthetic benchmark
4.	Copy an interactive FORTRAN program file to file INTR1
5.	Edit file INTR1 correcting a syntax error
6.	Edit file INTR1 correcting a logical error
7.	Compile file INTR1
8.	Execute file INTR1 which will interactively compute the prime numbers from 1-100
9.	Delete file INTR1
10.	Logoff the system

consists of commands representing two types of remote access interactive computing: transaction processing and interactive program development and execution.

The transaction processing was implemented in a COBOL program which executed a sequential search of a bibliographic database in order to retrieve a given set of entries. The database consists of 2400 records, each of which is 132 characters long. The transaction processing was accomplished by Item No. 2 in the scenario (see Table II). A synthetic module which was capable of being adjusted for varying amounts and types of CPU and I/O activity was executed next (Item No. 3 in the scenario). The last section of the scenario (Items No. 4-9) consists of commands to debug, compile and execute an interactive FORTRAN program which computes prime numbers.

The translation of the scenario into scripts executable on each computer service required interaction with personnel who possessed a thorough knowledge of each of the four respective operating systems. For each computer service two activities were required. First, it was necessary to establish permanent program and data files for use in each script execution. Then, instructions for the actual script execution were determined, sometimes including complicated control language constructs.

The specifications for running the scripts were typical of a real selection procedure. The user is concerned with the quality of a network computer service offered during a particular time period. If this quality is within acceptable limits, then service at all other times is assumed to be acceptable. In this case study, performance data was collected from each service only between 8:30 AM and 4:45 PM, Monday through Friday, in order to compare services under normal work day conditions.

Experimental data collection and analysis

Eight performance measures were chosen for computer service comparison in the case study. They are:

1. Cost.
2. Turnaround time for the entire script execution.
3. Response time for the bibliographic retrieval (No. 2 in the scenario).
4. Response time for the FORTRAN version of a synthetic benchmark (No. 3 in the scenario).

5. Response time for the copy command (No. 4 in the scenario).
6. Response time for the first edit command (No. 5 in the scenario).
7. Response time for the second edit command (No. 6 in the scenario).
8. Response time for the interactive calculation of all prime numbers less than 100 (No. 8 in the scenario).

The hardware/software configuration for the case study is illustrated in Figure 2. The scripts were executed on each system under the automatic control of a remote terminal emulator called the Network Access Machine.¹³ Turnaround time, response time and cost data were automatically collected for each execution of each script by a minicomputer called the Network Measurement Machine.¹⁴ (Many vendors distribute remote terminal emulators, and automatic measurement machines also are readily available on the market.) The data were analyzed using a statistical package called OMNITAB.⁹ Means, percentiles and proportions were calculated. Since all of the ranking and selection procedures used require independent measurements, correlation coefficients were also calculated. The method of ensuring independence for the experimental data is described in Reference 12.

Comparison results

A sufficient amount of independent data was collected from the computer services in the case study to allow for a reasonably high statistical confidence in the comparison results. The details of selecting services based on the various measurement criteria can be found in Reference 12. Based

on the full-scale case study, it was concluded that it is *technically* feasible to execute statistically sound measurement phases in a computer service comparison study. The question remains, "At what expense can the technology be applied?"

DATA COLLECTION AND ANALYSIS— ECONOMIC FEASIBILITY

The amount of money available for a comparison study depends on the economic resources and policies of those sponsoring the study. So the question of economic feasibility only can be answered in the context of a completely specified comparison experiment. However, cost estimates from the case study described above provide guidelines for addressing this question.

The total cost to collect measurement data and compare the four services in the case study was estimated to be \$16,600. Personnel and equipment costs are itemized according to different tasks in Table III. Each task was performed by one professional and one technical person working in cooperation on a full time basis. Personnel costs are estimated at \$200/day for each person. This represents the average "burdened" (all overhead included) cost to the federal government per day for each person.

In a typical procurement study, the equipment cost from developing and running the scripts would be absorbed by the vendors. Therefore, although these costs are noted in Table III, no figures are entered. It is assumed that the equipment costs for the data collection and analysis are part of a selection effort. These activities require a specialized hardware and software configuration for about six weeks, with an estimated cost of \$1,500.

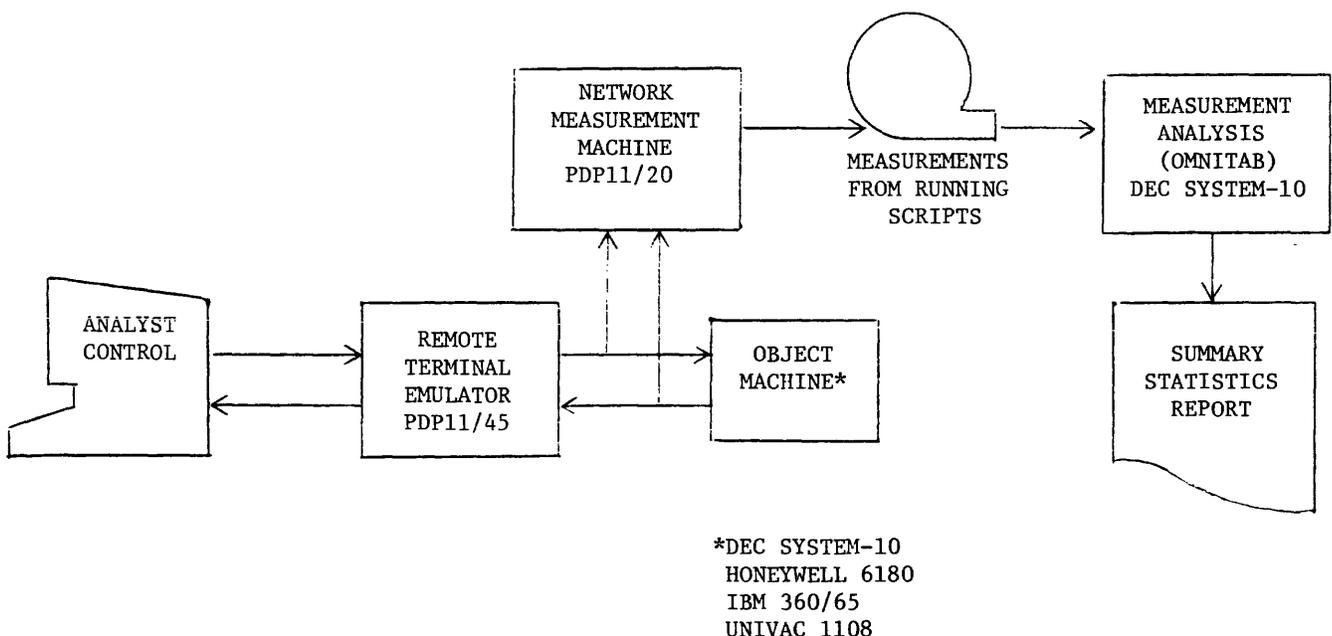


Figure 2—Hardware/software configuration for the case study.

TABLE III—Estimated Cost

A. PERSONNEL			
TASK	PERSON DAYS	COST (\$200/DAY)	
Scenario development	5	\$ 1,000	
Script generation (4 computer services)	12	2,400	
Control programs for remote terminal emulator	8	1,600	
Script runs (data collection)	30	6,000	
Data analysis	15	3,000	
Report generation	3	600	
Overhead (equipment failure, bad data, etc.)	2	400	
	75	\$15,000	
B. EQUIPMENT			
TASK	COST (\$)		
Script development (4 computer services)	*		
Script runs (150 runs/service)	*		
Data collection and analysis	\$1,600		
	\$1,600		

TOTAL COST = \$15,000 + \$1,600 = \$16,600

* Supplied by the vendor

Since care was taken to design and execute the case study in the same way that a real computer service comparison would be done, the total cost figure of \$16,600 covers the measurement phases of a full-scale computer service comparison study. This figure, therefore, can be used as a basis for a reasonable cost estimate when making a decision about using the selection methodology in alternative comparison environments. The impact on cost due to an increase in the number of computer services or due to a change in the number of selection criteria is discussed next.

If more than four computer services are to be compared, then various personnel and equipment costs will change, but in different proportions. Time for scenario development will remain the same since only one scenario is needed regardless of the number of services being compared. The time required for script generation and for writing control programs for remote terminal emulation will increase in direct proportion to the increase in the number of services being compared. This is because time is necessary to translate the scenario into a system compatible script for each service. The total elapsed time for data collection (script runs) may remain the same or increase slightly, inasmuch as the requirement for independent measurements forces interleaving of script runs on the different services anyway. As more time is required to collect measurements, more money must be allocated for data collection equipment. The total increase for both data analysis and report generation would be about two more person-days for each additional computer service.

A change in the number of selection criteria to be applied will not have much impact on the cost of the study. Most automatic measurement devices collect data about every interactive transaction, even if it is not all required. So, increasing or decreasing the number of criteria used in a

selection has little effect on the time and cost of data collection. Also, most statistical analysis packages are as easy to run for one variable as for ten, so changing the number of criteria will have only minor effect on the time and cost of data analysis.

Based on the case study, it can be concluded that subject to certain practical constraints, it is *economically* feasible to compare services.

CONCLUSION

The comparison of interactive computer services available through a network is a complicated process involving measurement and nonmeasurement phases. Measurement phases of a computer service comparison process can be validly executed if test workloads can be generated which are representative of real workloads, and if measurements can be made on the test workload using statistically sound techniques. This paper has discussed the theoretical, technical and economic feasibility of employing statistically sound data collection and analysis in a network computer service comparison. Based on the evidence presented and referenced here, it can be concluded that, given a representative test workload and sufficient funds to measure the performance of that workload as dictated by statistical ranking and selection procedures, accurate measurement phases can and should be included in a selection process.

REFERENCES

- Abrams, M. D., I. W. Cotton, S. W. Watkins, R. Rosenthal and D. E. Rippy, "The NBS Network Measurement System," *IEEE Transactions on Communications*, Vol. Com-25, No. 10, October 1977, pp. 1189-1198.
- Amer, P. D., and S. A. Mamrak, "Statistical Methods in Computer Performance Evaluation: A Binomial Approach to the Comparison Problem," *Proceedings Computer Science and Statistics: Eleventh Annual Symposium on the Interface*, Raleigh, NC, March 1978, pp. 314-319.
- Amer, P. D., "Experimental Design for Computer Comparison and Selection," Ph.D. Dissertation, Department of Computer and Information Science, The Ohio State University, March 1979.
- Anderson, M. A., Jr., and R. G. Sargent, "A Statistical Evaluation of the Scheduler of an Experimental Interactive Computing System," *Statistical Computer Performance Evaluation*, W. Freiberg (ed.), Academic Press, New York, 1972.
- Brown, R. S., "A Case Study of a Multi-Access Scientific Benchmarking Application," *Benchmarking*, N. Benwall (ed.), Hemisphere Publishing Co., Washington, DC, 1975, pp. 133-147.
- Guidelines for the Measurement of Interactive Computer Service Turn-around Time and Response Time*, FIPS Pub 57, National Bureau of Standards, August 1978.
- Fuchs, E., and P. E. Jackson, "Estimates of Random Variables for Certain Computer Communications Traffic Models," *Communications of the ACM*, Vol. 13, No. 12, December 1970, pp. 752-757.
- Gibbons, J. D., I. Olkin and M. Sobel, *Selecting and Ordering Populations: A New Statistical Methodology*, John Wiley and Sons, Inc., New York, 1977.
- Hogben, D., S. T. Peavy, and R. N. Varner, *OMNITAB II User's Reference Manual*, NBS Technical Note 552, National Bureau of Standards, October 1971.
- Joslin, E. O., *Computer Selection: Augmented Edition*, The Technology Press, Inc., Fairfax Station VA, 1977.
- Mamrak, S. A., and P. A. DeRuyter, "Statistical Methods for Comparison of Computer Services," *Computer*, Vol. 10, No. 11, November 1977, pp. 32-39.

-
12. Mamrak, S. A., and P. D. Amer, *A Methodology for the Selection of Interactive Computer Services*, NBS Special Publication 500-44, National Bureau of Standards, January, 1979.
 13. Rosenthal, R., "Accessing On-Line Network Resources with a Network Access Machine," *IEEE Intercon 1975*, pp. 25/3:1-4.
 14. Rosenthal, R., D. E. Rippy and H. Wood, *The Network Measurement Machine—A Data Collection Device for Measuring the Performance and Utilization of Computer Networks*, NBS Technical Note 912, National Bureau of Standards, April 1976.
 15. Totšchek, R. A., "An Empirical Investigation into the Behavior of the SDC Time-Sharing System," System Development Corporation, Report SP-2191, AD622003, Santa Monica, CA, 1965.
 16. Youden, W. J., "How to Pick a Winner," *Statistical Design and Industrial Engineering*, Reprint, December 1959, pp. 81-82.

Characterizing a workload for the comparison of interactive services

by DOMENICO FERRARI

*University of California
Berkeley, California*

INTRODUCTION

Characterizing the workload is a necessary part of any system performance evaluation effort. This necessity is a consequence of the sensitivity of the performance indices of interest, be they productivity or responsiveness indices, to the nature, composition and intensity of the workload processed by the system being evaluated. No matter how we define it, performance cannot be expressed by quantities independent of the system's workload.¹

Similarly, the performance of two or more systems cannot be meaningfully compared if these systems do not process the same workload. This statement applies not only to procurement problems, but also to problems of improvement and design, that is, whenever a performance comparison is to be made between two systems which differ from each other in some respect (see Figure 1). Since the purpose of the comparison is to expose the relative merits of the two systems, configurations, versions, architectures, or solutions through the differences between the values of some suitable performance indices, the workloads they process must be the same if their influences on these values are to be identical. Otherwise, the differences between the indices will reflect those between the workloads as well as those between the systems, and it will be difficult or impossible to distinguish the respective contributions.

The evaluation of a system's performance, or the comparison of the performances of several systems, should ideally be made under the workload that the system, or any of the competing systems, will have to process from the present to the end of its lifetime. This is, however, impossible, at least for most installations, since the workload is time-variant and its future evolution unknown. The only exception is that of those installations whose workload is constant in composition and intensity, and perfectly known (e.g., certain industrial control and manufacturing computers).

Moreover, even if the whole future workload were known (a very unrealistic assumption in the great majority of the cases), measuring or computing the performance of the system under its total workload would be impossible or impractical. A much more compact workload is therefore to be used in evaluations or comparisons. Since performance indices depend on the workload's nature and composition,

this more compact input to the system must be a model of the total workload it replaces;^{1,2} in other words, it should accurately represent those properties of the workload which are of interest in the context where the model is to be used. And this is where the difficulties begin.

THREE DEFINITIONS OF REPRESENTATIVENESS

In the technical literature, the term "representativeness" has been traditionally employed when referring to workloads (especially artificial ones, such as a set of benchmarks) used in performance evaluation studies. If we agree that, independent of their type, these workloads are in fact workload models, we should use the term "accuracy" instead. Perhaps, representativeness is usually preferred because of its vagueness, which very accurately reflects the lack of an exact definition of the underlying concept.

When is a workload model "representative" (or "an accurate representation") of a given workload? Probably, the most popular answer to this question is: When the model consumes the same physical resources at the same rates as the workload it tries to represent. This resource-oriented definition is unsatisfactory since

- a. It is heavily machine-dependent.
- b. The choice of the resources whose consumptions are to be considered and of the rate definitions influences the performance of the system (or systems) in a usually unknown way. For example, the set of the resources included in the characterization may be incomplete, or redundant; and, even if they are both complete and non-redundant, we generally do not know what consumption rates (overall means, moving averages, interval means, instantaneous means, or others) would be needed, or sufficient, to determine the performance indices of interest.

Drawback a is irrelevant in studies involving a system which is not supposed to be modified, for example when the capacity of an existing system is to be determined.³ However, it becomes increasingly important as we move from these problems to improvement to procurement. Drawback b is unfortunately present in all contexts.

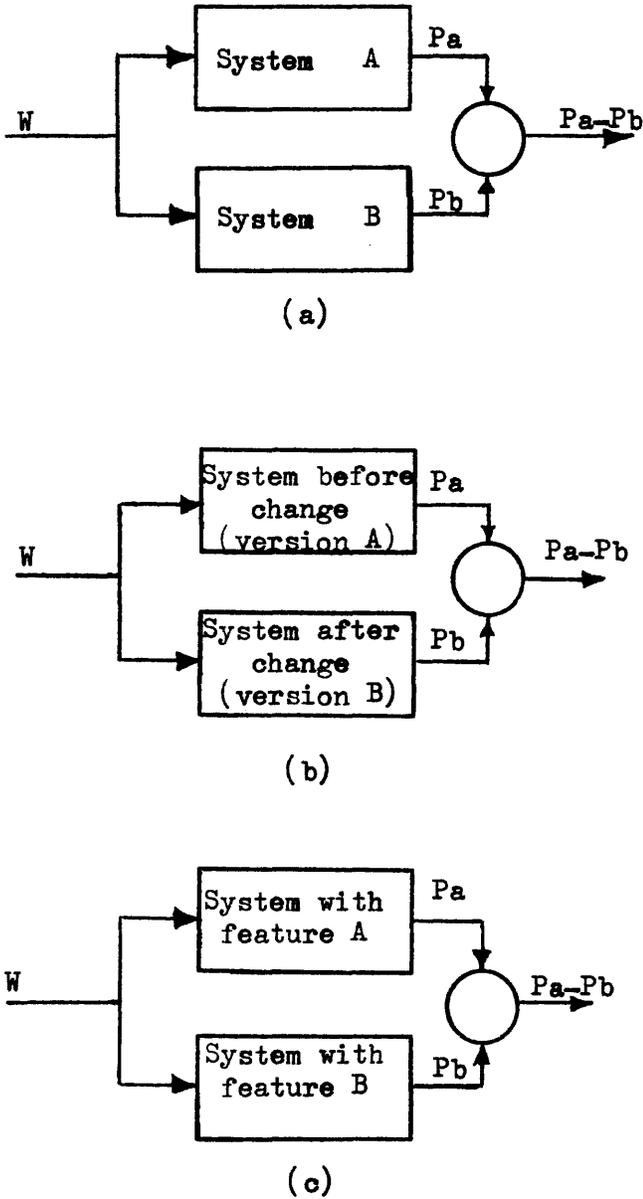


Figure 1—Conceptual schemes of performance comparison in (a) procurement, (b) improvement and (c) design environments. For the differences $P_a - P_b$ between performance indices to reflect the relative merits of the systems being compared, these systems must process the same workload W , which is to be an accurate model of the actual workload.

An alternative definition is the one based on functional considerations: Workload W_b models workload W_a if it performs the same functions as W_a . When the workload to be modeled performs relatively few well-known tasks which are repetitively executed on large volumes of data (e.g., payroll computation, sorting, accounting, compilation, and so on), such a functional definition is in principle applicable to the case we are interested in, the one where workload W_b is to be much more compact—that is, to execute in much less time—than workload W_a . In this case, W_b may be obtained from W_a by suitably reducing the volume of

data to be processed by each of W_a 's functions; however, neither the relative proportions of the various tasks in W_b nor the criteria for choosing the input data for them can be determined by applying this workload model definition. Otherwise, the one above can only be used as a functional definition of the equivalence¹ between W_a and W_b , without implying or requiring that W_b be a sufficiently compact model of W_a or vice versa. An important application of functional equivalence will be illustrated.

A third answer to the question we asked at the beginning of this section may be dictated by the intended usage of the workload models to which the question refers. Since these models are to be applied in performance evaluation studies, what matters is the values of the relevant performance indices. Workload W_b is an accurate model of workload W_a for system S and performance index P if the values of P produced by W_b when running on S equal (with some error, which may be taken as a measure of the model's accuracy) those produced by W_a running on the same system² (see Figure 2). Note that P may be a set of global performance indices or of functions of such indices.

This, like the previous ones, can be seen as a definition of workload equivalence; however, unlike the functional definition, it can be applied without restrictions (except in

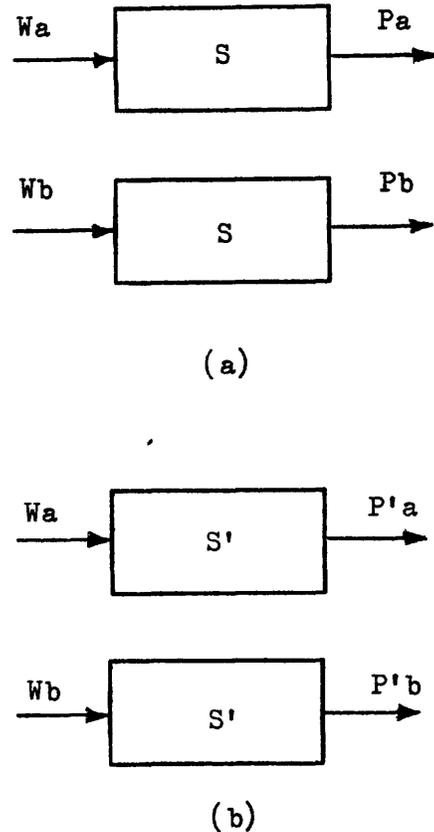


Figure 2—Performance-oriented definition of equivalence between two workloads W_a and W_b . W_b can be said to be a representative model of W_a with respect to S and P if $P_a \cong P_b$ (a). If $P'_a \cong P'_b$, then W_b is a valid model of W_a even with respect to system S' (b).

the unlikely case of improperly chosen indices) to the equivalence between a workload and a compact model of it. A drawback the performance-oriented definition shares with the resource-oriented one is system dependence, but in this case the problem is one of model validity.¹ A model calibrated by comparing its output with that of the modeled system under certain conditions is then used under a variety of different conditions. Can it be expected to be acceptably accurate even under these conditions? In the case of a workload model, the conditions include the system *S* with respect to which the model is equivalent to the modeled workload. Let us assume that both the workload and the model can be transported, possibly by converting them, to system *S'*, which is a different version or configuration of *S*, or a totally different system. Are they still equivalent when *S* is replaced by *S'*? A general answer cannot, of course, be given. In fact, very little, if anything, is known in this area. We conjecture that some of the methods used to increase the validity of other models can be applied also to workload models. For instance, use of real instead of synthetic components and functional similarity may enhance this model's validity. Also, calibration based on performance indices more detailed than those we are interested in often widens a model's domain of validity.¹ If, for example, the performance indices in *P* are the mean and the variance of response times, the workload model could be calibrated taking the distribution of response times as the calibration criterion.

A definite advantage of the performance-oriented definition over the functional one is the guidance it usually provides in the design of a suitable workload model, as shall be seen below. What makes the performance-oriented definition appealing is its operational verifiability coupled with its philosophical soundness. Unfortunately, in a procurement context, the question of model validity discussed above may make the functional definition more attractive, or even, when the new system or service to be procured does not have a predecessor whose performance can be taken as a reference, the only applicable approach. In general we feel that, whenever feasible, a performance-oriented solution should be preferred, but the workload model should be made as functionally similar to the modeled workload as possible in order to widen the model's domain of validity.

THREE LEVELS OF WORKLOAD MODELING

A workload and each one of its components (jobs, job steps, tasks, processes, interactions, transactions, and so on) may be modeled:¹

1. At the physical level, i.e., by the consumptions and/or consumption rates of the resources (CPU instructions or time, main memory space, I/O channel time, disk transfers or time, disk space, and so on) of the physical machine or machines on which it is processed.
2. At the virtual level, i.e., in terms of the amounts and/or rates of the resources (higher-level language statements, virtual memory space, file accesses, data base

accesses and so on) of the virtual machine or machines on which it executes.

3. At the functional level, i.e., at the level of the applications tasks the workload has to perform.

When going from Level 1 to Level 3, the dependence of the characterization on the system decreases, while its proximity to the viewpoint of the end user increases. Thus, the end user's perception of what a workload is roughly corresponds to its characterization at the functional level, but the programmer's (or interactive user's) viewpoint is better represented by a virtual-level model. Actually, a programmer's task just consists of translating functional descriptions of workload components into programs which utilize the resources of a given virtual machine. A Level-1 equivalent of such a program is then obtained when its virtual resource demands are automatically translated into physical resource demands by language processors, the operating system, and the hardware of the physical machine. Thus, to be processed, an executable workload model must always become a physical-level model, independently of the level at which it was designed. If a model is based on a functional characterization, it has to be translated into its virtual-level equivalent, which will in turn be transformed by the system's software and hardware into a physical-level equivalent.

Our knowledge about how the variables to be used to characterize a workload can be defined and measured decreases from Level 1 to Level 3. Thus, characterization is easier at a lower level than at a higher level, and models should generally be designed at the lowest acceptable level. Different types of evaluation studies require different minimum levels of characterization. To evaluate a system's performance for such purposes as performance testing or determining the residual capacity, a Level-1 workload model is usually sufficient. In the context of an improvement study, a Level-1 model may be dependent on system aspects which could be modified as a final or partial result of the study. If this is the case, a Level-2 model will be necessary, since the comparison between the modified and the unmodified system is to be performed under the same workload, a condition hard to verify if the modifications affect the characterization on which the model is based. A Level-3 characterization will not be needed if, as is usually the case, the virtual machine will not be modified by the study. On the other hand, several different virtual machines are normally to be compared in a procurement study, which will therefore require a workload model based on a functional (Level 3) characterization.

Clearly, there are relationships between the levels at which a workload may be characterized and the definitions of workload model representativeness discussed in the previous section. The verification of functional representativeness requires a Level-3 characterization for the modeled workload and for its model. Similarly, with physical-level characterizations a resource-oriented definition of representativeness only can be used. Both the resource-oriented and the performance-oriented approaches are applicable at

any characterization level. Their common requirement is the existence of a machine on which both the workload to be modeled and its model can (at least conceptually) be run.

WORKLOAD MODELING FOR THE COMPARISON OF INTERACTIVE SERVICES

The general discussion in the previous sections will now be applied to the problem of procuring interactive services. We assume that a number of suppliers of computer services (single installations and/or networks) could be capable of processing a given interactive workload. The choice among them is usually made according to criteria influenced by a variety of factors. An important class of factors is that of performance indices, which in the present context are usually concerned with the responsiveness of the service to user stimuli.⁴ One major difference between the equipment procurement problem and the service procurement problem is that system-oriented indices such as productivity and utilization, normally considered among the important ones in the former, are of no consequence in the latter.

Another difference, having the same origin, is that the workload the evaluators may analyze and, to some extent, control is only a fraction of the workload to be processed by the installations or by the networks being compared. Actually, in the procurement of computer services, we do not compare systems, but loaded systems. The fluctuations of the loads existing on the competing systems or networks make the statistical requirements of empirical comparisons, which are to be performed in an essentially uncontrolled environment, particularly demanding.⁵

As in all procurement contexts, the workload (which in this case is only part of the total load) must be characterized at the functional level. System independence (we should actually say "service-supplier independence"), which is necessary for the fairness of the competition, is normally achieved by preparing a scenario,⁶ that is, a description of the functions to be performed by the model. These functions are described in a natural language and in user-oriented terms, since a formal workload description language, perhaps in the same vein as problem-statement or requirement-specification languages,⁷ has not been developed yet. The system-independent scenario, which functionally characterizes the workload model, is then manually translated into a number of scripts, so that one or more scripts are obtained for each service to be compared. A script is a sequence of interactive commands to be input, containing information about the speeds at which they have to be typed in, and of think times between the output and the input of consecutive commands. Included in a script are also the programs whose execution is to be caused by its commands as well as the files and data bases to be accessed. Thus, the set of scripts (often consisting of only one) which are derived from the scenario for each service are, or ought to be, equivalent at the virtual level to the workload model represented by the scenario. This virtual model will then be translated into an equivalent physical model by the command interpreter, the operating system and the hardware when a remote terminal

emulator⁸ (or some human users) inputs the scripts into the system (see the solid-line part of Figure 3).

Even though in practice much effort may be required to make sure that the scenario is translated into the scripts without unduly favoring or penalizing certain services with respect to others, and not always the results of this effort are successful, the procedure just described is philosophically unobjectionable. However, assuming that the workload is known, is the scenario a representative model of it? The answer depends on the definition of representativeness which is adopted. Most workload model designers implicitly or explicitly choose the functional definition and try to convince themselves as well as others that their scenario, while not doing exactly what the original workload does (how could it?), is indeed functionally representative. Sessions and sub-sessions are categorized according to their functions (text editing, program debugging, file manipulation, data base accessing, and so on), the frequencies of the various categories are measured, and a scenario representing a scaled-down picture of this functional characterization is somehow constructed.^{9,10} No effort, however, can successfully conceal the large amount of subjectivity present in each decision. The lack of quantitative criteria and systematic methods makes it often necessary to resort to arbitrary choices. The situation is so hopeless that some evaluators find it easier to argue that representativeness is not important than to demonstrate that their scenarios are reasonably accurate models of a given workload. Can any improvements be made to this situation? If so, are they economically and technically feasible?

PRODUCING A MORE REPRESENTATIVE SCENARIO

Since the functional definition of representativeness is not easily applicable to compact workload models and does not suggest any systematic approach to model design, another definition not having these drawbacks should be considered. The performance-oriented definition is certainly an eligible candidate, but there is a major difficulty with it: What sort of performance should we use to determine whether a model is representative or not? If measurement of this performance is to be at least conceptually feasible (which is very desirable), we should choose to consider the responsiveness of a system on which the workload to be modeled can run. Assuming that such a system, to be called system Z, exists (for example, it might be the one on which the interactive applications constituting the workload were developed), there are at least two ways to proceed:

1. A scenario is designed as briefly described in the previous section: then, a set of scripts for system Z are derived from it (again as described above), executed on Z, and modified until they produce approximately the same performance as the entire workload to be modeled when running on Z: finally, either the scenario or the procedure to derive from it sets of scripts for

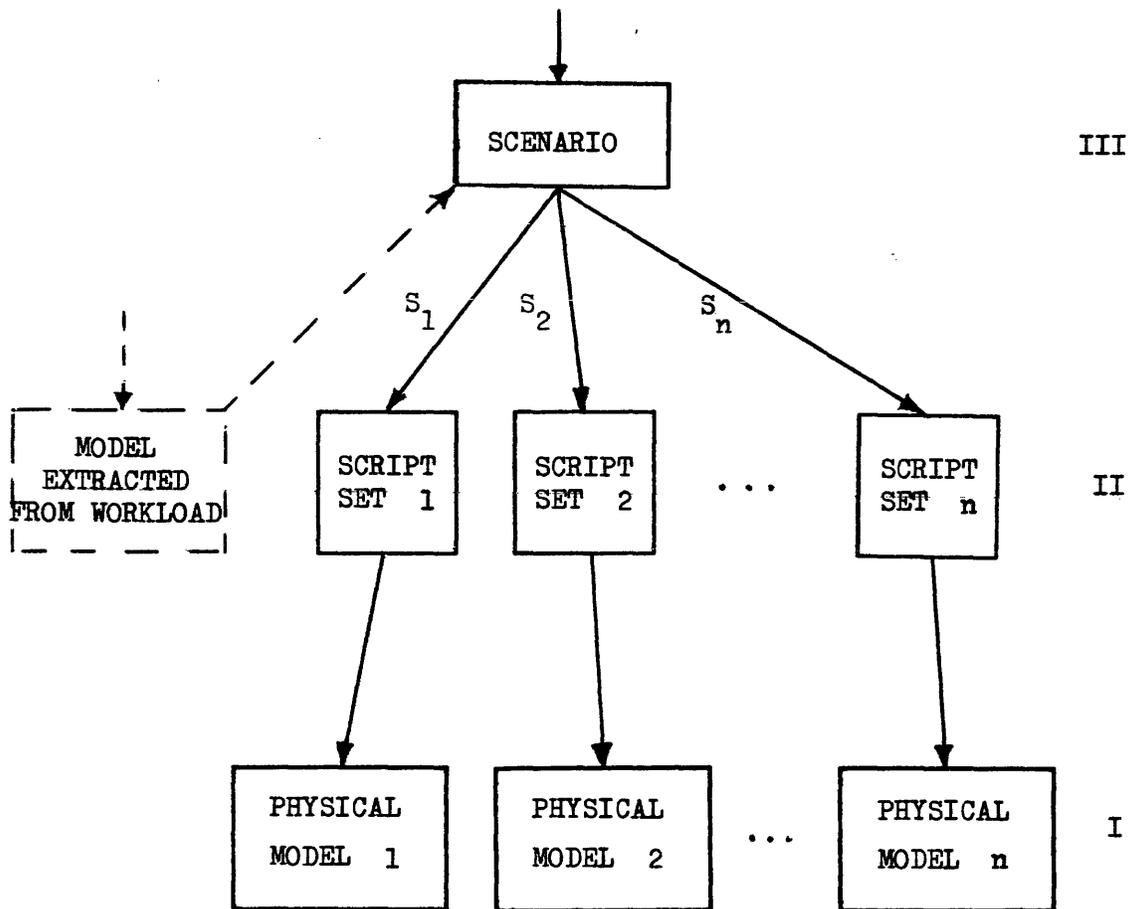


Figure 3—The tree of executable workload models for a procurement study of interactive systems or services S_1, S_2, \dots, S_n . Script sets often consist of only one script, which performs all the functions listed in the scenario.

other systems, or both, should be modified to reflect these changes.

2. A systematic method for extracting a representative scenario from a virtual-level characterization of the original workload is developed.

Approach 1 is likely to be quite cumbersome in practice, also because it probably involves a large number of measurement experiments. In principle, a method like 2, if one can be found, is certainly more attractive. The question is, therefore, whether a method for transforming an interactive workload into a much more compact one without appreciably affecting the responsiveness of a given system (Z) can be found. Note that both workloads are assumed for convenience to be processed alone by system Z . If such a transformation is possible, the workload model should then be translated into an equivalent scenario (see the dashed part of Figure 3), an operation which is not conceptually complex but may be less than straightforward in practice when the peculiarities of system Z 's command language, operating system, and organization are to be dealt with. The scenario, which should be expected to be much more detailed than usual and less explicitly reflecting the main func-

tions of the original workload, is finally translated into sets of scripts for all the competing services. Another fundamental question is whether a model representative with respect to a dedicated system (Z) continues to be such when running on another, non-dedicated system. This is the problem of model validity discussed in the second section. For one of the services being compared, that which is selected, it is possible to verify a posteriori whether the model's representativeness was affected by this drastic change of environment. For the others, not even this late verification is feasible. Thus, only the validity enhancement techniques mentioned in the second section can be applied, without any guarantee of success.

A number of obstacles must be overcome before an interactive-workload compaction method which leaves responsiveness unaffected can be obtained. Responsiveness indices are generally very sensitive to the command mix (types and rates) and to the order in which commands are input and processed. Thus, characterizing an interactive workload by the frequency distributions of command types and think times is not sufficient. Both the correlation among the commands issued by each user and the temporal relationships among the various command streams cannot gen-

erally be assumed to have a negligible influence on responsiveness. In order to preserve both to the largest possible extent, we may subdivide the workload to be modeled into a number of time intervals. Each time interval should be so long that the responsiveness index or indices chosen for defining representativeness can be meaningfully computed. If, for instance, the workload model's calibration criterion is based on the distribution of response times, enough interactions should be contained in each interval that

- The frequency distribution of their response times is statistically significant.
- The distortions due to the interactions in progress at the beginning or at the end of each interval have negligible effects on the corresponding distribution.
- The correlations between the indices (e.g., between the moments of the response time distributions) of consecutive intervals can be ignored.

Note that a similar subdivision is the basis of the "method of subsamples" used in simulation to compute the statistical accuracy of a given run's results or, alternatively, the minimum duration of a run which will produce results with a given accuracy.^{1,11} Assuming that all of the above conditions are satisfied, a subdivision yields a large number of sub-workloads, each having an associated performance index or set of indices. For example, this set of indices may consist of the relative frequencies of response times within the k classes defined by the values $t_0, t_1, \dots, t_k(t_{i-1} < t_i, i=1, \dots, k; t_0=0, t_k=+\infty)$:

$$F_{ji} = n_{ji}/N_j \quad (i=1, \dots, k), \quad (1)$$

where n_{ji} is the number of interactions with response time t such that $t_{i-1} < t < t_i$, and N_j is the total number of interactions in the j th sub-workload. Note that the value of one of the k indices F_{ji} is functionally dependent on the values of the other $k-1$ indices.

With all responsiveness indices of practical interest, the value of index P for the whole workload is given by

$$P = \frac{1}{N} \sum_{j=1}^M N_j P_j, \quad (2)$$

where P_j is the contribution of the j th sub-workload to P , M is the total number of sub-workloads, and N is the total number of interactions in the workload. For most indices, P_j coincides with the value of P for the j th sub-workload, but sometimes this is not the case. For instance, when P is the variance of response times, contribution P_j is the sum of this variance for the j th sub-workload and the square of the difference between the mean response time of the j th sub-workload and the overall mean:

$$P_j = \text{Var } j + (E t_j - E t)^2 \quad (3)$$

Each sub-workload is thus characterized by its contributions to the values of the overall performance indices, e.g., by the values of the F_{jis} , or of $E t_j$, or of $E t_j$ and $\text{Var } j + (E t_j - E t)^2$. This characterization may now be used to classify sub-workloads, that is, to cluster them into classes defined by values of the contributions relatively

close to each other. If more than one index is being considered, clustering methods^{3,12} or joint-probability scaling techniques¹³ will have to be applied. The objective of this operation is to reduce the original workload to a much more compact executable model by picking a relatively small number of representatives from each class. Note that, with joint-probability scaling, each class is composed of the sub-workloads contained within one of the regions into which the hyper-space of the characterizing variables has been divided.

Once the classes have been defined, how many representative sub-workloads should be selected from each class, and how should they be selected? Here again, the performance-oriented approach helps us find a satisfactory answer. Let C be the number of classes being considered, and let the h th class contain M_h sub-workloads ($h=1, \dots, C$). The contribution $P'h$ of each class to P can be defined in the same way as the contribution of each sub-workload, and Equation 2 will become

$$P = \frac{1}{N} \sum_{h=1}^C N'h \cdot P'h, \quad (4)$$

where $N'h$ is the total number of interactions in the h th class. In turn, $P'h$ can be expressed in terms of the contributions of the sub-workloads in the h th class as follows:

$$P'h = \frac{1}{N'h} \sum_{j=1}^{M_h} N_h j \cdot P_h j, \quad (5)$$

where $N_h j$ is the number of interactions in the j th sub-workload of the h th class, and $P_h j$ is its contribution to $P'h$. Note that, if P is the variance of response times, $P_h j$ will be given by (3) where $E t$ is to be replaced by $E t'h$, the mean response time of the class. Thus, (4) and (5) yield

$$P = \frac{1}{N} \sum_{h=1}^C \sum_{j=1}^{M_h} N_h j \cdot P_h j. \quad (6)$$

If the total number of sub-workloads in the model is to be m (m may be viewed as a design requirement), then the h th class will have to be represented by m_h sub-workloads such that

$$\sum_{h=1}^C m_h = m. \quad (7)$$

Assuming, with no loss of generality, that the first m_h sub-workloads are selected to represent each class, the model will produce a performance index

$$\begin{aligned} p &= \frac{1}{n} \sum_{h=1}^C \sum_{j=1}^{m_h} n_h j \cdot p_h j \\ &= \frac{1}{n} \sum_{h=1}^C n'h \cdot p'h, \end{aligned} \quad (8)$$

where $n_h j$ is the number of interactions in the j th sub-workload included in the model from the h th class, $p_h j$ is this sub-workload's contribution to the performance of the h th class in the model.

$$n = \sum_{h=1}^C \sum_{j=1}^{m_h} n_h j = \sum_{h=1}^C n'h \quad (9)$$

is the total number of interactions in the model, and $p'h$ is the contribution of the h th class to the model's performance.

The performance-oriented definition of representativeness states that the model is representative if

$$p=P \quad (10)$$

Sufficient conditions for satisfying (10) are, from (6) and (8),

$$(1/n) \sum_{(j=1, mh)} nh_j ph_j = (1/N) \sum_{(j=1, Mh)} Nh_j Ph_j, \quad (11)$$

or, from (4) and (8),

$$n'h \cdot p'h/n = N'h \cdot P'h/N, \quad (12)$$

for all h ($h=1, \dots, C$). Conditions (12) are satisfied if

$$p'h = P'h, \quad (h=1, \dots, C), \quad (13)$$

and

$$n'h/n = N'h/N, \quad (h=1, \dots, C). \quad (14)$$

Equations 13 are easy to satisfy, since the sub-workloads in each class have by construction very similar performances. Thus, it should not be hard to select mh out of Mh sub-workloads having approximately the same values of Ph_j so that the resulting $p'h$ is about equal to $P'h$. Equations 14 say that the fraction of interactions from the h th class in the model should be equal to the same fraction in the original workload. In other words, classes should be proportionally represented in terms of their number of interactions, no matter how "heavy" or "light" these interactions are. If all sub-workloads contained the same number of interactions, Equations 14 could be written

$$mh/m = Mh/M, \quad (h=1, \dots, C), \quad (15)$$

and mh would be proportional to Mh with a coefficient equal to the inverse of the scaling or reduction factor $r = M/m$. Otherwise, determining the mhs will involve some iterations, since n in (14) is not given, but is to be computed as the sum of the $n'h$. The problem may be approached by computing the approximate mhs from (15), selecting mh sub-workloads from each class, determining the values of $n'h$ and of n , and verifying whether equations (14), or rather (12), are satisfied. If not, the sub-workloads selected to represent each class should be replaced by others, and possibly their number should be changed, until Conditions (12) are satisfied within acceptable error bounds.

Once the sub-workloads to be included in the model have been selected, their ordering and their implementation as executable scripts are to be determined. If the assumption of negligible correlation between the performances of consecutive sub-workloads is satisfied, the relative order of samples from the original workload should be immaterial. Thus, this order can be dictated by various objectives, for instance by that of facilitating transitions from one sub-workload to the next in the executable model. Note that some of the statistical techniques which are applied in simulation can be used to verify the assumption mentioned

previously, as well as the significance of the sample taken from the original workload to construct the model.¹¹

Transitions between consecutive sub-workloads will have to be smoothed so that a continuous script to be input from each terminal during the execution of the model can be obtained. Partial interactions, those which had already begun when the sub-workload started or were not completed at the end of it, will be completed, or eliminated, or modified, in order to minimize the distortions due to a transition. In most transitions, a major amount of turbulence is to be expected, since a number of terminals will have to switch from one subsystem (e.g., an editor) to another (e.g., a compiler). The impact of this turbulence on the accuracy of the model can probably be reduced by suitably modifying the sub-workload involved. The magnitude of such an impact cannot be predicted without experimentation. However, it seems desirable, though perhaps not very easily feasible, that the phase relationships among the various command streams be roughly preserved within each sub-workload during execution, for instance by making proper use of the transition periods and possibly by acting on think times, so that command arrival rates are not appreciably influenced by the responsiveness of the service. Note that this will probably require remote terminal emulators considerably more intelligent than usual.

We have implicitly assumed that the executable model will consist of a number of command streams to be input by a trace-driven remote terminal emulator. An alternative approach is the one based on synthetic sub-workloads. A synthetic sub-workload is a set of synthetic scripts whose performance can be changed by properly selecting the values of certain parameters. Both a synthetic sub-workload at the virtual level and its functional equivalent would generally be less realistic than their natural counterparts. However, their use could substantially facilitate workload model implementation. This subject, which has not been studied yet, is certainly among those which deserve a great deal of attention.

A third approach to workload model construction, probably the most attractive one, is directly suggested by the model design method described above. Since all the sub-workloads in a class are performance-wise roughly equivalent to each other, the model could be reduced to only one representative (possibly synthetic) sub-workload from each class, and performance could be computed as a weighted sum of the performances of these isolated sub-workloads, the weights being proportional to the numbers $n'h$ of interactions in each class (see Equations 4, 8 and 14). A major difficulty with this method, which would on the other hand be very convenient since it would yield much more compact models and eliminate transition problems, is that of the bias introduced in the values of performance indices by the initial transients.^{1,11} This bias is likely to affect all measured performances and not only that of the first sub-workload as in the two previous approaches. Application of the methods used in simulation runs¹¹ may reduce to acceptably low levels the inaccuracies due to the initial bias. This is another one of the many aspects of the workload model design procedure proposed here which must be experimented with.

SUMMARY AND CONCLUSIONS

The road to more satisfactory and defensible workload characterizations is, especially in procurement contexts, a long and difficult one. A preliminary investigation into the feasibility of applying a performance-oriented approach in the procurement of interactive services has been described. After arguing that the accuracy (or, as it is usually called, the representativeness) of a workload model is essential in all evaluation contexts, three definitions of representativeness have been discussed. The three levels at which a workload may be characterized, and their relationships to the different types of evaluation studies as well as to the definitions of representativeness, have then been presented. Next, these general concepts have been applied to the comparison of interactive services for procurement purposes. A method for designing a representative workload model in such a context has been proposed. The method is based on the performance-oriented definition of representativeness and on the assumption that a model designed and calibrated at the virtual level of characterization for a given system is valid also when transported to the functional level and on other systems or networks offering interactive services. While this assumption, like many other aspects of the method, are still to be experimentally verified, the philosophies and procedures proposed here can probably be applied in a more straightforward, less expensive way to simpler environments and types of workload. The method described requires extensive measurements, the recording of long command streams, sophisticated remote terminal emulators, and is likely to produce substantially less compact models than desirable. However, it must be recognized that the problem is a very complex one, and that the current solutions to it, far from being satisfactory, are only made acceptable by the lack of any better approach. Furthermore, some ways of rendering the method and the model more feasible have been suggested. If the model can be reduced to one copy of a synthetic sub-workload per class, only the selected performance indices produced by the original work-

load have to be measured for each one of its sub-workloads, no tracing of command streams is necessary, neither transition problems nor complex phase-preservation issues arise, and maximum compactness is achieved. In spite of the number of important "ifs" still pending, the approach thus appears to hold some promise of practical applicability.

REFERENCES

1. Ferrari, D., *Computer Systems Performance Evaluation*, Prentice-Hall, 1978.
2. Ferrari, D., "Workload Characterization and Selection in Computer Performance Measurement," *Computer*, 5, 4, July-August 1972, pp. 18-24.
3. Artis, H. P., "Capacity Planning for MVS Computer Systems," in *Performance of Computer Installations—Evaluation and Management*, D. Ferrari (ed.), North-Holland, 1978, pp. 25-53.
4. Abrams, M. D., and Treu S., "A Methodology for Interactive Computer Service Measurement," *Comm. ACM*, Vol. 20, No. 12, December 1977, pp. 936-944.
5. Mamrak, S. A., and P. A. DeRuyter, "Statistical Methods for Comparison of Computer Services," *Computer*, Vol. 10, No. 11, November 1977, pp. 32-39.
6. Watkins, S. W., and M. D. Abrams, "Remote Terminal Emulation in the Procurement of Teleprocessing Systems," *AFIPS Conf. Proc.*, Vol. 46, 1977, NCC, pp. 723-727.
7. Teichroew, D., "Problem Statement Languages in MIS," in *System Analysis Techniques*, J. D. Cougar and R. Knapp (eds.), Wiley, 1974, pp. 310-327.
8. Wyrick, T. F., "Benchmarking Distributed Systems: Objectives and Techniques," in *Performance of Computer Installations—Evaluation and Management*, D. Ferrari (ed.), North-Holland, 1978, pp. 83-101.
9. Nolan, L. E., and J. C. Strauss, "Workload Characterization for Time-sharing System Selection," *Software—Practice and Experience*, Vol. 4, 1974, pp. 25-39.
10. McNeece, J. E., and R. J. Sobacki, "Functional Workload Characterization," in *Proc. CPEUG 13th Meeting*, D. Conti and J. L. Walkowicz (eds.), October 1977, pp. 13-21.
11. Fishman, G. S., *Concepts and Methods in Discrete Event Digital Simulation*, Wiley, 1973.
12. Agrawala, A. K., and J. M. Mohr, "Some Results on the Clustering Approach to Workload Modelling," in *Proc. CPEUG 13th Meeting*, D. Conti and J. L. Walkowicz (eds.), October 1977, pp. 23-38.
13. Sreenivasan, K., and A. J. Kleinman, "On the Construction of a Representative Synthetic Workload," *Comm. ACM*, Vol. 17, No. 3, March 1974, pp. 127-133.

Control of computing funds and resources in a networking environment*

by BEVERLY O'NEAL and RONALD SEGAL

EDUCOM**
Princeton, New Jersey

INTRODUCTION

Users or potential users of computing services at research and educational institutions are currently trying to meet their computing needs in cost-effective ways that will also provide them with high-quality services. As these individuals attempt to match their needs against the expanding menu of available alternatives (mini- and microcomputers; central campus facilities; and local, regional and national networks), they may face restrictions on their choices. Some institutions treat all users alike and, subject to budget constraints, they are allowed the same access to all services or face the same restrictions. At other institutions, some users and not others face restrictions on the amount of available computer funds and on the types of resources that can be purchased.

Often the treatment of users depends on how the institution bills for computing services. For example, access to local computing services is "free" at many institutions, and only very superficial efforts are made to allocate or account for computer usage. At others, users are given "funny money" accounts (accounts that represent funds pre-allocated to the computer center) so that local usage can be monitored and controlled. In both cases, there is little provision for the use of services other than the central facility. When "real" money (operational funds) is used to pay for computer services, users often have more flexibility in selecting services. Thus, institutional financial policies may be a major factor in determining how well the computing requirements of the user community are satisfied.

One often-proposed method of meeting some user needs for computing services is through a national resource sharing computer network. Such a network could provide access to specialized computing resources, minimize duplication of software development and promote a widespread exchange of resources among educators, researchers, administrators, and students.¹ Sixteen institutional participants worked

closely with EDUCOM in a recently-completed project² to investigate these issues in more detail. The participants included large universities and small colleges, public and private schools and teaching and research-oriented institutions. Interviews with key people at each institution concentrated on obtaining a deeper understanding of the institution's policies and practices with respect to computer usage and possible computer networking.

Although this paper is mainly concerned with the implications of the various types and levels of user control of funds on network participation, the discussion is relevant to most currently available computing options.

INSTITUTIONAL PRICING STRATEGIES

The pricing structure established by an institution's computer center can be used to implement various objectives.³ A common goal, for example, is to recover a fixed percent of the overall costs of the center (usually 100 percent). Since the net price to users often represents the sum of many component resource prices, the relative charges for these component services presents another degree of choice within the primary goal of cost recovery. Some institutions attempt to relate these prices to actual costs—i.e., the charges for memory, CPU, peripherals, etc. are directly proportional to their costs. Alternatively, the component charges may be directed towards encouraging efficient system utilization, allocating scarce resources, maximizing system usage and/or revenue or achieving a variety of other management goals.

Instead of setting prices by the separate components of a service, circumstances may indicate other strategies for selected services. These include charging a flat rate (usually hourly), pricing by service and unit input or output pricing (i.e., by item retrieved from a data base or by transaction processed). All of these approaches can be scaled to a given cost recovery rate if desired, and can be tuned to meet the objectives described earlier.

The usual approach in pricing is first to establish a base or standard price. Variations of this price can then be used by management to attain operational goals. For example, shift differentials are common procedures to encourage load

* This work was supported by the National Science Foundation under Grant Number MCS75, "A Simulation and Gaming Project for Inter-Institutional Computer Networking."

** EDUCOM is a consortium of more than 270 colleges, universities and non-profit organizations that serve higher education. It was founded to help its members make the most efficient use of computer communications technology.

leveling, and priority differentials are a traditional way to use the price mechanism to determine scheduling. Often there are variations in price to reflect institutional policy towards various user groups or categories. For example, surcharges are frequently placed on all outside users, with even higher surcharges if the user is not from an educational institution. It is common to bill users with outside funds at a rate equal to full cost recovery, while other internally-funded users may be charged a lower rate (i.e., they are partially subsidized by the institution). In this context, a billing percentage of zero for some users implies that they receive free computing. This approach is preferable to no billing since it permits monitoring and accountability. More important, it has been shown that when users know the actual cost of their computing, even if they don't have to pay it, they function in much the same manner as those paying the full charges.

SOURCES OF USER FUNDS

In the past, many institutions offered computing free to their internal user communities. Although this practice still exists, at least partially, at many institutions, the current trend is towards charging in some fashion for computing services. Even if the charges do not amount to full cost recovery, they are useful for the purposes of allocating a scarce resource. At institutions that charge for computing services, a prospective user must look to either internal (institutional) or external sources of funding to pay for these services. At any institution, if a user does not feel that the internal services are appropriate, and wants to purchase services from an outside source, he must find a means of paying for these services.

Internal sources of funds

Internal funds may be allocated in different ways within the institution. Frequently, some or all computing is provided as a "free" resource. This was true for the main institutional facility for two of the small colleges in the project. At most institutions, this occurs primarily with special purpose micro- or minicomputer facilities. Usually, the "free" facility is a dedicated operation serving a small, homogeneous community within the institution such as a department or research project. Depending on their location in the organization and function, such facilities may be capitalized by state funds, government or foundation grants, tuition revenues, or departmental operating funds. In lieu of charges, there is usually an implicit allocation of resources depending on one's status or role in the organization, size or type of job, or the total system workload compared with the computing capacity (i.e., service may deteriorate as usage increases).

A second method used at many universities is to give money directly to the computer center for operating expenses. Users (generally for academic or internal administrative purposes) then receive accounts against which they

may draw. These funds, since they actually have been allocated to the computing center, may not be spent for any purpose other than computing at the local facility. At half of the institutions studied, computer charges for most local users were handled through a combination of free computing and "funny money." Although there may be an upper limit to the total amount of funds available to a user, it is usually a straightforward process to obtain additional funds.

There is a growing trend towards a third approach: Considering money for computing as part of the operating expenses of the department or college. Over a third of the institutions studied treat their computer budgets in this way. The decision on how much money is to be spent on computing is made (often at budget time) at the departmental level. Operating funds are usually considered as "real money" and the department, if not the user, has reasonable freedom to spend them as it sees fit—e.g., for an internal system, at a service bureau, for networking, etc.

These approaches have different implications for both the computer center and the user. In the "free resource" and "funny money" cases, the computer center has a fixed income with which it can budget its activities. Although this income may or may not meet what the computer center management perceives as its needs, it is a known amount that is negotiated periodically between the computer center management and the administration. The job of the computer center is then to provide the best possible service for the available money. In the third case ("real money"), however, the income is not so certain, since the computer center must compete with other resources that are vying for the users' dollars. This puts the center in a very different light. It must now stand or fall based on the quality and price of the service it provides relative to available alternatives, i.e., the computer center is running a "business."

For the user, the first two cases represent degrees in his perception of a free good. The former may appear to him more unlimited than the latter (that is, if he could just get his turn to use the facility!), but there is no sense in either environment of being able to exchange the use of this service for any other that he might need. That must be done by special appeal or process, and at an incremental cost. With operating funds, the tradeoffs are much more obvious and concrete. The user measures local services against alternative sources of computer services, if not also against the cost of new books or journals, graduate students, or secretarial help. He must then make a conscious evaluation of how much the local computer services are worth to him when compared to other options.

There are obviously various combinations of these methods. Many institutions, for example, use a combination of "real" and "funny" money so that users are billed at a specified percentage of the actual cost of their computer usage and university funds make up the difference.

External sources of funds

External sources of computing funds available to the user are usually research grants or contracts sponsored by gov-

ernment, industry, or other funding agencies. The process for obtaining these funds depends on the particular source involved. Typically, the user must take the initiative obtaining those funds, and often, in the process of obtaining them, they are earmarked for computing. In general, users' control of their externally-supplied money more nearly approximates the "real money" method of handling institutional funds. Again, the tradeoffs must be weighed by the user himself and a choice made.

For the computer center, users supported by external funds may represent a relatively unpredictable source of income. When a new grant is requested, the potential recipient decides what portion is to be spent for computing services. There is no guarantee that all of the funds indicated for computer services will be spent as planned, or that the grant will be awarded in the anticipated time frame. A majority of these funds come from granting agencies that can fluctuate in the level of their awards from year to year and there is no sure continuity. This uncertainty is one price that the individual researcher must pay for the added control that comes with outside funding. In aggregate, however, the income from such outside sources is normally fairly predictable, and the computer center management becomes very adept at estimating what this will be.

USER OPTIONS FOR OBTAINING COMPUTING RESOURCES

Depending upon the policies established by the institution for the expenditure of computing funds, a potential user with computing needs and funds is often faced with a choice of options. The money may be spent at the local computing facility (which may be a service purchased externally by the institution); to purchase services from an external source such as another university, a research institution, or a service bureau; or to purchase his own hardware, such as a minicomputer.

At institutions participating in the project, users with external sources of funds had the most freedom to exercise options that they selected. Those who use institutional operating funds to purchase computing services face more restrictions but usually have some degree of choice. Users with internal funds for computing, however, find it very difficult to purchase from any source except the local computing center.

Local services

Most institutions historically have had a single general purpose computer center that satisfies most of the needs of local users. As the cost of hardware drops, institutions are frequently offering several alternate centralized facilities. A common example is that of running student jobs and CAI (Computer Assisted Instruction) on a particular machine that is often a minicomputer. It is still the case, however, that most users are restricted to internal services and that,

although there may be multiple campus facilities, the choice for any given user or application is usually quite limited.

External services

There are several factors that must be faced in deciding whether or not to purchase a service externally. If the desired service is not available locally, the first question is usually, "Can it reasonably be added to the local service offerings?" Larger institutions offer most standard services. Services that they do not, and possibly cannot, offer locally include such things as large data bases, specialized application packages, and bibliographic retrieval systems. If there were a large demand for any of these, the institution might consider acquiring the service. However, for a growing number of specialized services, it is not cost-effective to do so when both installation and maintenance costs are considered. Such services are likely candidates for outside purchase.⁴

Another consideration is the quality of service. This is not as important in measuring local service as it should be but usually enters into the decision in evaluating external suppliers. As more external options become feasible, the issue of their comparison with local services will arise more frequently. For example, are external services a user option if they are "better" or less expensive than the local service? How does the quality of service, support, documentation, etc. of the external supplier compare with that of the local facility? How reliable is the external supplier as compared to the internal facility? Not only must the user considering an external source be aware of such factors, he must be able to evaluate their importance to him.

Dedicated computing

The number of applications in which the purchase of a mini- or microcomputer system makes sense is increasing. Hardware costs are decreasing rapidly, while performance is improving. Minicomputers can now meet many user needs at a low cost, while offering a degree of control and flexibility that the user cannot exert over the local computer center. In addition, minicomputers often can provide an interface to the central and network computers in addition to providing dedicated, low-cost, stand-alone service.

In addition to their role in satisfying specialized campus-wide needs (CAI, administrative systems, etc.), the price performance ratio of small systems justifies serious consideration at the project or departmental level. Typical applications include word-processing, introductory programming courses, monitoring and control of research experiments, specialized scientific software packages, and small simulations used by multiple students.

It is beyond the scope of this paper to comment on the problems (often hidden) associated with small computers, such as estimating their full cost, maintenance difficulties, support problems, software shortcomings, etc. At the user level, constraints on the availability of such resources are

similar to those for acquiring external services. Those users with internal operating funds or specific outside sources of funding are usually the only ones that can consider such acquisitions. Often the review process is more stringent and specific than for outside services since there is a capital investment involved.

At many institutions such acquisitions are not dealt with in an organized fashion and this has led to a diversity of incompatible equipment and duplication of capabilities. Given the total cost and net impact on the organization of these mini- and microcomputers, the need for institutional-level policies and coordination is becoming critical.

AVAILABILITY OF COMPUTING RESOURCES TO USERS

Machine resources

If more than one computer is available at an institution, certain classes of users may be restricted to particular facilities. The most common examples of these are the use of specific computers for class instruction or administrative data processing. Even if there is only a single computer, the institution may impose restrictions on the resources available to certain user groups. Samples include limits on CPU time, maximum memory allocations, limits on disk storage space, or job priorities. Systems with restrictions such as these usually have ways in which the limits can be raised by special request.

At institutions that use "funny money," or no prices at all, controls are important and may be quite involved. Several institutions imposed restrictions, for example, on the types of jobs that could be run by the various user categories. Others imposed resource limitations also based on type of user. One institution had developed an involved cutback algorithm whereby overusage by one class of users impacted only that class and no others. Another operated by dividing the day into periods for administrative usage and periods for academic usage. Within the academic time slots at this institution, usage was on a first-come-first-served basis.

When computing is financed partially or totally by "real," or operating, money there is less need for these types of controls. At such places, the pricing mechanism usually serves as a simple, effective allocation mechanism. To the extent that a free market environment is considered desirable, the motivation for using real money for computing increases.

Support services

There are two major areas in which human resources are required—consulting and programming. A few institutions provide extensive support free of charge in both these areas, while most only offer casual consulting free and charge for programming or extensive consulting. Some institutions offer minimal consulting and no programming whatsoever.

Different users at a single institution may receive different levels of support. For example, students can usually get information about running jobs but cannot obtain programming help from the consultants. On the other hand, a user with outside funds can often pay for both programming and additional consulting services from the computer center staff.

At most institutions in the project, limited consulting was available free of charge. Most of them allowed all users to buy programming services with computing funds although one institution did not allow student funds to be used to purchase such services and two institutions provided no programming at all.

INSTITUTIONAL REVIEW PROCESS

Most institutions consider it necessary to have a review group to provide an institutional view on the use of facilities other than the local centralized facility, whether they be external services or the acquisition of individual minicomputer systems. A variety of titles are in use, such as "Computer Review Board," "Computer Advisory Committee," or "Computer Center Oversight Committee." In addition to its role in reviewing outside purchases, it often also functions as a committee overseeing or advising the local computer center on purchases and budgets. Typically, members represent a variety of functional areas including faculty, department heads, heads of large research projects, computer center management and administrators. Usually the group is an institution-wide committee, although it may function on a college basis. At state institutions, there may be parallel legislative and institution-specific groups. At smaller institutions the review process may be performed by one person, such as the computer center director.

The areas of concern to this committee also vary. In some cases it reviews only proposed purchases of hardware, or only cases in which institutional funds are to be spent on external services. It may, however, have responsibility for all aspects of computing at the institution.

At some institutions, as long as the money involved is from outside sources, approval is almost automatic. Thus, a user with a research grant may have no difficulty in getting authorization to purchase a mini-computer or to buy time from a service bureau, whereas a faculty member wanting to use departmental computing funds to buy access to a specialized package offered elsewhere may find this extremely difficult.

About a third of the institutions had established a formal review committee as described above for the purchase of outside service. Another third had a review process that was performed either by the computer center director or through an administrative office such as the vice president for research or administration or the university's finance committee. The rest of the institutions had no such review process, either because all users had operating funds and were unrestricted, or there was such a low level of outside usage that any cases were handled on an *ad hoc* basis.

Computing at an institution is perceived differently by the

various constituencies that come in contact with it. The administration often has one perception, while the computer center director may have a second, and the users a third. The review process should be the place where these sometimes conflicting viewpoints can be resolved and translated into institutional priorities and policies.

There is a clear requirement for a review process at most institutions, and many are moving in this direction. However, emphasis to date has been on the approval or disapproval of specific requests, rather than on the establishment of mechanisms for providing more general policy guidance. As the number of options increases, institutions are recognizing the benefit of establishing overall policies for guiding and coordinating their computing decisions. For example, recommendations for standardization of time-sharing terminals have come from several committees, policies for network access permission have come from others, and a few have taken the initiative and promoted policies for acquisition and operation of campus minicomputers.

USER CATEGORIES

Users at any given institution can be grouped in a variety of ways to reflect source of funds, computing activity or homogeneity of grouping. Institutions in the project were asked to define categories that represented the major levels at which funding and usage are controlled. In general, the categories selected represented line items on the institutional budget. Although many of the variations were unique to individual institutions, the many common elements in the groupings reflect the similarities in how computer usage is perceived at educational and research institutions. These groupings exist because most institutions do not treat all users alike. Instead, the rules that apply to a user, and the control that he has over how and where his computing money is spent, depends largely on the category into which he falls.

The groupings used were surprisingly similar across the institutions. Almost all of the educational institutions, for example, had a category entitled "instruction." This included classroom use of the computer both in teaching a programming language and as a tool in discipline-related subjects. Another grouping used by the majority of the institutions was that of institution-funded research. This represented student and faculty work for which there was no outside source of funds. One institution separated such research work into that done for masters' theses and that done for Ph.D. dissertations. Typically, these two categories, instruction and internally-funded research, had great difficulty receiving permission to do computing on anything except the centrally provided facilities. Even in "real money" environments, central review committees were less than sympathetic to most requests for external expenditures or local capital investments by these users.

Three-quarters of the participants identified externally-funded research separately, and one even separated government-funded research from other externally-funded research. Those that did not use this category to distinguish

users either had almost no externally-funded research (the very small teaching-oriented colleges) or felt that this was not a useful distinction since all users had "real money," i.e., this category was merged with institution-sponsored research. In general, funded researchers had the most control over their own computing destiny. By definition they have hard dollars to spend, their expenditures do not place a drain on institution revenues, and hardware acquisition or outside computing is often an explicit part of their grant or contract. Even so, there are often pressures to use the local facility if at all possible.

Administration was also a separate category at three-quarters of the institutions. At most locations administrative work was performed on the same facility used for academic applications. By virtue of their place in the decision-making hierarchy, this group of users should be expected to have great flexibility of choice in making computing decisions. Up until now the choice seems to have been made between separate administrative facilities and the campus computer center. Security and privacy concerns, the cost of running large processing jobs, and the perceived need for control and flexibility have all contributed to this limitation on options.

Two-thirds of the institutions identified computer center staff usage as an explicit item. The rest generally assumed that since these users were not billed, there was no need to explicitly identify or budget for their usage. The philosophy was that staff usage was a fixed and necessary part of the facility workload. Except for explorations of remote resources by user services personnel on behalf of their customers, there is rarely any justification for computer center staff to seek alternatives to the facility that they operate.

The final category identified by most sites was that of external users of the local facility. About a fifth of the organizations further separated outside educational use from other external use. Where this was done, non-educational users were billed at a higher rate than educational users. In general, even outside educational users paid a somewhat higher rate than internal users. Obviously, this group of users usually has no difficulty in "going elsewhere" for services whenever it chooses.

Although these were the most common categories of usage, there were a few variations and one notable exception in categorization. Instead of following these breakdowns, one institution reported usage by schools and administrative units. This reflected the philosophy of decentralized control, i.e., each school had to account for computing expenditures as part of its operating funds.

CONCLUSIONS—NETWORKING IMPLICATIONS FOR USERS AND THEIR CONTROL OF RESOURCES

Groups likely to benefit from networking

Outside users. It is evident that some classes of users are more likely to use a network, or other alternatives to the central computer center, than others. The most obvious

example is that of the outside (non-institutional) users of a computer center. These users are not directly affiliated with the institution and have no restrictions to prohibit their purchasing services elsewhere. Often they are purchasing raw computing power rather than the entire package of service and user support. In these cases, they are likely to go elsewhere for service anytime that they perceive it is in their best interest to do so. At most educational institutions these users represent a very small fraction of the total usage and are generally considered only as purchasers of excess capacity.

Externally-supported researchers. Users whose research is supported by external research funds are likely to both use and benefit from a network. The nature of their work often requires special packages or services that may not be offered at their institution. In most cases they are faculty members or graduate students who are familiar with those services that are useful in their work and that exist outside their own institution. This category of user, therefore, is likely to know about applicable remote resources (via professional or discipline contacts), and is also in a position to take advantage of them since the institution is likely to have less control over how their funds are spent than it does over other users.

Users of specialized resources. Any user who needs a specialized or unique resource is a likely candidate to benefit from a network. Examples of specialized resources include unique hardware, CAI systems, statistical packages, econometric models, planning models, and specialized data bases. These are the resources that may not be available at the local institution, and would be very expensive, if not impossible, to establish locally. Users of specialized resources may fall into any user category. Unfortunately, it is their category rather than their need that is likely to be the primary determinant of whether or not they may obtain these services via a network. There are indications, however, that this situation is slowly changing. For example, review committees are much more sympathetic to requests for outside service when the resource is not available locally (many state this as a formal policy). Institutions, even those that provide local service as a free good or on a "funny money" basis, are recognizing the need to support outside usage in these circumstances.

Groups unlikely to benefit from networking

Student users. Except for special packages like CAI materials, student instructional needs are less likely to be met through networking. Programming courses and the use of simple models and packages are the kinds of applications that can be accommodated very effectively at most local centers. In many cases a dedicated minicomputer may best serve such a group. Particularly in programming courses, access to outside services will not enhance the basic quality of instruction. An exception to this may occur at smaller schools that can not support the full range of programming languages. In such cases, the use of other facilities over a network may provide the only access to that service.

Computer center staff. The staff of the computer center is likely to be the most knowledgeable about networking possibilities and availability. They may be able to get free (or very cheap) trial accounts at other institutions and may try out new packages over the network for their user community. Their own need for these services, however, is limited since they are employees of the central facility and their job is to facilitate its use. Their budget in "real money" is likely to be extremely limited, and although they may be a major source of information, they are unlikely to become a major purchaser of network services.

Traditional administrative users. Administrative data processing, for a number of reasons, is less likely to be performed over a network. Primary reasons are the perceived need for control over the computing resources used, the concern for security, and the volume of input and output. Administrative applications are often very time-dependent and usually receive highest priority in the case of a crisis. On a network, such work has no assurance of special treatment and must be adapted to the schedule of the host computer. The second concern, security, is often mentioned but is actually less valid. Unauthorized access to such data may, in fact, be more difficult at an institution that is accustomed to protecting the data of a variety of outside customers. At present, networks are most effective for applications that require only a nominal amount of data input and output. Consequently, communications capacity limitations and the cost of data transmission currently impose severe barriers to administrative applications.

It does not appear that the overall situation is likely to change in the near future for traditional administrative applications. However, with the advent of decision-making tools such as planning and forecasting models, and specialized hardware dedicated to "office automation," there are indications that new, non-traditional applications will be carried out in the most appropriate manner. Again, the place of this user community in the decision-making hierarchy of most institutions makes it likely that it will be able to acquire both funds and authorization as the need arises.

Groups that could benefit from a network but are unlikely to use one

Student users. Some student instructional work might benefit greatly from expanded network services, and yet be unlikely to have access to a network. In particular, the use of discipline-oriented computer programs as teaching devices falls into this category. Marketing models, political science data bases, chemical reactor simulations and econometric statistical packages represent computer resources of this nature that are not offered at every institution, and yet could represent a useful supplement to course work.

Internally-funded researchers. Internally-funded researchers, although they could also benefit from a network, are less likely than funded researchers to fully utilize one. The attractiveness of a network here lies in the specialized services that can thus be obtained; i.e., the need is similar to

that of externally-funded researchers, but the funds and flexibility are not available.

The primary problem that the above groups face when considering networking is that services purchased over a network must be paid for with real money, and this represents an apparent cash drain on the institution. To the extent that the user can demonstrate that the value justifies the total cost, and adequate funds are available, this is not a serious problem. In many cases, although there is a net cost saving to the institution by not doing the work internally, this is difficult to demonstrate, particularly in an environment where the user pays less than the full cost for local computing. The solution lies in reorienting institutional accounting and billing procedures so that these trade-offs can be made explicit. In other words, if it is cheaper overall to go outside than to do the work internally, one should be able to show this in a straightforward manner.

Needed organizational changes in a networking environment

Computer center focus on providing services. Computer centers must gradually change their images from that of providers of general computing capacity to that of providers of computing services. In this context, they must focus on how they might best help the user to satisfy his computing needs, rather than trying to adapt his needs to fit their offerings. This concept is much easier to state than to implement. Hardware budgets, number of employees and sophistication of equipment are traditional measures of responsibility that are easy to quantify and evaluate. Service or cost-effectiveness of service are generally accepted concepts, but very difficult to measure. Administrative officers must find a way to motivate their computer center to focus on the latter concepts in looking at its performance.

Expanded responsibility for user services. Particularly in a networking environment, the role of user services must be greatly expanded. It must be able to direct the user to available service options and alternatives, to assist in the selection process, and to assist the user in utilizing remote resources. All this is in addition to their more traditional services relative to the local facility. Although this new role is a difficult one, it is necessary before the computer center can effectively function as a provider of service as previously described.

More direct control for users. The current situation with respect to outside services at most educational institutions can be summarized as a financial consideration: Those with "real" or outside money usually may spend it where they like; those with "funny" or internal money are usually constrained to the local facility. This institutional posture may

not show any relation either to need or to cost/benefit considerations. In order to effectively function in a service environment, control over choice, and the responsibility for those choices, must shift from the institution to the user (or at least to the department). The user must be given both the motivation to examine alternate modes of obtaining service, and the authority to act on his decision. His decisions need not be based on cost alone, but could also include considerations of reliability, user support, suitability of service to his needs and ease of use.

Make economic implications of choices more explicit. One of the major barriers to implementing the concepts just described is that the true economic implications of alternatives are rarely very explicit. Users are often faced with a choice, for example, between "free" internal computing and relatively expensive outside alternatives. In reality the incremental cost of providing the internal service may very well be higher than that of going outside. The evolution from free computing to real money will help this situation, as will an environment that motivates the computer center to focus on cost-effective service instead of merely the internal provision of service.

Implications for networking. Although networking offers a very attractive alternative for meeting certain computing needs, it will probably never be a very large percent of the computing usage at the average institution. However, for the user who cannot find what he needs locally, it represents a significant alternative for meeting his requirements through the purchase of external services. Such usage will grow slowly as institutions experiment with this mode of usage and accept it gradually where it proves successful. Financial concerns will continue to be a major factor, and users with their own source of funds will have more ready access to the network. This will gradually change as institutions begin to view computing as a service rather than a facility, and users acquire more control over their selection of such services.

REFERENCES

1. Emery, J. C., "Implementation of a Facilitating Network," *Policies, Strategies and Plans for Computing in Higher Education*, EDUCOM, Princeton, New Jersey, 1976.
2. Final Report to the National Science Foundation on the Simulation and Gaming Project for Inter-Institutional Computer Networking. Grant DCR75-03634, EDUCOM, Princeton, New Jersey, (in preparation).
3. Bernard, D., J. Emery, R. Nolan and R. Scott, *Charging for Computer Services: Principles and Guidance*, PBI Books, Petrocelli, New York, New York, 1977.
4. Nielsen, N., and R. Segal, "Implications of a National Computer Network for Higher Education and Science Research," *Proceedings National Computer Conference*, Anaheim, California, June, 1978.

The economic impact of network affiliation upon institutions of higher learning

by NORMAN R. NIELSEN

SRI International
Menlo Park, California

INTRODUCTION

Educational and research institutions are increasingly becoming concerned with determining the best means by which their computing requirements can be satisfied, since these needs are becoming increasingly varied and are continuing to expand. Simply installing more processor power is no longer a solution. More types of software support, ranging from languages to specialized applications packages, are needed. Various types of data bases must be made available. Each of these entails the development of the necessary facilities as well as continuing maintenance. While technological developments are continuing to reduce the cost of raw computing power, the cost of software is continuing to rise relentlessly. Accordingly, it is becoming economically impossible for many institutions to support the breadth and variety of software that their users desire.

One oft-talked-about solution to these growing problems is the computer network. For example, by linking institutions together it would be possible to share the development and maintenance costs of software and specialized data bases over a much broader user base. Clearly, it is unlikely that there would always be a nice fit between the services economically supplied locally and the services available over a network. On the other hand, it is clear that a national educational and research computing network would contribute toward the solution of the escalating support costs being encountered. It would also address the problem of nonlocal availability of unique or specialized resources.

There are obviously drawbacks to such a national network. Yet, the potential is so great that considerable interest exists. For example, EDUCOM (a consortium consisting of more than 250 colleges, universities, and nonprofit organizations and dedicated to helping its members make the most effective use of computer and communications technology), with the financial support of the National Science Foundation, organized a series of General Working Seminars on the subject of computer networking in higher education and research. Invited participants were drawn from the ranks of university administrators, computer center directors, users from key disciplines, and computer scientists.

The results of the General Working Seminars have been

set forth in a very readable book¹ and shall not be repeated here. However, a general observation concerning the participants' conclusions is pertinent. Regardless of their professional role, all participants agreed that it is now technically feasible to create a national network linking computer facilities at colleges, universities, and research institutions. While technological problems do remain to be overcome, these were viewed as minor in relation to the non-technical difficulties confronting such a network, difficulties involving economic, political, and organizational considerations. Accordingly, it was believed critical to obtain a clear understanding of these factors before embarking upon any large-scale network development.

A variety of projects have been conducted to examine various aspects of networking. For example, a study by Weingarten, Nielsen, Whiteley and Weeg² examined the effects which the National Science Foundation's Regional Networking programs have had upon institutional computing activities. Berg³ has examined the exchange of computing services in relation to comparative advantage and international trade concepts from economics. Heller⁴ studied the relative price differentials between institutions for processing different types of standardized jobs. This represents an empirical study of the comparative cost advantages of different computing facilities, independent of the advantages of greater service offering availability.

No one project, however, has taken a comprehensive look at the overall impact of a national network linking institutions of higher education and research. The concerns voiced at the General Working Seminars, coupled with the networking interests of many institutions, led to the formation of a comprehensive three-year investigatory project funded by the NSF and conducted by EDUCOM.

Experimentation on a national basis with a prototype network was rejected for a number of reasons. Such an effort would be very costly, would severely restrict the alternatives that could be investigated, would disrupt the normal operations of the affiliated institutions, and would require a significant commitment of energy and personal time from many key individuals.

Accordingly, a simulation approach was taken to investigate the non-technical issues confronting a national edu-

cation and research computing network. This paper reports upon just one aspect of the overall study, namely the economic impact which affiliation with such a network might have upon member institutions.

THE NETWORK SIMULATION PROJECT

The network simulation project conducted by EDUCOM was divided into three phases. The first phase consisted of basic data collection. Eighteen education and research institutions participated in the project, each contributing a voluminous amount of data with respect to the computing resources which they maintained and operated (hardware and software facilities), as well as data with respect to current operations (pricing schedules, priorities, demand levels by service type, and turnaround times as a function of demand levels). In addition, data was collected on user profiles. A network simulation model to operate upon this data was constructed. The results of this project phase have been reported previously.⁵⁻¹¹

Phase two expanded and updated the accumulated data base. Visiting teams interviewed administrators, users, and computer center personnel at each institution. Computing policies, practices, and decision rules, as well as institutional computing goals, were deduced from the interview data, confirmed with each institution, and added to the data base.

Phase three enhanced the basic simulation model, so that it could be operated in three modes:

- As a basic simulation model, with all decisions at each institution being reflected by programmed decision rules.
- As a "straw network" game, with the player making decisions for his institution but with pre-programmed decision rules making all other decisions.
- As a full network game, with all decisions being made in parallel by institutional representatives.

The basic simulation model was used to investigate a number of issues such as network stability to induced shocks; traffic flows as a function of price, service levels, and policy decisions; and migration and usage patterns. The straw network game was used by each of the participating institutions to learn about the possible effects of network membership and to experiment with alternative policy decisions. The full game was employed during a three-day session held at EDUCOM with representatives attending from each of the participating institutions. In addition, these attendees participated in a number of workshops. Project findings are summarized in Reference 12.

Although institutions have a variety of reasons for seeking to join and participate in a national computer network, all share a common concern—namely, the likely economic impact of participation. Economics is the glue that binds all portions of the institution together, and the economic impact of national network membership is something that would be felt indirectly if not directly by all members of the institution.

Accordingly, it is important to consider in some detail the possible economic consequences of network membership.

CASH FLOWS

When "cash flow" is mentioned in connection with a network, the associated thought is often "unplanned cash flow imbalances." It is very likely that there will be a net cash flow (either inflow or outflow) between any one institution and the other institutions on the network, for it is very unlikely that an institution's exports of computing services would exactly match its imports. Further, because of the variations in demand and supply, the net cash flows are very likely to differ from the desired or budgeted level.

Cash flow imbalances

Whether a net cash flow will or will not pose a problem for a specific institution is a function of three characteristics—the size of the net cash flow (relative to the total computing budget and to the budgeted level of net cash flow for that institution), the size of the cash inflow and outflow individually relative to the total computing budget, and the rate at which the magnitude of the net cash flow is changing.

Relative size of net cash flow

A net cash inflow or outflow will have little impact if it totals only a small percentage of an institution's total computing expenditures or of an institution's budgeted level of net cash flow. An institution may plan to run a small surplus to relieve budget problems or to run a large deficit because it chooses to purchase services externally rather than maintaining a larger on-campus computing capacity. In either case, so long as the difference between planned and actual is relatively small, the net cash inflow or outflow is not likely to impact the institution significantly.

A large net cash flow, relative to budget, may have a significant impact. If steps were not taken to increase (decrease) capacity or to restrict the purchase (sale) of services, there would necessarily be an effect upon an institution. An unexpected net cash outflow would require funds to be diverted from other activities or would require deficit financing on the part of the institution. An unexpected net cash inflow may be used to support other activities (or it may be law flow to another organization, e.g., the state treasurer). It may also impact the service received by internal computer users and may cause other side effects (e.g., tax implications, violation of restrictions on unrelated income or source of income).

Size of the cash flow components

Independent of whether cash inflows match cash outflows, there should be a concern for the size of the cash inflows and outflows relative to the institution's overall com-

puting expenditures. For example, a high level of external service purchases should raise questions concerning continued availability of these services to the institution's members and the possibility of negotiating volume discounts from volume suppliers. On the other hand, a high level of sales to external users should raise questions concerning the likelihood of continued demand for these services. In either case there should be a concern about possible contractual arrangements, quantity discounts, volume guarantees, and the like. The sudden termination of a major supply source could seriously impact the continuation of the users' educational and research computing while the termination of a major demand source could remove revenue needed to support relatively high fixed costs of the computing facility. In either case, there is a potentially significant economic impact upon the institution, and the risk must be recognized and dealt with accordingly.

However, unless an institution happens to offer a very popular specialized service, it is expected that the cash flows will be relatively small. Even in those cases where the cash flow is intended to be relatively large, there are mechanisms to protect against the increased risk. Thus, risks previously described should not discourage economically sound arrangements. It is just that the importance of proper contractual arrangements to protect the interests of both buyer and seller becomes much greater in these circumstances.

Rate of change of net cash flow

The fixed costs associated with the operation of a computing facility are relatively large. Thus, in the short run, a rapid change toward greater cash inflow would result in deteriorating service or response time, while a change toward lesser cash inflow would result in operational losses (or reduced surpluses) with such budgetary shortfalls having to be met with other institutional funds. Because of the relative inflexibility of short term cost/capacity adjustments in a computing facility, a rapid change in net cash flow can have potentially serious consequences.

On the other hand, more slowly evolving changes offer the potential for lesser impact, since there is more time for the institution to make appropriate adjustments in purchase/supply levels.

Areas improved by differential cash flows

There are a number of impacts which may stem from the above-described factors. These impacts are controllable in the sense that cash flows can be controlled with management attention. The key is to make sure that proper and timely control is exercised.

Minimum acceptable central facility

Most institutions feel for reasons of control, local service, prestige, and so forth that there is a minimum size and capability level below which they would not want to reduce their central computing facility. (Often, this minimum size

is not very far below current capacity.) Such a capacity floor introduces a number of complications into network membership.

For example, the existence of a minimum capacity target implies that the central facility should operate at greater than or equal to the specified minimum capacity. Otherwise, the institution would be paying more than it should for computing, representing a direct drain on the institution's budget. The only alternative to an outright subsidy (paying for the operating deficit) is to attempt to increase net income through price adjustments. However, higher prices may encourage additional business to flow to the network (thereby decreasing revenues and requiring that prices be raised again in a continuing cycle) and lower prices may not generate sufficient additional demand to overcome the reduced revenue from existing business.

A minimum capacity level may also result in a number of institutional goals being overridden. For example, an institution may desire to provide its entire user community with ready access to a national network so as to enhance the effectiveness of teaching and research. However, if this results in insufficient business being directed to the institution's own computing facility, then pressures will develop to restrict network access. Network access may be rationed, outside usage may be taxed (to make internal use look economically more attractive to the user), or network use may be prohibited for certain types of computing (or for certain types of users such as students).

Hard versus soft funding for computer services

Many institutions fund internal computing with so-called soft money. That is, the institution directly supports the computing facility's budget and then allocates or rations usage by distributing time or "funny money" credits to users. The "funds" so distributed to users have no value for anything other than local computing, hence the term "soft." In contrast, all network computing will involve "hard" or real dollar expenditures. Thus, institutions must find additional hard money to fund those network services that are utilized, as additional cash outflows will be triggered.

The funds to support network usage may come from four sources. They may come from hard dollars earned from the sale of computing services to the network (analogous to earning foreign exchange in international trade), or they may come from a reduction in support for the institution's own computing facility. Additional hard funding may be obtained from reductions in other programs or activities at the institution, or it may be obtained from new funding sources. However, in pursuing new sources of funding, network usage would potentially have to compete with all other institutional programs seeking additional funding support.

Network sales viewed as an asset exchange

An institution that is selling computer services to network users is in essence performing an asset exchange, exchange-

ing a portion of its computing facility for dollars. Whether or not this is advantageous depends upon the institution. For example, many state universities currently receive a fixed allocation from the state legislature to support computer operations; in turn, all revenues from services go to the state. Computing service sales to the network under these conditions would be disadvantageous, for the institution would essentially be losing part of its computing capability without receiving any benefit in exchange.

Revenue source

Membership in a network is often looked upon as a revenue source. There is a widespread belief that institutions can "make a profit" by selling computer services. (Clearly, not all institutions can be net sellers of services, although obviously some will be.) However, even for net sellers of service, the opportunity to generate net income is not without cost.

First, if capacity is expanded to support outside service sales, then increased capital and operating costs must be considered along with the investment risk (users may switch to another facility at a later date, leaving the institution with excess capacity). Second, if capacity is not expanded proportionately, the institution's own users will be "paying" for the outside usage in terms of longer turnaround times and increased congestion. In fact, if the institution offers a priority surcharge scheme, internal users may literally subsidize outside usage through the priority charges required to maintain the same turnaround time.

BUDGETING PROCESS

The budgeting process will also undergo change in a networking environment. At the present time most institutions make an estimate of sponsored research computing (based upon research contracts in-house and upon proposals outstanding) and an estimate of the appropriate level of educational, administrative, and unsponsored computing. These figures are compared with the budget requirements of the computing facility in order to serve the aggregate workload, and a final budget is established for each type of computing. Funds for these activities are then allocated to the prospective users, using either soft or hard dollars, depending upon the policy of the institution.

With a national networking environment, two new factors appear—use of the local computing facility by external users and use of external computing services by local users. The former is a source of hard dollars, while the latter consumes hard dollars. The budgeting process still goes through the estimation phase as before. However, estimates must also be made of external use of the local facility. This estimate is more difficult to make, since an institution will generally have little knowledge of the computing plans of external users for the forthcoming budget period. (This is another reason why service guarantee/minimum volume agreements can be advantageous.) Since less is known about the volume

and timing of external demands, there is likely to be a greater error surrounding their estimate. Hence, the institution should be prepared to accept a greater shortfall of revenue or a greater excess of demand at its computing facility than customary.

The budget preparation process must also address the needs of internal users for outside services. The use of network services by internally-funded users can be tightly controlled, since the institution is in a position to dictate limits on the amounts that can be spent. The real risk arises if the institution ties the level of external use by these users to the level of service supplied to external users over too short a time span, so that fluctuations in external sales would disrupt the educational and research process.

PRICING PRESSURES

Most institutions presently operate their computing facilities as a quasi-monopoly. A variety of constraints (both funding and administrative) are established to restrict users to the local facility. However, participation in a national network implies much greater freedom for users to move to external computing sources to satisfy their computing needs, and institutions will be subject to a number of new pressures—pressures that will constrain some of the ways in which they might seek to price their computing services.

Specialization

As one would expect, different types of computers have different relative advantages in processing a specified workload. As the study of Heller⁴ showed, no one installation was least expensive for all types of work. Each had a relative advantage for some types and a relative disadvantage for others. Thus, no matter how well an institution is operating its local computer facility, there will be external services that will likely be more cost-effective for certain types of work.

As a result there is likely to be a trend toward specialization of the services offered by different facilities. As various segments of one facility's business are drained off by other facilities having a comparative advantage, the institution is forced to compete in those areas in which it has a comparative advantage. Thus, there will be pressures to specialize. This may or may not be in keeping with the institution's objectives.

Price structure

Differences between facilities are not limited to computer power alone. Many installations operate in a bundled fashion, with service and support factored into the computer rate. Accordingly, by shopping around, a user can select one service offering good support for program development and another service offering "cheap cycles" for those occasions when turnaround and support are not important.

Thus, pressures will develop to move each member institution toward a common, unbundled pricing structure (though not toward the same prices). To the extent that an institution fails to price some component of a service, there is a significant loss potential. Business that does not take advantage of the favored resource (but pays for it) will be lost, and business that does use the favored resource (but doesn't pay for it) will be gained. Revenue is likely to be lost, unless the institution adjusts toward an unbundled price structure. Such a change may or may not be in keeping with an installation's operating philosophy.

Cost-recovery disparities

No two institutions are exactly alike, and their accounting systems are likely to be even more dissimilar. Thus, the costs that are included in the rate-setting process will vary. Even if two institutions have identical computing equipment, staff salaries, and rate structures, the values calculated for those rates are likely to differ due to differences in the cost bases or cost-calculation procedures used. Even among the eighteen participating institutions in this project, we found wide variations in reported costs in environments that would appear similar or comparable. Thus, many institutions are likely to feel there is "unfair competition" from institutions whose accounting systems might yield a lower cost base for budget and rate purposes.

Local service advantages

To place the previous factors in perspective, it must be remembered that every institution has many significant competitive advantages with respect to its local user population. For example, there is the general desire of users to "do it here" where things are more familiar and where they may have greater control or can obtain better responsiveness from the "establishment." There is also the inertia factor. Computing is not a uniform commodity, and there are significant conversion barriers to switching sites and services. Consulting and other user assistance can be provided better locally. There is also a cost associated with using a network (e.g., transmission costs), so each institution can offer a built-in cost advantage to its local user community. Thus, there will always be room for an institution "to do its own thing" without fearing the immediate departure of most of its local users.

CONSTRAINTS ON A FREE-MARKET NETWORK

In theory, the national network should be a competitive market place that permits competition between facilities to squeeze out operational inefficiencies and to encourage new entries when service offerings are inadequate. Users in such an environment would shop for the facility offering the greatest comparative advantage for their computing needs. As a result, more computing would be performed for the same

cost, and everyone would be a winner. In practice, however, this may not happen. The stage is set, but there are a number of constraints which may preclude such competition.

Non-comparable services

Many of the computing services to be made available over the network are expected to be unique or specialized services. By their very nature these services will not be offered by many institutions. Hence, a competitive market for those services is unlikely to develop.

Legal

A variety of legal constraints, ranging from prohibitions on serving commercial organizations to prohibitions on serving out-of-state organizations, may restrict the ability of a computer facility to sell its services externally over a national network. The constraints will vary, depending upon educational discount contracts, state laws, and institutional charters.

Tax

Most educational institutions are exempted from property and income taxes with respect to their educational facilities and activities. To the degree that various taxing jurisdictions declare network service activities and revenue to be a non-educational use of facilities, an institution's ability to participate in the network may be sharply curtailed. Whether any commercial work will be tolerated and whether the provision of services to other educational institutions will be viewed as an educational or a business enterprise will depend upon the local taxing authorities' interpretations of their laws and regulations. However, the economic impact upon an institution could be significant, as could the constraints upon network affiliation.

Federal

An internal computing facility serving federally-funded computing users is required to share its costs fairly among the various users. There is an elaborate body of procedures associated with the definition of "fair costs." Normally, so long as the federal government pays these "fair" rates and other users, internal or external, pay no lower rate, the institution may charge whatever rate it desires.

However, will federally-funded network users from other institutions still be considered outsiders on whom surcharges may be levied, or must these users be considered part of the internal federal sharing community which must be charged the lowest rate? Also, the importing and exporting of computer services by an institution may be viewed simply as an exchange of services, in which case an institution would have to make representations to its auditors about the fair-

ness of another institution's charges. (Under the exchange interpretation, the computing charges of the external computing facility would have to be included in the local computing facility's cost base.)

One of the features often mentioned in connection with a national network is a provision for software royalties. This would enable the developer of a program or routine to establish a usage fee, providing an incentive to individuals to develop and share high quality software. It is anticipated that an individual could specify any royalty level he wished, depending upon his perception of the quality of his work and its value to others. However, arbitrarily established surcharges for internal users are not likely to be acceptable to government auditors, and the consequences could be severe if a broad set of external users (e.g., those with federal funding) were to be considered "internal users" for rate purposes. Thus, federal regulations may have a significant impact upon one of the planned mechanisms to stimulate the development and sharing of quality software.

Loss of freedom

The decision to supply service to external users may result in a loss of freedom for the supplying institution. If the supplier were to see itself as offering a full-fledged service rather than temporarily selling excess capacity, it would lose the freedom to modify network service availability unilaterally in order to accommodate local needs.

By the same token, unless the user of an external service has some form of service availability agreement, he may be giving up control over his computing supply. Thus, some other organization, with which he is not affiliated, would have the power to change the availability or conditions of his access.

Growth

Even if an institution were able to compete freely in a national computing marketplace, there are a number of constraints upon growth (both upwards and downwards) of its computing facilities that may limit network participation.

- Institutional charter—The institution may not be able to operate an auxiliary, revenue generating enterprise, so that external business would have to be minimal.
- Desired facility size—The institution's goals for a local facility may prevent the facility from being reduced to its economically optimal size.
- Entrepreneurial risk—The institution may not be willing to accept the entrepreneurial risk associated with installation of additional computing capacity and the sale of computing services to an external market.
- Capital investment—An institution may not be able to expand facilities to serve external users because of an inability to obtain capital funds.

EVOLUTION OF NETWORK COMPUTING FACILITIES

All of the factors discussed above appear to dictate against a significant change in the size of an institution's computing facility. However, there are ways around these problems. The following subsections suggest ways in which network computing facilities might evolve.

Evolution of major network suppliers

The establishment of a separate, wholly-owned corporation to serve network demands (when business exceeds the institution's growth limitations) is a likely mechanism for providing popular computer services to a national marketplace. Many institutions have indicated a desire to make their services available to other educational and research users. However, should these services prove very popular and should demand exceed what can conveniently be served, we have observed a willingness of institutions to consider setting up a separate organization to serve a national network community.

This mode of network facility development has already been observed in practice. For example, library cataloging services started out as small offerings on the host institution's computer facility. However, in the case of BALLOTS and OCLC, as the volumes of these services grew, they were spun off by their host institutions into separate service-providing organizations. Thus, the national network could evolve to a number of institutions providing minimum service volumes and a number of commercial organizations providing larger volumes of specialized services.

Evolution of major network buyers

The facility that is a successful network service buyer is likely to have evolved along one of two paths. On the one hand, it may have become a smoothly-operating, specialized facility that provides a limited range of services and imports the bulk of its requirements. The computing facility (hardware, software, and staff) would have become oriented toward providing a selective number of high quality, specialized services. Other services would be supplied to users via network purchases. The facility itself might be large or small, depending upon the volume of those specialized services it might sell to others.

On the other hand, the network buyer may have become a very small, limited computing facility. The institution may have reached this point either via a conscious decision to install limited local hardware capability, or by a decision to reduce the hardware configuration and scale of operation of an earlier facility. In either case network services would be used to enhance the breadth and cost-effectiveness of the computing services offered to the local user community.

COST CONSIDERATIONS

The availability of a wide array of computer services will have a number of cost-related impacts upon an institution and its internal user community. In particular, the benefits (costs) will accrue to (will be borne by) different portions of the institution. The following represent a sample of the considerations which an institution should evaluate in reaching a decision on network participation.

User cost savings

The previously-mentioned study by Heller⁴ indicated that the total cost to run a standard benchmark job at each of a number of institutions spanned a wide range. Variations of more than 10:1 were observed in job cost from one institution to another. This variation provides a significant opportunity for a user to "shop around" and obtain meaningful cost savings for his particular type of work.

On the other hand, if an institution has the capability and the capacity to do the work locally but the user takes his work elsewhere for processing, the total amount spent externally will be a direct loss of income for the institution. Note that the loss to the institution will exceed the user's savings. In order to maintain the same "net income" position, the institution must reduce the size of its own computing facility (e.g., hardware, software, staffing level reduction) and/or obtain a compensating volume of business from other external network sources. Thus, when evaluating the user benefits that might accrue, an institution must also consider the actions that it will have to take to compensate.

User cost increases

Computing is not a uniform commodity, so one cannot substitute "Brand A" for "Brand B" effortlessly. Therefore, in order to obtain the savings indicated by the job processing-cost differentials described above, it will be necessary for the user to investigate the relative costs and advantages/disadvantages of different facilities. In addition, when he makes a decision to use "Facility X," the user must learn the command language and control statements for that facility and learn to work with a remote user consulting organization.

All of these activities require an expenditure of the user's resources. The institution or the network may provide some assistance. For example, data on processing costs for common benchmark programs at all facilities might be provided, so that the user would have information on which to limit his field of investigation. However, learning to operate with a new facility is a cost the user must always bear. Even if this did not involve an accounting charge, there would still be the cost of lost time, personal effort, and energy. Thus, the potential cost savings must be reduced by these added costs.

Facility cost savings

The availability of network revenues may permit an institution's computer facility to expand to a more economic size and take advantage of economies of scale. This would permit unit cost savings to be passed on to all users, internal and external alike. Usage growth may also permit the facility to upgrade its support levels for certain services, or to support new services which were not economic at lower usage levels.

Similarly, the availability via the network of external service suppliers may also permit an institution's computer facility to obtain cost savings by not expanding (or by actually contracting) its scale of operation or its offerings. For example, a facility may be able to avoid a costly upgrade to new hardware by off-loading some of its growing local demand to outside service suppliers. Or a facility may be able to reduce its support costs by dropping services whose support is very costly or difficult to provide or whose use is infrequent. Such services can be eliminated locally, since they could still be obtained remotely via the network. Thus, a network connection offers a variety of cost saving alternatives.

Institutional benefits

The availability of specialized computer services via a network connection may enable an institution's research community to be more competitive in obtaining external research funding. That is, researchers might be able to undertake activities that had been foreclosed to them previously due to lack of proper computational facilities. Greater research breadth and facilities' availability may also enhance the institution's educational program, aid student and faculty recruitment, and enhance the institution's reputation. Further, to the degree that researchers are successful in attracting additional external research support, the indirect or overhead charges against these research contracts will make a positive contribution to the support of the institution's facilities and general operations.

PERSPECTIVES

The various types of economic impacts that have been discussed must be characterized as potential impacts; they might or might not occur. Their significance will depend upon the particular institution, the type of network, and the operating environment. However, each impact can potentially occur, so it needs to be considered explicitly.

On the other hand, reading the list of potential impacts gives an overall negative impression, for there are many things that can go wrong. Yet, many of these negative events are not likely to occur. Consider, for example, the long list of things that could go wrong with a jet airplane. Fortunately, most of the possibilities on that list never occur, and we continue to use commercial air transport safely. Therefore, let us consider what types of networking situations are

likely to occur, based upon the data provided by the participating institutions.

Initial network usage is likely to involve specialized services that are not available locally. (Such specialized services would include application packages, data bases, as well as "cheap raw cycles.") There is little expectation that major portions of an institution's internal users' workloads would be processed externally. Likewise, there is little expectation that any supplier would capture a major portion of another institution's processing load. There will, of course, be exceptions. For example, Harvard not too long ago gave up a major portion of its local computing capability, choosing to rely instead upon service purchased remotely from MIT. However, it is expected that network flows for exports and imports of computer services will be in the range of five percent to 15 percent of an institution's overall processing budget.

At this level of interaction, many of the potential funds flow problems discussed above become manageable. At worst, an institution might face a 15 percent reduction in business due to internal users going out on the network for service and no external users choosing to make use of the facility. While no one looks forward to a 15 percent budget cut, it is not likely to cause severe dislocations. The impact may be further softened, since many institution's workloads are growing and since network utilization is expected to phase in gradually.

Similarly, the other worst case (no internal users make use of network services but network users add 15 percent to the processing load on the facility) does not pose a significant danger. Most facilities either have some excess capacity available or have planned various small enhancements that will increase capacity at moderate cost. Thus, the increased demand should be accommodated without undue service deterioration, expense, or risk.

Pressures stemming from service-level comparisons (e.g., turnaround time, software quality) are likely to be present but muted due to the high conversion costs between facilities and the existence of user inertia. As long as these pressures remain moderate, they will exert a healthy influence on operations. However, unlike the other areas, this one has a much greater potential to become serious. If an institution is not operating close enough to the "customary" service levels, or if it is operating old, inefficient equipment, it might well be placed under rather severe competitive pressures.

The various constraints on a free-market network represent very real issues. Many institutions believe that these issues will not become serious so long as network business represents a "small enough" proportion of their business. On the other hand, network business amounting to 15 percent of revenues may not be "in the noise level," especially for facilities with multi-million dollar annual budgets. It is impossible to speculate as to the likely outcome. Impacts of some of these issues will depend upon the rulings and interpretations of local governmental jurisdictions; thus an inconsistent pattern of effects may be seen across the country. In other cases, similar situations have not arisen previously, so there are no precedents from which to extrapolate.

The constraints upon facility growth, while formidable

sounding, are not expected to be a severe limitation on network development. Most facilities are not expected to face significant growth problems. The few that might face very high external demand levels are likely to be those with a pre-existing entrepreneurial disposition, sites that would be willing to create a new business enterprise to provide the demanded services to the network user community. This has been an observed growth pattern in university computing operations, and administrators generally expect analogous behavior in a networking situation.

The cost savings and other benefits that have been described appear for the most part to be real. That is, it appears that users will be able to gain the benefits described within the bounds of a reasonable expenditure of effort. Thus, the promise of a national network in many respects appears realizable.

ACKNOWLEDGMENT

This work was sponsored by the National Science Foundation under Grant number MCS75-03634, "A Simulation and Gaming Project for Inter-Institutional Computer Networking." The opinions expressed are those of the author and do not necessarily reflect the views of the Foundation.

REFERENCES

1. Greenberger, M., J. Aronofsky, J. L. McKenney and W. F. Massey (eds.), *Networks for Research and Education: Sharing of Computer and Information Resources Nationwide*, MIT Press, Cambridge, Massachusetts, 1974.
2. Weingarten, F. W., N. R. Nielsen, J. R. Whiteley and G. P. Weeg, "A Study of Regional Computer Networks," University of Iowa Report, Iowa City, Iowa, 1973.
3. Berg, S. V., "Planning for Computer Networks: The Trade Analogy," *Management Science*, Vol. 21, No. 12, August 1975.
4. Heller, P., "Benchmarking the Price of Computing: Results of a Survey," *Computer Networks*, Vol. 1, No. 1, June 1976.
5. Report to the National Science Foundation of Year 1 of the Simulation and Gaming Project for Inter-Institutional Computer Networking, Grant DCR75-03634, EDUCOM, Princeton, New Jersey, July 1976.
6. Segal, R., and N. White, "Representation of Workloads in a Network Environment," *Proceedings of the Summer Computer Simulation Conference*, Washington, D.C., July 1976.
7. Segal, R., and N. White, "Management of a Large Computer Network Simulation Project," *Proceedings of the Fourth Annual Symposium on the Simulation of Computer Systems*, Boulder, Colorado, August, 1976.
8. Segal, R., and N. White, "A Simulation Model of a National Computer Resource-Sharing Network for Education and Research," *Proceedings of the Computer Networks Conference*, Gaithersburg, Maryland, November 1976.
9. Nielsen, N. R., and R. Segal, "Modeling an Inter-Institutional Computer Network," *EDUCOM Bulletin of the Interuniversity Communications Council*, Vol. 11, No. 4, Winter, 1976/77.
10. Nielsen, N. R., and R. Segal, "Simulation of Institutional Behavior in a National Networking Environment," *Proceedings of the Winter Simulation Conference*, Gaithersburg, Maryland, December 1976.
11. Emery, J. C., N. R. Nielsen, B. O'Neal and R. Segal, "A Simulation Model of a National Network for Education and Research," *Proceedings of the 15th Annual AEDS Convention*, Fort Worth, Texas, April 1977.
12. Final Report to the National Science Foundation of the Simulation and Gaming Project for Inter-Institutional Computer Networking, Grant MCS75-03634, EDUCOM, Princeton, New Jersey (to be published Spring 1979).

Approaches to concurrency control in distributed data base systems*

by PHILIP A. BERNSTEIN and NATHAN GOODMAN

Harvard University
Cambridge, Massachusetts

INTRODUCTION

Whenever multiple users or programs access a data base concurrently, the problem of concurrency control arises. The problem is to synchronize concurrent interactions so that each reads consistent data from the data base, writes consistent data, and is ultimately processed to completion. In a distributed data base this problem is exacerbated because a concurrency control mechanism at one site cannot instantaneously know about interactions at other sites. No fewer than 30 papers on this topic have appeared to date. Our purpose is to survey this literature, concentrating on three approaches—locking, majority consensus, and SDD-1 protocols—which together subsume the bulk of the literature.**

Distributed concurrency control is complex and our treatment is, of necessity, sketchy. We urge the interested reader to consult the source materials listed in the bibliography.

BACKGROUND

Preliminary definitions

A **distributed data base management system** (abbr. DDBMS) is a collection of sites interconnected by a network. Each *site* is a computer running a local (i.e. non-distributed) DBMS, and the *network* is any computer-to-computer communication system. We assume that sites are widely dispersed geographically, so the network must employ long-distance communication media. Consequently, inter-site communication is qualitatively slower and more costly than intra-site computation.

We define a *data base* to be a collection of data items. In practice, a *data item* may be a field, a record, a file, etc. This “level of granularity” is important, but does not impact concurrency control and so we leave it unspecified.

Each data item may be stored at any site in the system,

and moreover each may be stored *redundantly* at several sites. Redundant data improves the robustness and performance of a DDBMS and must be supported by general purpose systems. Unfortunately, it is also a major source of complexity. A stored copy of a data item is called a *stored data item*. Though it is impossible for all stored copies of a data item to be identical at every instant of time, it is essential that all “converge” to the same final value. We use the term *logical data item* when the distinction between “data item” and “stored data item” requires emphasis.

Users interact with the DDBMS by entering *transactions*, by which we mean a program or on-line query that accesses the data base. Transactions have two important properties in our model—(1) We assume they represent complete and correct computations: i.e. each transaction, if executed alone on an initially consistent data base, would terminate, output correct results, and leave the data base consistent. (2) We assume transactions obtain data from the data base by issuing Read commands to the DDBMS, and modify data by issuing Write commands. The arguments to these commands are *logical data items* and it is the responsibility of the DDBMS (a) to choose *one* stored copy of each data item for Reads, and (b) to update *all* stored copies of each data item for Writes. We model a transaction as a sequence of Read and Write operations paying no attention to its internal computations.

The *read-set* of a transaction is the set of logical data items it reads, and its *write-set* is the set of logical data items it writes. Two transactions *conflict* if the write-set of one intersects the read-set or write-set of the other. Similarly, two operations conflict if one is a Write and they operate on the same data. It is a fundamental theorem of concurrency control that two transactions require synchronization only if they conflict. (The converse need not be true, as we shall see in the fifth section).

Serializability

A *log* is a sequence of Reads and Writes. A log is *serial* if the Reads and Writes for each transaction are contiguous (see Figure 1). Such a log represents an execution in which no transactions execute concurrently. Since we assume each

* This work was supported by the National Science Foundation under Grant MCS-77-05314 and by the Advanced Research Projects Agency of the Department of Defense, contract number N00039-78-G-0020.

** References on these approaches are listed in the bibliography by topic. We will limit our use of in-text references in the interest of readability.

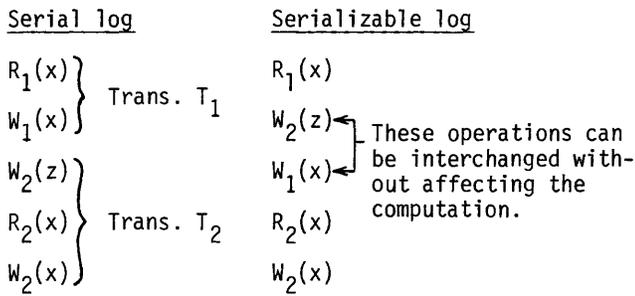


Figure 1—Serial and serializable logs.

transaction preserves consistency if executed alone, a serial sequence of transactions also preserves consistency. A log is *serializable* (abbr. *SR*) if it is "equivalent" to a serial log, meaning that for all initial data base states it produces the same output and the same final data base state as some serial log. Since serial logs preserve consistency, and SR logs are equivalent to serial logs, SR logs preserve consistency as well.

In a DDBMS, each site processes a different log. We define a *distributed log* (abbr. *dlog*) to be a set of logs, one per site. A *serial dlog* is a dlog in which each component log is serial and reflects the same total ordering of transactions (i.e., all transactions are in the same relative order in all logs in which they appear). A dlog is *serializable* if it is equivalent to a serial dlog.

Serializability has been adopted almost universally as the correctness criterion for DBMS concurrency control; all the approaches we describe follow this convention. Alternate correctness criteria are discussed in References 20 and 22.

Other aspects of concurrency control

In addition to ensuring serializability, a concurrency controller must guarantee *termination*; it must operate *robustly*; and it must operate *efficiently*.

A transaction may fail to terminate for one of three reasons—(1) *Deadlock* may occur, i.e. two or more operations might be forced to wait for each other. (2) Some operation may be *indefinitely postponed* by an unexpected conspiracy of events. Or (3) *Cyclic restart* might be experienced, meaning that the transaction repeatedly reaches a blocked state and is aborted and restarted. Every concurrency control approach is susceptible to some combination of these problems.

With respect to robustness, all approaches face essentially identical problems. We discuss this issue in the sixth section.

The efficiency of a distributed concurrency controller is determined principally by how much inter-site communication it requires. Typically, message delays in long distance networks are tenths of seconds, and network capacity is the scarcest system resource. In analyzing the performance of a controller, then, it is reasonable to study its communication behavior, and ignore other aspects. We compare the performance of various approaches in the Conclusion section.

DISTRIBUTED LOCKING ALGORITHMS

Locking is the most widely used concurrency control technique. We describe locking first in the centralized DBMS context and then present several extensions for distributed systems.

Centralized locking

Locking synchronizes transactions by explicitly detecting and preventing conflicts. When a transaction issues a Read or Write command, the DBMS attempts to "set a lock" on the desired data item; the lock is "granted" only if no other transaction holds a conflicting lock. If the lock is not granted, the requesting transaction waits until the lock is available and can be granted.

Since the DBMS processes all Read and Write commands from every transaction, it can automatically generate lock requests for each command. This is important because it allows programmers to ignore concurrency control issues when writing their transactions.

Eswaran et al.¹⁶ prove that locking is sufficient to ensure serializability provided no transaction requests new locks after releasing a lock. This usually amounts to having transactions hold all locks until they finish execution.

Since transactions are made to wait for unavailable locks, the possibility of deadlock exists (see Figure 2). Deadlocks can be detected by maintaining a *deadlock graph* in the DBMS. The nodes of the graph represent transactions and the arcs represent the "waiting-for" relationship; an arc is drawn from transaction T_i to transaction T_j if T_i is waiting

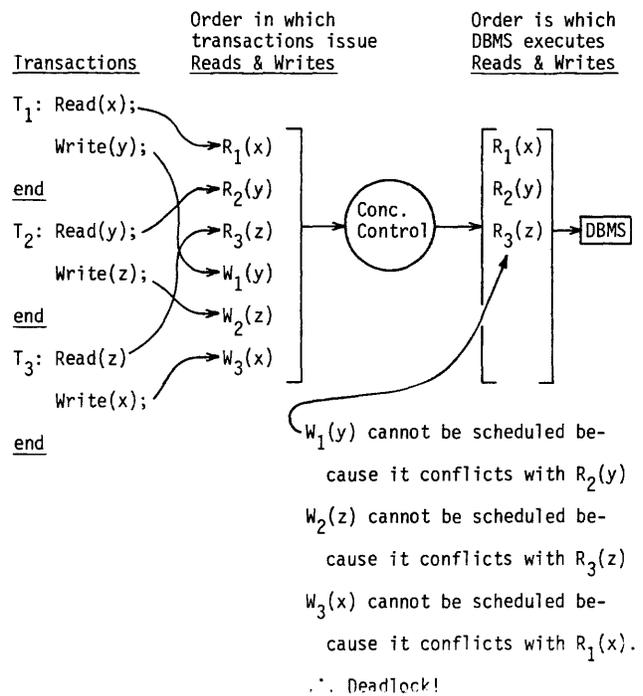


Figure 2—Deadlock

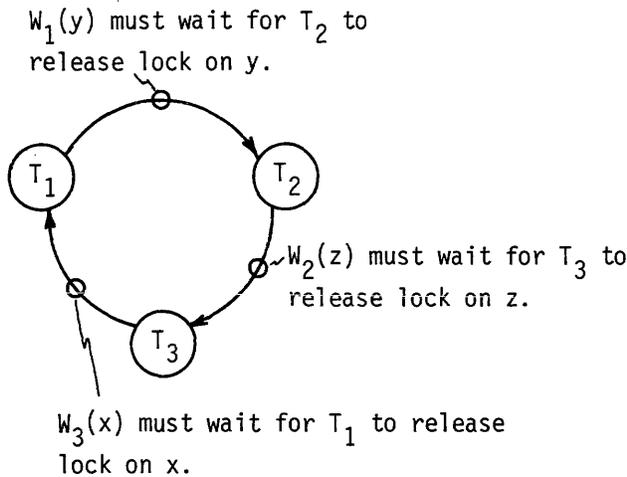


Figure 3—Deadlock graph for Figure 2.

for a lock held by T_j (see Figure 3). There is a deadlock in the system if and only if the deadlock graph has a cycle.⁵ If a deadlock exists, some transaction in the cycle is backed out and restarted. This deadlock elimination technique can potentially lead to cyclic restart. A simple way of avoiding this problem is to always abort the “youngest” transaction involved in the deadlock. Other solutions to cyclic restart are described in the section on Conflict-Driven Restarts.

Indefinite postponement can be prevented in a locking system by processing lock requests on a first-come-first-served basis. Other solutions are discussed in References 14 and 19.

Primary site locking

Primary site locking is a simple extension of centralized locking. One site of a DDBMS is designated to be the “primary site” and it manages all synchronization. When a transaction wishes to access data at any site, a lock is requested from the primary site. The primary site processes lock requests exactly as described in the previous section, the only difference being that lock requests come in over the network. Similarly, issues of termination are handled by the primary site exactly as in centralized locking.

Although locks are centralized at the primary site, the data base is, of course, distributed. Once a transaction is granted a lock, it may access data at whatever site has a copy. It is important that if a transaction updates a data item that has many stored copies all copies are actually updated before the lock is released; otherwise another transaction can read a copy of the data item before the first update propagated there. It is also important that read-only transactions follow the locking discipline, or else they could read inconsistent data (see Figure 4). This point is often overlooked in discussions of distributed locking, yet is important because most applications predominantly consist of read-only transactions.

The principal drawback of primary site locking is that the

primary site tends to be a bottleneck—the capacity of the primary site to process locks bounds the capacity of the entire distributed system.

Primary copy locking

Primary copy locking is an extension of primary site locking that eliminates the primary site bottleneck. For each logical data item, one copy is designated the “primary copy”; when a transaction wishes to access a data item, it locks the primary copy. Since the primary copies of different data items may be stored at different sites, no single site is primary in any sense. This eliminates the bottleneck, but introduces a new problem—deadlock detection.

To test for deadlock, all sites with some primary copy must participate. For example, Figure 5 illustrates a deadlock involving two sites which cannot be recognized locally by either site. The solution is to designate one site of the DDBMS as the “deadlock detector”; periodically each other site sends it a list of newly granted or released locks, and newly pending requests. The deadlock detector then operates as in the centralized case.

As with primary site locking, if a transaction writes into a data item, all copies must be updated before the lock is

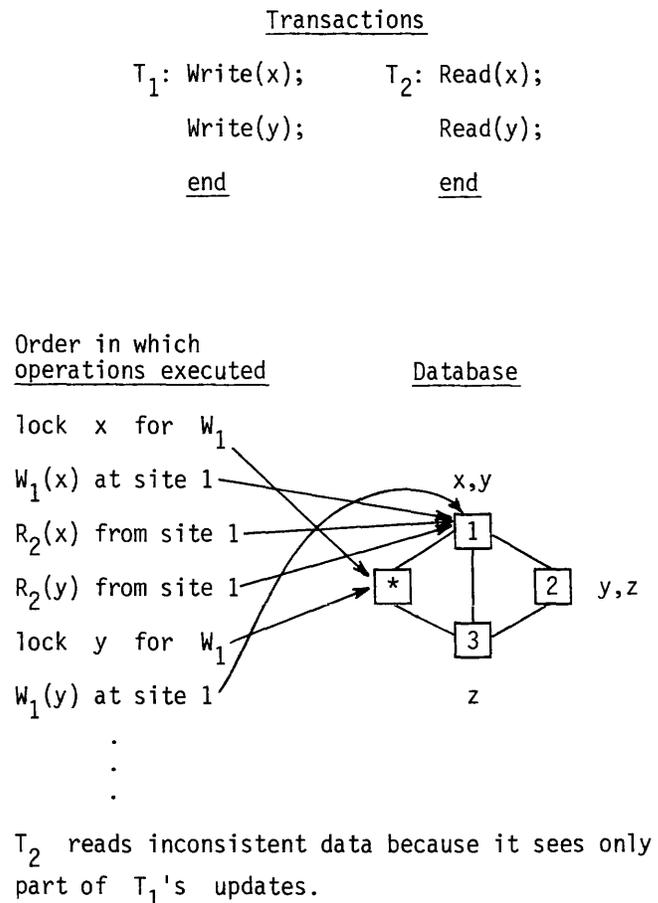


Figure 4—Read-only transactions must lock, too.

<u>Transactions</u>		
T_1 : Read(x);	T_2 : Read(y);	T_3 : Read(z);
Write(y);	Write(z);	Write(x);
<u>end</u>	<u>end</u>	<u>end</u>

Order in which locks are requested at each site

site 1

lock x for R_1

• lock x for W_3

site 2

lock y for R_2

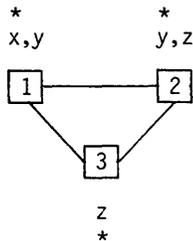
• lock y for W_1

site 3

lock z for R_3

• lock z for W_2

Database



*denotes primary copy

None of the •'ed locks can be granted, hence the system is in deadlock. But deadlock graphs at each site are acyclic:

site 1

site 2

site 3

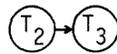
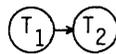
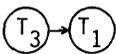


Figure 5—Multi-site deadlock.

released; and read-only transactions must follow the locking rules, too.

Conflict-driven restarts

An interesting variant of primary copy locking has been described by Rosenkrantz, Lewis and Stearns.³³ The mechanism, which we call *conflict-driven restart*, uses a model of transaction execution in which each transaction is active at only one site at a time and moves from site to site during its execution. When a transaction wishes to access a data item at a site, it tests whether it conflicts with a previous access made by an in-progress transaction. If it does conflict, one of three actions is possible—it waits, it is restarted, or the other transaction is restarted. Since testing for conflict

is equivalent to asking whether a data item is *locked*, this approach is essentially a locking mechanism.¹²

The analysis of conflict-driven restart yields interesting observations about termination problems. If the system responds to conflict by making the requesting transaction wait, deadlock is possible. To avoid deadlock Rosenkrantz et al. propose two mechanisms that substitute *restarts* for waiting. Both mechanisms require that transactions be assigned unique "timestamps" when they are submitted. Intuitively, timestamps correspond to the time a transaction was submitted, and have two important properties—timestamps assigned at any particular site must strictly increase with time, and timestamps assigned at different sites must be different. Timestamps are used to resolve conflicts as follows. In one mechanism, called the *Wait-Die System*, the requesting transaction waits if it has a smaller timestamp (i.e. is older); else it is restarted. In the second mechanism, called the *Wound-Wait System*, the requesting transaction waits if it has a larger timestamp (i.e., is younger); else the transaction it conflicts with is restarted.

Rosenkrantz et al. prove that both mechanisms avoid cyclic restart, but the details of their behavior is quite different. In a Wound-Wait system, an old transaction may be restarted many times, whereas in a Wait-Die system old transactions are never restarted. It is suggested that Wound-Wait produces fewer overall restarts, but the justification is more intuitive than analytic.

MAJORITY CONSENSUS ALGORITHM

The majority consensus algorithm of R. Thomas³⁷ was one of the first distributed concurrency control mechanisms proposed. Many of Thomas's ideas have found their way into more recent designs.

The majority consensus algorithm as presented by Thomas assumes a fully redundant data base, meaning that every site has a stored copy of every logical data item in the data base. A transaction executes at one site. Its Read commands access stored data at its site, and do so *without* locking or any other synchronization. Whenever the transaction issues a Write command, the name of the data item being updated and its new value are recorded in an *update list*; the data base itself is *not* modified at this time. When the transaction completes, the update list is sent to all sites and each site votes on it. If a majority of the sites vote "Yes," the transaction is *accepted*, and the updates are installed at all sites. Otherwise the transaction is restarted. The heart of the algorithm is the rules that determine how each site votes.

A site votes "Yes" on transaction T if

1. The data items read by T have not been modified since T read them (the algorithm requires that a data item must be read before it can be written).
2. T does not conflict with any transaction T' that is *pending* at the site (T' is *pending* if the site has voted "Yes" but T' has not yet been accepted or rejected system-wide).

One way to meet Condition 1 is to use locking; but the majority consensus algorithm uses a timestamping technique instead.

Transactions are assigned timestamps as in "conflict driven restart," and each stored data item is tagged with the timestamp of the most recent transaction that has updated it. Also, update lists are augmented to include the name of each data item read by the transaction and its timestamp. Now, when a site receives an update list it can compare timestamps to determine whether Condition 1 holds. Since augmented updated lists specify transactions' read-sets and write-sets, Condition 2 is easily checked as well.

If Condition 1 is not satisfied, the site "vetoes" the transaction and it is restarted. If (1) is satisfied but (2) is not, the site cannot vote on this transaction until the pending one is resolved. Since different sites receive update lists in different orders, they vote in different orders and deadlock could result. To avoid deadlock, the site votes "No" if (1) holds, (2) does not hold, and the transaction has a larger timestamp (i.e. is younger) than the pending one. If a majority of sites vote "No," the transaction is restarted.

The voting rules ensure that two conflicting transactions are both accepted only if one has read the other's output. Since both transactions received a majority of "Yes" votes, some site, say *S*, must have voted "Yes" on both transactions. Since they conflict, *S* must have installed one before voting on the other; this guarantees that the second read the first one's output, for otherwise *S* would not have voted "Yes." This is sufficient to guarantee serializability.

THE SDD-1 APPROACH

The SDD-1 DDBMS³⁹⁻⁴² employs a qualitatively different approach to concurrency control. Each of the preceding methods synchronizes *all* conflicting Reads and Writes. However, not all conflicts can violate serializability (see Figure 6). SDD-1 exploits this fact by means of two mechanisms—*conflict graph analysis*, and *timestamp-based protocols*.

Conflict graph analysis

Conflict graph analysis is a technique for determining which conflicts require synchronization. The method begins with the definition of *transaction classes*. A transaction class is defined by a read-set and write-set. A transaction is a member of a class if the transaction's read-set and write-set are contained in the class's read-set and write-set (respectively). Associated with each class is a *transaction module* (abbr. TM), a software DBMS component that serially processes transactions from that class. Since transactions in a single class run serially, only transactions in different classes can "interfere." Hence, only inter-class conflicts need be considered.

Due to the way classes are defined, transactions in different classes can conflict only if their corresponding classes conflict. Class conflicts are modelled by an undirected *con-*

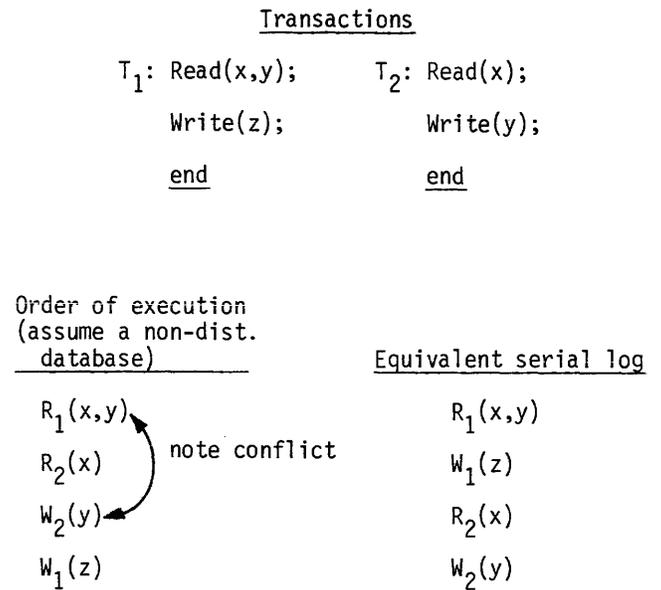


Figure 6—A conflict that does not violate serializability.

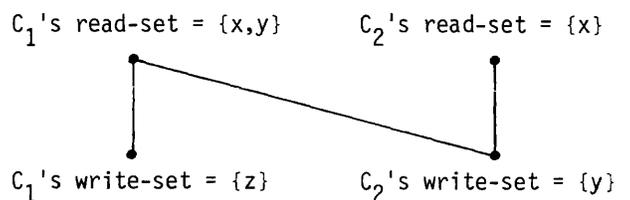
conflict graph whose nodes represent class read-sets and write-sets, and whose edges represent conflicts. (There is also an edge between the read-set and write-set of each individual class. See Figure 7.) The important property of a conflict graph is that transactions that do not lie on a cycle are always serializable and do not need synchronization. Only transactions that lie on cycles require synchronization.

In a conflict graph system, the conflict graph is constructed and analyzed statically when the data base and classes are defined. Classes that do not lie on cycles are noted; the TMs corresponding to these classes are 'told' not

Class definitions:

- C_1 : read-set = { x,y }, write-set = { z }
- C_2 : read-set = { x }, write-set = { y }

Conflict graph



Note: Transactions T_1 and T_2 in figure 6 are in classes C_1 and C_2 resp. Since the conflict graph is acyclic, their conflict cannot violate serializability (see text).

Figure 7—Conflict graph.

to use synchronization when executing transactions. The remaining TMs must synchronize their transactions.

Locking is one correct way to synchronize transactions that lie on cycles. If all transactions on cycles use locking, all executions are serializable.¹² However, other synchronization mechanisms are possible.

Timestamp-based protocols

SDD-1 uses timestamp-based synchronization protocols in place of locking. While the details of these protocols are too involved for this paper, their basic structure can be sketched.

Each edge from one class's read-set to another's write-set represents a conflict that must be synchronized if the edge lies in a cycle. This synchronization occurs during the processing of Read commands. Suppose the read-set of class i conflicts with the write-set of class j , and suppose T_i is a transaction in class i . To process a Read for T_i at site S , the concurrency controller waits until S has processed all Writes for transactions in class j that are "older" than T_i ,*** but no Writes for transactions in class j that are "younger" than T_i . This has the same effect as locking the data shared by i 's read-set and j 's write-set.

The advantage of timestamp-based protocols lies in the wide range of protocols that can be used. There is a special protocol for read-only transactions which is more efficient than locking. There is a special protocol for infrequently run transactions that places a heavier synchronization burden on these transactions while reducing the synchronization required for common transactions. In addition, all protocols use timestamps to resolve conflicts, so deadlock is prevented without the overhead of a detection algorithm.

The correctness of conflict graph analysis and the SDD-1 protocols is proved in Reference 41.

ROBUSTNESS

Component failures are inevitable in a DDBMS, and any practical concurrency controller must operate correctly despite them. Problems of three types arise—(1) A failed site may hold information needed to synchronize in-progress transactions. (2) A failed site may hold stored copies of data items being updated by a transaction. (3) A transaction that is updating data at several sites may fail after performing some updates but not all of them. No mechanism yet developed attains 100 percent robustness and it is believed that no such mechanism is possible.⁴ Given this apparent limitation, one cannot prove that a concurrency controller is, or is not, robust; all one can do is express the *level* of robustness it attains.

*** SSD-1 assigns unique timestamps to transactions in the same manner as majority consensus.

Loss of synchronization information

When a site holding synchronization information fails, there are two options. One is to *abort* all in-progress transactions that depend on the information. If transactions are short and failures occur infrequently, this simple approach is satisfactory. The alternative is to maintain redundant copies of synchronization information. Techniques for managing these redundant copies have been proposed by Alsberg and Day²⁶ and Menasce et al.³¹ The techniques are presented in the context of primary site or primary copy locking, but could be adapted for other approaches.

The Alsberg and Day technique employs a *back-up* for each primary copy. When transaction T wishes to access data item x , it requests a lock against the primary copy as described in the section on Primary Copy Locking. If the concurrency controller decides to grant the lock, it forwards this information to the back-up, which records the lock in memory. Only when the lock is safely recorded at the back-up is transaction T permitted to access x . If the site containing the primary copy fails, the back-up can immediately take over and a new back-up selected. This scheme offers 100-percent protection against single-site failures, but of course is susceptible to multi-site failures. Protection against multiple failures can be improved by using multiple back-ups, although Alsberg et al.²⁷ argue that one back-up is sufficient for most applications.

Menasce et al.³¹ propose a similar mechanism designed for multiple back-ups. The heart of their approach is a communication procedure for ensuring that all locks are received by all back-ups, and a procedure for reconstructing consistent lock tables following site failures.

Non-availability of stored data items

Suppose a transaction issues a Write command against a logical data item x , and some stored copy of x is unavailable. Since the DDBMS must update *all* stored copies of x , we have a problem. The DDBMS could delay the Write until all stored copies were simultaneously available, but this might never happen. Or it could abort the transaction, but then the availability of a data item would *decrease* as more copies of it were maintained. The solution is to buffer Write operations against non-available sites and to perform them when the failed site recovers. By buffering the Writes at multiple sites, increased protection against multiple failures can be achieved. This technique is sometimes called *spooling*.⁴⁷

Transaction failures

If a transaction fails before completion, a serious concurrency control problem is created—every Write performed by the transaction must be backed out to avoid leaving partial results in the database. The usual technique for doing this is called *two phase commit*.⁴

While a transaction executes, all Writes it performs are placed in temporary files and not the permanent data base. When the transaction completes, it issues Commit messages to each site holding temporary files, whereupon the temporary file is merged into the permanent data base. If the transaction fails before sending the first Commit, no updates are installed. If it fails after sending some but not all Commits, the sites holding temporary files can recognize the situation and can consult the other sites. If any site did receive a Commit, all sites will perform the update.

This technique achieves 100-percent protection against failures of the transaction alone, but is not fully robust with respect to multi-site failures. Hammer and Shipman⁴⁷ describe mechanisms for improving the multi-site robustness of two phase commit.

CONCLUSION

We have presented several approaches to distributed concurrency control, and the obvious question is, "Which one is best?" We have no clear answer to this question, but a comparison of the methods may be helpful.

First, all methods presented here are *correct*—they all guarantee serializable executions. Second, the methods offer slightly different degrees of concurrency—conflict-driven restart and majority consensus offer slightly less concurrency than conventional locking; conflict graph analysis combined with locking offers slightly more concurrency than conventional locking; and conflict graph analysis coupled with SDD-1 timestamp protocols offers an "incomparable" degree of concurrency, meaning it allows some executions the other techniques prohibit, while prohibiting some executions the others allow. Termination issues are best understood in the context of locking, and locking is the only technique for which termination can be proved. Majority consensus is susceptible to cyclic restart, and conflict graph analysis coupled with SDD-1 protocols can lead to indefinite postponement; in practice, however, the probability of non-termination can be made acceptably small. With respect to robustness, all approaches share the same problems, and the same techniques. So the approaches compare almost identically on these four issues, at least.

The remaining area of comparison is *performance*. As explained in the section on Other Aspects of Concurrency Control, the key determinant of performance is communications behavior. Unfortunately, few quantitative performance results are available and we shall limit ourselves to basic observations.

Primary site locking requires inter-site communication whenever a lock is requested or released. In principle, primary copy locking could require the same amount of communication, but in a well configured system we would expect to do better. The reason is *locality of reference*—in many distributed applications, the majority of transactions access data *local* to the site at which they run. If that data is the primary copy, all lock requests can be processed locally. Of

course, this advantage cannot be realized for data items that are heavily accessed from multiple sites.

The performance of the SDD-1 technique similarly depends on application-specific factors. If many transactions run in classes that do not require synchronization, the system will require few synchronization messages. Since the class definitions are tunable, classes can (in principle) be designed so that frequently-executed classes do not lie on cycles. However, if most transactions run in classes that do require synchronization, the communication overhead involved will be comparable to locking.

The performance of majority consensus is comparable to primary site locking with these differences—all locks are in effect requested in a single message; in return for this savings, though, multiple restarts may have to be endured.

The material we have presented on each approach is a bare outline. We have left out many important details and variations, and we urge the interested reader to consult source materials directly.

ACKNOWLEDGMENTS

We would like to thank Clarice Louis for editorial assistance in preparing this paper. We also thank S. Kimbleton and R. Muntz for prodding us to produce this tutorial survey.

BIBLIOGRAPHY

Note: References are listed by topic, and many do not appear in the text.

General

1. Aho, A., J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1975.
2. Date, C. J., *An Introduction to Database Systems*, 2nd ed., Addison-Wesley, Reading, Mass., 1977.
3. Deppe, M. E. and J. P. Fry, "Distributed Databases: A Summary of Research," *Computer Networks*, Vol. 1, No. 2, North-Holland, Amsterdam, Sept. 1976.
4. Gray, J. N., *Notes on Database Operating Systems*, unpublished lecture notes, IBM San Jose Research Laboratory, San Jose, Calif., 1977.
5. King, P. F., and A. J. Collmeyer, "Database Sharing—An Efficient Mechanism for Supporting Concurrent Processes," *Proc. 1974 NCC*, AFIPS Press, Montvale, New Jersey, 1974.
6. Martin, J., *Computer Database Organization*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975.
7. Martin, J., *Principles of Database Management*, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.
8. Peebles, R. et al., "System Architecture for Distributed Data Management," *Computer*, Vol. 11, No. 1, Jan. 1978.
9. Rothnie, J. B., Jr., and N. Goodman, "A Survey of Research and Development in Distributed Database Management," *Proc. Third Int. Conf. on Very Large Databases*, IEEE, 1977.
10. Rothnie, J. B., Jr., N. Goodman and T. Marill, "Database Management in Distributed Networks" in F. F. Kuo (ed.), *Protocols and Techniques for Data Communication Networks*, Prentice-Hall, Englewood Cliffs, N.J., 1978.
11. Stonebraker, M., and E. Neuhold, "A Distributed Database Version of INGRES," *Proc. 2nd Berkeley Workshop on Distributed Databases and Computer Networks*, May, 1977.

Concurrency control

12. Bernstein, P. A., D. W. Shipman and W. S. Wong, "A Formal Model of Concurrency Control Mechanisms for Database Systems," *IEEE Trans. on Software Engineering*, to appear.
13. Chamberlin, D. D., R. F. Boyce and I. L. Traiger, "A Deadlock-Free Scheme for Resource Allocation in a Database Environment," *Info. Proc. 74*, North-Holland, Amsterdam, 1974.
14. Courtois, P. J., F. Heymans and D. L. Parnas, "Concurrent Control with Readers and Writers," *Comm. ACM*, Vol. 14, No. 10, Oct. 1971.
15. Everest, G. C., "Concurrent Update Control and Database Integrity," in *Database Management*, J. W. Klimbie and K. L. Koffeman (eds.), North-Holland, Amsterdam, 1974.
16. Eswaran, K. P., J. N. Gray, R. A. Lorie and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," *Comm. ACM*, Vol. 19, No. 11, Nov. 1976.
17. Garcia-Molina, H., "Performance Comparisons of Two Update Algorithms for Distributed Databases," *Proc. 3rd Berkeley Workshop on Distributed Databases and Computer Networks*, August 1978.
18. Gelenbe, E., and K. Sevcik, "Analysis of Update Synchronization for Multiple Copy Databases," *Proc. 3rd Berkeley Workshop on Distributed Databases and Computer Networks*, August 1978.
19. Greif, I., "Formal Problem Specifications for Readers and Writers Scheduling," *Proc. MRI Symp. on Computer Software Engineering*, Polytechnic Inst. of N.Y., 1976.
20. Gray, J. N., R. A. Lorie, G. R. Putzulo and I. L. Traiger, "Granularity of Locks and Degrees of Consistency in a Shared Database," IBM Research Report RJ1654, September 1975.
21. Lamport, L., *Time, Clocks, and the Ordering of Events in a Distributed System*, Massachusetts Computer Associates, CA-7603-2911, Wakefield, Mass., March 1976.
22. Lamport, L., *Towards a Theory of Correctness of Multi-User Database Systems*, Massachusetts Computer Associates, CA-7610-0712, October 1976.
23. Papadimitriou, C. H., P. A. Bernstein and J. B. Rothnie, Jr., "Some Computational Problems Related to Database Concurrency Control," *Proc. Conf. on Theoretical Computer Science*, Waterloo, August 1977.
24. Papadimitriou, C. H., *Serializability of Concurrent Updates*, TR-14-78, Center for Research in Computing Technology, Harvard University, Cambridge, Mass., 1978.
25. Stearns, R. C., P. M. Lewis and D. J. Rosenkrantz, "Concurrency Control for Database Systems," *Proc. Conf. on Foundations of Computer Science*, 1976.
33. Rosenkrantz, D. J., R. E. Stearns and P. M. Lewis II, "System Level Concurrency Control for Distributed Database Systems," *ACM Trans. on Database Systems*, Vol. 3, No. 2, June 1978.
34. Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," *Proc. 3rd Int. Workshop on Distributed Databases and Computer Networks*, August 1978.
35. Stucki, M. J., J. R. Cox, Jr., G. C. Roman and P. N. Turcu, "Coordinating Concurrent Access in a Distributed Database Architecture," *Proc. 4th Workshop on Computer Architecture for Non-Numeric Processing*, August 1978.
See also Reference 4.

Majority consensus

36. Johnson, P. R., and R. H. Thomas, "The Maintenance of Duplicate Databases," Network Working Group RFC#677 NIC #31507, January 27, 1975.
37. Thomas, R. H., *A Solution to the Update Problem for Multiple Copy Databases which Uses Distributed Control*, BBN Report No. 3340, Bolt, Beranek & Newman, Cambridge, Mass., July 1975.
38. Thomas, R. H., "A Solution to the Concurrency Control Problem for Multiple Copy Databases," *Proc. COMPCON 1978*, IEEE, N.Y.

SDD-1

39. Bernstein, P. A., J. B. Rothnie, N. Goodman and C. H. Papadimitriou, "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (the Fully Redundant Case)," *IEEE Trans. on Software Engineering*, Vol. SE-4, No. 3 (May 1978).
40. Bernstein, P. A., D. W. Shipman, J. B. Rothnie and N. Goodman, *The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases*, Tech. Rep. CCA-77-09, Computer Corp. of America, Cambridge, Mass., December 1977.
41. Bernstein, P. A., and D. W. Shipman, *New Analysis of Concurrency Control in SDD-1: A System for Distributed Databases*, Tech. Rep. CCA-78-08, Computer Corp. of America, Cambridge, Mass., June 1978.
42. Rothnie, J. B., and N. Goodman, "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases," *Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks*, May 1977, pp. 39-57.

Other

26. Alsberg, P. A., and J. D. Day, "A Principle of Resilient Sharing of Distributed Resources," *Proc. 2nd Int. Conf. on Software Engineering*, October 1976.
27. Alsberg, P. A., G. G. Belford, J. D. Day and E. Grapa, "Multi-Copy Resiliency Techniques," Center for Advanced Computation, AC Document No. 202, University of Illinois at Urbana-Champaign, May 1976.
28. Ellis, C. A., "A Robust Algorithm for Updating Duplicate Databases," *Proc. 2nd Berkeley Workshop on Distributed Databases and Computer Networks*, May 1977.
29. Gouda, M. G., "A Hierarchical Controller for Concurrent Accessing of Distributed Databases," *Proc. 4th Workshop on Computer Architecture for Non-Numeric Processing*, August 1978.
30. Menasce, D. A., and R. R. Muntz, "Locking and Deadlock Detection in Distributed Databases," *Proc. 3rd Berkeley Workshop on Distributed Databases and Computer Networks*, August 1978.
31. Menasce, D. A., G. J. Popek, and R. R. Muntz, "A Locking Protocol for Resource Coordination in Distributed Databases," 1978 ACM Int. Conf. on Management of Data, May 1978.
32. Minoura, T., "Maximally Concurrent Transaction Processing," *Proc. 3rd Berkeley Workshop on Distributed Databases and Computer Networks*, August 1978.
43. Badal, D. Z., and G. J. Popek, "A Proposal for Distributed Concurrency Control for Partially Redundant Distributed Data Base System," *Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks*, 1978, pp. 273-288.
44. LeLann, G., "Algorithms for Distributed Data-Sharing Systems Which Use Tickets," *Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks*, 1978, pp. 259-272.

Robustness

45. Belford, G. C., P. M. Schwartz and S. Sluizer, *The Effect of Back-up Strategy on Database Availability*, CAC Document No. 181, CCTC WAD Document No. 5515, Center for Advanced Computation, University of Illinois at Urbana-Champaign, February 1976.
46. Lampion, B. and H. Sturgis, *Crash Recovery in a Distributed Data Storage System*, Tech. Report, Computer Science Laboratory, Xerox Palo Alto Research Center, Palo Alto, Calif., 1976.
47. Hammer, M. M., and D. W. Shipman, "An Overview of Reliability Mechanisms for a Distributed Data Base System," *Proc. 1977 COMPCON*, IEEE, N.Y.
See also References 4, 26, 27 and 31.

Access control mechanisms for a network operating system*

by HELEN M. WOOD and STEPHEN R. KIMBLETON

National Bureau of Standards
Washington, D.C.

INTRODUCTION

The growing recognition of the need for computer system security has resulted in the design, development and installation of "patches," packages and even new operating systems intended to provide higher degrees of data and systems protection. With the increased utilization of computer networks and current developments in the area of network operating systems (NOSs)^{40,19,30} the requirements for security in networking environments are also coming under investigation.^{26,42} While research and development are still ongoing in the NOS area, it is vital to ensure that requirements for the security and integrity of data are well specified and that mechanisms for achieving the needed levels of systems protection are included in the design of the NOS. This will ensure that subsequent production versions of NOSs incorporate such mechanisms—thus, charting a course away from the otherwise inevitable "retrofit security" situation.

Military applications have been the source of much of the computer and communications security knowledge. However, what is cost-effective in enhancing systems security for a military application may impose an untenable burden on a commercial or public system. To assist the manager/analyst in identifying appropriate security mechanisms, this paper identifies a set of access control capabilities which should be considered for inclusion within a general purpose NOS. Before identifying the actual mechanisms involved in enhancing NOS security, an NOS environment is briefly described. An overview of computer network security requirements is then provided and suggested approaches are referenced.

The second section identifies the specific access control functions required by the type of network operating system described in the first section. The implementation of these access control mechanisms in the NBS Experimental Net-

work Operating System (XNOS) is presented in the third section, followed in the fourth section by a discussion of current status and future plans.

The NBS experimental network operating system

Network operating systems are commonly viewed as the mechanism for masking system differences from users. The functional objective of a NOS is to support and simplify access to existing services and to expedite the construction and subsequent accessing of new services by simplifying interaction among systems and between systems and users. A major design goal for implementing a NOS on an existing computer network is that the NOS is transparent to the participating host systems. This goal is achievable through a consolidation of NOS support functions into a Network Interface Machine (NIM), as suggested by Kimbleton.²⁹

The National Bureau of Standards has developed an Experimental Network Operating System, XNOS, to demonstrate the feasibility of such general purpose NOSs and to facilitate the investigation of the capabilities and limitations inherent in such systems.³⁰ Figure 1 illustrates the user view of the network, while Figure 2 identifies the current XNOS configuration.

The major computing requirements of XNOS are fulfilled by an XNOS Network Interface Machine (XNIM). The XNIM is, in fact, the focal point for user-system and system-system interactions. It serves, among other things, as a translator for commands (e.g., MOVE <file>, DELETE <file>) and a transformer for data flowing between network processes. The first role provides the XNOS user with a standardized view of network resources by supporting a common command language for all participating hosts.¹⁸ The latter capability, termed Remote Record Access (RRA), provides (1) the means for transmitting data between systems, and (2) the mechanisms for accessing and preserving the meaning of structured data as it is transmitted across heterogeneous systems.^{46,47}

While the remainder of this paper will discuss access controls within the context of the NBS XNOS implementation, it should be noted that the functionality of the solution approach applies to the general class of NOSs represented by the NBS system.

* This work is a contribution of the National Bureau of Standards and is not subject to copyright. Partial funding for this work was provided by the U.S. Air Force Rome Air Development Center (RADC) under Contract No. F 30602-77-F-0068. Certain commercial products are identified in this paper in order to adequately specify the procedures being described. In no case does such identification imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the material identified is necessarily the best available for the purpose.

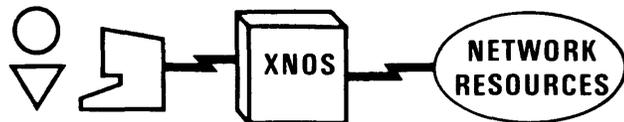


Figure 1—User view of network.

Network security requirements

The growth of time-sharing and other forms of computer networking have increased the vulnerability of data in computer systems. Not only are more systems being accessed by remote users, but large amounts of data are shipped between systems.

Systems without adequate protection mechanisms are vulnerable to threats including theft, fraud and vandalism. Potential losses range from unauthorized use of computing time to the unauthorized access, modification, or destruction of valuable data. Perpetrators of such abuse may be otherwise honest individuals wishing to play a few computer games, or sophisticated corporate spies, hoping to learn trade secrets or perhaps acquire the list of a competitor's top ten accounts. (Computer crime is extensively treated in the literature.)³³⁻³⁶

In order to appreciate networking-specific security needs, it is useful to identify the relevant components of a network and their vulnerabilities. For these purposes, a network may be defined as consisting of three parts—host system(s), data communications equipment (e.g., communications nodes and links—often termed "communications subnet") and terminal systems.

At the terminal equipment level, certain security techniques such as controlled access to terminal rooms or the incorporation of an identification signal within each terminal may be useful in "securing" a network, but certainly are not adequate in themselves. It is crucial, however, to know who the person using any particular terminal is in order to determine that person's rights and privileges on the network. The process of verifying the claimed identity of a user is termed user authentication.

Access control is the cornerstone of computer network security. It includes not only authentication, but also authorization—the granting of the right to access an object to a user. An object may be a person, program, or process. A user is either a person or a program (process) working for the benefit of that user.

When data communications facilities are involved, as in

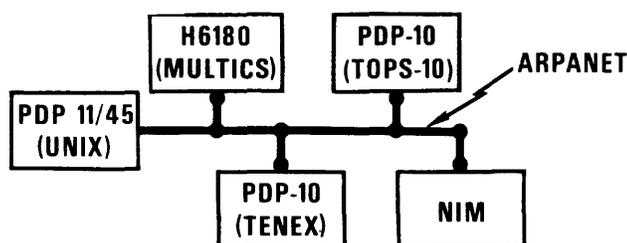


Figure 2—XNOS initial configuration.

the case of a network, the vulnerability of data and systems is greatly increased. Categories of threats in a communications environment include passive and active wiretapping, between-lines entry and piggy-back infiltration. Petersen and Turn describe a number of such threats.³⁷ Encryption is one of the techniques used to provide protection in a communications environment. It may be applied in a link-oriented or end-to-end fashion.

The federal government has adopted a Data Encryption Standard (DES) for use in the protection of transmitted data.¹⁶ Numerous works discuss such modern encryption techniques.^{6,20,12-14,3} Karger identifies some of the limitations of end-to-end encryption and Kent presents protocols for the protection of interactive communications.²⁶⁻²⁸ Included are techniques for key distribution and resynchronization following channel disruption.

There are many aspects to computer and network security. We have highlighted several security methods and techniques in an effort to establish a framework for a discussion of NOS access control mechanisms. This is certainly not, however, a comprehensive treatment of computer security. Numerous excellent guides into this area are available (e.g., References 8, 42, 41, 24, 7, 31 and 2).

Network security center approach

Before beginning the discussion of NOS access control functions, we shall briefly describe one example of a suggested approach to computer network security. This approach, the Network Security Center (NSC), is of special interest here because of the possibilities suggested by the common functions that an NSC and a NOS with access control capabilities share, as well as the powerful system that would result from a combination of NSC and NIM functions.

The NSC utilizes one or more dedicated minicomputers, termed Network Security Centers (NSCs), for validating user identity (authentication) and to some extent checking user access rights (authorization) to network resources. This approach, which has been extensively treated elsewhere, is briefly summarized in the remainder of this section.^{4,5,22,9}

The NSC is responsible for the following functions: (1) authentication, (2) authorization, (3) establishment of a secure communications path for user access to network resources and (4) the collection and/or distribution of appropriate information relative to this connection (e.g., audit data collected, user profile information supplied to host). Fundamental to the concept of a NSC is the assumption that a highly secure encryption algorithm, such as the DES, will be used to protect all information transmitted between participating systems. Such algorithms employ both enciphering and deciphering operations which are based on a binary number called a key. The key consists of a string of binary digits used directly by the algorithm. A unique key chosen for use in a particular application makes the results of encrypting data using the algorithm unique.

Intelligent Cryptographic Devices (ICDs) must exist at a level below the NSC. An ICD is used to establish a protected

connection between two network entities after authentication and authorization checking have taken place. This ICD must be capable of being remotely keyed—by the NSC only. Furthermore, whenever a dialog is completed the corresponding connection must be broken to ensure that other users do not have an opportunity to “piggy-back” an authorized user and thus gain access to restricted resources. Figure 3 presents a simplified view of a network incorporating a Network Security Center.

In order for a user to gain access to a computer on a network which incorporates an NSC, the following steps must occur:

1. The user initially connects to the NSC.
2. The NSC performs authentication and authorization checking.
3. After validation of the user's claimed identity and successful authorization check, NSC then “keys” the ICD for the user's device (e.g., terminal) and the target host with a “one-time” session key.
4. The user and target host then communicate directly over encrypted lines.

The protocol for establishing secure communications links in an architecture incorporating both XNIM and NSC would be somewhat different in that two or more host-to-host connections would be required for the same work session.

NOS SECURITY REQUIREMENTS

In order to provide data protection in a NOS environment, one must first support overall network security, as discussed in the preceding section. The requirement that a NOS be nearly transparent to the host precludes any extensive development of host specific, access control software for the purpose of improving NOS security. The focal point for NOS security, then, becomes the NOS support computer—the NIM.

Achieving/enhancing security in a NOS environment through use of a NIM gives rise to two issues—(1) determination of the appropriate extent or degree of NIM security services and (2) identification of the security capabilities/services that a NIM can provide.

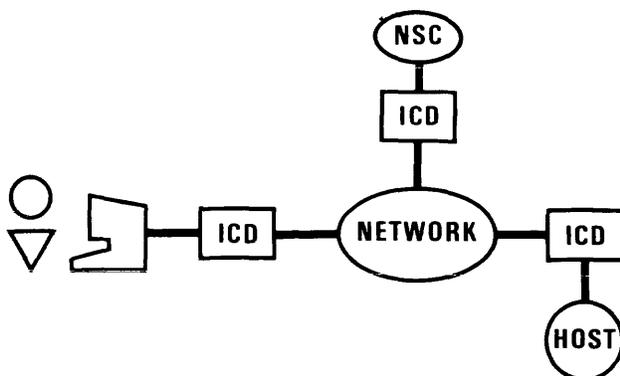


Figure 3—Network security center configuration.

Level of security

Increasing the effectiveness of one component in protecting the system usually results in diverting an attacker's penetration efforts to another, more vulnerable, system component. Since the NIM is the component which changes an “ordinary” computer network into a NOS, it is vital that the NIM does not become the “weak link” in the security structure of the net. This implies that the level of security provided by an NIM should not be less than that of any other network component.

As no capability comes without cost, it would not be cost-effective to provide security “overkill” and far exceed the highest degree of protection offered elsewhere in the network. Thus, an appropriate level of NIM security support would appear to be a level equal to or *slightly* greater than that of the most secure of the remaining network components.

Support computer security functions

A NOS provides an opportunity to enhance the security of a computer communications network. This is primarily because, as noted by Linden, “a large class of security problems can be solved by putting a level of indirection between a subject and the object it is seeking to access.”²⁵ Within the category of NOSs being considered, the NIM acts as an intermediary between users and systems and systems and systems. Thus, the NIM can take advantage of its relationship with network hosts by properly utilizing the authentication mechanisms provided at each host. It can also control access to network resources via authentication and authorization checking techniques at the NIM level.

Use of host security mechanisms

Several opportunities exist for the XNIM to capitalize on host-supported mechanisms to enhance overall NOS security. Some of the obvious techniques are discussed in this section.

Authentication

When remote users are granted access to computer resources, authentication of the user's identity is critical. Due to their relatively low cost and ease of implementation, passwords are the most commonly used mechanism to achieve personal identity authentication.^{43,44} In fact, passwords are used by nearly all multi-user systems within the federal government and all commercial time-sharing systems.¹ Other automated techniques (e.g., dynamic signature verification, hand geometry) are also beginning to make an appearance.^{17,23,10}

The problem with passwords is that they are not used to full advantage by humans. Easy-to-guess passwords such as children's names, initials, birthdates and mothers' birth

names are among those commonly selected. Even the practice of using randomly selected passwords from the English language reduces the password space significantly. As a result, such passwords are easier to determine by exhaustive searching than random strings of characters. In addition, changing passwords frequently is a nuisance to most users.

Since the NIM's relationship with the NOS-participating host is similar to that of a terminal user, it is a trivial matter to enhance the protection of NOS objects (i.e., data and programs) maintained on the host by incorporating good password utilization procedures into the NIM. For example, "remembering" frequently changing passwords is certainly no problem for the NIM. Moreover, protection of these passwords within the NIM system can be accomplished through encryption, memory protection and other such techniques.

Making the password difficult to guess is best accomplished by careful selection of the size of the password and of the alphabet from which the password is constructed. Anderson provides the following guideline for the determination of an adequate password size, S (in characters).

$$(R/E)(4.39 \times 10^4) \times (M/P) <= A^S \quad (1)$$

where R is the transmission rate of the line in characters per minute, E is the number of characters exchanged in a logon attempt, P is the probability that a password will be found during M , the period over which the systematic testing is to take place (in months of 24 hours per day operation) and A is the alphabet size.²

Of course the NIM could be directed to use the largest password space available for each supported system. With the resulting fixed password size per system, one then specifies an acceptable probability P for the NOS-supported network host, and determines the maximum lifetime (M) allowed to ensure that level of confidence. Solving for M in Formula 1 then yields

$$M <= (E/R) \times A^S \times P / (4.39 \times 10^4) \quad (2)$$

As an example, assume that the probability of finding a six character password on a NOS-participating host must be no greater than .001, the logon attempt involves the exchange of 100 characters and the line speed is 1800 characters/minute. Then, for a 26-character alphabet (e.g., English), the password lifetime M must be no longer than a third of a month (i.e., about 10 days).

Requiring would-be users to re-dial the computer system after several (e.g., three) unsuccessful logon attempts would increase the effective password lifetime in the above example. These and other factors should be taken into consideration when attempting to determine the "right" mix of variables needed to provide the required degree of protection. Still, from this discussion it is clear that the NIM can easily surpass the average user in the practice of effective password techniques. Other techniques for better utilization of password schemes are thoroughly discussed by Wood.⁴³⁻⁴⁵

File naming

Assume that a network attacker has succeeded in masquerading as the XNIM or another, valid user and is now logged directly onto a network host (i.e., the XNIM is not in the path from user to host system). At that point, a number of factors interrelate in determining the degree of damage that an individual can do. Should his or her goal be the indiscriminate destruction of data files, then only the access controls of the host system, coupled with the access rights of the authorized user being imitated, can limit the success of the attacker. The XNIM certainly has no control, direct or indirect, in this situation.

However, if a successful system penetrator is instead seeking to locate specific information, his job can at least be made more difficult by the use of non-informative file naming conventions. For example, if there is one NOS-owned account on each participating host system, then the files of all NOS users would be maintained in the same account space with such nondescript names as "NOS001," etc. No indication need be made at the host level of the actual "owners" of these files, since from the host's viewpoint, the NOS is the owner. If possible the NOS should also be the only party granted any access privileges to these files.

Compartments

To further complicate the efforts of masqueraders to locate information, an NOS could maintain two or more accounts (i.e., compartments of information) on each participating host. Thus, for N account spaces, the probability of selecting the correct account on the first try would be reduced to $1/N$. With the probability of successfully penetrating any account through masquerading kept at a minimum by effective use of passwords, the net result is an increase in the protection of the NOS files.

File compartments also exist at a finer degree of granularity. Consider the case of a penetrator successfully masquerading as an authorized XNOS user, when accessing network resources through the XNIM. Since XNOS functions only operate on an XNOS-user-account basis, access to or destruction of all XNOS-maintained files requires possession of the appropriate set of access rights. Thus, total destruction of XNOS-maintained files would require emulating the set of users whose pooled access rights cover all XNOS files.

NIM intermediary functions

Access to NOS resources that are utilized via the NIM can be controlled at the time of system access and data access. Data access, in turn, may be effected at the file, record and data item level. Within this paper, file access will be the smallest degree of data granularity addressed in detail. Record and data item access control within a NOS

environment are currently under investigation and therefore will only be briefly mentioned.

System access

System access usually involves user authentication techniques. Within the NOS, system access takes place at what is commonly called "login" or "logon" time, at both the NIM and the participating host systems. At the time a user logs into the NIM, password or other user authentication mechanisms should be used. If, in the course of a NOS session, multiple NIMs need to communicate, then likewise each NIM must authenticate itself to the other in a like manner. The fact that each NIM-host communication is in support of a single, specific user, serves to restrict the damage that can be done by an imposter.

The issue that now arises is, "What identity does/should the NIM process claim?" The alternatives are (1) a special (NOS) account (or set of accounts) could be maintained on each participating host for NOS sessions or (2) a NOS user could be required to maintain individual accounts on each system to be used on his behalf. In the latter case, the user would have to provide the NOS with all personal user-ids and associated passwords. Furthermore, in the event that two or more users request networking sessions involving the same target host, an equivalent number of connections would have to be effected between the XNIM and the host.

The former case, however, has the consequence of maintaining all NOS-user files for a given system within the same target system NOS account(s), as previously discussed. Providing effective access controls in this situation necessarily requires trust in the NOS-NSC process handling the account. This is not an unreasonable requirement for so critical a component of the NOS.

Data access

Access control may be enforced through discretionary or non-discretionary systems. The former implies a system wherein the "owner" of an object (e.g., file, program) can grant access rights to individual users.

Non-discretionary systems, on the other hand, deny users such rights. Instead, access to objects is granted on the basis of labels which specify the access class of the given object and the clearance of the subject requesting access. These labels may be changed only by the System Security Officer or equivalent.

Pure non-discretionary systems are inappropriate for most civilian NOSs as they preclude flexibility and selectivity in object sharing. Conversely, discretionary systems require object owners to explicitly identify individuals or groups to whom access rights are granted. Therefore, it seems desirable to aim for a combination or hybrid access control system for an NOS. Labels (i.e., clearances and classifications) would provide a security filter or first line of control to network resources, while object owners retain the right to

selectively grant or deny access privileges to individuals and groups within that general framework.

At the highest level in such a hybrid scheme, the rights of a user or process to access an object (e.g., file, record, data item) can be determined by a set of rules which specify the clearance (i.e., label) required to access an object of a given access class. A lattice security model has been proposed to enforce such controlled access to data among distributed computer systems.^{26,27} This model is briefly described in the next section.

Given that the flow of data through a network is governed by an appropriate non-discretionary model, then the particular rights of individuals and processes to access and operate on that data must be determined. Such rights are often called usage restrictions and define the "mode of access" permitted to the data. Example modes of access include READ, WRITE, APPEND and EXECUTE. The mode of access allowed by a given user (subject) to a set of data (objects) may be defined by an access control list (ACL), as implemented in the Multics operating system.^{39,32}

Other data access issues include local vs. remote authorization checking and the complications that result from the capability of one party to grant access rights to another. The mechanisms for determining authorization can become quite complex when, for example, person A grants person B a set of access rights to a given object X and among that set of rights is the right to grant access to object X. In this context, the implications of granting a user READ privileges to an object should be fully considered. Once a user has been given READ rights to a file, for example, it is conceivable that a user process will copy the file. In such a case, it would be inappropriate for the NIM to expend much effort facilitating multi-level GRANT capabilities. This differs from the case for NOS-supported data base access.

Data flow control

The applicability of the "lattice security model"¹¹ for controlling access to networked systems has been examined.²⁶ This model was originally derived from the military classification system and is intended to be a non-discretionary access control system. Therefore, objects are assigned access classes and subjects are assigned clearances. For a subject to gain access to an object, it must be "cleared" for the object. A subject does not have the discretion to grant access to objects to other subjects who are not cleared for the objects. The basis of the lattice model is a set of partially ordered access classes from which subject clearances and object classifications are chosen. There must be a lowest access class that is strictly less than any other access class and a highest access class such that it is strictly greater than all other access classes.

An example of a simple lattice would be the set of access classes {SECRET, PUBLIC}. In this case the ordering is a total ordering—PUBLIC<SECRET. The military security lattice, on the other hand, has two components—a sensitivity level and a category set. Sensitivity levels are {UN-

CLASSIFIED, CONFIDENTIAL, SECRET and TOP SECRET}. Categories involve the further segmentation of sensitivity levels into collections of information requiring special access permission ("need to know").

To ensure the integrity of the data, an individual or process with a given clearance is not allowed to WRITE to a process or data file with a lower access class. READs from a set of data at a lower access class are allowed. Although these restrictions do not inhibit the flow of data from, say, an unclassified file to a process with secret clearance within an individual host, complications arise when data is being accessed across host boundaries. In order for a process on one host (HOST A) to "read" data from another host (HOST B) HOST A must send a request for data to HOST B. This request is analogous to a WRITE to HOST B, and thus we have a violation of the confinement property.²⁶

Acceptable solutions to this problem depend on the nature of the application and the security characteristics of the participating hosts. However, it is worth noting at this point that in most civilian NOSs the potential existence of trap doors (the primary threat against which much of this model is directed) is not a pressing concern.³⁸ Therefore, a less rigid approach to controlling access would quite possibly be sufficient to achieve the required level of NOS security.

XNOS SECURITY MECHANISMS

Security-enhancing mechanisms have been incorporated in the NBS XNOS to control access to (1) the XNOS monitor, (2) network files and (3) data elements within network files and data bases. The first two are discussed in this paper. The third is still under investigation.

A secure operating system was not available for the XNIM implementation. However, it is assumed that such a system is a prerequisite for an operational NOS which intends to provide access control to network resources. There are current efforts ongoing in the development and verification of a secure operating system for a minicomputer of the same architecture as the NBS XNIM (i.e., DEC PDP 11/45).³⁸

System access

Consistent with conventional operating systems, access to XNOS is granted via passwords at XNOS "login" time. Since the XNOS-monitor is resident on the XNOS NIM (XNIM), which in turn is implemented on a real operating system (Bell Laboratories' UNIX time-sharing system), the user must first be recognized as a valid XNIM (unix) user. At this stage, authentication is accomplished by the possession of the appropriate pair (unix-user-id,unix-user-password). The user must then be identified as a qualified XNOS user by presenting the XNOS subsystem with an additional (preferably different!) password. In accordance with recommended password techniques,⁴³⁻⁴⁵ the XNOS password table is maintained in an encrypted form.

To facilitate the prototype development and feasibility demonstration, XNOS currently maintains only one account

on each supported host system. All user files, therefore, are kept in one account space and it is only necessary for XNOS to supply its own password to the target system when logging on. XNOS then keeps internal directories associating XNOS users with particular files in each target host account.

File level access

File level access control is accomplished in two stages—(1) access control specification (for files, users, and commands) and (2) XNOS command processing.

Access control specification

The decision to incorporate both discretionary and non-discretionary access control mechanisms within the XNIM-based XNOS monitor implies the need for the following information: (1) Object security classifications, (2) User clearances and (3) The specific rights to access specific objects possessed by specific users.

XNOS maintains a profile of each authorized user. This profile includes that user's security clearance category (e.g., "3-B,C"). Only the XNOS "super-user" (i.e., System Security Officer) has the ability to change a user's clearance.

Within XNOS, access control lists similar to those in Multics are used to contain the first and third of the above information items.³⁹ At present, four classification levels are supported for all XNOS files—numbered zero through three. The assignment of classification names is arbitrary. For purposes of discussion, we can assume the following classification scheme:

- 3—Top Secret
- 2—Secret
- 1—Confidential
- 0—Public

This particular scheme represents a strict partial ordering, since each security classification level is greater than the next lower level. In order to support the general lattice security model for data flow control, it must be possible to specify disjoint, "need-to-know" subclassifications (i.e., compartments) at any given (other than the greatest or lowest) level. Thus, Level 1 (Confidential) could be further divided into parts 1.A, 1.B, and so forth. Classification 1.A could correspond to *Confidential: Government Contracts* or *Confidential: Payroll*, for example.

At this, the non-discretionary level of control, a person attempting to access (or grant access to) a file must have a security clearance at least as great as the classification of the object. Furthermore, if the person's clearance is equal to that of the object, then the "need-to-know" category of the prospective user must match the subclassification of the object. Thus, the user who has a Confidential level clearance to see confidential objects relating to Payroll (classification 1.B) would not be granted access to files classified 1.A.

To support discretionary access control, owners of files

ACCESS RIGHTS FOR:	testfile
SECURITY LEVEL:	1.A
OWNER:	wood
NAME	MODE
wood	r ea
kimbleton	r

Figure 4—XNOS file profile and access control list.

(as indicated in each file's access control structure) may grant up to four modes of access privileges to specified users. The available modes are read (R), write (W), execute (E), and append (A). With this capability, we now can fully specify the requirements and capabilities needed to access each file. Such a specification is shown in Figure 4. Note that a user must explicitly authorize access privileges for one's self to one's own file. This helps the user protect his files from himself.

XNOS command processing

In order to ensure flexibility—a mandatory design requirement for an experimental system such as XNOS—the security-related input/output characteristics for XNOS commands are maintained in a table. This table specifies, for each XNOS command, the minimal set of access privileges a user must possess to invoke the command with the given parameters (e.g., file names). A portion of the command access requirements table is shown in Figure 5.

The decision to grant or deny access is made at run-time. Thus, for example, although user Smith may have been explicitly granted the right to read file F1, classified at level 3.C, the clearance level of Smith at the time of the access request may result in the denial of access. Smith may have had clearance 3.C at the time F1's owner granted read privileges to him; however, it is possible that Smith's clearance changed in the interim. Such run-time decision-making allows the XNOS system to be reasonably dynamic, in that the latest security-related information is used in determining access rights.

On the other hand, the decision to grant or deny access needs to be made as soon as possible in order to minimize the computational and communications-related burden on the XNIM and the rest of the supported network. Therefore, the user's access rights are matched against the classification of both input and output files *before* any data access operations are begun. For example, to COPY <file1> TO <file2>, the user would have to be cleared at least to read file1 *and* to write to file2 *before* the transfer would be initiated.

Data item-level access

One of the two basic functional objectives of a network operating system is preservation of meaning in transmitting structured data, e.g., records, between heterogeneous systems.³⁰ Accomplishing this objective requires four major types of supporting information in addition to the binary string representing the record. These are (1.) the structure of the record and the storage order of its data elements, (2.) data element types, (3.) data element representations (which may be dependent on both computer and compiler) and (4.) data element names (to support differing source and destination names or differences in source and destination storage order representations).

Given knowledge of data element names, one potentially has the opportunity to control access to these data elements. Such control can prove highly desirable since it supports enhanced sharing of data subject to unrestricted access while providing required access restrictions and without necessitating separate files, as would otherwise be mandatory.

As is the case for file level access control, data element-level access should also be supported by both discretionary and non-discretionary controls. This implies the need for managing a significant amount of access control-related information. As a result, although the access control decision is implemented by XNOS—e.g., if access is denied XNOS will not support the retrieval and transmission of the data element—in practice such decisions are likely to be closely related to the utilization of database management systems (DBMSs).

NBS is currently developing an Experimental Network Data Manager (XNDM) to investigate the problems of interfacing network users to dissimilar DBMSs. An XNDM

COMMAND	INPUT MODE	OUTPUT MODE
append	ra	ra
compare	r	----
copy	r	rw
delete	rw	----
erase	rw	----
rename	rw	rw
type	r	----

Figure 5—XNOS command requirements.

provides an excellent opportunity for examining issues related to data element-level access control. Consequently, a key component of the NBS XNDM is an Access Control Mechanism supporting discretionary and non-discretionary controls. Revocation of grants of access privileges uses a mechanism similar to that described in References 21 and 15 based on time-stamping.

In addition to supporting both discretionary and non-discretionary controls, the XNDM Access Control Mechanism also differs from counterparts for individual DBMSs through permitting distribution of access information across systems. Thus, in processing a query emitted by a source program *S*, one of the key steps is collecting the access rights of *S*. These rights are then transmitted, together with the query to a destination system where they are compared against the access requirements of the specific data elements. If access is not rejected, the appropriate data is retrieved. Otherwise, a rejection notice is returned to the calling program.

A major problem in implementing network wide data element-level access controls is resolving the issue of how much trust one will place in an operating system using unverified code. To circumvent this problem, all XNDM access control information is also resident in the XNIM. Thus, the arguments which one would make in establishing the appropriate level of trust are analogous to those used in discussing file-level access control issues earlier in this paper.

CONCLUDING REMARKS

The goals behind the XNOS access control mechanism implementation were much the same as those motivating the entire XNOS project—to demonstrate the feasibility of the concept and, while doing so, identify the design alternatives inherent in the development of such a capability. While we feel that these goals have been achieved, there is no intent to claim that the implementation approach is in itself the definitive one. In fact, as a part of the ongoing NBS investigation of NOSs and related higher-level communications protocol issues, the XNOS implementation previously described by Kimbleton is currently undergoing a major restructuring and re-implementation.³⁰

Given the type of NOS architecture exemplified by the NBS XNOS, network operating systems can be used to enhance network security. The skillful utilization of existing host security mechanisms, such as password techniques, can result in an immediate increase in protection, as the NOS support computer can be relied upon to use the pre-programmed, recommended procedures for selecting, changing and protecting its passwords. Furthermore, by adding a level of indirection between network resources and those wishing to access them, the NOS not only achieves a higher degree of resource protection, but does so without requiring changes to the participating host systems. It remains for system and network managers to determine the required level of security for a particular system or network.

ACKNOWLEDGMENTS

The authors wish to acknowledge the efforts of Larry Gelberg, who assisted in the design of the file-level access control mechanisms and programmed and installed these security features in the NBS Experimental Network Operating System.

REFERENCES

1. Anderson, James P., *On Centralized Distribution of One-time Passwords in Resource Sharing Systems*. James P. Anderson and Co., Fort Washington, Pa., August 1971, 8 pp.
2. Anderson, James P., "Information Security in a Multi-user Computer Environment," *Advances in Computers*, Vol. 12, Academic Press, Inc., New York, 1972, pp. 1-36.
3. Baran, Paul, *On Distributed Communications: IX. Security, Secrecy, and Tamper-free Considerations*, Rand Corporation, August 1964, AD-444 839, 39 pp.
4. Branstad, Dennis K., "Security Aspects of Computer Networks," *Proceedings of AIAA Computer Network Systems Conference*, American Institute of Aeronautics and Astronautics, New York, N.Y., April 1973, 8 pp.
5. Branstad, Dennis K., "Encryption Protection in Computer Data Communications," *Proceedings of the Fourth Data Communications Symposium*, IEEE Computer Society, October 1975, pp. 8-1, 8-7.
6. Branstad, Dennis (Ed.), *Computer Security and the Data Encryption Standard*, National Bureau of Standards, Special Publication 500-27, February 1978.
7. Browne, Peter S., "Computer Security—A Survey," *Proceedings of the National Computer Conference*, AFIPS Press, Montvale, N.J., 1976, pp. 53-63.
8. Bushkin, Arthur A., *A Framework for Computer Security*, System Development Corporation, McLean, Va., AD-A025 356, June 1975, 158 pp.
9. Cole, Gerald D., *Design Alternatives for Computer Network Security*, National Bureau of Standards, Special Publication 500-21, Vol. 1, January 1978, 173 pp.
10. Cotton, Ira W., and Meissner, Paul, "Approaches to Controlling Personal Access to Computer Terminals," *Proceedings of the 1975 Symposium Computer Networks: Trends and Applications*, IEEE Computer Society, 1975, pp. 32-39.
11. Denning, Dorothy E., "A Lattice Model of Secure Information Flow," *Communications of the ACM*, Vol. 19, No. 5, May 1976, pp. 236-243.
12. Diffie, W., and Hellman, M. E., "New Directions in Cryptography," *IEEE Trans. on Info. Theory*, Vol. IT, No. 22, November 1976, pp. 644-654.
13. Diffie, W., and Hellman, M. E., "Multiuser Cryptographic Techniques," *National Computer Conference, AFIPS Conference Proceedings*, Vol. 45, June 1976, pp. 109-112.
14. Diffie, W., and Hellman, M. E., "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *Computer*, June 1977, pp. 74-84.
15. Fagin, Ronald, "On an Authorization Mechanism," *ACM Transactions on Database Systems*, Vol. 3, No. 3, September 1978, pp. 310-319.
16. *Data Encryption Standard*, National Bureau of Standards, FIPS PUB 46, January 1977.
17. *Guideline on Evaluation of Techniques for Automated Personal Identification*, National Bureau of Standards, FIPS PUB 48, April 1977.
18. Fitzgerald, M. L., *Common Command Language for File Manipulation and Network Job Execution: An Example*, National Bureau of Standards, Special Publication 500-37, August 1978, 32 pp.
19. Forsdick, H. C., R. E. Schantz and R. H. Thomas, *Operating Systems for Computer Networks*, BBN Report No. 3614, Bolt Beranek and Newman, Cambridge, Mass., 1977.
20. Gait, Jason, *Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard*, National Bureau of Standards, Special Publication 500-20, November 1977.
21. Griffiths, Patricia P., and Bradford W. Wade, "An Authorization Mech-

- anism for a Relational Database System," *ACM Transactions on Database Systems*, Vol. 1, No. 3, September 1976, pp. 242-255.
22. Heinrich, Frank, *The Network Security Center: A System Level Approach to Computer Network Security*, National Bureau of Standards, Special Publication 500-21, Vol. 2, January 1978, 69 pp.
 23. Herbst, N. M., and Liu, C. N., "Automatic Signature Verification Based on Accelerometry," *IBM J. Research Development*, May 1977, p. 245-263.
 24. Hoffman, Lance J., *Modern Methods for Computer Security and Privacy*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1977.
 25. Linden, Theodore A., "Operating System Structures to Support Security and Reliable Software," *Computing Surveys*, Vol. 8, No. 4, December 1976, pp. 409-445.
 26. Karger, Paul, "Non-discretionary Access Control for Decentralized Computing Systems," SM thesis, M.I.T. Dept. of Electrical Engineering and Computer Science, May 1977. (Also available as MIT/LCS/TR-179, Laboratory for Computer Science, M.I.T., May 1977, NTIS AD A040808.)
 27. Karger, Paul A., "Non-discretionary Security for Decentralized Computing Systems," *Proceedings of 1978 Trends and Applications: Distributed Processing*, IEEE Press, May 1978, pp. 33-39.
 28. Kent, Stephen T., *Encryption-Based Protection Protocols for Interactive User-Computer Communication*, (Master's Thesis), Massachusetts Institute of Technology, Cambridge, Mass., AD-A026 911, May 1976, 122 pp.
 29. Kimbleton, S. R., and R. L. Mandell, "A Perspective on Network Operating Systems," *Proceedings 1976 National Computer Conference*, AFIPS Press, Montvale, N.J., Vol. 45, 1976, pp. 551-559.
 30. Kimbleton, Stephen R., Helen M. Wood and M. L. Fitzgerald, "Network Operating Systems—An Implementation Approach," *Proceedings 1978 National Computer Conference*, AFIPS Press, Montvale, N.J., Vol. 47, 1978, pp. 773-782.
 31. Martin, James, *Security, Accuracy, and Privacy in Computer Systems*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973, 626 pp.
 32. Organick, Elliott I., *The Multics System: An Examination of Its Structure*, MIT Press, Cambridge, Mass., 1972.
 33. Parker, Donn B., *Threats to Computer Systems*, Lawrence Livermore Laboratory, UCRL-13574, March 1973, 118 pp.
 34. Parker, Donn B., Susan Nycum and S. Stephen Qura, *Computer Abuse*, Stanford Research Institute, PK-231 320, November 1973, 181 pp.
 35. Parker, Donn B., "Computer Abuse Perpetrators and Vulnerabilities of Computer Systems," *Proceedings of the National Computer Conference*, AFIPS Press, Montvale, N.J., 1976, pp. 65-73.
 36. Parker, Donn B., *Crime by Computer*, Charles Scribner's Sons, New York, 1976, 308 pp.
 37. Petersen, H. E., and Turn, R., "System Implications of Information Privacy," *Proceedings of the Spring Joint Computer Conference*, Thompson Book Co., Washington, D.C., 1967, pp. 291-300.
 38. Popek, Gerald J., *Security in Network Operating Systems: A Survey*, an interim report sponsored by the Institute for Computer Sciences and Technology, National Bureau of Standards, July 1978.
 39. Saltzer, Jerome H., "Protection and the Control of Information Sharing in Multics," *Communications of the ACM*, Vol. 17, No. 1, July 1974, pp. 388-402.
 40. Thomas, R. H., "On the Design of a Resource Sharing Executive for the ARPANET," *Proceedings of the National Computer Conference*, AFIPS Press, Montvale, N.J., 1972, pp. 155-164.
 41. Walker, Bruce J., and F. Blake Ian, *Computer Security and Protection Structures*, Dowden, Hutchinson & Ross, Inc., Stroudsburg, Pa., 1977, 142 pp.
 42. Winkler, Stanley, and Lee Danner, "Data Security in the Computer Communication Environment," *Computer*, February 1974, pp. 23-31.
 43. Wood, Helen M., "On-line Password Techniques," *Proceedings of Trends and Applications: 1977—Computer Security and Integrity Symposium*, IEEE Computer Society, May 1977.
 44. Wood, Helen M., *The Use of Passwords for Controlled Access to Computer Resources*, National Bureau of Standards, Special Publication 500-9, May 1977.
 45. Wood, Helen M., "The Use of Passwords for Controlling Access to Remote Computer Systems and Services," *Proceedings of the National Computer Conference*, AFIPS Press, 1977, pp. 27-33.
 46. Wood, Helen M., and Stephen R. Kimbleton, *Remote Record Access: Requirements, Implementation and Analysis*, National Bureau of Standards, Special Publication, in press.
 47. Wood, Helen M., and Stephen R. Kimbleton, *A Framework for Specifying Structured Data Transfer Protocols*, National Bureau of Standards, Special Publication, in press.

Public key vs. conventional key encryption*

by CHARLES S. KLINE and GERALD J. POPEK

University of California
Los Angeles, California

INTRODUCTION

As distributed computer systems grow and their convenience attracts uses for which maintenance of privacy and security is important, the means by which encryption is integrated into these systems also becomes important. Encryption is the only practical way by which secure, private communication can be conducted while employing untrusted media to carry the transmission. The interest has spurred developments in the use of conventional encryption algorithms and there is even a federal standard algorithm for commercial use.⁴ In addition, an innovative approach to encryption, called public key algorithms, has recently been proposed as a way to address many of the key distribution and other problems which are present in conventional algorithm-based approaches.

Here we examine the general classes of functions desired of encryption algorithms as they are integrated into computer systems, and discuss the characteristics and properties desired in each case. We then look at the two general approaches—conventional algorithm-based, and public key-based. In every case we examine, we conclude that the two approaches are essentially equivalent. Neither approach has any particular advantage over the other. This conclusion is surprising, since one's intuition may suggest that public key algorithms are intrinsically superior because of their potential additional flexibility. Certainly, many public key proponents claim so. Yet, upon closer examination, the advantages of public key systems evaporate when one actually integrates them into a larger system.

In fact, we will see that the major unsolved problems are not concerned with key distribution, or the development of trusted software, but instead with the need for strong algorithms, whatever their form, and for reliable authentication methods: ways by which the human being can be effectively identified by the computer system.

PUBLIC KEY AND CONVENTIONAL ENCRYPTION ALGORITHMS—GENERAL CHARACTERISTICS

Encryption provides a method of storing data in a form which is unintelligible without the "key variable" used in

the encryption. Basically, encryption can be thought of as a mathematical function

$$E = F(D, K)$$

where D is the data to be encoded, K is the key variable, and E is the resulting enciphered text. For E to be a useful function, there must exist an F' , the inverse of F ,

$$D = F'(E, K)$$

which has the property that the original data can be recovered from the encrypted data if the value of the key variable originally used is known.

However, the use of F and F' is valuable only if it is difficult to recover D from E without knowledge of the corresponding key K . A great deal of research has been done to develop algorithms which make it virtually impossible to do so, even given the availability of powerful computer tools.

The "strength" of an algorithm is traditionally evaluated using the following assumptions. First, the algorithm is known to all involved. Second, the analyst has available to him a significant quantity of matched encrypted data and corresponding cleartext. He may even have been able to cause messages of his choice to have been encrypted. His task is to deduce, given an additional, unmatched piece of encrypted text, the corresponding cleartext. All of the matched text can be assumed to be encrypted through the use of the same key variable which was used to encrypt the unmatched segment. In particular, therefore, the difficulty of deducing the key used in the encoding is directly related to the strength of the algorithm.

Recently, Diffie and Hellman¹ proposed a variation of the conventional encryption methods that may in some cases have certain advantages over standard algorithms. In their class of algorithms, there exists

$$E = F(D, K),$$

as before, to encode the data and

$$D = F'(E, K')$$

to recover the data. The major difference is that the key K' used to decrypt the data is not equal to, and cannot be easily derived from, the key K used to encode the data. Presumably there exists a pair generator which based on some input

* This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-77-0211.

information produces the matched keys K and K' with high strength (i.e. resistance to the derivation of K' given K , D , and matched $E=F(D,K)$).

The value of such a public key encryption algorithm lies in some potential simplifications in initial key distribution, as well as for "digital signatures." The key K used to encrypt the data is expected to be publicly known, and is referred to as the public key. The key K' used to decrypt the data would be kept secret and is referred to as the private key.

SYSTEM FUNCTIONALITY

There are a number of privacy and security-related functions which are desired from a distributed computer system. Each of the classes of functions which have been suggested will be discussed, together with the properties desired of the function.

The assumptions made regarding the threats against which protection is desired include tapping of lines, introduction of spurious traffic and retransmission of previously-transmitted genuine traffic. It is assumed that malicious attacks are expected, and that there do not already exist secure, high bandwidth paths between those who wish to communicate in a private manner. This typically is the case when common carrier transmission media, including packet-switched services, are employed.

Private communication

This is the conventional use of encryption. In the distributed systems being envisioned, such as a large-scale computer network, it is viewed as desirable to separately protect each different conversation from every other. That is, the goal is to guarantee that each user/user connection is protected via encryption separately from all other connections. Therefore, each conversation needs its own key pair in conventional methods, and each user must have his own public/private key pair in the public key schemes. As a result, there can be a formidable key distribution problem. Any solution to this problem should not, in the view of many, depend on a single, central key distribution center, since the security of the entire network could depend on it, and the center contains high potential for abuse.

Internal authentication

In constructing a distributed system, it is not uncommon for the sites in the system to be connected using common carrier circuits, so that a potentially large amount of switching mechanism may be involved in the links. Hence, the sites may wish to exchange several messages each time that they recommence operation, to assure that they are each connected to the right sites. The usual method seen in computer systems today, where one member of the pair reveals some secret information to the other, is unacceptable in

general if one is concerned about spoofing. The conventional form of this problem concerns authentication of the user to the system, via some form of a login protocol.

In networks, however, the problem is mutual—each "end" of the channel may wish to assure itself of the identity of the other end. Quick inspection of the class of methods used in centralized systems show that straightforward extensions are unacceptable. Suppose one required that each participant send a secret password to the other. Then the first member that sends the password is exposed. The other member may be an imposter, who has now received the necessary information to pose to others in the network as the first member. Whoever goes first potentially reveals his secret to a spoofer, who can then masquerade and collect other secrets. Obviously, extension to a series of exchanges of secret information will not solve the problem. It only makes necessary a several-step-posing procedure by the imposter. A different approach is in order.

Datagrams

In the private communication function, it is generally understood that the parties wishing to communicate are willing to pay some reasonable amount of overhead to get the private conversation established, so that a key distribution algorithm involving several messages would be quite suitable, for example. However, in the case of short messages, or datagrams, it is generally viewed as unreasonable for the actual transmission of the short message to require significant overhead, such as several preceding messages to set up the channel. On the other hand, some queuing delays at the sending or receiving site may well be acceptable if the number of overhead messages can be significantly reduced. Also, the datagram function is similar to mail, in that the receiver need not be active, or logged in, at the time the message is received.

Digital signatures

The goal here is to provide a way by which the author of a digitally-represented message can "sign" it in such a fashion that the "signature" has similar properties to the analog signature written in ink for the paper world. Without a suitable digital signature method, many have argued that the growth of distributed systems will be seriously inhibited, since large classes of applications would be precluded.

The properties required of a digital signature method include the following:

1. **Unforgeability**—It should only be possible for the author to create the signature for any given message.
2. **Authenticity**—There must be a straightforward way to conclusively demonstrate the validity of a signature in case of dispute, even long after authorship.
3. **No repudiation**—It must not be possible for the author of signed correspondence to subsequently disclaim authorship.

4. Low cost and high convenience—The simpler and lower-cost the method, the more likely it will be used.

Minimum trusted mechanism: Minimum central mechanism

In all of these functions, it is desirable that there be minimum trusted mechanism involved. This desire occurs because the more mechanism, the greater the opportunity for error, either by accident or by intention (perhaps by the developers, maintainers, etc.). One wishes to minimize the involvement of a central mechanism for analogous reasons. This fear of large, complex and central mechanisms is well justified, given the experience of the failure of large central operating systems and data management systems to provide a reasonable level of protection against penetration. All the Kernel-based approaches to software architectures have as their goal the minimization in size and complexity of central, trusted mechanism. Others are distrustful that a centralized, governmental (presumably) communication facility, or even a large common carrier can be trusted to assure privacy and other related characteristics. These general criteria are quite important to the safety and credibility of whatever system is eventually adopted. They also constrain the set of approaches that may be employed.

PRIVATE COMMUNICATION—KEY DISTRIBUTION

There are several requirements which any encryption protocol must satisfy. First, the encryption algorithm must be strong. Second, the keys to be used must be chosen and stored securely. Third, the keys must be communicated securely. Finally, authentication must be provided to separate this conversation from others which may use, or have used, the same key. Of these issues, the main problem is key distribution.

Conventional and public key systems solve these problems in different ways. In conventional systems, a new key is typically requested for each new communication. A key controller (which may or may not be centralized) chooses all keys, and performs any protection policy checks. The key controller only communicates the keys, establishing the communication channel if the protection checks succeed. The key controller communicates the keys chosen utilizing the same communications system as will be used for data transfer, but using previously arranged secret keys between the controller and the parties in the planned communication. However, it is straightforward to design the system so that the secret keys are stored in read only storage of the encryption units and never revealed. No authentication mechanism is needed to separate the new secure channel from prior ones, since the new keys chosen effectively form an authentication—no prior messages are useful.

Public key advocates claim that one of the advantages of public key algorithms over conventional algorithms lies in potentially simplified key distribution. Simply put, public key advocates argue that an automated "telephone book"

of public keys can generally be made available, and therefore whenever user x wishes to communicate with user y , x merely must look up y 's public key in the book, encrypt the message with that key, and send it to y .¹ Therefore, there is no key distribution problem at all. Further, no central authority is required initially to set up the channel between x and y .

It is clear, however, that this viewpoint is incorrect—some form of a central authority is needed and the protocol involved is no simpler nor any more efficient than one based on conventional algorithms.⁵ First, the safety of the public key scheme depends critically on the correct public key being selected by the sender. If the key listed with a name in the "telephone book" is the wrong one, then there is no security. Furthermore, maintenance of the (by necessity machine-supported) book is non-trivial because keys will change; either because of the natural desire to replace a key which has been used for high amounts of data transmission, or because a key has been compromised through a variety of ways. There must be some source of carefully maintained "books" with the responsibility of carefully authenticating any changes and correctly sending out public keys (or entire copies of the book) upon request.

Needham and Schroeder⁵ exhibit protocols to provide the desired properties for public key systems, and show that there are equivalent protocols for conventional algorithms. The protocols are equivalent both in terms of numbers of messages required as well as in the mechanisms which must be trusted. In particular, the public key must be requested from the central authority (be it implemented in a centralized or distributed manner) and transmitted in a way which guarantees that the right key is received. Since the public key is reused, some authentication mechanism, such as a sequence number, is required to isolate this communication from others which may have used the same key. The communications required to retrieve the key and to establish the authentication mechanism make the public key distribution algorithm entirely equivalent to conventional algorithms.

Some public key advocates have suggested ways to avoid requesting the public key from the central authority for each communication. First, a cache of keys can be kept (a small local "telephone book") and frequently used keys will be found there.

Second, a concept known as certificates has been suggested.³ A user can request that his public key be sent to him as a certificate. A certificate is a user/public key pair, together with some certifying information. For example, the user/public key pair may be stored as a signed message from the central authority. When the user wishes to communicate with other users, he sends the certificate to them. They each can check the validity of the certificate using the certifying information, and then retrieve the public key. Thus, the central authority is only needed once, when the initial certificate is requested.

Both the previous approaches have several problems. First, the mechanism used to store the cache of keys must be correct, since it will be relied upon. Second, the user of the certificate must decode it and check it (verify the sig-

nature) each time before using it, or must also have a secure and correct way of storing the key. Perhaps most important, as keys change the cache and old certificates become obsolete. This is essentially the capability revocation problem revisited.¹⁰ Either the keys must be verified (or re-requested) periodically, or a global search must be made whenever invalidating a key. Notice that even with the cache or certificates, an internal authentication mechanism is still required.

Public key systems also have the problem that it is more difficult to provide protection policy checks. In particular, conventional encryption mechanisms trivially allow protection policy issues to be merged with key distribution. If two users are not to communicate, then the key controller can refuse to distribute keys.** However, public key systems imply the knowledge of the public keys. Methods to add protection checks to public key systems add an additional layer of mechanism.

INTERNAL AUTHENTICATION

There are a number of straightforward encryption-based authentication protocols which provide reliable mutual authentication without exposing either participant. The methods are robust in the face of all the network security threats mentioned earlier. The general principle involves the encryption of a rapidly changing unique value using a prearranged key. In the following we outline a simple authentication sequence between nodes *A* and *B*. At the end of the sequence, *A* has reliably identified itself to *B*. The analogous sequence is needed for *B* to identify itself to *A*. Typically, one expects to interleave the messages of both authentication sequences.

Assume that *A* uses a secret key, associated with itself, in the authentication sequence. The reliability of the authentication depends only on the security of that key. Assume that *B* holds *A*'s matching key (as well as the matching keys for all other hosts to which *B* might talk).

1. *B* sends *A* in cleartext the current time of day as known to *B*.
2. *A* encrypts that time of day using its authentication key and sends the resulting ciphertext to *B*.
3. *B* decrypts *A*'s authentication message, using *A*'s matched key, and compares it with the time of day which *B* had sent. If they match, then *B* is satisfied that *A* was the originator of the message. If the received time of day is not much older than the current time of day, *B* is satisfied that the message has not been delayed and retransmitted.

This simple protocol does not expose either *A* or *B* if the encryption algorithm is strong, since it should not be possible for a cryptanalyst to be able to deduce the key from

** This approach blocks communication if the host operating systems are constructed in such a way as to prohibit cleartext communication over the network.

the encrypted time of day, even if he knew what the corresponding cleartext time of day was. Synchronized clocks in the network are not required. Further, since the authentication messages change rapidly, it is not possible to record an old message, retransmit it, and have it treated as valid by the recipient.

Authentication protocols such as these require the prior distribution of secret keys. If data security is the goal, no formal authentication protocol is actually required when all data transmissions are encrypted, since possession of the key serves as prima facie evidence that the participants are the appropriate ones, as well as providing the mechanism empowering the communication. Nevertheless, authentication protocols can give immediate assurance, and protect against the playback of previously recorded traffic.

DATAGRAMS

Datagrams are short messages from one user to another. These messages should be delivered with relatively low overhead if services such as electronic mail are to be practical. In addition, it is desired that buffering be performed at the recipient site. That is, the mail should be delivered as soon as possible to the recipient site, and stored there, even if the desired user is not logged in.

Assume that a user at one site wishes to send mail to a user at another site. Using conventional encryption algorithms, the first user would request a connection to the second user, and a new key would be chosen and distributed by the key controller to each end of the communication channel. That key is sent using the secret keys of the two users.

However, since the second user may not be signed on at the time, a daemon process is used to receive the mail and deliver it to the user's "mailbox" file for his later inspection. It is desirable that the daemon process not need to access the cleartext form of the mail, for that would require the mail receiver mechanism to be trusted. This task can be accomplished by sending the mail to the daemon process in encrypted form and having the daemon put that encrypted data directly into the mailbox file. The user can decrypt it when he signs on to read his mail. In that way, the daemon only needs the ability to append to a user's mailbox file.

In order for the user to know the new key used for this mail, however, the key distribution algorithm described earlier must be modified. Rather than send the key for this connection to both the sender and the receiver, the key controller sends the key twice to the sender, one copy encrypted with the sender's secret key and one copy encrypted with the receiver's. The sender can prepend the copy of the key encrypted in the receiver's secret key to the mail before transmission. When the recipient signs on, his own mail program will examine the mailbox file, find the key message encrypted with his secret key, decrypt it to obtain the key for that message, and then use that key to decrypt the remaining text.

In the case of public key encryption algorithms, the mail problem is somewhat simplified since the recipient knows

what key to use in decryption (his private key). However, authentication is not possible since the recipient is not present when the message is received. Thus, it may be a replay of a previously sent message. This problem can be prevented in the conventional encryption algorithm case via various protocols with the key managers, for example, by time-stamping the mail and having the recipient keep track of recently used mail keys.

Both mechanisms just outlined do guarantee that only the desired recipient of a message will be able to read it. However, as pointed out, they don't guarantee to the recipient the identity of the sender. This problem is essentially that of digital signatures, and is discussed in the next section.

DIGITAL SIGNATURES

The need for digital signatures has by now become apparent to many. At first, it appeared that public key methods would be superior to conventional ones for use in digital message signatures. The method, assuming a suitable public key algorithm, is for the sender to encode the mail by "decrypting" it with his private key and then send it. The receiver decodes the message by "encrypting" with the sender's public key. The usual view is that this procedure does not require a central authority, except to adjudicate an authorship challenge. However, two points should be noted. First, a central authority *is* needed by the recipient for aid in deciphering the first message received from any given author (to get the corresponding public key, as mentioned). Second, the central authority must keep all old values of public keys in a reliable way to properly adjudicate conflicts over old signatures (consider the relevant lifetime of a signature on a real estate deed for example).

Further, and more serious, the unadorned public key signature protocol just described has an important flaw. The author of signed messages can effectively disavow and repudiate his signatures at any time, merely by causing his secret key to be made public, or "compromised." This fact has also been pointed out by Saltzer.¹³ When such an event occurs, either by accident or intention, all messages previously "signed" using the given private key are invalidated, since the only proof of validity has been destroyed. Because the private key is now known, anyone could have created any message claimed to have been sent by the given author. None of the signatures can be relied upon.

Hence the validity of a signature on a message is only as safe as the *entire* future history of protection of the private key. Further, the ability to remove the protection resides in precisely the individual (the author) who should not hold that right. That is, one important purpose of a signature is to indicate responsibility for the content of the accompanying message in a way that cannot be later disavowed.

The situation with respect to signatures using conventional algorithms initially appears slightly better. Rabin⁹ proposes a method of digital signatures based on any strong conventional algorithm. Like public key methods it too requires either a central authority or an explicit agreement

between the two parties involved to get matters going.^{***} Similarly, an adjudicator is required for challenges. Rabin's method, however, uses a large number of keys, with keys not being re-used from message to message. As a result, if a few keys are compromised, other signatures based on other keys are still safe. However, that is not a real advantage over public key methods, since one could readily add a layer of protocol over the public key method to change keys for each message as Rabin does for conventional methods. One could even use a variant of Rabin's scheme itself with public keys, although it is easy to develop a simpler one.

However, *all* of the digital signature methods described or suggested above suffer from the problem of repudiation of signature via key compromise. Rabin's protocol or analogues to it merely limit the damage (or, equivalently, provide selectivity!). It appears that the problem is intrinsic to any approach in which the validity of an author's signature depends on secret information, which can potentially be revealed, either by the author or other interested parties. Surely improvement would be desirable.

A reliable digital signature method

A simple, obvious solution is to interpose some trusted interpretive layer between the author and his signature keys, whatever their form. For example, suppose the list of keys in Rabin's algorithm were not known to the author, but instead were contained in a secure Unit (hardware or software). Whenever the author wished to send a signed message, he merely submitted the message to the Unit, which selected the appropriate keys and then used the signature algorithm. Each author would have access to such a Unit.

The loading of each Unit requires some examination. In particular, the means which are used to select keys and insert them into each Unit must be correct if mail challenges are to be handled satisfactorily. That is, there must be some trusted Source of keys (and matching "standard message" in the Rabin protocol), and the key list for each author/recipient pair must be deliverable in a correct, secret way to the appropriate Units. We will call the collection of Units and the Source(s), together with their internal communication protocols, a Network Registry (NR). Such an NR appears required to solve the problems raised earlier. Note that some secure communication protocol among the components of the Network Registry is required. However, it can be very simple; low-level link style encryption would suffice.

For safety and efficiency, the NR functions presumably should be decomposed and distributed throughout the network. In particular, the failure or compromise of a local NR would then only have local consequences. One can even

^{***} In his paper, Rabin describes an initialization method which involves an explicit contract between each pair of parties that wish to communicate with digitally-signed messages. One can easily instead add a central authority to play this role, using suitable authentication protocols, thus obviating any need for two parties to make specific arrangements prior to exchanging signed correspondence.

construct local NR components of the Network Registry in a decentralized way so that compromise of more than one component would be required before a message signature was affected.

The Registry concept is quite common in the paper world. A local government's real estate recorder's office is probably the most commonly known example.

*Simplification of the proposed signature architecture—
Specialized digital signature protocols unnecessary*

Once the necessity of a Network Registry is recognized, including a guaranteed authentication mechanism, it appears that simplifications in the mechanisms required for digital signatures can be made that seem to remove the need for specialized digital signature protocols. Instead, any of a collection of simple methods will suffice.

In particular, in order for the Network Registry to operate satisfactorily (including performing user authentication), it clearly must be distributed, and clearly must be able to communicate securely internally among the distributed components. Given that such facilities exist, then the following is an example of a simple implementation of digital signatures which does not require a specialized protocol or encryption algorithm:

1. The author authenticates with a local Network Registry component, creates a message, and hands the message to the NR together with the recipient identifier and an indication that a registered signature is desired.
2. A Network Registry (not necessarily the local component) computes a simple characteristic function of the message, author, recipient and current time, encrypts the result with a key known only to the Network Registry, and forwards the resulting "signature block" to the recipient. The NR only retains the encryption key employed.
3. The recipient, when the message is received, can ask the NR if the message was indeed signed by the claimed author by presenting the signature block and message. Subsequent challenges are handled in the same way.

This simple protocol involves little additional mechanism beyond that which was needed by the Network Registry anyway. It does require that the Network Registry be involved in every message signature and validation. However, recall that all of the unadorned signature methods reviewed earlier require involvement of some form of a Network Registry for at least the first message between any two parties. Public key protocols must check the "telephone book," and Rabin's method requires either a contract or a Network Registry. Furthermore, when one adds a more complete Network Registry on top of those other signature methods to correct their repudiation problem, all methods involve the NR for each message. Note that this protocol also does not require the NR to maintain any significant storage for signature blocks.

Performance and safety

Certain elementary precautions should be taken in the design of the Network Registry to avoid unnecessary internal message exchanges and to assure safety of the keys used to encrypt the signature blocks. Performance enhancements presumably would involve distributing the signature block calculation. Safety enhancements could include the use of different keys at each distributed site, replicating sites, and employing a signature block computation which requires the cooperation of multiple sites. Each of these facilities is straightforward to build and so they are not discussed further here.

It has been speculated that the task of constructing a Network Registry would be simpler using public key systems, since only the secret key of the Registry needs to be stored securely. However, using conventional encryption the Registry could encrypt all the private keys using a master key which belongs to the Registry. Thus, it is again the case that only the master key of the Registry needs long-term secure storage.

From the preceding discussions, we conclude that the digital signature algorithms proposed heretofore are unsatisfactory, and the improvements required to correct their inadequacies make the use of a specialized digital signature algorithm unnecessary.

We note here that the safety of signatures in this proposal also depends on the future history of protection of keys as before—in this case those held by the Network Registry. However, there are several crucial differences between this case and previous proposals. First, the authors of messages do not retain the ability to repudiate signatures at will. Second, the Network Registry can be structured so that failure or compromise of several of the components is necessary before signature validity is lost. In the previous methods, a single failure could lead to compromise.

USER AUTHENTICATION

While digital signatures are important, it is necessary to realize that there still *must* exist a guaranteed authentication mechanism by which an individual is authenticated to the NR (presumably directly the local Unit). Any reasonable communication system of course ultimately requires such a facility, for if one user can masquerade as another, all signature systems will fail. What is required is some reliable way to identify a user sitting at a terminal—some method stronger than the password schemes used today. Perhaps an unforgeable mechanism based on fingerprints or other personal characteristics will emerge.

Once the user has been correctly identified to the system, public key systems also must deal with the problem of retrieving the recipient's private key. That key *must* be securely stored, either in part in the user's head (not a very secure place), or somewhere else. Various forms of storage are possible: for example a simple card the user carries around with him, or in the system itself. However, the

storage must be in a form which is not useful to anyone else. For example, if the user loses the box containing his key, no one else should be able to use it, nor decipher its contents. This requirement means that the box itself can not be used as the sole user authentication mechanism. In addition, the key must be stored in the box in an unreadable form, presumably encrypted using some system key. The user must first authenticate himself to the system, have the system read the box, decode the key, and store it securely for use.

Once it is recognized that the system must be able to store keys securely, it becomes clear that the box just suggested can be dispensed with, except possibly as part of a user authentication mechanism. The system then would store users' private keys. Once a user has authenticated himself, the system can retrieve the key. This approach avoids the problem of requiring the user carry around a card, and makes revocation/change of keys simpler.

Thus, there appears to be no advantage in the use of public key systems over conventional ones for user authentication or private key storage, since keys must be securely stored in either case. In fact, in conventional encryption, only the keys used for initial connection establishment with the key controllers require long-term storage. Others only remain as long as the connection is in use.

CONCLUSIONS

Based on the preceding discussions, we draw several conclusions. First, the debate over the relative advantages of public versus conventional key encryption algorithms is just not very important, at least for the class of applications discussed in this paper. In either approach, there must exist a similar amount of secure mechanism that must be trusted. Public key algorithms do not aid that problem to any significant degree. In any event, a strong algorithm is needed. Whether it is public key-based or a conventional one doesn't matter much at all, compared to the overriding necessity that it be strong. If strong conventional algorithms are easier to develop, as has been speculated,¹² research would be better devoted to that area rather than public key systems. Once suitable algorithms are available, the remaining weak link in the principles of secure, distributed systems lies with the requirement to accurately authenticate the user to the system.

Also, it seems that the digital signature methods which have been proposed, both public key- and conventional algorithm-based, do not adequately protect recipients of signed documents from repudiation of signatures by the author revealing the secret key(s) employed. The difficulty appears intrinsic to the approaches being taken. An alternative is available which overcomes this problem; however, that involves a small amount of trusted software.

The necessary underlying mechanism required to support improved digital signature methods, as well as other user-visible secure network communication protocols, is relatively well understood, and takes account of the important requirement that the amount of trusted mechanism involved be minimized for the sake of safety.⁸

In more global terms, this discussion of network security has been intended to illustrate the current state of the art. Assuming a common carrier philosophy, then general principles by which secure, common carrier-based, point-to-point communication can be provided are reasonably well in hand. Of course, in any sophisticated implementation, there will surely be considerable careful engineering to be done.

However, this conclusion rests on one important assumption that is not universally valid. Either there exist secure operating systems to support the individual processes and the required encryption protocol facilities, or each machine operates as a single protection domain. A secure implementation of a Key Distribution Center or Registry is necessary in any case. Fortunately, reasonably secure operating systems are well on their way, so that this intrinsic dependency of network security on an appropriate operating system base should not seriously delay common carrier security.

REFERENCES

1. Diffie, W., and M. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, November 1976, pp. 644-654.
2. Kent, S., "Encryption-based Protection Protocols for Interactive User-Computer Communication," Laboratory for Computer Science, MIT, TR 162, 1976.
3. Kohnfelder, L., "Towards a Practical Public-key Cryptosystem," Bachelor of Science Thesis, Massachusetts Institute of Technology, 1978.
4. National Bureau of Standards, *Data Encryption Standard*, Federal Information Processing Standards Publication 46, January 1977.
5. Needham, R., and M. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM*, December 1978.
6. Popek, G. J., and D. Farber, "A Model for Verification of Data Security in Operating Systems," *Communications of the ACM*, September 1978.
7. Popek, G. J., and C. S. Kline, "Design Issues for Secure Computer Networks," in *Operating Systems, An Advanced Course*, R. Bayer, R. M. Graham and G. Seegmuller (eds.), Springer-Verlag, 1978.
8. Popek, G. J., and C. S. Kline, "Issues in Kernel Design," *Proceedings of the National Computer Conference*, AFIPS Press, 1978.
9. Rabin, M., "Digitalized Signatures," *Foundations of Secure Computing*, R. Demillo, et. al., (eds.), Academic Press, 1978.
10. Redell, D. D., and R. S. Fabry, "Selective Revocation of Capabilities," *Proceeding of the IRIA Conference on Protection in Operating Systems*, Rocquencourt, France, August 13-14, 1974.
11. Rivest, R. L., A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," MIT Laboratory for Computer Science Technical Memo LCS/TM82 Cambridge, Mass., April 4, 1977 (Revised August 31, 1977).
12. Rivest, R., private communications, 1977.
13. Saltzer, J., "On Digital Signatures," *ACM Operating Systems Review*, April 1978.

SIGMA—An interactive message service for the Military Message Experiment

by ROBERT STOTZ, RONALD TUGENDER and DAVID WILCZYNSKI

USC/Information Sciences Institute
Marina del Rey, California

and

DONALD OESTREICHER

Xerox Corporation
El Segundo, California

MME OVERVIEW

The increasing sophistication of military systems and decreasing time frame for making decisions make it essential to provide the military commander better quality information faster. With today's technology, messages can traverse several thousand miles in fractions of a second, but hours are lost at either end, both in entering the message into the communications system and in delivering it to the person who can act on it. Even after the message is delivered, an officer acting on it requires background information to formulate a proper response. More often than not, that information is available only after time-consuming searching through ponderous files. The response is usually an outgoing message which must be coordinated with other people, many of whom are not in the immediate vicinity of the message drafter. Hand-carrying the draft to these people slows the response still further. In times of crisis this system can easily become overloaded, throwing the entire operation into disarray.

This message management problem seems an excellent candidate for automation. Users of the ARPAnet have had a form of on-line message service for more than seven years. There is no question that the technology exists, but whether it will be cost-effective in the military environment is not so clear.

In December 1975 the Defense Advanced Research Projects Agency (DARPA), Commander Naval Telecommunications Command (NAVTELCOM), Commander Naval Electronic Systems Command (NAVELEX), and Commander-in-Chief, Pacific (CINCPAC) signed a Memorandum of Agreement⁶ stating their intention to conduct an experiment at CINCPAC Headquarters whose express goal was to "evaluate the utility of interactive message service capabilities in a military environment." The experiment is called the Military Message Experiment (MME).

To have the military conduct an experiment of this sort is highly unusual. More traditionally the user community

would state their requirements in a Request for Operational Capability (ROC), which is interpreted and converted by some agency of the service into a system specification in the form of a Request For Proposal (RFP). The RFP is subject to further interpretation by the various contractors, first in their proposals in response to the RFP and later in the implementation by the winning contractor(s).

Although this procedure has apparently served well for procurement of more traditional systems, experience indicates it has been less successful in the field of computer automation, especially in data management systems like a message service, where the requirement is seldom well understood and the many levels of interpretation between the ROC and the final system lead to products poorly suited to the real requirement. The implications of a particular system design are subtle; if some aspect is inappropriate, it is often virtually impossible to change. ARPAnet experience has shown that the effectiveness of a message service is strongly dependent on ease of access to the system, how often people look at their messages, how "official" such messages are considered to be, *ad infinitum*. Further, the message service is often just the tip of the iceberg as users begin to understand the capabilities the system offers outside pure message processing. If the message service is used extensively, it intimately affects individuals' work style in ways that are difficult to predict. Thus it is very risky to try to specify "requirements" when replacing a paper, pencil and typewriter world with modern "office automation" tools. Attempts at management information systems have shown it is particularly difficult to provide a user interface that is acceptable to high-level managers, so it is not even clear precisely who will sit at the terminals or how these people will interact with senior officers.

The MME is a computer-world equivalent of a "fly-before-buy" test of a message service in an operational military environment. The test is relatively small and inexpensive compared to the cost of multiple installations of a "production" system. The hope is the experiment will provide

enough experience and understanding of interactive message handling that subsequent production systems will be successes rather than expensive lessons in how not to automate the process. In June 1977 the U.S. House Appropriations Committee put a moratorium on virtually all new development of "message systems" by the Department of Defense until results from the MME can be evaluated.

The MME is being conducted at the CINCPAC Headquarters, Camp Smith, Oahu. The test community is approximately 100 officers and staff personnel in CINCPAC's command center and "operations" directorate (called J3). Twenty-four video display terminals are provided for user interaction with the service. Seven printers are located throughout the headquarters for local hard copy. The host processor is a PDP-10 (KL processor) manufactured by Digital Equipment Corporation running the TENEX operating system developed by Bolt, Beranek and Newman.⁴ The PDP-10 connects to CINCPAC's AUTODIN (AUTomated Digital Information Network) terminal computer, called the LDMX (Local Digital Message eXchange). The LDMX supports the current message handling service at CINCPAC; it prints copies of all CINCPAC's incoming AUTODIN messages and accepts outgoing messages through an optical character reader, sending them to their specified addressees via AUTODIN.

TECHNICAL RESPONSE TO THE MME

The message service being used for the MME is called SIGMA, developed at the University of Southern California's Information Sciences Institute especially for this experiment. As such it is an "experimental" system to be used in an "operational" military environment to test the effectiveness of a "secure" "interactive message processing" service. Consider some of the issues implied by these terms.

"Experimental"

The primary purpose of the MME is to determine the effectiveness of interactive message processing in a military environment and to provide a technology-transfer path to apply the knowledge and techniques gathered to future generations of military systems. Many design philosophies are implied in such a context. The system developed must be flexible enough to adapt to a changing understanding of the problem and additional requirements imposed by the user community. It must concentrate on issues of functionality and suitable user interfaces. This does not imply that other issues such as sizing and performance are not important—the system must still be responsive and large enough to support a meaningful experiment—but the system need not be cost-justified itself, merely sufficient to gain understanding of the functional and cost issues. And, perhaps most important, the system must be highly instrumented to allow collection of various data reflecting the manner in which users operate the message service. Analyses of these data allow evaluation of user performance and usefulness of ser-

vice facilities, and provide a further understanding of the potential role of interactive message services in the military environment. All of these factors were considered and respected in the development of SIGMA.

"Operational"

To gain the most accurate picture of the MME's potential impact on the military community, an early decision was made to perform the experiment in an actual military environment, the CINCPAC headquarters. Since CINCPAC already has an effective manual message system whose use is well understood by its personnel, SIGMA presents a message processing model which is intuitively similar to the existing manual one in order to gain early user acceptance. This decision implied choosing terminology which matched standard military usage,⁵ and providing functions which, where possible, were similar to the manual ones. Since the military users operate the on-line system as only a part of their normal jobs, SIGMA has been designed to be highly self-instructive.

"Secure"

An important requirement for the SIGMA service is that it meet military security specifications. Although this test system will be operated only by personnel classified at Top Secret, it is a test objective that the message service address the multi-level security issues identified by previous research.^{3,1} To satisfy this objective, SIGMA implements a security model which behaves as though SIGMA were running in such a "provably secure, kernel-based" operating environment; this model is described in Reference 2. Although the provably secure environment does not yet exist on TENEX, SIGMA's emulation of it allows the users to interact in a manner virtually identical to that they would encounter if SIGMA actually ran in the secure environment.

"Interactive message processing"

SIGMA has been designed to be a "complete" interactive message service for CINCPAC. Its user interface responds to the needs of computer-naive users in several ways. Since this community will not receive much special training in SIGMA's use, an online Tutor and Help facility has been designed to take on the bulk of this responsibility.⁷ The Tutor, like the rest of SIGMA, takes advantage of the specially designed MME terminal,⁸ which features multi-window displays and two-dimensional editing.

The command formats are defined by SIGMA's Command Language Processor (CLP). The user instructs SIGMA through a set of function keys or by typing commands in a predesignated "command window" on the screen of the MME terminal. The CLP parses and interprets these instructions; it is table-driven so instructions may be added or modified easily. The CLP expands commands and parame-

ters that are only partially entered and corrects misspelled words to the degree that it can, based on the user's personal directory of named objects as well as the command table. A Prompt facility is provided which allows the user to ask about required parameters for a given command without losing any of his operational context. This powerful command language interface is an essential ingredient in providing the user the highly supportive environment needed for users with little or no experience with computers.

SIGMA—APPLYING INTERACTIVE TECHNOLOGY TO MILITARY MESSAGE PROCESSING

In common with other styles of mutual communication, message processing has distinctly cyclic characteristics—a message is initiated; its contents are refined from draft to final form; it is approved and sent to its intended recipients; a recipient reads it, perhaps forwards it to colleagues; the information contained within invites (or requires) a reply; the reply is initiated, and the process repeats. Message processing as practiced at CINCPAC differs from other styles primarily in its highly formal nature, with guidelines governing nearly every aspect.

The entire message processing task can be roughly separated into three areas of closely related activities: Message management, incoming message processing and outgoing message processing. As implied by the cyclic nature of the communication process, none of the activities in one area is disjoint from those in other areas. For the purposes of presentation, however, the facilities of the SIGMA message service are described according to these three areas.

Message management

A message service must facilitate all phases of message management. The following sections summarize SIGMA's support in this area.

Messages

Messages are SIGMA's fundamental concern. They are composed of a diverse set of fields; a field's contents depends on its type. For example, a "TO" field contains a list of addressees, a "TEXT" field contains a succession of uninterpreted paragraphs, while a "SUBJECT" field can only have a single line of text (of arbitrary length). Although AUTODIN traffic is its primary focus, SIGMA also supports formal in-house communications (*Memos*) and informal messages (*Notes*). They differ slightly in the fields they contain and the ways in which SIGMA processes them.

Folders, entries, and selectors

Folders are the users' basic mechanism for organizing and storing collections of messages in SIGMA. They contain *en-*

tries which are pointers to messages. A folder entry, an abstract of its message, contains information such as the message's precedence, security, sender, type and subject, which are considered by SIGMA as attributes of the entry. An entry for an incoming AUTODIN message might look like

```
22 R UU Auto 04222Z DEC 78 From:
    JCS WASHINGTON DC INCOMING Act: J3
    Subject: AIRCRAFT INFORMATION
```

The above entry is number 22, with routine (R) precedence, unclassified (UU) security, AUTODIN type, whose date/time is 04222Z DEC 78, etc. When a folder is DISPLAYed (the capitalized part of verbs are SIGMA commands), a numbered list of entries is put on the terminal's screen.

For use in commands, entries from folders can be identified in three ways: 1) By their number, 2) by default to the current entry, 3) by HEREding the entry number, i.e., putting the terminal cursor into an entry and depressing the "HERE" key. With this scheme most folder entry commands—DISPLAY, DELETE, FORWARD, etc.—have three forms. Using DISPLAY as an example, they are

```
DISPLAY ENTRY entry-number
DISPLAY ENTRY
DISPLAY NEXT ENTRY
```

The first is a typed command. The second is a function key which can take a HEREd entry; without one it will display the current entry. The last is also a function key. Some commands like DELETE and FILE (which copies entries from one folder to another) take entry lists, in addition to simple entry numbers, as parameters.

As objects themselves folders can be CREATED, DISPLAYed, DELETED, RESTORED (the inverse of DELETED), and FILEd into. In addition, user directories of folders can be VIEWed and assuming access is permitted, folders can be shared among users.

Often a user wants to extract from a file a class of entries which have some uniform characteristics. For example, he might wish to work on his messages according to their precedence. SIGMA provides *selectors* for this task. Selectors are boolean expressions composed of attributes of entries. When applied to folders they act as filters, returning lists of entries whose members satisfy their criteria. When the user has a folder displayed he can use the two selector commands, RESTRICT and AUGMENT, to change his display to exactly those entries he has selected. Thus after DISPLAYing a folder, a user can see all secret entries from CINCPAC by typing the command

```
RESTRICT SELECTION SECRET AND FROM
CINCPAC
```

The user can AUGMENT his display in a similar manner. If he now wants to add to this display those entries whose

precedence is routine, he types

AUGMENT SELECTION ROUTINE

AUGMENT and RESTRICT commands are "stacked" so that the user can always back up to his previous display using the BACKUP function key.

At any point during a sequence of RESTRICTs and AUGMENTs the user can CREATE a named selector which reflects the logical "ANDing" (for RESTRICT) and "ORing" (for AUGMENT) which led to the current state. In addition, he can CREATE a named selector directly by typing in the boolean as an additional parameter. DELETE, RESTORE, and VIEW commands apply to named selectors. A user's directory of named selectors can be VIEWed, and if access is permitted selectors may be copied from other users.

The richness of entry attributes makes selectors easy to use for creating relevant folder displays. SIGMA commands that operate on entries apply only to those entries currently in view, i.e., selected. So if entries 5, 15, 22, 27 were selected, four uses of the DISPLAY NEXT function key would step through only those messages.

Comments

Message fields and folder entries may be annotated with arbitrary text strings by means of the COMMENT command. Comments are identified by the user making them and have additional access properties; they can be public, private (to the commentor), or restricted to a named user. Comments are created by pulling a HERE in the desired message field or folder entry. The COMMENT command is then entered with the access specification and in response SIGMA will create a new field ready for editing.

Editing and text objects

Fields can be edited in two ways—by modifying the display with local editing functions provided by the terminal, and by various SIGMA commands. Suffice it to say the terminal provides a full complement of editing capabilities.⁸ In addition to those capabilities, SIGMA has its own editing commands. The PICKUP function key command deletes the characters between two HEREs, putting them into an unnamed buffer. The PUT function key inserts the contents of the same text buffer at the current cursor position. The MOVE function key, a composition of PICKUP and PUT, moves the text between the two HEREs to the current cursor position. COPY is the same as MOVE except the characters are not erased. These commands give the user the capabilities to erase, move and copy large amounts of text conveniently.

Text that is to be reused in messages or commands can be created and stored as a named *text object*. A text object is nothing more than a series of uninterpreted paragraphs, and can be CREATED and edited using all the capabilities

described above. A user directory of text objects can be VIEWed; they can be DISPLAYed, DELETED, RESTORED and, if the access is correct, copied from other users. Named text objects can also be used in conjunction with the PICKUP and PUT commands.

The content of a text object is unrestricted. No semantics are applied until it is put into an interpretable field. Once it is there, SIGMA acts on it depending on the type of field into which it was put. For example, text in an address list is checked for legal user names; when put into a field in which multi-paragraphs are allowed, the text is formatted.

SIGMA has not explored all the potential of text editing. So while it has a FINDSTRING command, it doesn't have one for text substitution. The experimental use of SIGMA at CINCPAC will provide feedback related to the adequacy of our editing model.

The SIGMA display

SIGMA divides the MME terminal screen into four windows. The FLASH window at the top of the screen contains three lines. The first is updated every minute and gives general operating information, time of day, and so forth. The FEEDBACK line tells the state of SIGMA processing and conveys error information. The STATUS line will be described below. SIGMA commands are typed into the two line COMMAND window below the FLASH window.

The remainder of the screen is the user's working space, which may be occupied by one or two windows. When a user DISPLAYs a folder, text object, or message, the object is "opened" and put into the EDIT window. If another object of the same type is already opened, then it is FINISHED, i.e., stored away with all new edits saved. If the open object is of a different type, then it is moved off the screen, though still opened, to make room for the newly DISPLAYed one. The STATUS line names all the open objects with the first name on the list identifying the one currently on the screen. Three function keys, SHOW FOLDER, SHOW MESSAGE and SHOW TEXT, can be used to put the appropriate object back into the EDIT window.

The VIEW window shows objects which the user names with the VIEW command. It is cleared by the CLEAR VIEW function key. This window is not editable and is used only for reference (text can, however, be copied from it). It is shown at lower intensity to distinguish it from the EDIT window. The EDIT window occupies the full screen when nothing is viewed; otherwise, both share the screen.

This section has given a functional view of SIGMA by describing its objects and some of its legal operations. The next two sections will give a more structured view of the tasks which compose message processing.

Incoming message processing

In the military, formal AUTODIN messages are sent from the commander of an organization to the commander of

other organizations, never between individuals within the organizations. This practice requires the receiving command to determine the appropriate recipients within the command for every incoming message. Naturally the correct assignment of recipients is a critical part of the incoming processing task.

CINCPAC employs a content-based scheme to determine the correct recipients. The first stage of this process is implemented by the LDMX message processor. By scanning the header and selected fields of the message contents, LDMX makes a preliminary assignment to the top management level (directorates). Although LDMX is capable of more detailed assignment, CINCPAC chooses to allow its directorates to perform their own routing internally. Within the MME target population (J3, the Operations directorate), this next level of routing is performed manually by an administrative office (J301). Using both catalogued tables of routing assignment and his specialized training, J301 scans each incoming message and determines its disposition. Such disposition can be any or all of the following:

Action	Typically each message is assigned to an <i>action officer</i> . He is responsible for any actions or response to be made by the J3 directorate, and is said to <i>have the action</i> for the message.
Info	In addition to a responsible officer, the contents of the message may be of interest to other officers as well. Such officers are said to receive an <i>information</i> copy of the message.
Readboard	Certain messages may be of interest to large groups within the directorate, and occasionally to all of J3. Such messages are placed in binders called <i>readboards</i> , which are then circulated through the directorate.

While the J301 assignment is generally accurate, it is neither complete nor infallible. An officer receiving a copy of a message may determine that other officers not designated by J301 should also receive copies. Occasionally the action assignment for a message is not appropriate; after seeing a message, an action officer may decide that another officer is better qualified to handle it and "sells the action" to him. Thus, the propagation of copies and/or action of a message may continue for several stages beyond the J301 assignment. Based on data compiled at CINCPAC, an average of 40 copies of a message is required to reach all its recipients.

SIGMA supports incoming message processing with a variety of facilities. They can be roughly divided into the three areas of delivery, reception, and redistribution.

Delivery

SIGMA was designed to merge naturally into the existing message processing milieu at CINCPAC. Through the special LDMX interface, SIGMA's Reception Daemon receives the text of incoming AUTODIN messages, parses them, builds the SIGMA internal representation, and stores the re-

sulting SIGMA-formatted messages in its message data base. To allow methodical retrieval of messages by arrival times, SIGMA also places an entry in a special folder called a *Date File*, a new instance of which is created each day to contain entries for all AUTODIN messages received during that day. A user can then see an index of all messages received on a particular day by simply DISPLAYing the corresponding Date File.

The high fan-out of incoming messages makes it impractical to provide a separate copy of each message to each eventual recipient. The scheme adopted in SIGMA was designed to minimize on-line storage requirements while still providing convenient access to messages. Messages are stored only once, in the central message data base. Each recipient receives not a copy of the message, but an abstract containing a useful subset of the message's contents. An instance of this abstract, called a *citation*, is created for each message transaction between users. Each citation sent to a folder causes a new folder entry to be appended (indeed, the terms *citation* and *folder entry* are often used interchangeably), a task performed by a special SIGMA background process, the Citation Daemon. A citation is small (approximately five percent the size of a message) and thus is much more economical to replicate than the full message.

All users access and modify the single copy of a message. Obviously such activity cannot occur unrestricted, or the integrity of the message contents and the users' intended changes could not be preserved. To allow such operations, a special scheme correctly assimilates parallel modifications, preserving both the consistency of the message and the users' intentions.⁹

Since messages flow into the J3 directorate constantly (approximately 1000 per day), the available secondary storage would soon fill unless appropriate steps were taken to reduce the number online. To make room for incoming messages, an archival scheme has been implemented. Using frequency of access as a rough guide, SIGMA moves inactive messages onto bulk storage (magnetic tape), from which users needing access to messages can request retrieval. Mechanisms are also provided to allow shorter retention periods for selected messages.

Reception

Once citations have been sent to a user, he must be allowed to see them, access their referenced messages, and dispose of them as he sees fit. These capabilities revolve around a repository for incoming citations, a special SIGMA folder known to each user as his *Pending File*. Analogous to a mail in-basket, a user's Pending File receives all citations destined for him.

Physically, a Pending File is implemented as a SIGMA folder, and can thus be manipulated by the wide variety of folder operations—DISPLAY of referenced messages, COMMENTing, cross-sectioning via RESTRICT and AUGMENT, etc.

Since all citations for a user are appended to his Pending File, he must eventually delete nearly all of them, lest he

exceed the folder size restriction (which is in excess of 6000 entries). This does not imply that a user must lose references to important messages, however, since a user may create an arbitrary number of other folders where he may FILE them.

Frequently, messages of great urgency to a user may arrive. In such cases the user would like to be notified immediately, rather than wait until he happens to notice it appear in his Pending File (which might be some time if he were DISPLAYing some other folder). To allow a user to specify criteria for incoming messages for which he wants immediate notification, SIGMA provides the Alert facility. The user activates the Alert facility by creating a SIGMA selector named ALERT_SELECTOR. If such a selector exists, each incoming citation is matched against it to determine if it meets the Alert criteria. If so, the citation is added to a special *Alert List*, the format of which is very similar to that of a folder, and the bell at the user's terminal is rung. The user can then display the Alert list without disturbing his open folder, and access any of the referenced messages in a manner similar to that for folder entries. In addition, a count recording the number of active alerted citations is maintained in the SIGMA flash line.

Redistribution

As explained in the description of the Delivery task, the routing provided by LDMX is not sufficient to reach all appropriate recipients. Additional routing is provided by the administrative J301 function, which supplies the bulk of the specific routing assignment, and by individual officers, who either supplement or correct the J301 assignments. SIGMA provides this flexibility by means of its redistribution facilities.

To effect the bulk routing assignment at the directorate level, SIGMA provides the ROUTE command. With this single command, J301 can specify action assignment, info distribution, and readboard creation for an entire group of messages. Using the RESTRICT and AUGMENT operations to select a class of similar messages, J301 can then perform a complete routing assignment for the whole class in a single step.

Individual officers needing to perform further redistribution have two more limited redistribution commands. The FORWARD command allows one user to send an information citation ('FOR_INFO') to another. The ACTION command is similar, but implies that the originating officer transfers the action assignment to the designee (by means of the 'FOR_ACTION' citation). Additionally, the ACTION command causes an entry to be placed in the issuing user's *Action Log*, a special folder which contains a record of all action assignments he has made. Since CINCPAC wishes to keep a central accounting of action assignments, users normally share a single Action Log (via SIGMA's shared folder capability).

All redistribution commands account for the further distribution of messages by appending records to certain message fields. In each message a *Distribution* field records each user who has received an info citation, while an *Action*

field records the full history of action assignments. Thus it is possible to ascertain all users involved in a message's redistribution by examining its appropriate fields.

Outgoing message processing

In the existing manual system, CINCPAC officers deal exclusively with so-called "record traffic." Even when the contents of a message are routine, the onus of representing an entire command's viewpoint adds a measure of importance. Consequently highly formalized procedures have developed at CINCPAC to ensure that messages transmitted from the CINCPAC organization have been thoroughly reviewed and approved by a responsible authority.

In addition to supporting an on-line implementation of the review/approval process, SIGMA has augmented the media of communication. In addition to the existing formal traffic (AUTODIN messages), SIGMA has added two new message formats, formal internal and informal.

- Formal internal messages (memos) are similar to content and form to AUTODIN messages, but the addressees are other SIGMA users. This provides CINCPAC personnel with a formal (recorded) medium to send official communications within the CINCPAC organization.
- Informal messages (notes) provide an off-the-record message medium for informal communication. Such messages, which are not reviewed or recorded, provide an alternative to face-to-face or telephone communication.

The outgoing message processing in SIGMA is roughly divided into four phases: drafting, coordination, release and transmission.

Drafting

During the drafting phase the original message is composed. The sources of the original contents of the body and various fields vary, depending on the type of message being prepared. SIGMA supports the following commands for message drafting:

- | | |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CREATE | An empty message form is created, with blanks for the editable message fields. The contents of any desired fields must be filled in by the drafter. |
| COPY | This command, which requires an existing message as a parameter, copies all of the non-header fields into the new draft message. It is useful for pro forma messages which are sent frequently and whose contents are basically similar. |
| REPLY | This command also takes a message parameter, and creates a new draft in reply to the subject message. In this case, the addressees |

are derived from the subject message, the subject is copied, and the referenced message cites are copied with a cite to the subject message appended as an additional reference.

Once he has created his draft message, the author has the following alternatives:

- He can save the draft for later use (via the FINISH function key). This can be done to hold an incomplete draft for later completion, or to save a pro forma message to make it available for later COPYING. To make later retrieval of such saved drafts convenient, SIGMA puts a citation referencing the draft in the user's Pending File, so its existence is remembered and it remains easy to access.
- He can send the message for review if it is a formal message. This process, called Coordination, will be described in more detail.
- If authorized, he can cause the message to be transmitted to its addressees. This will be described in the section on Release.

Coordination

Within CINCPAC there exists a formal procedure for review and revision of a message prior to its release. The drafter can request several other officers to review the message, make comments, suggest changes and give a general disposition regarding the message. This procedure, called *chopping*, permits CINCPAC to acquire a consolidated opinion from a cross-section of responsible officers before a message is sent.

SIGMA supports a style of coordination more general, although perhaps less flexible, than the manual CINCPAC procedure. The message drafter may designate any number of users in a field called the *Chop List*. With the special COORDINATE command, the drafter can specify that any or all of the users on the Chop List be requested to act as reviewers for the message (they are referred to as *coordinators*), causing a special "FOR_CHOP" citation to be sent to each of the designated users.

A coordinator is notified of the drafter's request to review a message by receipt of the FOR_CHOP citation in his Pending File. He can display the message, and will see the drafter's most recent version. The coordinator can make comments or suggest revisions to the message; if so, the changes are not applied to the drafter's copy, but rather to a copy belonging solely to the coordinator. In deciding upon his changes he has access not only to his own version and the drafter's, but to other coordinators' as well.

When a coordinator decides that he in turn would like comments from other users (perhaps subordinates or other colleagues), he may further designate other coordinators. This "sub-coordination" is exactly analogous to that initiated by the drafter. In this case his sub-coordinators see his version of the message when they first display it.

When a coordinator has finished his review of a message

he may indicate his global disposition of the message by means of the CHOP YES and CHOP NO function keys. These commands cause a "CHOPPED" citation indicating the appropriate disposition to be sent to the drafter (or higher level coordinator), who is thereby notified that this coordinator has finished his review.

During the coordination process, the drafter (or a higher-level coordinator) can monitor the coordination process by means of a status field, which indicates the progress of each coordinator. When coordinators have finished their reviews he can view their versions and note suggested changes and comments. He can incorporate changes by duplicating them in his own version or by copying the changed sections from the coordinators' versions. If he is not satisfied with the resulting message or wishes to elicit further review, he can initiate another coordination cycle, which will result in additional FOR_CHOP citations being sent. If the drafter is satisfied with the content of the message, he can initiate the Release process.

Release

Because of the formal nature of record communication, certain officers, designated *release authorities (releasers)*, are solely empowered to approve outgoing record traffic. SIGMA provides the same enforcement by checking each attempt to transmit a message against a list of authorized releasers (since the three different message formats have different levels of formality, a separate list is maintained for each).

When a drafter has determined that a draft message is ready for transmission, he must gain the approval of an appropriate releaser (unless he himself is one, in which case he can release it himself). He does this by using the COORDINATE command after designating the releaser's name in a special *Release* field, causing a "FOR_RELEASE" citation to be sent to the releaser. After receiving this citation a releaser has options similar to those of a coordinator. He can display the drafter's and coordinators' versions, seeing comments and suggested changes. In particular, he may examine the "chop" disposition of the various coordinators to determine whether he is satisfied that there is sufficient agreement among them. If he is not satisfied he can make his own comments and changes and specify CHOP NO, in which case a citation is sent back to the drafter. But if the message is in order and the releaser is satisfied, he can initiate transmission via the RELEASE command, whereupon the message leaves the preparation phase and is sent for transmission processing.

Transmission

When approved by a releaser, a draft message is prepared for transmission by the SIGMA process called the Message Daemon. First the draft is marked as transmitted; this prevents it from being further modified or transmitted again. A new message is then created to contain the transmitted ver-

sion of the draft message. Fields which are appropriate for transmission are copied from the draft; others which do not belong in transmitted messages (such as comments, chop lists) are omitted.

When the contents of the transmitted message are prepared, the appropriate transmission medium is determined. If the message is destined for AUTODIN, the message is sent through the LDMX interface to be transmitted to the AUTODIN network. If it is an internal message, the transmitted message (in internal SIGMA format) is entered into the SIGMA message data base, and "INCOMING" citations are sent to each of its addressees.

CONCLUSIONS AND USER REACTIONS

Our initial opinion after studying the CINCPAC environment was that an interactive message service could be extremely effective. The CINCPAC staff was enthusiastic about the possibilities and endorsed the experiment to the point that they were willing to serve as the test-bed for it. Now the experiment is underway and we are beginning to learn whether our optimism has been well founded.

Although at the writing of this paper formal results are not yet available, CINCPAC users have been using the service for six months. They have already asked for changes and extensions to the service; some, like ROUTE, have been implemented. As expected, the use of such a service is altering the style in which many officers operate.

Probably the most dramatic effect is on J301 who previously required seven hours to process the new messages that arrive overnight. Using SIGMA this process is reduced to less than an hour-and-a-half. Furthermore the feeling is the assignments made are generally better, primarily because the same assignment is made to entire classes of messages at once, thereby assuring uniformity.

Another group of users that has been heavily influenced by SIGMA normally get their messages from J301 about 9:00 AM, two hours after they come in in the morning. They have found that with SIGMA they can go directly to the Date Files for the day and, using Selectors, get the messages of interest without waiting for J301 to distribute them. They are also able to find messages requiring their action that have been assigned incorrectly to others, messages that they simply never saw before SIGMA was available to them.

There are still many improvements requested by CINCPAC users which SIGMA has not yet addressed. Indeed, the list is already large even at this early date:

- The ROUTE command was put into SIGMA in response to a direct request from J301. Other composite commands can be visualized. It would be nice to have a powerful facility for building such "macros" from existing commands. However, such a feature touches heavily on many difficult user interface issues.

- The ALERT mechanism is fundamental but acts only on incoming messages. Some users have expressed interest in a general facility based on a variety of different events.
- Users have expressed a desire for the ability to search the full message database with a mechanism like selectors. SIGMA has no model to support this expensive operation at this time.

Although the experiment is just beginning to collect useful information, it is clear that SIGMA is having an impact on the message processing at CINCPAC. SIGMA appears to be rich and flexible enough to support the goals of the experiment to gain insight for future military message systems. As the users become more involved with interactive message handling their awareness of its capabilities and potential is being sharpened and their requests for functional enhancements are more accurately based on realistic needs.

The injection of a research project, like SIGMA, directly into an operational military environment is an unusual event. This approach offers the military a more active role in developing relevant software for sophisticated applications. The MME effort is showing that the transition from the laboratory to an operational setting can be accomplished for such an experiment, which should dramatically shorten the normal technology-transfer path.

REFERENCES

1. Ames, S. R., and W. W. Plummer, "TENEX Security Enhancements," MTR-3217, MITRE Corporation, April, 1976.
2. Ames, S. R., and D. R. Oestreicher, "Design of a Message Processing System for a Multilevel Secure Environment," *Proceedings of the National Computer Conference*, AFIPS, 1978.
3. Bell, D. E., and E. L. Burke, "Secure Computer Systems: Mathematical Foundations and Model," M74-224, MITRE Corporation, October, 1974.
4. Bobrow, D. G., J. D. Burchfiel, D. L. Murphy, and R. S. Tomlinson, "TENEX, a Paged Time Sharing System for the PDP-10," *Comm. ACM*, Vol. 15, No. 3, March 1972, 135-143.
5. Heafner, J. F., and L. H. Miller, "Design Considerations for a Computerized Message Service Based on Tri-Service Operations Personnel at CINCPAC Headquarters," Camp Smith, Oahu, ISI/WP-3, USC/Information Sciences Institute, September, 1976.
6. Memorandum of Agreement between Director, Defense Advanced Research Projects Agency (DARPA), Commander Naval Telecommunications Command (NAVTELCOM), Commander Naval Electronics Systems Command (NAVELEX), and Commander-in-Pacific (CINCPAC). Unpublished memorandum.
7. Rothenberg, J., "On-Line Tutorials and Documentation for the SIGMA Message Service," *Proceedings of the National Computer Conference*, AFIPS, May, 1979.
8. Stotz, R., P. Raveling and J. Rothenberg, "The Terminal for the Military Message Experiment," *Proceedings for the National Computer Conference*, AFIPS, May, 1979.
9. Tugender, R., "Maintaining Order and Consistency in Multi-Access Data," *Proceedings of the National Computer Conference*, AFIPS, May, 1979.

This research was performed for the Advanced Research Projects Agency under Contract No. DAHC 15 72 C 0308, ARPA Order No. 2223. The views and conclusions expressed in this paper are not necessarily those of any person or organization except the author(s).

The SIGMA experience—A study in the evolutionary design of a large software system

by DAVID WILCZYNSKI and RONALD TUGENDER

*USC/Information Sciences Institute
Marina del Rey, California*

and

DONALD OESTREICHER

*Xerox Corporation
El Segundo, California*

INTRODUCTION

Anyone who has been a part of a large software effort knows of its peculiar afflictions and special problems. The literature teems with guidelines, rules and conventions for designing and managing such systems; in fact, there are probably more books written about large systems than there are large systems. This paper will make no attempt to add to this literature; instead it will simply report our experience over the last five years in the design and implementation of SIGMA.³ This introspective study is meant to be not a pedagogical paper but a reflective, often humbling, diary.

SIGMA is the interactive message processing system built at the University of Southern California Information Sciences Institute (ISI) for the Military Message Experiment (MME).² It is currently in active use at Camp Smith in Hawaii, supporting about 100 users, of which 24 may be concurrently online. SIGMA comprises about 270 modules totaling some 2500 routines. This makes approximately 200,000 lines of source code or some two edge-feet of listings. The point, of course, is that SIGMA is big by any metric. The software was designed and written by a revolving group of five to seven expert programmers, making SIGMA about a 30 man-year program. This is by no means a gigantic effort by industrial standards, but large when judged by research community criteria.

Our experience should not be taken lightly. Deadline pressures notwithstanding, the ISI environment is conducive to high-quality output. Each project member has a private office, an HP2640A CRT terminal, access to an inhouse library and all the facilities one expects at a computer research center. Our successes or failures are generally a result of our own ingenuity and intelligence or lack thereof, respectively.

SIGMA ARCHITECTURE OVERVIEW

The SIGMA message service runs on a Digital Equipment PDP-10 under the TENEX timesharing system¹ and is written in the BLISS system implementation language.⁶ The SIGMA system is divided into two functional areas: the user jobs, which interact with the message service users; and the daemons, a collection of background processors that perform non-interactive functions.

The SIGMA user job

An instance of the SIGMA user job is created for each user who logs into the system. As seen in Figure 1, the user job is composed of five major components. The Terminal Driver interfaces the specially modified HP2649A terminal⁴ to the rest of the user job. The Command Language Processor (CLP) reads command input, parses it, builds a command specification called an Execution Request Block (ERB), and passes it to the Functional Module (FM), through a protocol called EC99. The FM is responsible for the actual execution of commands, and thus it has two main tasks: to manage the display of information on the terminal, and to manipulate SIGMA's objects.* The former task is performed by a module called the Virtual Terminal (VT), which builds and maintains display lists for the terminal. The latter function is done by the FM directly for text objects and selectors, and by two

* SIGMA supports four kinds of objects. *Text Objects* are lists of uninterpreted paragraphs. *Messages*, of which there are several kinds, are lists of various message fields. *Folders* are lists of message citations; a citation is an abstract of its associated message. *Selectors* are boolean expressions applied to folders in order to access only those citations whose attributes match those named by the selector.

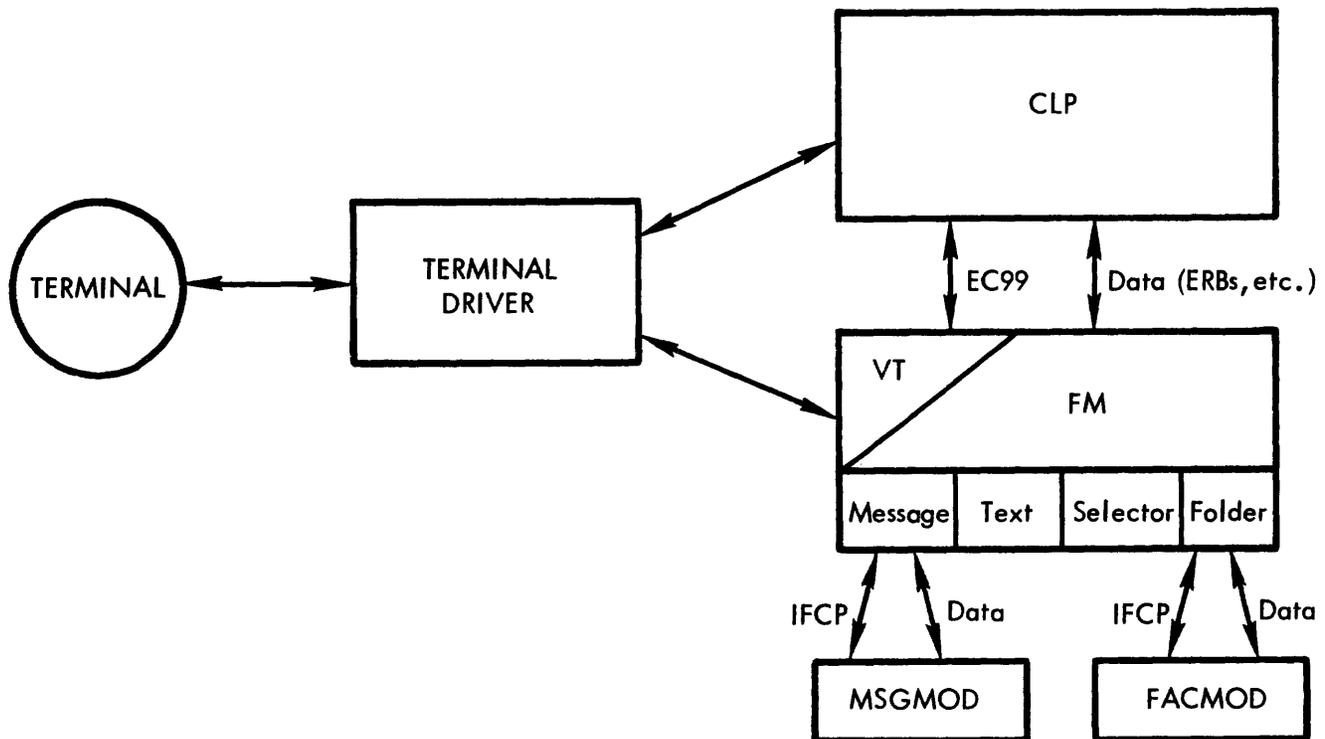


Figure 1—SIGMA user job.

special-purpose modules called the Folder Module (FACMOD) and Message Module (MSGMOD) for folders and messages, respectively. Because of address space limitations, FACMOD and MSGMOD are located in separate TENEX forks (processes) and require a special Inter-Fork Communication Protocol (IFCP) to communicate with the FM.

The SIGMA daemons

The SIGMA daemons are responsible for tasks that require no direct user interaction and can thus be performed in background. These include management of the shared data bases (messages and folders), message reception and transmission, archive retrieval, and printer spooling. The daemons process requests received through input queues; the source of such requests may be the user jobs or other daemons. To provide operations personnel with the ability to control the daemons, a Configuration Control Program (CCP) communicates operator requests to daemons.

A SIGMA TIMELINE

Figure 2 represents our first effort at producing a milestone diagram for SIGMA's development. This event-oriented chart provides a general impression of how SIGMA developed.

The project, which started in September 1973, was first

called Information Automation (IA). Approximately one year later a series of six IA papers were produced defining the components of what was to be SIGMA and by January 1975 a full system design was published. This 200-page document represented about ten man-years of effort culminating the period we now call "designing in a vacuum."

The first SIGMA, which we will call SIGMA 0 (although it was both unnumbered and unnamed), made its debut a year later in December 1975. It didn't do much, but it did have a front-end that talked to users, a minimal FM that executed a few message manipulation commands, a terminal simulator that was to be functionally equivalent to the terminal being designed for our project, a primitive debugging mechanism, and a simple text editor.

The next year was spent getting SIGMA 1.0 ready for a system evaluation which was to take place in Boston in February 1977. This pubescent period saw SIGMA take on functional substance. The first daemons were written; the concepts of folders, text objects, and selectors were implemented; and the FM was rewritten to encompass all the new objects. SIGMA 1.0 was slow and bulky, but showed enough promise to be the service selected for the Military Message Experiment. We knew even then that the daemon design and implementation were hopeless, but other things had to be done first. SIGMA's performance had to be improved.

Performance was the project watchword for SIGMA 1.75. Since the functionality had become stable, we were able to carefully analyze the system's flow of control, data paths, communication demands and so forth, a study which led to

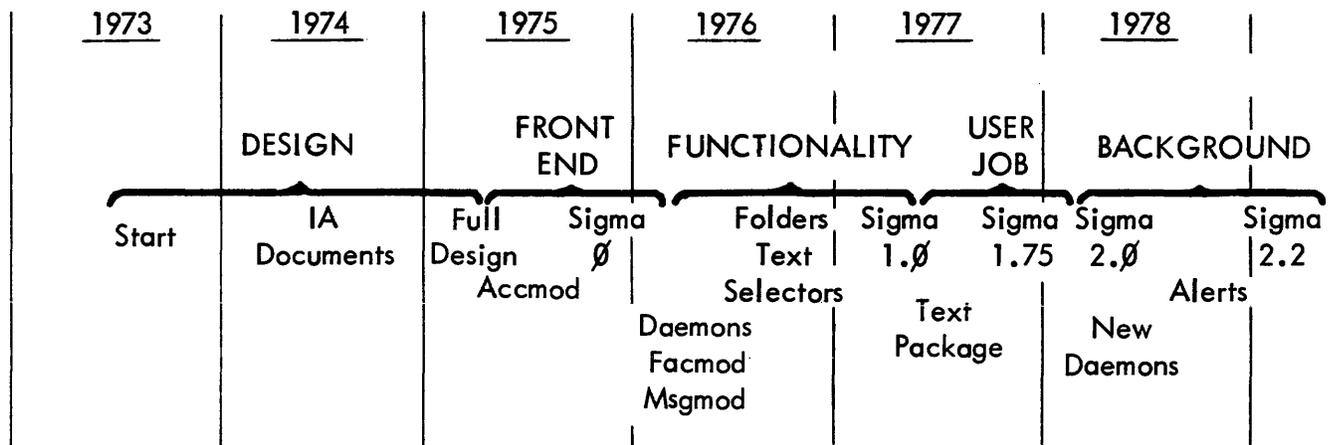


Figure 2—A SIGMA timeline.

optimizations spanning all parts of the user job. This performance transition was particularly trying. Still, by December 1977 it was running at an acceptable speed. The changes we made were so dramatic that SIGMA 1.0 seemed like ancient history.

The time had finally come to deal with the daemons. Their original design was based on principles that led to an overly-complicated implementation whose data flow can euphemistically be described as baroque. The irony is that by SIGMA 1.75 we realized that all the requirements that led to this design were either too expensive or superfluous. This time the daemons were redesigned and rewritten based on our experience of what background support was actually needed.

The new daemons had a great effect on our project. The 1.75 daemons were hopeless to maintain, difficult to debug, and resistant to change; the 2.0 daemons of July 1978 were elegant by every programming standard. SIGMA could finally be called a mature system. Now and only now could we think about functional enhancements that involved the daemons.

The remaining part of 1978 was spent preparing for SIGMA 2.2. Among the new features was Alert processing, an addition which made the online user immediately aware of activity in his personal pending file. The Alert concept had been knowingly lacking since SIGMA 1.0 but was not practical to implement until we redesigned the daemons and had a stable user job.

While in this timeline we seemed to be responding to individual and isolated problems as they appeared, a more abstract view of the development reveals a coherent top-down design. The first year we designed a message service in great detail. The first SIGMA was produced in year two with the emphasis on the front-end components, the CLP and the terminal. The functionality, defined and implemented during year three for SIGMA 1.0, was demonstrated in Boston. Following that came the architectural wirebrushing of the user job, culminating in SIGMA 1.75 in year four. The new functional daemons came during the fifth year together with a round of improvements for SIGMA 2.2.

Even though we actually followed a sensible path in SIGMA's development, our failure to realize what was happening proved to be both costly and frustrating. No doubt all retrospective analysis is concise and relevant; still, early recognition of certain principles would have prevented the chaotic nature of our progress. The next section will present those principles as a pseudo-mathematical theory of software development.

LARGE SYSTEM DESIGN THEORY

Now that SIGMA has matured we compared its current design and implementation with the one planned in the initial system design. This comparison shows that about the only thing that survived all those years is the system diagram found in Figure 1. There is something futile about the first axiom:

Axiom 1—Early system design should identify only large namable components.

Perhaps this is the best that you can do when designing in a vacuum. SIGMA 0 wasn't very good but it did show that our top-level design was sound. Lacking some sharp insight we believe that:

Axiom 2—A complete running system is necessary to verify even a top-level design.

The implications of this axiom are severe. When the system is being put together, a placeholder is needed for every component regardless of expense. We knew, for example, that even though the SIGMA terminal was scheduled for late 1976, we needed a simulator for it in order to design the CLP and FM. The man-years spent on this known throw-away piece of software were eminently worthwhile, substantiating the following:

Theorem 1—Every major component of a design must be implemented in some form.

Our FM in SIGMA 0 produced the necessary results. It showed that the CLP/FM interface was sound and the terminal model was maintainable by the VT. Even though the

FM barely worked, nothing would have been learned if it didn't run. Trite as it may sound:

Corollary 1.1—All implemented components must work.

SIGMA 0 showed us that some of our high-level concepts were designed correctly. However, the implementation experience also taught us other things.

The original FM had too many features, among them a concept called Placemarks. These were named locations within message bodies that the user could assign and address for various purposes. They were hard to implement, and in fact did not survive the transition to SIGMA 1.0. Placemarks may or may not have proved useful, but this was not the time to find out; suffice it to say that much valuable time was wasted here.

ACCMOD was the first complete message manager for SIGMA. It had a data base model for the way it serviced the FM. In other words, it owned the message being displayed and was involved in every facet of its editing. Besides just having performance problems, ACCMOD was a huge, unwieldy piece of code. In the simpler file model, MSGMOD gets the message to the FM who "owns" it during the editing session, and then MSGMOD stores it when the user is done. The important knowledge we obtained from ACCMOD was that its model was wrong. However, a simpler implementation would have shown this as well.

One thing we designed right was the communication of common data between the forks of the user jobs. Data (such as who was logged on, the ID of the open message, etc.) was kept in memory shared between them. The effect was like a FORTRAN COMMON block. This simplest of interfaces served so well that we still use it as originally designed. Of course, we understand its limitations, but we haven't reached them yet.

This Placemark/ACCMOD/Common experience shows clearly that a project should:

Theorem 2—Start small in early design and implementation phases.

It is hard to overestimate the pragmatic value of **Theorem 2**. Its realization leads to a natural evolution in which the designers can cheaply reflect on their basic ideas and perhaps modify them before they run out of funds or energy. We should note that there were no daemons at this time. We were still designing them as if they would suddenly appear like Pallas Athena, springing full-grown from the forehead of Zeus. An understanding of **Theorem 2** would have led us to a more conservative first-pass design.

The work which led to SIGMA 1.0 represented a new direction for the project, one that was not fully appreciated at the time. SIGMA 0 verified our front-end model; SIGMA 1.0 would define the system's functionality as needed by the FM. Had we realized that this was the real effort, we could have avoided the complexity found in the daemons. In other words:

Axiom 3—You can't aim high on every component at the same time.

Many things happen during a design cycle: Ideas are tested, system modules are inspected, some things work well, and some things are thrown out. Typically, some things stabilize. The CLP did by the time SIGMA 1.0 was released.

The FM was beginning to at the same time. Unfortunately, what had stabilized throughout the user job was our text-handling routines called ZT. ZT was a loose collection of low-level functions. Given these primitive routines each implementer developed his own set of macros and functions, all based on the ZT structure. By the time we realized its performance and design implications, ZT was hard to remove from the user job. The TEXT package that replaced ZT simplified SIGMA by imposing a coherent text-processing model on the project. What we should have realized is:

Axiom 4—Recognize which component is stabilizing during a design cycle and aim high for it.

These two axioms are the heart of our evolutionary design theory. Even with unlimited resources, we believe a project will do better to focus its attention on one major component at a time. Some scientific reasons, ranging from interface issues to manpower expenditure, are easy to generate. But one which is often missed is that very little is learned when an overdesigned module is rewritten or modified—it is better to build up from minimum capabilities than tear down from unneeded ones. This "hourglass design," in which one starts high and must then strip away features before a redesign is possible, is a painful way to develop a system.

Consider the daemons. Their original design was based on four requirements:

1. SIGMA would run in a distributed network environment.
2. High-priority requests would need to be processed quickly.
3. Individual long requests could not be allowed to hold up the daemons.
4. Error results must be returned to the user in a synchronous mode.

The ramifications of these early requirements were severe. The first implied a logon procedure so that the daemons knew the location of its users. This meant that when the daemons went down so did all the users, bad in operational use in Hawaii and hopeless for development at ISI. The second requirement led us to an implementation of duplicate processors with different priorities in case a high-priority request came in. The "multiprocessor" environment was also prepared for the third requirement in which a request could monopolize a processor. Finally, returning requests to the user, as necessitated by (4), implied a single output process for distributing results.

By SIGMA 1.75 all those requirements were shown to be spurious and the code supporting them was removed. What was left were daemons with the barest capabilities. From them together with our experience with 1.75 we were able to redesign and rewrite the daemons to be elegant and functionally relevant in just four months. This lesson is our first major result:

Theorem 3—Avoid the hourglass design syndrome.

Figure 3 depicts this theorem graphically. The syndrome is seductive and debilitating. The natural ego of a designer/programmer drives him to build to the hilt. We've all seen this happen in design meetings. A few capabilities are clearly required; some others are suggested and incorporated. Once

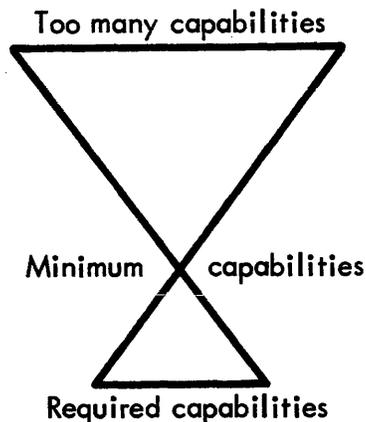


Figure 3—The hourglass design.

rolling, this juggernaut is hard to stop, and a concise design suddenly belongs to a committee. PL/I is an old example; SIGMA's former daemons are our contribution.

FACMOD is an example of how smoothly evolution can take place. Folders contain message entries and are the place from which messages are displayed. Besides just being displayable, entries can be keyworded, deleted, filed into other folders, commented, selected by various criteria, etc. Instead of building in all these capabilities, the first FACMOD had only the barest set. FACMOD came up quickly and grew with our needs. Naturally, many of the requirements we foresaw were added later, but many were not. The message is clear:

Theorem 4—Underdesigned components are needed during a system's evolution.

We have talked about system design in terms of focus on mainline modules, aiming high in selected pieces, underdesigning others. *Axiom 4* also alluded to stabilizing factors within all parts of the system. Those are important to watch for. As important are sections that are critical for one reason or another. Two examples in SIGMA are the IFCP package and the Terminal/Terminal Driver communication protocol.

The IFCP is the data channel between the FM and both MSGMOD and FACMOD. It was independent of application and unlikely to change regardless of the direction of any of those components. The IFCP had to be robust and solid; it was serving many masters. It was recognized as a critical component of SIGMA and designed accordingly.

The terminal protocol raised different issues. Even though the transmission lines at ISI between the terminal and the Driver were virtually perfect, we knew that lines at other installations might not be. Without knowing how bad lines could be, we anticipated the worst and built a robust protocol that included acknowledgments, timeouts, retransmissions, and checksums. Our experience in the Boston review and in Hawaii made us glad that we did. So, underdesign is good, but:

Theorem 5—Aim high for critical sections.

Part of aiming high is to understand in detail what is expected of the component under examination. Before the daemons were redesigned in aim-high mode, a firm require-

ments specification was written for them. This document gave our design and implementation a well-defined target. The VT never has received this treatment and remains loosely organized. It seems that part of aiming high means that:

Corollary 5.1—Firm requirements are needed for mainline components.

Requirements are one thing, and documentation is another; the former precedes implementation while the latter follows it. Once a major component of the system has been written, complete documentation should be generated for it. Much lip service is paid to this liturgy; we are no exception. All the good reasons for documentation are easy to list but one that is understated is that undocumented components take on an orphan flavor if the writer leaves. This happened to the VT. It passed from hand to hand, each programmer leaving his mark, but no one documenting anything. So:

Corollary 5.2—Fully document mainline components as they are written.

As important as **Theorem 5** and its corollaries are, their duals are just as important:

Theorem 6—Aim low for noncritical system components.

Corollary 6.1—Noncritical components need only tentative requirements.

Corollary 6.2—Small documentation effort should be spent on noncritical components.

Noncritical system components should get aim low treatment—the daemons and ACCMOD should have but didn't, FACMOD and COMMON should have and did, the VT and ZT shouldn't have but did. The technical issues here are not controversial. However, the psychological one of producing work that is less than your best arises. Perhaps a good manager should assign his best programmer to produce such code, since ego is less likely to be a problem. Even though the code is known to be throw-away, it still has to work (Corollary 1.1).

The theme of aim-low also brings out questions of performance. The development leading to SIGMA 1.0 paid virtually no attention to performance. We used flexible data structures, clean interfaces between modules, and straightforward coding techniques. SIGMA 1.0's performance was poor but not unexpectedly so. When performance became an issue, we analyzed SIGMA using Program Counter (PC) samples, a technique that takes snapshots of running code to tell what code is being executed. We found, for example, that our storage management package needed tight optimization, since it was active all the time. That was expected, but we also found that some heavily used operating system facilities took several orders of magnitude more time than we expected. This stunning information taught us that:

Axiom 5—A priori, it is impossible to know in which sections of code performance will be critical.

With the PC samples as a guide optimizing SIGMA's code in the user job was straightforward: the character output to the terminal was rewritten at a 10-to-1 saving, the FM's folder handling was changed from linked lists to an array structure, FACMOD and MSGMOD calls were tuned to specific needs, and a TEXT package was incorporated to replace ZT. These changes were extensive and revolution-

ary; still, they didn't affect the overall design. Assuming that the early design gets you within one or two orders of magnitude of your target, we believe that:

Axiom 6—Performance is "easy" to add later.

Once all these changes were made, we had a much faster, sleeker, less flexible User Job. At this point another PC sample test showed that the Text package had an inefficient look-up routine that needed recoding. *Axiom 5* tells us about anticipating performance, but here we have a case where the code didn't even exist at design time. The evidence is clear:

Theorem 7—Performance should not be an early design goal.

This theorem promotes an ideal which served SIGMA poorly. In following it we found ourselves with a SIGMA 1.0 that was very slow when compared to one of our competitors during the Boston review. The "fast" system did not have the capabilities of SIGMA, although that fact together with the above performance principles was lost on some (fortunately not all) of the reviewers. The performance of SIGMA 1.75 vindicated our development strategy.

Until now the discussion has addressed design theory with an undercurrent of implementation examples. With the evolutionary principles we are proposing, design and implementation are inextricably tied together. Now the focus will shift to the implementation side.

The literature rightly pays much attention to storage handling, searching and sorting, queue modeling, etc., because of their widespread use. To have poorly implemented packages of this sort is ludicrous, whether you are aiming high or low. Besides avoiding code duplication, these modules have a unifying effect on a system. Our careful implementation of free storage, lexicons, and queues were based on the principle that:

Axiom 7—Good support packages are essential and well understood.

Support tools are important, but a good supporting environment for testing is essential. From the beginning we realized that since BLISS had no debugging facilities we would have to provide our own. A system-error package was built to take over control when an error was detected. The programmers had an ASSUME (boolean,message,data) statement available that invoked the system-error mechanism if the boolean was false. The package was small when we started (no hourglass design here) and grew as SIGMA evolved. Note however, that small and aim-low are not the same:

Theorem 8—Aim high on debugging tools.

It is time to formalize an assumption that conveys our theory. We have been cavalier about the implementation effort required by our evolutionary model. Yet each design cycle, whether it was the CLP, the FM or the daemons, was followed by a painless implementation phase. Once understood:

Theorem 9—System components are easy to build with good support tools.

This optimistic theorem runs contrary to prevailing, perhaps self-serving, opinion. Our contention that well defined modules can be cranked out comes from innumerable cases,

from both SIGMA and other sources. It's just not hard to build "one." Yet what we have tried to show throughout this paper is that a large system needs to evolve, since many decisions can be wrong. We have shown how things can be designed or modeled poorly.

Even knowing where to put functionality can be a problem. As an example, the original SIGMA design included a component called the Personal Daemon (PD). An instance of this background process was created for each user job, and its intended purpose was to provide a background processing capability which was active even when the user was not. Alerts were originally designed to be a PD function. Since we neither knew all the potential functions of such a process nor had the time to develop them, we (correctly) gave it aim-low treatment. As the design of SIGMA was reworked from 1.0 to 2.2, each of the existing hypothesized PD functions was reassigned to another area. With nothing left to do, the PD was removed. Unfortunately:

Axiom 8—Often you guess wrong.

Every component of SIGMA has been rewritten. Large pieces of code were abandoned: ACCMOD (twice), the FM, the daemons, the PD. It is the nature of the evolutionary design theory that:

Theorem 10—Almost everything gets thrown away.

Even aim-high modules can be re-examined and thrown away. SIGMA had a directory scheme for storing and retrieving messages based on a lexicon package (height-balanced AVL trees). It worked from the beginning and never was looked at until our performance cycle. When it got dumped for a simpler scheme we knew nothing was sacred:

Corollary 10.1—Don't be afraid to throw stuff away.

Once this fear is conquered, a lot of pressure is removed from a design/programmer staff—certain kludges are accepted, simplicity encouraged. The aim-low paradigm coupled with **Theorem 10** means that each component will get a second chance during an aim-high cycle. The CLP got it by SIGMA 0. Functionality made the FM the focus of SIGMA 1.0. The entire user job got aim-high treatment for 1.75. Finally, the daemons were the target for 2.0. Each cycle left a little more of SIGMA hardened, i.e.,

Corollary 10.2—Every development cycle will focus on a permanent component.

Even though we didn't realize it at the time, SIGMA followed an orderly development cycle. It would be nice to attribute this outcome to our careful long-range planning, but that is not the case. The things we did were circumstantially good or bad. Performance, redesign, and new requirements all imply that a large system is a moving target. Simply stated, the first fundamental result is:

Theorem 11—No one can implement a large system in one pass.

It is seductive to think that you can. Even more seductive is the notion that a long, studious design is the answer; we tried and failed. Every aspect of SIGMA proved the statement made by our last theorem:

Theorem 12—A large mature system must evolve; it cannot be designed.

This second fundamental result is borne out by all large

systems. There is no good reason to suspect that anyone can design a big system down to the bit level using any known methodology.

CONCLUSIONS

The theme of this paper is applicable only to large, multi-year projects. Though many of the principles are relevant to any effort, the theory applies to a "design a little, implement a little, design some more, implement some more . . ." paradigm, rather than the "design it, implement it" one. The most important early decision is to recognize which model is appropriate. Remember the two fundamental theorems.

A relevant issue, not previously mentioned, is the choice of a programming language for a project of SIGMA's magnitude. When we faced this decision, INTERLISP⁵ was brought up. Though it comes with a marvelous programming environment, its lack of speed and address space problems made it impractical to use. Once BLISS was chosen for SIGMA, all code was written in it. Perhaps a better scheme would have been to write an aim-high interface, in the IFCP manner, to exist between forks written in BLISS and others written in INTERLISP. Then aim-low modules, perhaps written in INTERLISP, could have been put together very quickly. Using this strategy implies a strong commitment to this design philosophy.

We all guess wrong (*Axiom 8*) but perhaps the main reason that this theory is not in general use is that people don't like to reprogram modules they have coded before. But a large

software project has many people on it, so the solution is obvious. Let the aim-low implementation drive the design; when aiming higher, let someone else implement it. The daemon and FM rewrites were perfect illustrations of this point.

As with top-down programming, egoless programming, or chief programmer teams, no magic is offered. This theory is not a panacea for all software problems. Lousy designs and poor programmers will still defeat any methodology. What we do offer is a guess as to how to make a long-term project less painful and more rewarding.

REFERENCES

1. Bobrow, D. G., J. D. Burchfiel, D. L. Murphy and R. S. Tomlinson, "TENEX, a Paged Time Sharing System for the PDP-10." *Comm. ACM* Vol. 15, No. 3, March 1972, pp. 135-143.
2. Memorandum of Agreement between Director, Defense Advanced Research Projects Agency (DARPA), Commander Naval Telecommunications Command (NAVTELCOM), Commander Naval Electronic Systems Command (NAVELEX), and Commander-in-Chief Pacific (CINCPAC). Unpublished memorandum.
3. Stotz, R., R. Tugender, D. Wilczynski and D. Oestreicher, "SIGMA: An Interactive Message Service for the Military Message Experiment," *Proceedings of the National Computer Conference*, AFIPS, May, 1979.
4. Stotz, R., P. Raveling and J. Rothenberg, "The Terminal for the Military Message Experiment," *Proceedings of the National Computer Conference*, AFIPS, May, 1979.
5. Teitelman, W., *Interlisp Reference Manual*, Xerox Palo Alto Research Center, December, 1975.
6. Wulf, W. A., D. B. Russel and A. N. Habermann, "BLISS: A Language for Systems Programming," *Comm. ACM* Vol. 14, No. 12, December 1971, pp. 780-790.

This research was performed for the Advanced Research Projects Agency under Contract No. DAHC 15 72 C 0308, ARPA Order No. 2223. The views and conclusions expressed in this paper are not necessarily those of any person or organization except the author(s).

The terminal for the Military Message Experiment

by ROBERT STOTZ, PAUL RAVELING and JEFF ROTHENBERG

USC Information Sciences Institute
Marina del Rey, California

INTRODUCTION

One of the technical challenges faced in the Military Message Experiment (MME) is providing a system that is easy to learn and operate by the typical action officers who are the users of the message service. These people have no computer background and have neither the time nor the inclination to master a complex system in order to accomplish a simple task such as reading their message traffic, which they already do effectively. The system must offer some new capabilities to make it attractive, but above all it must be comfortable and natural to use. A most critical ingredient of the user's interface to the system is the terminal.

To provide the desired naturalness and ease of use, we want to make available the same sorts of facilities that the user has at his desk, where he is able to scan quickly through large amounts of data, see whole pages of text at once, and easily change his attention back and forth between pages of different documents. He is able to view several documents at the same time and edit or annotate a message at any spot by simply writing there. With paper and pencil these abilities are effortless and natural. We would like to be able to give him the same capabilities in our on-line system with the same simplicity of use.

Unfortunately we are constrained by such practical considerations as the cost of the terminal hardware, its maintainability, the amount of desk space that it occupies, etc. Like most such devices, the MME terminal is a compromise between conflicting goals.

In the early days of the MME program it was envisioned that users would be distributed all over the island of Oahu and that the host computer would be on the mainland. The long delays involved (a satellite hop and an unknown number of network nodes) argue for providing buffering and processing power in the terminal itself in order to guarantee responsiveness to the user (an essential attribute of a natural interface). Economics restricts the speed and memory size of the terminal processors. The terminal that has resulted is different in concept than any currently on the market.

OPERATING WITH THE MME TERMINAL

The style of interaction we have attempted to achieve is to have the user feel he is talking directly to the application

program, with the terminal transparent to him. Simple but powerful two-dimensional editing functions are available in the terminal with a one-to-one relation between a keystroke and the execution of a function. The editing operations automatically format the screen dynamically so at all times the user sees a well ordered presentation of his data. Because of instant response to these operations, the user feels he is editing the document directly (rather than executing abstract commands to a computer which makes the changes for him, which is really what is happening).

To further this impression, the terminal masks the user from its memory limitations. The user can scroll through a full document without having to break it into pieces that will fit on screen or in memory. To achieve this the terminal has considerably more display buffer memory than screen space. The terminal gives the user the impression that it holds the entire document locally, even though it is really bringing new data in from the host as needed.

The naturalness of the interface rests on the principle that "what you see is what you get." The user merely edits a text image on screen to cause the system to make the semantic changes to the database that those editing changes imply. This editing often has the side effect of controlling some operation; for example, the addressees for a message are thought of as simply names in the "To" field of the message, which is filled in by editing the screen along with all the other message fields. The user is not forced to think of these names as active arguments to the message sending process, which the application program must parse, check for validity, and perhaps correct, before accepting them.

To allow flexibility in the data presentation made to the user, the terminal has a variety of highlighting facilities, such as inverse video, underlining, and half-intensity. Basic editing and dynamic formatting are done in the terminal, but the application program has control over how text is highlighted, what format constraints apply and where editing is allowed to occur, with granularity down to a single character. Thus the data sent to the terminal contains editing, highlighting and formatting attributes, as well as the text itself.

To give a user the ability to view several objects simultaneously or quickly switch to views of different objects, the terminal provides a facility called "windows." Each window can hold a different data object and can be independently scrolled, giving the user the impression that he is

working directly on the entire document. Windows may be established by the application program to occupy only part of the screen, so several can be displayed at the same time. They may be moved on or off screen almost instantaneously, making it easy for the user to shift his attention between objects.

The terminal also supports a variety of command language styles. The terminal has function keys whose actions are assigned by the application program, but it also allows typed command input, menu selection (using the cursor to point), and an ability to refer to arbitrary characters on screen as arguments to commands.

IMPLICATIONS FOR THE APPLICATION PROGRAM

The terminal's style of interaction has profound effects on the application program. First it implies a rich data structure. A sequential file does not lend itself to arbitrary editing conveniently. Whatever representation the application program has of the documents it deals with, it must be able to extract and send to the terminal the controls for editing, highlighting and formatting along with the text. It also must be able to change its database to match the changes reported from the terminal, and make correct semantic interpretation of these changes. We call this application resident model of the data in the terminal the "Virtual Terminal" (VT). There must be a VT for each active window in the terminal.

In most alphanumeric applications, changes to the database are made only as a result of the execution of some command through a command interpreter. The semantic content of the command is extracted, the change is made to the database, and then the appropriate display information is generated from the new data and sent out to the terminal. SIGMA, the application program for the MME², operates in this manner on many of its commands.

SIGMA, however, also allows changes to the database through screen editing. In this case the changes to the text that are sent by the terminal go to the virtual terminal, where their semantic impact is interpreted, and the database updated. In this way the meaning of the database can be changed as a side effect of editing, rather than by direct command execution. For example, while filling in a message form the user enters the contents of a message address field. So far as the user or the terminal is concerned this is just text, like any other text in the message. But since it is an address field, SIGMA parses the text, extracts user names and builds an address structure in the database, one element per addressee. If the text or format of the address field is changed as a result of SIGMA's semantic interpretation (e.g., addressees' names might be corrected), the necessary modifications are sent out to the terminal.

Since the terminal does its own formatting, the application program considers text of a paragraph as a continuous stream. Only where the structure of the text dictates (e.g., paragraph beginnings) does the application force format controls. The usual ASCII control characters CR (carriage return) and LF (line feed) do not appear in SIGMA's representation of text.

THE USER'S VIEW

The MME terminal is a Hewlett-Packard 2649A CRT Terminal (an OEM version of the HP2645A terminal¹), with microcode supplied by ISI. The CRT holds 24 lines of text. SIGMA permanently assigns the top three lines as Status lines, where system and user status is continuously reported and error messages appear. The most frequently used SIGMA commands are assigned to 30 function keys. Other commands must be typed in the Instruction window, which occupies the next two screen lines below the status lines. The remaining 19 screen lines make up the working space; they may contain a View window for reference only, a Display window for editing, or a split between Display and View to allow referring to one object while working on another.

Two keypads next to the standard keyboard keys control the local terminal operations, including cursor movement (by character, word, line or window), character insertion (any of the normal typewriter printing keys), deletion (by character, word or line), scrolling of independent windows, and a special function called HERE. The terminal maintains the screen format, automatically breaking a long line at a word boundary and wrapping the remainder onto the next line. The carriage return key forces subsequent text to start on a new line.

FUNCTIONAL DESCRIPTION OF THE MME TERMINAL

Since the 2649 is microprogrammable, the functional operation is entirely defined by the microcode. Communication between the application program in the host computer and the terminal is done in blocks of data, representing a complete command from the application to the terminal (dispatch) or a complete report of some new condition in the terminal to the application (notice).

The terminal is basically a half-duplex device, in that it is either in input state (keyboard active) or output state (host computer active). During input state the user has at his disposal the full screen-editing capability. The terminal switches to output state whenever a function key is pushed (e.g., the EXECUTE key, which causes SIGMA to interpret and execute the contents of the Instruction window, is a function key). During output state the keyboard is disabled. The terminal is returned to input state when the host sends a special "Continue" dispatch. Strictly speaking, the system is not half-duplex because in output state the terminal does send certain control notices required to maintain consistency between the terminal's database and the host's model.

Communications between the terminal and the host is really one computer talking to another. Each transmission must be error-free; otherwise the computer's model and the terminal's model may not match. To insure the needed reliability of data across potentially noisy lines, a fully synchronized block retransmission protocol is used.

Windows

The MME terminal is designed to hold up to seven separate items of text in areas called "windows." Windows are allocated and deallocated by the host (never by the terminal). They are of arbitrary size so long as the total contents of all the allocated windows does not exceed the memory capacity of the terminal. Although the host fills the windows by sending data to them, the terminal does its own memory management and decides what data to keep when it nears its memory limits.

Windows may be thought of as numbered text buffer areas. A window may be assigned to occupy any contiguous portion (full horizontal lines) of the screen, such as lines 15 through 23, with an operation called "map." Normally a window contains more lines of text than will fit on the mapped area (it may also contain less). "Map" places that portion of the data from the window that will fit onto the screen, while any excess data beyond the screen area is stored in "margins," areas logically considered to be above and below the window's mapped screen area. Data scrolls on-screen from these margins.

Several windows may be mapped on different areas of the screen at the same time, which allows a user to view several text objects simultaneously. A window may also be unmapped, which means it remains in the terminal memory, but is not visible to the user. Figure 1 illustrates the terminal with four windows, three of which are mapped. The host may switch the contents of the screen from one text item to another very quickly by "unmapping" the displayed object and mapping the new object.

Mapped windows scroll independently. The ROLL UP or ROLL DOWN keys cause scrolling in whatever window the cursor is in. This operation is done entirely within the terminal, without telling the host. Therefore, although the host always knows what text is in each window and what windows are mapped onto which screen lines, it usually does not know exactly what text is visible on screen at any time.

Domains

In the MME terminal all text is stored in "domains," the atomic unit for the communication of text between the host and the terminal. Each window is made up of a contiguous string of domains. Domains may be any length up to 100 characters. Any character stored in the terminal can be uniquely identified by its window, domain identifier within the window, and character position within the domain. Domains have format, highlight, and control attributes assigned when the domain is created. The user is not aware of the domain structure of text, except as domain attributes are apparent to him. Figure 2 illustrates the domain structure for part of a SIGMA display.

Normally each domain starts at the character to the right of the last character of the previous domain and may wrap around onto the next line. However, the application program may set a domain to be "formatted," which makes the domain start on the next line at the left margin of the screen,

regardless of where the previous domain ends. In this case the blank space to the right of the previous domain is essentially undefined to the application program, since it cannot be identified by domain ID and character position. The terminal will not allow the cursor to move to an undefined location. If a user attempts to move his cursor into such an area, it will jump to the next enterable domain.

The 2649 allows any combination of blinking, underlining, inverse video and half brightness on a character-by-character basis. The MME terminal limits this highlighting to a domain basis; that is, all characters in a domain are highlighted the same. Character set selection is also done as a domain attribute.

Domains also have editing control attributes set by the host when domains are created. They control whether the cursor may enter the domain, whether the domain is editable (from the keyboard), whether characters within the domain may be marked with a HERE, and whether the domain will accept carriage returns. Space is left for assigning other attributes.

When a user edits text he is changing the contents of some domain. If, for instance, he inserts or deletes characters, the domain expands or contracts appropriately and the domain is recorded as "Changed." Nothing is sent to the host until the user begins to edit another domain, at which time the terminal will send the host the new contents of the previously edited domain (via a "Changed Domain" notice) and record this new domain as the Changed domain. Thus the host computer may be, at most, one domain change behind what is in the terminal. Eventually the user will push a function key, carriage return, or the HERE key. The first action the terminal takes on these keys is to lock the keyboard and send out the pending Changed Domain notice if there is one. It then processes the key pushed, which ensures that the host always has the up-to-date state of the terminal before performing any operation on the data (all SIGMA commands are initiated by a function key).

Most often the data displayed come from the application program, in which case the host computer creates the domains and sends them to the terminal. However, the terminal will generate domains in three special instances.

1. When user editing causes a domain to exceed 100 characters, the terminal will generate a new domain and tell the host its location, ID, and contents in the form of a special "Extraction" notice.
2. A carriage return creates a new "formatted" domain and a special "EOL" notice is sent to the host, reporting the location and the ID of the new domain.
3. The HERE key marks the character at the cursor position as a parameter for a subsequent command by splitting it off into a new, one-character domain with inverted video and non-editable attributes. This mark will stay associated with that character regardless of any editing the user might do thereafter before the command is executed. Merely reporting the character position in the domain and the domain ID is not sufficient. The "HERE" notice contains all the information needed to identify the position of the marked character,

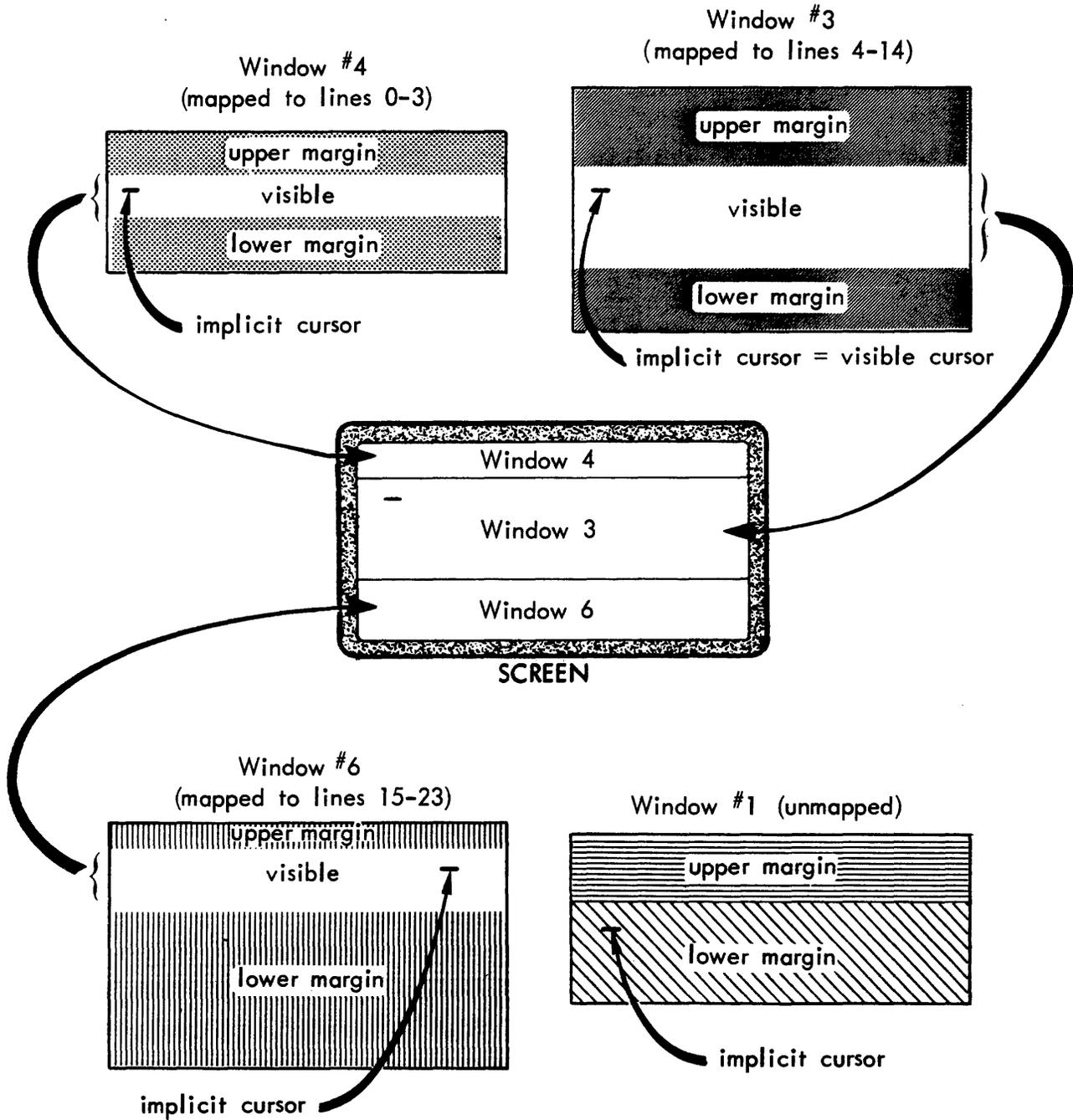


Figure 1—Multiple windows.

how the old domain was split, and the ID of the new domains created.

Flash lines

The terminal has an eighth window called the Flash window. If it exists at all, it is assigned to the top of the visible screen, and can be set by the application to occupy from zero to 24 lines. It has no domain structure and has fixed

attributes. It is always non-enterable, and has no highlight or formatting properties. SIGMA uses the Flash window for status and other output-only information, since it does not have to keep around a corresponding data structure for this window.

Cursor control

Although there is only one visible cursor, each window may have an implicit cursor position. When more than one

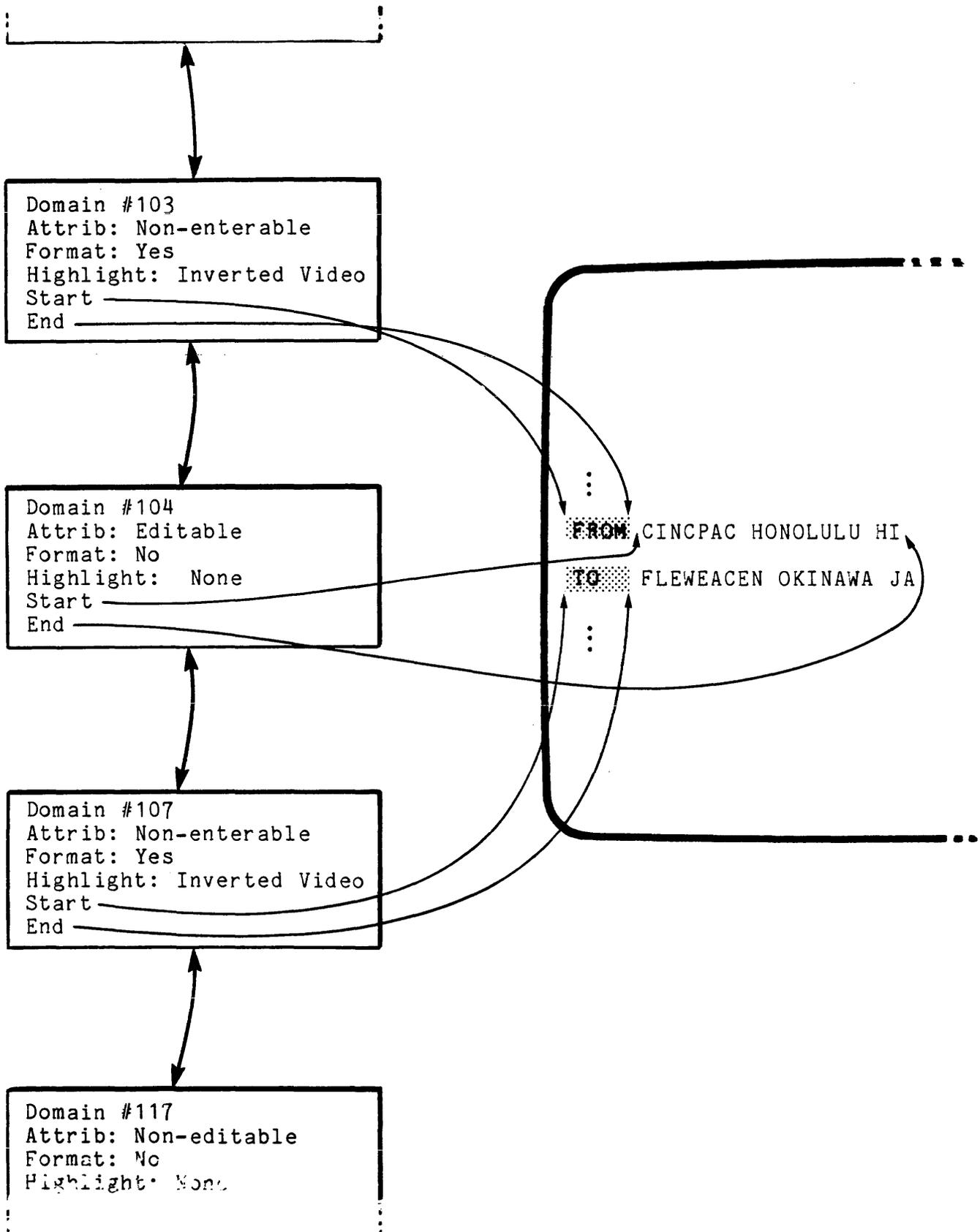


Figure 2—Domain structure.

window is mapped on screen, the UP WINDOW or DOWN WINDOW key causes the cursor to move to the implicit cursor position of the adjacent window.

The host has two controls over the implicit cursor for a window: 1) On what character in the window it should be and 2) on what line on the screen (for mapped windows) it should be. For unmapped windows, the latter translates to "on what line on the screen it would be if the window were mapped." A separate dispatch is provided for each. This limited form of cursor control is the only way that the host can establish what data is shown on screen. Since the user can scroll the screen contents, this is transient control at best. The host also has a dispatch to put the actual visible cursor into the desired mapped window (at the implicit cursor position).

Scrolling

The terminal's basic heuristic for mapped windows is to keep extra lines of text in margins above and below the lines that are on-screen. This lets the user scroll in either direction without having to go to the host for more data. Whenever the terminal assesses that a margin is getting too small, it will send a "Vacancy" notice to the host asking for more data for that margin. The terminal calculates the number of lines of data to ask for based on the size of the on-screen area, the number of lines in the margin, and the amount of memory left in the terminal.

Memory management

The 12K bytes of display memory in the terminal are allocated as necessary for each dispatch. When the remaining memory is reduced to a prescribed limit, the terminal tries to reclaim memory from unmapped windows, and if that does not yield enough, from large margins of mapped windows. Memory is reclaimed by deleting domains and their contents from the edge of the margin and then telling the host through a "Scroll" notice. A Scroll notice identifies the last domain deleted and from which margin it came. It is important that the terminal be able to generate Scroll notices even when the keyboard is locked and the host is in control, for it is during this state that the host will send the data that reaches the memory limit. The terminal must be free to reclaim memory right away in order to have room for the next dispatch, which may already be in the terminal's input buffer.

It is possible for the terminal to try to reclaim memory from the same margin into which the host is writing. To prevent this, the host can set special "No Reclaim" controls for each margin of each window, and the terminal will not reclaim memory from a window margin so marked. The host must be careful not to leave these No Reclaim controls on, or the terminal will quickly run out of margins from which it can reclaim memory and the terminal memory will become full.

CONCLUSIONS

The MME Terminal is now operating with SIGMA at CINCPAC Headquarters. It appears to be successful in providing an interface that is quick to learn and natural to use. Editing is easy to master, since the operations provided are simple and their results immediate. The user deals with large messages or files as single continuous entities, which appear to be completely contained within the terminal. The user never has to consciously "send" an editing change to the host; he simply makes the change, just as he would on paper.

The terminal's most obvious limitation is the small size of the screen (24 lines of 80 characters), compared to the full page printed on paper that users are used to seeing. This is particularly noticeable when operating in split screen mode looking at two objects. This limitation, however, points out what we consider to be a significant technical contribution of the MME terminal.

By dealing in terms of windows and domains, and letting the terminal format the text and control the flow of data (via Vacancy and Scroll notices) we have achieved a division of labor which makes the application program almost completely independent of the terminal characteristics. The high-level protocol that achieves this also supports the natural user interface style we sought. The application program has no knowledge of, or concern for, the amount of memory in the terminal. Since the terminal manages its own memory and tells the host when and how much data to send, the application program never needs to consider whether the text "will fit."

Furthermore, the application program assumes very little about the size or characteristics of the display screen. A small section of code in SIGMA, which controls the mapping of windows to screen lines, knows that the terminal has 24 lines. It has no idea how many characters fit on a line, or whether the terminal has proportional spacing, or what kind of display technology is employed.

The application program is also completely unconcerned about details of the terminal's editing facilities. For instance, SIGMA does not know the terminal does not have a "type-over" mode, or that it does have a "word delete" key. The terminal could scroll a line at a time or a page at a time. Adding a positioning device like a "mouse" or a joystick would not affect SIGMA at all. In theory, one could add a "replace string" function internal to the terminal if he wanted without affecting the application program (we chose not to because we feel such operators should be global to the entire document and therefore belong in the host).

The protocol isolates the physical and functional features of the terminal from the application program, which allows us to take advantage of new, more capable terminals with more memory, larger screens and more powerful editing features without having to rewrite the application software. At the same time we maintain the desired interactive coupling between the terminal and the host.

We believe a protocol which provides this independence is needed to foster the use of "intelligent" terminals for network applications. We are advocating the development

of a protocol to support communication with such powerful terminals, which we are calling the Network Virtual Processing Terminal protocol (NVPT). We do not propose that the MME terminal protocol in its current form is sufficient for NVPT, since it does not yet provide adequate format or editing controls. It needs to accommodate a variable number of display windows and visible screen lines, and issues such as backward compatibility with teletypes and down-load capabilities must be considered. We do feel, however, that the protocol in MME provides a good starting point for

discussion of an NVPT and we invite constructive criticism directed toward achieving such a goal.

REFERENCES

1. Hewlett-Packard, "2645—A Display Station Reference Manual," Hewlett-Packard Company, 1976.
2. Stotz, R., R. Tugender, D. Wilczynski and D. Oestreicher, "SIGMA: An Interactive Message Service for the Military Message Experiment," *Proceedings of the National Computer Conference, AFIPS*, May, 1979.

This research was performed for the Advanced Research Projects Agency under Contract No. DAHC 15 72 C 0308, ARPA Order No. 2223. The views and conclusions expressed in this paper are not necessarily those of any person or organization except the author(s).

On-line tutorials and documentation for the SIGMA message service

by JEFF ROTHENBERG

*USC/Information Sciences Institute
Marina del Rey, California*

INTRODUCTION

The highest goal of the SIGMA service has been to successfully make the transfer from the computer science research environment to the working military message processing environment for which it was intended. One of the problems leading to the failure of such technology transfer in the past has been a lack of appreciation for the importance of the way a system is introduced into a working environment, how users are trained in its use, and how it is documented from the users' point of view. This paper describes SIGMA's approach to these issues.

The target users of SIGMA are military personnel at CINCPAC engaged in message processing. The functional aspects of this military message processing environment are described elsewhere.⁴ The target users understand and competently perform complex and important tasks. The training and documentation material for SIGMA assumes that the users are already expert at their jobs and that they will quickly become proficient at using SIGMA if it is presented to them in familiar terms.

The fact that target users cannot afford the time for extensive scheduled classes or other means of collective instruction suggested that on-line tutorial and help facilities be developed to allow a user to learn at his own pace and to a self-determined depth, and to minimize the need for classroom or one-to-one personal instruction. A short introductory lecture gives an overview of SIGMA and shows users what it can do for them. From there on, the user refers to on-line aids for most of his training, using human consultants only when necessary. While reference aids also exist in hardcopy form, it was strongly felt that on-line references should be available whenever the user is working with SIGMA. On-line references have the advantage that they are guaranteed to be available, they can be kept dynamically up to date more readily than hardcopy, and they can exploit knowledge of the user's state to pinpoint what he is most likely to need help with. The initial design proposal for these training and documentation facilities was published in May of 1975,³ and a requirements document for the actual documentation was published in June of 1976.²

The strategy for training and documentation in SIGMA involves three avenues of attack. The Command Language

Processor (CLP), which parses the user's input commands, implements a Prompt facility that allows the user to see which commands are legal in his current state. The Help facility provides on-line access to detailed reference material. The Tutor provides a complete curriculum of on-line Lessons and Exercises.

THE SIGMA TERMINAL

The SIGMA (MME) terminal is described elsewhere,⁵ but a brief description is in order here. The terminal consists of a keyboard with a number of special function keys and a CRT screen divided into several horizontal (full-width) windows.

SIGMA uses the top two lines of the screen for alert and status information called the Flash and Status Windows, respectively. The third line is a Feedback Window which SIGMA uses to give feedback to commands the user executes (e.g., error messages). Following the Feedback Window is the Command Window, in which the user enters typed commands.

The remainder of the screen is used as a working area consisting of a Display Window (where data objects can be displayed and edited), and/or a View Window (where data objects can be shown for reference but not edited).

The terminal supports multiple windows that may or may not be "mapped" onto the screen. For example, the user can have a data object displayed in the Display Window and can press a key to split the screen and map another object into the View Window. The documentation facilities described below make use of this mapping feature to take over the screen for documentation purposes (at the user's request), and later restore it to its original state.

PROMPT

When the user presses the key labelled PROMPT on the SIGMA terminal keyboard, the CLP remaps the working area of the screen as a Prompt Window and shows all commands that are legal in the current state. (When the user hits PROMPT again, the original state of the screen will be mapped back again.) If the user cannot recall the arguments

or form of a command he is typing, he can hit PROMPT and be shown the various forms of the command he has begun. If he types an ambiguous command in the Command Window and tries to execute it, the CLP will respond with an error message (in the Feedback Window) telling the user that the command is ambiguous and suggesting he press PROMPT. If the user then presses the PROMPT key, the CLP shows Prompting of all those commands which it cannot yet disambiguate based on what has been typed in the Command Window. Commands that are close to what the user typed (e.g., which match the number and types of the given parameters) are shown highlighted.

For example, suppose the user has a file named Pending and a text object named Papa. If he types "D P" in the Command Window and hits PROMPT, the Prompt facility will show four possible commands:

```
Display Text <Existing Text Name>
Delete Text <Existing Text Name>
Display File <Existing File Name> <Security>
Delete File <Existing File Name>
```

With each command is shown a brief description of its function and a stylized form of the command with its parameters.

Once a command (or list of commands) is shown by Prompt, the user can select one of the commands with the cursor and press PROMPT again; this expands the description of that particular command, showing the syntax and meaning of each of its parameters.

ON-LINE HELP

If the user needs a more detailed description of a command or cannot remember which command to use, he can request the next level of documentation: Help.

Compatibility of on-line and hardcopy reference material

Considerable effort was devoted to making the on-line Help database semantically identical to the hardcopy printed Reference Manual so that the user would not find these alternate forms of documentation incompatible or confusing. At the same time, however, syntactic differences between the printed form of a Reference Manual and an on-line reference database had to be allowed for. For example, the printed form requires page numbers, section headings and chapters, whereas the on-line form is accessed interactively, making this organization irrelevant and cumbersome.

```
HELP      SERVICE FACILITIES
BACK     FORWARD
```

The Current Term field shows the Term for which HELP is currently displayed. The New Term field allows the user to type in a new Term for which he wants Help. Spelling correction is provided by means of the same algorithm employed by the CLP.

The documentation facility was designed to automate semantic compatibility while allowing syntactic variation so that documentation writers (documentors) would find it easy to maintain both on-line and hardcopy reference material in an up-to-date and interlocked form.

Using on-line help

SIGMA documentation is organized under a large set of "Terms," which are words or phrases that name various aspects of the system. Terms include all command and parameter names, classes of data objects and operations, and procedural matters relevant to the user's work situation. They are essentially a set of keywords and key-phrases that semantically cover SIGMA and the environment in which it is embedded. Several synonymous Terms may refer to the same information. The actual documentation for a given Term consists of text called a "cell" (a word SIGMA users never see). The display of a cell can be scrolled; it is not broken into frames in the sense of screenfuls or pages.

Requesting help

If there is a command or part of a command in the Command Window when the user presses the HELP key, Help is provided for the Term corresponding to the command word. If the Command Window is empty, pressing HELP results in a "top-level" Help display which is just a description of how to use the Help facility itself. (This is implemented simply by having the Help facility provide Help for the Term "HELP.")

Use of the screen in help

When Help is activated, the Feedback Window displays a message telling the user that Help is being shown below and how to "get out" (that is, how to return to what he was doing before he hit HELP). The Flash and Status Windows are unaffected. The Command Window and work area (Display Window and/or View Window) are mapped away, and the remainder of the screen is divided into two windows analogous to the Command Window and Display Window. (When the user returns from Help to his previous state, the screen is returned to the state it had before he hit HELP.) The lower of the two Help windows shows the actual documentation cell for some Term.

The topmost of the Help windows consists of two lines which appear as

```
Current Term: Some Key Phrase
New Term:
```

Selectable terms

The other four fields are shown in inverse video highlighting. The convention followed in the Help facility (and explained in the top-level Help display) is that anything that

appears on the screen in inverse video is itself a Term for which the user can get Help simply by selecting the highlighted field with the cursor and pressing HELP again. Thus the Terms HELP and SERVICE FACILITIES are always available whenever the user is getting Help; he has only to move the cursor into either of these fields and press HELP. SERVICE FACILITIES shows an Index-like list of the major topics and commands in SIGMA; when it is selected, the lower Help Window shows what is effectively a menu of topics on which Help is available. Any Term that appears highlighted (in inverse video) in this list can be selected similarly with the cursor. The user can thus use Help as a menu-driven access facility, or he can type in specific Terms to be accessed (at New Term). Whenever the Help display is changed, the Current Term field is changed to show the Term whose documentation cell is being displayed.

The fields BACK and FORWARD are also shown highlighted. These are not really Terms, but are "virtual function keys" which allow the user to retrace his steps through previous Terms for which Help has been displayed.

Documentation writing and maintenance

As explained above, the design of the documentation facility included a concern for how documentation is written and maintained for two syntactically disparate but semantically identical forms: hardcopy and on-line Help. The printed Reference Manual for SIGMA is generated using a traditional formatting system driven by commands embedded in the text to be formatted.

The on-line version must display selectable Terms as highlighted words or phrases which the user can select as described above. It must also allow the definition of Terms. The documentation text itself is generally the same as that in the hardcopy Manual. In order to make it possible for the user to select occurrences of Terms within the text, the documentor can flag words or phrases as potential Terms. An occurrence of a Term is shown as selectable only if the documentor flags it as a potential Term AND the Term is actually defined in the current database.

In order to make writing and maintaining documentation easy, a single combined source is used for both forms. Special bracket pairs control documentation preprocessing by signalling that characters are intended for either the hardcopy Manual or the on-line database or both.

THE TUTOR

The final level of detail in documenting SIGMA consists of a full curriculum of on-line Lessons and Exercises covering most of the features of the system. The primary goal in designing the Tutor was that the user be able to take Lessons on-line and try out various commands in a Protected Mode. That is, the Tutor guarantees that the user can do no harm to any real data (his own or anyone else's) when taking a Lesson. However, it was felt that any attempt to interpret or simulate the action of commands would sooner or later

result in divergence between the Tutor's simulation and the real behavior of SIGMA. The only way to keep the user's faith that a Lesson describes the system as it really works is to effectively let the system simulate itself.

The Tutor supports two related features: Lessons and Exercises. A Lesson is a detailed description of some aspect of SIGMA. As of this writing some dozen Lessons have been written. The user can take any Lesson at any time by using the command "LESSON j." No order is enforced, though the Lessons are arranged in a logical sequence for most users' needs. A user can retake a Lesson any number of times, can quit in the middle, and can start up in the middle the next time.

On-line lessons

The user asks for a lesson with the command "LESSON" which takes a lesson number. The available lessons are listed in the hardcopy Reference Manual and in on-line Help. When the user executes the LESSON command, the Lesson text is displayed in the working area of the screen. The Feedback Window shows a message telling the user he is in a Lesson and how to get back to what he was doing before he entered the Lesson. The Lesson text shown in the working area can be scrolled through, just like any display on the SIGMA terminal. Lessons provide complete discussions of the most important topics having to do with using SIGMA. Most of the Lessons have Exercises associated with them, which they suggest the user take.

On-line exercises

An Exercise is generally a very short and specific task that the user can try in the Tutor's Protected Mode (a phrase the user never sees). Lesson 2, for example, discusses a user's special data object called the Pending File and then suggests that the user try Exercise 1 to display a Pending File. Later in the Lesson, the user is shown how to display messages from a file, and Exercise 2 is suggested, which allows the user to display a message. In keeping with the nonassertive philosophy of the Tutor, the user is not coerced in any way into trying the Exercises. He can skip some or all of them, can take them in any order and can retake them any number of times.

The user takes an Exercise by typing the command "EXERCISE k" in the Command Window. The Exercise number automatically refers to that Exercise for the Lesson in progress. When the user enters the Exercise, the working area of the screen (which had displayed the Lesson) is remapped to display the Exercise. The Feedback Window displays a message telling the user he is in an Exercise and how to get out of it.

An Exercise describes how to specify some particular command or set of commands and suggests that the user try them. In order to try them, the user simply moves the cursor into the Command Window and types the command, just as he would if he were not in the Tutor at all. At this point, the

Command Window is simply parsed by the CLP as always. However, the resultant parsed command is not immediately executed but is first passed to the Tutor for scrutiny. The Tutor compares the command with the list of allowed commands for this Exercise. If there is a match, the Tutor allows SIGMA to execute the command; otherwise, it displays a message in the Feedback Window telling the user the command he typed did not match any of those in the Exercise.

The protected mode

The screening of commands by the Tutor is performed after the CLP has parsed them. Thus the match is not between what the user typed and what the Exercise said he should type but rather between the parse of what he typed and a parsed form encoded with the Exercise. The result of this approach is that the semantics of what the user typed are compared with the desired semantics in the Exercise. Any alternate form that the CLP can recognize as the same target command will pass the Tutor's inspection as the desired command. This behaves correctly in a pedagogic sense, in that the user is considered to have completed the Exercise successfully if he achieves the desired result, regardless of how he achieved it. (The next level of this approach¹ is to actually allow the command to execute and compare the result of its execution with the desired state. This comparison is harder to define and perform, and allowing execution makes it harder to guarantee the Protected Mode. It was felt that for efficiency and safety command-level matching was the best solution for SIGMA.)

In order to provide the Protected Mode, the Tutor must not only screen which commands can be executed, but must in some cases switch the data objects on which a command operates so that "real" data is never used. If real data were used, it would also be hard for the Exercise to refer to what the user was seeing when he displayed data. By using special Tutor data, the Tutor both protects the real data and has a handle on what the user sees when performing the Exercise. The Tutor's only interference with the execution process is to switch data objects in the parsed command after the CLP has parsed it and before it is given to SIGMA to execute.

Use of the screen in exercises

When the user executes a command within an Exercise, SIGMA is unaware that the Tutor is involved, and the screen is used just as it would be if there were no Exercise displayed. The Tutor is responsible for mapping away its own display and allowing SIGMA to display what it will. However, once the user has executed a command he must be able to return to the Exercise to see what to do next. A toggle is provided which allows the user to switch the screen between the "normal" state (as it appears after executing a command) and the "Exercise state" (which maps the Exercise

text into the work area). To prevent confusion the Tutor uses the Feedback Window to keep the user aware of what he is looking at and how to toggle to look at the opposite state of the screen. In practice this requirement for toggling in an Exercise is no worse than flipping back and forth between a figure and a description of that figure on two different pages in a printed lesson.

Tutor summary

The Tutor has proved a powerful training device whose use has greatly facilitated the introduction of SIGMA to users in a working environment. It frees users to learn at their own pace and their own convenience. It provides the security of always knowing that there is a detailed and dynamically up-to-date description of the operation of SIGMA available at the user's fingertips whenever he is running SIGMA.

RESULTS AND CONCLUSION

Formal statistics have not yet been gathered on the Military Message Experiment as a whole or the training process in particular. To date about 100 users have been exposed to the introductory lecture on SIGMA, and about two-thirds of those users have taken at least some of the on-line Lessons. As expected, the earlier Lessons have sufficed to get users started using SIGMA; most users have not found it necessary to take the later Lessons which deal with the full range of features in SIGMA. It is too early to give quantitative results as to the efficacy of on-line training for SIGMA, but indications are that users are able to gain competence and facility with the system by means of the supplied on-line training aids combined with human interaction and consulting.

The success of introducing any system into a real-world environment depends on many factors. An attitude of respect for the end user is crucial in all phases of producing a user-oriented system, and it permeates the design of SIGMA. The awareness of the need for on-line Help and Tutorial facilities is only half the battle for good user documentation: equally important is the style in which the documentation and training material are written. All condescension must be avoided, and if the documentor is a member of the Computer Science community he must keep in mind that he is writing for users whose expertise is different from—though probably no less than—his own.

SIGMA's three-pronged attack on documentation includes a Prompting facility, a fully integrated on-line Help and hardcopy Reference Manual, and a complete curriculum of on-line Lessons and Exercises under the Protected Mode of a Tutor. This approach has proved an efficient way for users to learn how to use SIGMA and to access reference materials while using it.

REFERENCES

1. Grignetti, Mario C., L. Gould, A. Bell, C. Hausmann and J. Passafiume, "Mixed-Initiative Tutorial System to Aid Users of the On-line System (NLS)," *Semiannual Progress Report (Phase I)*, Bolt, Beranek and Newman, Inc., May, 1974.
2. Miller, D., "Military Message Handling Experiment Training Requirements," MTR-3263, MITRE Corporation, June, 1976.
3. Rothenberg, J. G., "An Intelligent Tutor: On-line Documentation and Help for a Military Message Service," ISI/RR-74-26, USC/Information Sciences Institute, May, 1975.
4. Stotz, R., R. Tugender, D. Wilczynski and D. Oestreicher, "SIGMA: An Interactive Message Service for the Military Message Experiment," *Proceedings of the National Computer Conference*, AFIPS, May, 1979.
5. Stotz, R., P. Raveling and J. Rothenberg, "The Terminal for the Military Message Experiment," *Proceedings of the National Computer Conference*, AFIPS, May, 1979.

This research was performed for the Advanced Research Projects Agency under Contract No. DAHC 15 72 C 0308, ARPA Order No. 2223. The views and conclusions expressed in this paper are not necessarily those of any person or organization except the author(s).

Maintaining order and consistency in multi-access data

by RONALD TUGENDER

USC/Information Sciences Institute
Marina del Rey, California

MULTI-ACCESS DATA—A LONG-STANDING PROBLEM

The problem of controlling simultaneous access to shared data runs throughout the history of computer science. In order to preserve the consistency and integrity of such data, computer scientists and programmers have developed locks,⁶ semaphores,³ the notion of critical sections,^{4,3} monitors,⁵ and innumerable other techniques, both concrete and abstract. The great interest in such mechanisms throughout the computer community, both in the literature and in practice, is indicative of the importance of the problem.

Classical examples

Control over simultaneous access is necessary in countless practical applications, one of the most common of which is the need to guarantee strictly sequential access to critical resources by competing—conceptually or literally—parallel executing processes. To provide the necessary interlocking, implementors have used techniques such as the simple lock and its more sophisticated cousin, the mutual exclusion semaphore. In each of these, before being allowed to access the critical resource, a competing process must pass a decisive test, engineered to ensure that

- Only one process can pass the test, no matter how many processes are competing or how frequently they attempt access.
- Once a process has passed the test and is granted access to the resource, no other process can pass until the first has “released” the resource.

A different type of control over simultaneous access is necessary to satisfy another problem, referred to in the literature as the “Readers and Writers Problem.”^{2,1} In this case several readers accessing shared data wish to ensure that the data they are reading remains constant during the reading period. It can be summarized by the following rules:

- Any number of Readers may access the data simultaneously, but if any Reader has access, no Writer may also have access until all Readers are finished.

- Once any Writer has gained access, no Reader or other Writer may have access until the Writer has finished.

Multi-access objects in SIGMA

Neither of the above techniques is appropriate if any of the competing processes need extended access to the resource. Any process doing so would block other processes from completing their tasks, potentially lengthening the time required to complete them significantly and, in an interactive environment, slowing the response time. In a situation in which a process needs read access to a data object for a long time, possibly concluding with a need to modify the object, neither of the above techniques can offer the guarantee of integrity of the data as well as provide timely interactive response when many processes are competing.

In the SIGMA message service^{8,7} the two most important and frequently used data objects, *messages* and *folders*, are both typically shared among several users. At any time, any or all of the users allowed to share one of these objects may “open” it (request read and potentially write access), read through it for an arbitrarily long time and modify it in any of several ways. A scheme had to be developed which allowed simultaneous access by several users for arbitrary lengths of time, preserved a consistent image of the object for each user even while it was being modified by others, and both permitted and correctly assimilated modifications made in parallel by several users. At this point a description of the users’ perception of the two SIGMA shared objects is in order, to illustrate the issues involved in parallel modification.

SIGMA messages

A SIGMA message is conceptually similar to a formal business letter with a sender, addressees, a body and various other information. In draft form (i.e., before it is sent), it may be read and revised by several reviewers before it is actually approved for sending. Several people may be involved during the draft/revision process, and for time efficiency it is often desirable for them to work in parallel. SIGMA provides this parallelism by giving each reviewer an

up-to-date copy of the current draft message, allowing him to edit, revise and comment as he sees fit. Each user's rendition of the message, called a *messagette*, contains all the original, unmodified parts of the original draft, his own modified sections replacing their original counterparts, plus any new text and comments.

After the message has been sent (referred to as a transmitted message), its character changes. It is no longer a draft entity, subject to revision. Just like a letter which has been dropped into a mailbox, it has become an official document, the contents of which are now on record. Users may read it and make comments, but are not allowed to change its contents in any way.

SIGMA folders

A SIGMA folder can be likened to a file into which messages are placed. In SIGMA, however, the folder contains not the messages themselves but rather abstracts, called *entries*, containing a pointer to the actual message (for retrieval purposes) and a subset of the information contained in the referenced message. When messages are sent to a user, the entries referencing them are automatically placed in a particular folder named "Pending" (analogous to a mail in-basket). A user can create other folders and copy entries between them, where the number of folders needed or the significance of the entries placed into them is left entirely to the user's discretion. To help him locate specific entries within folders, a user is provided a rich set of searching tools, including the ability to associate entries with user-chosen keywords. A user can also place comments on entries and delete entries that are no longer needed in the folder.

A user may permit any or all of his folders to be accessed by other users, in which case the other users are allowed the same searching facilities available to the owning user, as well as to read the abstracts, make and read comments, and retrieve referenced messages. To allow users to peruse folders conveniently, SIGMA also maintains a place-marker in a folder for each user, marking the last entry he has referenced; when he next accesses that folder, he will be returned to the same entry.

Further complications

The requirement that many users be allowed to modify an object in parallel posed several logical as well as implementation problems. It became apparent that the solution involved more than simply providing the correct form of interlock apparatus; it also had to preserve as much as possible the intent of the modifiers. This caused two complications well beyond the scope of the classical techniques described earlier:

1. "Whoops, where'd it go?"—From the application system standpoint, even assuming that writers are prevented from simultaneous modification, logical incon-

sistencies can still occur if they can write arbitrarily when it is their turn to write. Consider an object represented as a linked list, where the types of modification allowed are *add*, *replace* and *delete* elements of the list. What does it mean to *replace* an element just *deleted* by another writer, or to *add* an element adjacent to one just *deleted*?

2. "That's not how I remember it!"—Just as important as the ability to produce a logically consistent object is the need to have the updated object conform to each user's expectations of how it should appear after modification. Consider the typical situation which occurs when several authors or reviewers are allowed to edit a draft document in parallel, and they specify disparate sets of changes to a common secretary. When the updated draft appears, one or more authors may be surprised that the new draft does not reflect the changes they specify.

Clearly the above situations could be avoided if some restraints were placed on the types of modifications allowed. In the latter case, for example, the problem lies not in the authors nor in the secretary, but rather in the revision process which allowed parallel modification of a common data object. To avoid the confusion the secretary could have requested that each author confine himself to a different section of the document during a given review cycle, which would ensure that the various sets of modifications would not seriously conflict. While not a perfect solution, some restraint on the types of modification permitted was necessary to allow SIGMA to preserve the basic intent of the modifying users.

AUGMENTING THE CLASSICAL SOLUTIONS

In light of the previous observations, the approach taken to provide parallel modification comprised two main components. The first was to carefully constrain the types of modifications permitted to the several users so they would cause neither irreconcilable conflicts nor unexpected results. The second involved finding a representation by which the modifications could be expressed.

Limiting modification

Determining in which ways to limit modification was a delicate issue. If the limitations were not strict enough or not along the correct dimension, the several sets of changes would conflict too much; if too restrictive, the users would be prohibited from expressing the changes they wished (and should be allowed) to make. The data objects involved, messages and folders, and the operations supported on them were thus designed with the goal of making the necessary limitations seem natural rather than confining.

Limitations applied to draft messages

In draft messages, SIGMA imposes its limitations on parallel modification by slightly constraining the draft/revision

model. Rather than portray a message as a shared entity, with all reviewers attempting to edit the one version into the form they want it, SIGMA effectively gives each user his "own" rendition of the message, to modify as he wishes. The process of successively refining a draft involves the selective reading and inclusion of desired segments of various users' messegettes, reading and possibly taking action on comments and suggestions, eventually resulting in a new draft. While the assimilation of the various changes and suggestions would normally be expected to be performed by the original author, any reviewer would have access to the same information and tools to create his own rendition.

The approach of giving each user his own rendition of a draft message avoids the complications described earlier.

1. Messegettes are logically and physically separate. Since each user may write only into his own messegette, modifications specified by several users contain no conflicts to produce logical inconsistencies.*
2. Each user works with his own rendition of the message. While he has access to other users' messegettes, from which he can incorporate desired sections, his message always accurately reflects only changes *he* has made.

Users are thus allowed a free hand in composing or revising draft messages, while still retaining the ability to read, reference and comment upon other users' versions.

Limitations applied to transmitted messages

As previously described in the analogy to conventional mail, the content of transmitted messages is not subject to arbitrary modification. Users are restricted to a small set of operations.

- *Comment*—A user may place comments on any desired part of a transmitted message.
- *Forward*—A user may specify that the message be forwarded to one or more other users. A notation of the users receiving such forwarded copies is appended to certain message fields.

The limited scope of these operations avoids the undesirable complications.

1. All the types of modifications that users can specify are non-conflicting.
 - Inserting a new comment is strictly additive, requiring no change to the message's basic contents. Multiple comments specified in the same place in the message are simply added one after the other.
 - Editing an existing comment causes no conflict, as each user may modify only his own comments.
 - Forwarding is also an additive operation, simply appending the names of the forwarded addressees to the end of the appropriate field.

2. With the possible exception of the order in which comments or forwarding entries appear in the message (since users may comment or forward in parallel), the updated transmitted message conforms to each user's expectations.

Limitations applied to folders

The structure and use of folders prohibit providing a copy for each user, as in draft messages. The replication of storage to keep the multiple copies would be too expensive (folders tend to become quite large), as would the processing necessary to add each new entry to all of the copies. Consequently, all users access and modify the same folder object. Clearly, not all users may be allowed to modify a folder arbitrarily. As shown earlier, such a situation would lead to chaos unless some limitations on modification are imposed.

Fortunately, a compromise was found which provided the appropriate limitations without unduly constraining the capabilities of the users. Since it was logical to permit the owner of a folder more latitude in modifying it than other users, the limitations imposed differed, as follows:

- *Owner*—The owner is allowed all possible modification capabilities, including the abilities to
 - Delete the folder entirely
 - Delete and modify entries
 - Add keywords to entries
 - Append entries to the end of the folder
 - Add or modify comments (his own)
 - Keep a place-marker in the folder
- *Non-owner*—Non-owning users are permitted only the latter three capabilities. Note that these are all basically append-like operations, causing no structural changes to the folder.

This two-level capability scheme avoids the undesired complications described earlier.

1. The owning user alone can specify "dangerous" (structure-modifying) changes, so conflicts cannot occur. As in messages, appending data to the end or modifying data pertaining only to a specific user (comments, place-markers) do not produce conflicts.
2. The only departures from users' expectations occur when entries are deleted by the owner: comments specified by other users for deleted entries simply disappear; a place-marker specifying a deleted entry is adjusted to the nearest entry remaining. In all such cases the behavior is reasonable and does not constitute a significant departure from users' expectations.

Representing and applying parallel modifications

Once the appropriate limitations had been established to eliminate textual inconsistencies between the modifying users, there remained the issue of providing a mechanism

* Subject to additional considerations discussed later.

which allowed changes to be assimilated into a common data object regardless of the number of users reading or updating the same object in parallel. It was also considered desirable to preserve a consistent image for readers during their access of an object (the goal in the Readers and Writers Problem); changes performed by other users should not cause objects to "change underneath them."

To provide the consistent image of an object during a user's session, a temporary copy of the object is made upon access. The reading and modification are then performed upon the copy, ensuring that no unexpected changes occur during the session. However, when a user's modifications have been completed, the net effect of the specified changes must then be performed on the central copy (called the *base* copy), which constitutes the "real" object to the rest of the users. Note that the modified copy cannot in general simply be substituted for the base copy; if any other users were making changes in parallel, such a strategy would cause all sets of changes but the last to be lost. Rather, the mechanism developed for SIGMA centered upon a construct called a Δ -file.

The Δ -file concept

When referring to a change in a variable (say x), mathematicians often express it as Δx , meaning "the change in x ." To obtain the new, updated value of x (call it x_{new}), one must take the old value (x_{old}) and apply the change (Δx), which in mathematics is done by addition, i.e.,

$$x_{\text{new}} = x_{\text{old}} + \Delta x$$

The notion of Δ -files in SIGMA is conceptually similar to the mathematical Δ . Each user, when accessing an object, is given a local copy. As he modifies it, his changes are remembered. When his changes are complete, SIGMA places into the Δ -file a record of the effective changes applied by the user to the object. A Δ -file thus represents the distillation of changes made by a user to his local copy of an object in an editing session, which, if "added" to the base copy, would produce the changes specified by the user.

The analogy to the mathematical Δ is complicated by the possibility of having several Δ -files produced in parallel by different users, each referring to the same base copy. Since the users work independently, the order in which the Δ -files are applied cannot be guaranteed. If one user makes changes which conflict with those of another user, the consistency of the base file and maintenance of users' intentions cannot be guaranteed. But, as previously described, the types of modification allowed cause no significant conflicts.

What's in a Δ -file?

When a user modifies an object, the Δ -file produced contains not the new contents of the object but rather a specification of the modifications that need to be performed on the base (original) copy to make it conform to the user's changes. The following types of change specifications used

in SIGMA Δ -files, called Δ -operations, are sufficient to describe all possible changes to SIGMA objects.

- *Add*—Add a new item of data to the object adjacent to some other data item.
- *Delete*—Delete a data item from the object.
- *Replace*—Replace the contents of a data item with a new value.

For efficiency in applying the changes to the base copy, the identification of the data items to be affected by the Δ -operations is done by an absolute, rather than symbolic, addressing scheme. This avoids costly searching to locate affected items, but requires that the addresses in the base copy at the time the Δ -operations are performed match those which existed at the time they were generated. The constancy of these absolute address references is guaranteed by the non-conflicting nature of Δ -files and the internal structures of messages and folders, which do not require address manipulation in response to modifications.

Mechanics of assimilating Δ -files

Once the various Δ -files have been generated, the task remains of applying them to the base copy. Rather than assign this task to the SIGMA user processes, it was decided to create separate processes (one for messages, one for folders) to execute this assimilation function, implemented in the SIGMA system as shared background processes known as *daemons*. When a SIGMA process creates a Δ -file as the result of user changes, it enqueues a request to the appropriate daemon, supplying the name of the object to be modified and the name of the Δ -file in the form of a physical location identifier (PLID), an operating-system-dependent path name describing the location of the Δ -file information. The daemons execute these requests in order, finding the referenced PLIDs and applying the contained Δ -operations to the named objects. A diagram depicting the process of Δ -file incorporation is shown in Figure 1 (although the SIGMA message service processes many shared objects, the figure concentrates on just one such object, in the process of being modified in parallel by several users). Note that the order in which Δ -files are processed is arbitrary, but the consistency of the resulting updated object is preserved by the non-conflicting nature of the changes they contain.

The division of labor in the updating task between the SIGMA processes and the daemon provides several benefits. It allows the Δ -file incorporation task to be performed by a separate background processor, insulating the SIGMA processes from a significant processing burden (hence user-perceived response time delay) which they would otherwise encounter whenever an update operation was performed. And since the SIGMA processes have no need to modify the base copy of an object (all writing is done in the Δ -files instead), the daemon can be given exclusive write access to the base copy. Combined with the "copy-on-access" discipline imposed to maintain a consistent view of objects for

SIGMA operations not pertaining to that object, unlike (2) does.

Since in practice it is rare that a user attempts to re-access an object before the daemon has had the opportunity to perform the Δ -file assimilation, Approach 3 was chosen. While not an elegant solution, the infrequent temporary denial of access to a specific object poses a negligible inconvenience and is thus a minimal impediment to users.

CONCLUSIONS

The multi-access scheme described in this paper has been in operation within the SIGMA message service for over three years. This experience has shown the approach to have successfully satisfied the requirements needed to provide multi-access to SIGMA's shared data objects. Following are several of the most successful aspects of the SIGMA multi-access methodology:

- Parallel access to shared objects by an arbitrary number of users (processes) occurs with no conflict.
- Each user is satisfied that his changes to a shared object are faithfully recorded.
- Users accessing shared objects are not confused by other users' changes during their own editing sessions.
- The limitations imposed on modifications are natural rather than cumbersome, and do not overly constrain users.
- The concept of a Δ -file and the resulting non-conflicting, incremental update of shared objects is a powerful technique to apply to the multi-access problem.
- The division of labor between the foreground (SIGMA)

and background (daemon) processes provides two significant benefits—the ability to avoid expensive reader/writer interlocks on shared objects, and the shifting of the processing burden away from the user process to achieve better user-perceived response.

While this methodology was developed specifically for the SIGMA message service application, the concepts involved are much more general. Similar techniques could be successfully applied to many other shared data applications, such as command-and-control, data base management systems, information retrieval systems, or any other application in which many users can have access to common data and in which preserving the intent (and hence the trust) of those users is an important concern.

REFERENCES

1. Brinch Hansen, P., "A Comparison of Two Synchronizing Principles," *Acta Informatica*, Vol. 1, 1972, pp. 190-199.
2. Courtois, P. J., F. Heymans and D. L. Parnas, "Concurrent Control with 'Readers' and 'Writers'," *Comm. ACM*, Vol. 14, No. 10, October 1971, pp. 667-668.
3. Dijkstra, E. W., "Cooperating Sequential Processes," in *Programming Languages*, F. Genuys (ed.), Academic Press, 1968, pp. 43-112.
4. Dijkstra, E. W., "Solution of a Problem in Concurrent Programming Control," *Comm. ACM*, Vol. 8, No. 9, September 1965, p. 569.
5. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," *Comm. ACM*, Vol. 17, No. 10, October 1974, pp. 549-557.
6. Saltzer, J. H., "Traffic Control in a Multiplexed Computer System," MAC-TR-30, Project MAC, Massachusetts Institute of Technology, 1966.
7. Stotz, R., R. Tugender, D. Wilczynski and D. Oestreicher, "SIGMA—An Interactive Message Service for the Military Message Experiment," *Proceedings of the National Computer Conference, AFIPS*, May, 1979.
8. Tugender, R., and D. R. Oestreicher, "Basic Functional Capabilities for a Military Message Processing Service," ISI/RR-74-23, USC/Information Sciences Institute, 1975.

This research was performed for the Advanced Research Projects Agency under Contract No. DAHC 15 72 C 0308, ARPA Order No. 2223. The views and conclusions expressed in this paper are not necessarily those of any person or organization except the author(s).

Exact solution for the initialization time of packet radio networks with two station buffers

by DANIEL MINOLI

*Bell Telephone Laboratories
Holmdel, New Jersey*

INTRODUCTION

The Packet Radio Network (PRNET), a store-and-forward packet-switching system sharing a single radio channel via multi-access techniques and spread spectrum, is an effective communication medium for data and voice transmission in situations requiring fast deployment, non-fixed hardware locations, encryption and anti-jamming in hostile military environments.^{1-5,14,15}

Many investigators have developed analytical solutions for the performance of channel access schemes typically employed (pure ALOHA, CSMA, etc.).⁷⁻¹³ These studies have shown, among other things, that PRNET's can be easily saturated and become unstable unless efficient routing and flow control algorithms are used. To enable point-to-point packet transportation the network station assigns a code (label) to each repeater; the process of assigning such labels is referred to as "network initialization." The initialization procedure assumes that the network topology is not known a priori and is changing with time. Thus, the initialization procedure involves mapping of network topology, determining network structure (labels for repeaters) and transmitting labels to the repeaters.

Notwithstanding its importance,¹⁹ the problem of initialization has not been studied extensively except in References 5, 16, 17 and 18.

In this paper we present a Markov chain model which enables one to obtain in closed form the optimal rates at which repeaters and station must transmit initialization packets and labels to minimize the network initialization time in a one-hop network, two buffer station. This is an extension of the work reported in References 17 and 18.

In Reference 17 a Markov Chain model for initialization of 1-hop packet radio networks was discussed; with this model the total initialization time must be obtained numerically by solving a set of linear simultaneous equations. For an m repeater network the number of such equations turns out to be $0(m^3)$. Hence, this model is not applicable for analyzing large networks.

The complexity of such models can be reduced by modifying the label queue management from a random selection discipline to a first-in, first-out (FIFO) discipline; the number of linear simultaneous equations that need to be solved

can be reduced from $0(m^3)$ to $0(m^2)$. In this case we can actually go a step further, and obtain closed-form solutions. In Reference 18 the exact solution for the initialization time with one station buffer for the label queue was derived.

In this paper we formulate a new Markov Chain initialization model based on FIFO queue management and we derive exact solutions for this new model when the station has two buffers for the complete interference case. Our most important result is that the network initialization time is relatively insensitive to the station transmission rate, but the repeater transmission rate must be carefully chosen to achieve rapid initialization.

INITIALIZATION AND INITIALIZATION PROTOCOL

Network initialization must be performed whenever the network resumes operation from cold, or whenever the network topology changes. Such topology changes may occur quite frequently; this may be due to decreases in repeater transmission ranges due to battery power drainage or equipment failure; or due to the severe variations in received signal strength caused by the topology of the terrain, man-made structures, foliage, multipath distortion and fading. In addition to this problem of monitoring RF connectivity, the potential mobility of the network must be taken into consideration; the initialization algorithm and its efficiency are particularly significant in such cases.⁴

The initialization procedure considered in this paper—and which is typical for this operation—consists of the following steps: a non-initialized repeater transmits Repeater-On-Packets (ROPs) informing the station of its existence and unique identification; a station program determines a label for the specific repeater and the station places a Label Packet (LLP) in its Label queue for transmission to the repeater. We refer to this queue as the station buffer. After the repeater has received the LLP and acknowledged its receipt, the repeater is considered labeled. The time required to initialize all repeaters is the network initialization time.

We consider a PRNET with a single station and m repeaters in which all devices can communicate directly; a slotted ALOHA access scheme is assumed. In such an environment, the available channel is time slotted into segments of

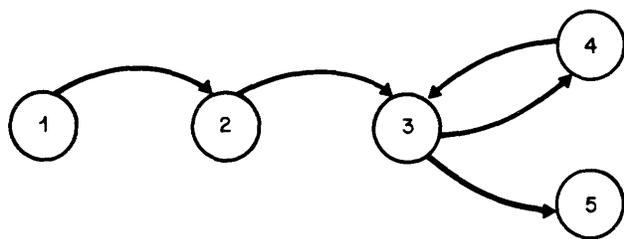


Figure 1—States of a repeater during initialization and possible transitions from status 1 (non-initialized) to status 5 (initialized).

and where:

p = Repeater transmission rate

q = Station transmission rate

$I(i)$ = Average number of interfering repeaters, given i repeaters are initialized.

The expression is exact for complete interference, i.e., $I(i) = m - 1 - i$.

NUMERICAL RESULTS FOR THE TWO-STATION BUFFER CASE

Figures 2, 3, and 4 depict the optimal parameter values and initialization time for complete interference, for $I(i) = 2$, and no interference, respectively, for an m -repeater network. The optimal values were obtained by point-by-point search and are accurate to two significant digits.

Figure 5 shows the initialization time as a function of m , parameterized on $I(i)$. The initialization time for a one-buffer system is shown for comparison.

The following can be observed:

1. The optimal station transmission is nearly constant at $q^* \approx 0.40$, showing a slight tendency to decrease as m increases.

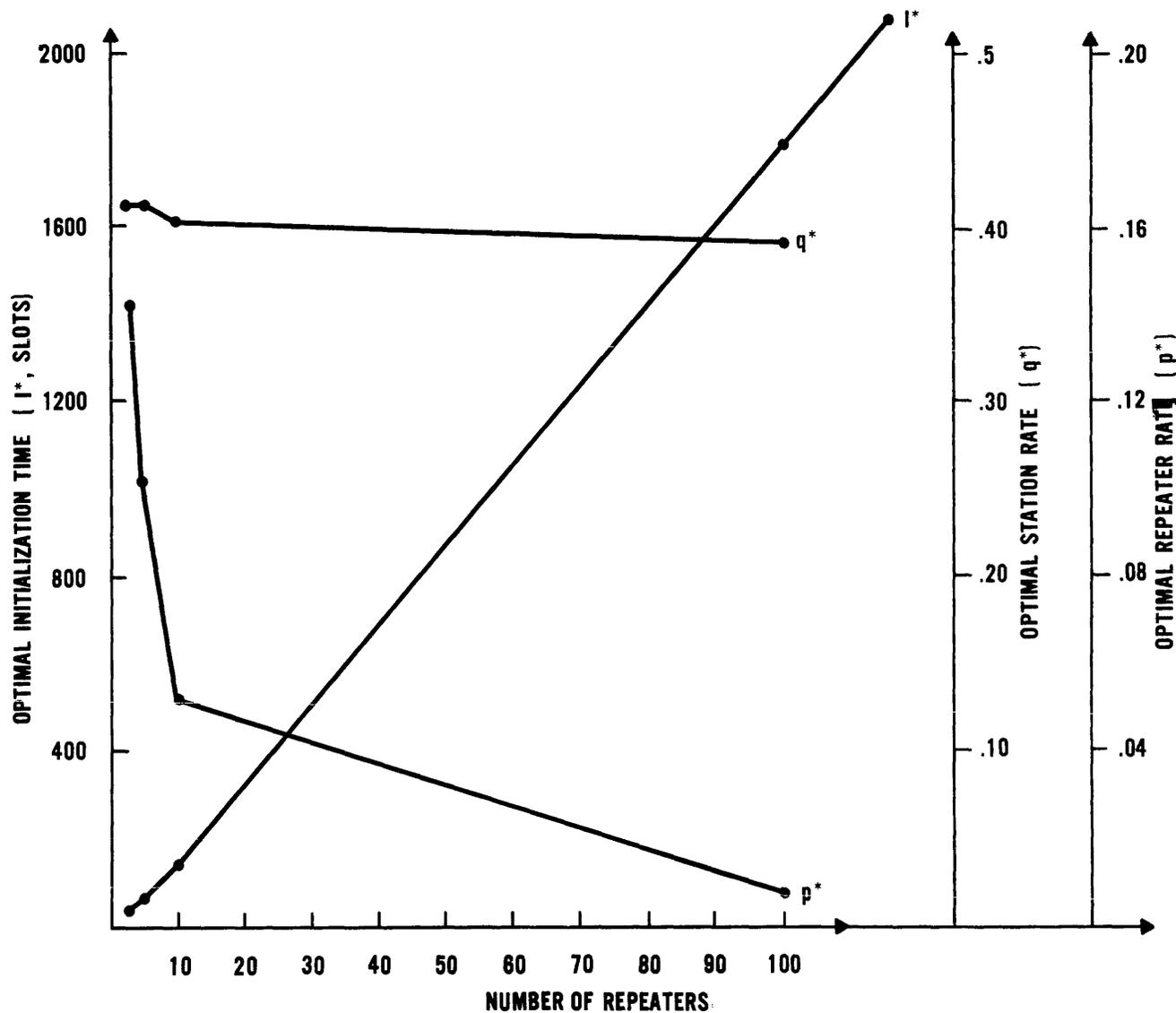


Figure 2—Complete interference, two buffers.

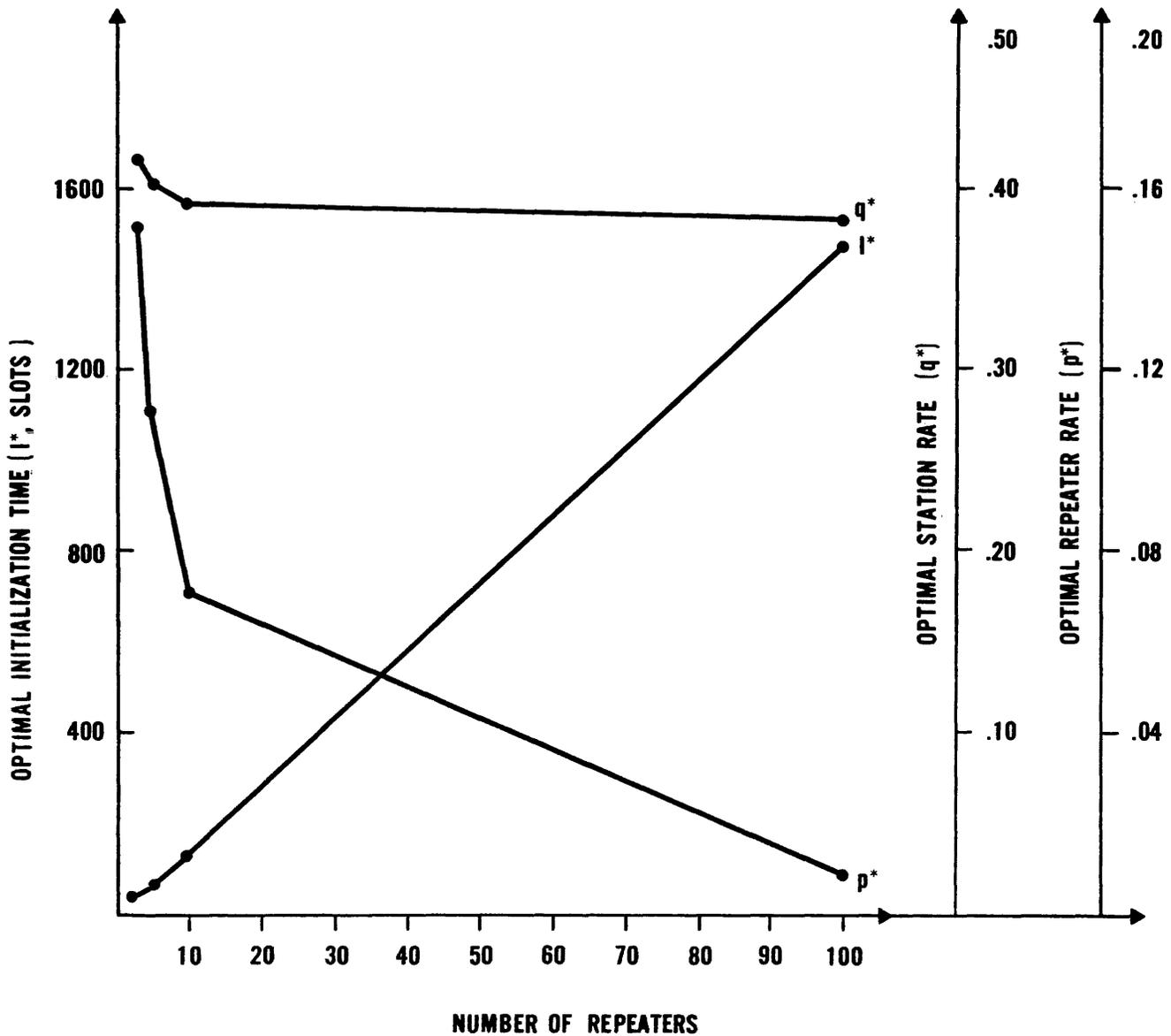


Figure 3—Partial interference, $I(i)=2$, two buffers.

2. There is only a very small difference in the optimal station rate at low or high interference.
3. For complete interference $p^* < 1/m$, and for large m , $p^* \approx 0.7/m$ (empirically determined).
4. As the interference decreases, p^* increases (as expected); $p < 1/m$ and $p^* \approx 0.8/m$ for large m .
5. As the interference decreases, the initialization time decreases. There is a reduction of about 15 percent in initialization time as we go from complete interference to zero interference.

COMPARISON OF ONE-BUFFER AND TWO-BUFFER CASES

Single station buffer

It was shown in Reference 18 that the initialization time for a single hop m -repeater packet radio network is given

by:

$$I = \sum_{i=0}^{m-1} \left\{ \frac{1}{(m-i)p(1-p)^{m-i-1}} + \frac{1}{q(1-p)^{i(i+1)}} + \frac{1}{(1-q)(1-p)^{m-i-1}} \left(1 + \frac{[1-(1-q)(1-p)^{m-i-1}]}{q(1-p)^{i(i)}} \right) \right\} \quad (4)$$

A comparison between the previous cases shows:

1. The station's optimal transmission rate decreased 10 percent as we went from one buffer to two buffers. This is explained by the fact that in a two-buffer situation there is no pressing urgency to clear the one occupied buffer since another is available for accepting ROPs.
2. The repeater's transmission rates are unchanged on the average, indicating that the repeaters can be ignorant of the station's buffer number.

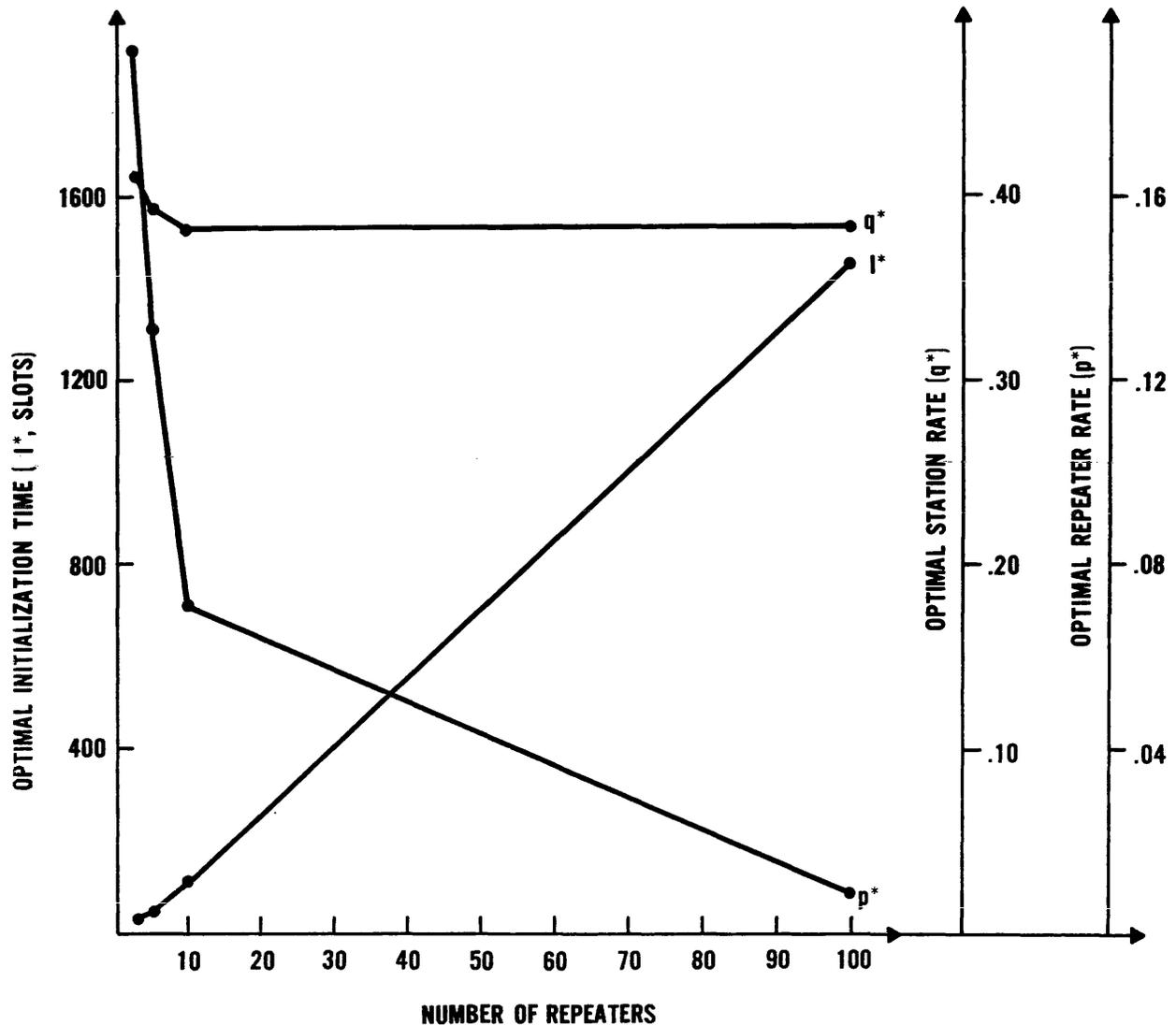


Figure 4—Zero interference, two buffers.

3. The initialization time decreased as we went from one buffer to two buffers. Approximately 15 percent less time is required with two buffers than with one buffer.

CONCLUSIONS

Closed-form solutions for the initialization time of a single-hop packet radio network were obtained. The solutions are for the case in which the station has one or two buffers for storing and sending labels and when it uses a first-in-first-out queue management strategy. The slotted ALOHA access scheme was assumed. The optimum values of repeater and station transmission rates as a function of the interference pattern of repeaters were experimentally obtained. These optimum rates result in minimum initialization times.

The following conclusions emerge from the studies:

1. The initialization time with two buffers at the station

is approximately 15 percent smaller than with one buffer, for the same interference pattern and network size.

2. The optimal station transmission rate, q^* , is nearly independent of m (number of repeaters), for both values of the buffer size; however, q^* decreased 10 percent as we went from one buffer to two buffers. Thus, q is a function of the station's architecture only.
3. The optimal repeater transmission rates were independent of the buffer size on the average, indicating that the repeaters need not be aware of the station's buffer structure.
4. The optimal repeater transmission rate increases about 20 percent as interference goes from complete to zero; the initialization time decreases about 15 percent as we go from complete interference to zero interference.
5. In both cases for large m optimal repeater transmission rates were proportional to $1/m$ where m is number of repeaters in the network.

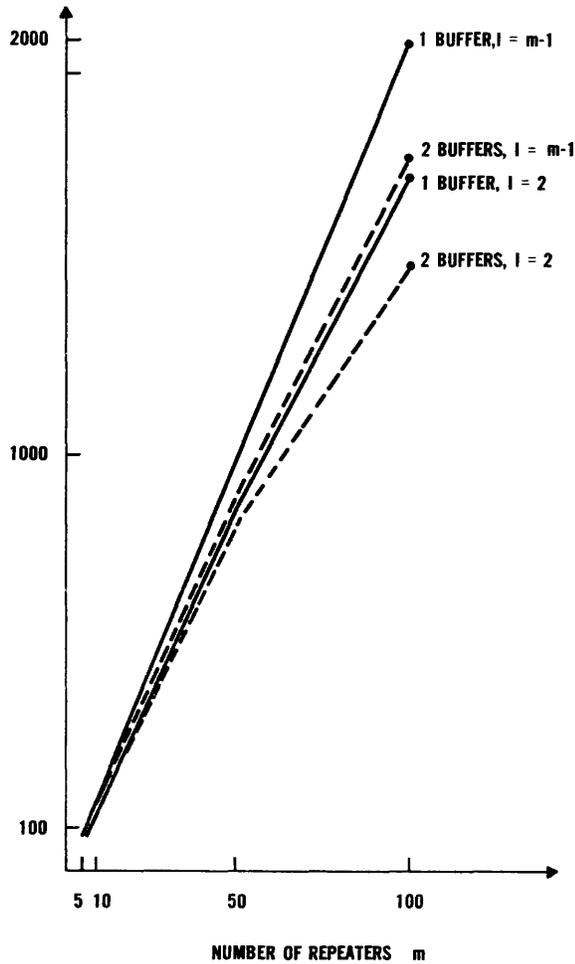


Figure 5—Initialization time as a function of m , $I(i)$ and b .

REFERENCES

1. Frank, H., I. Gitman and R. VanSlyke, "Packet Radio System-Network Considerations," *National Computer Conference*, May 1975, pp. 217-231.
2. Kahn, R. E., "The Organization of Computer Resources into a Packet Radio Network," *National Computer Conference*, May 1975, pp. 177-186.
3. Gitman, I., R. M. VanSlyke and H. Frank, "Routing in Packet Switching Broadcast Radio Network," *IEEE Transactions in Communications*, August, 1976.
4. Minoli, D., and I. Gitman, "Combinatorial Issues in Mobile Packet Radio Networks," *IEEE Trans. on Comm.*, December, 1978, pp. 1821-1826.
5. Minoli, D., and I. Gitman, "Monitoring Mobile Packet Radio Devices," *IEEE Trans. on Comm.*, February 1979, Part II, pp. 509-517.
6. Gitman, I., "On the Capacity of Slotted Aloha Networks and Some Design Problems," *IEEE Trans. on Communications*, Vol. COM-23, No. 3, March 1975.
7. Kleinrock, L., and F. A. Tobagi, "Packet Switching in Radio Channels: Part I—Carrier Sense Multiple Access Modes and Their Throughput-Delay Characteristics," *IEEE Trans. Communications*, Vol. COM-23, December, 1975, pp. 1400-1416.
8. Tobagi, F. A., and L. Kleinrock, "Packet Switching in Radio Channels: Part II—The Hidden Terminal Problem and the Carrier Sense Multiple Access Mode with a Busv Tone," *IEEE Trans. Communications*, Vol. COM-23, December 1975, pp. 1417-1433.

9. Lam, S., "Packet Switching in a Multi-Access Broadcast Channel with Applications to Satellite Communication in a Computer Network," Sch. of Eng. and Appl. Sci., Univ. of California, Los Angeles, UCLA-ENG-7429, April 1974.
10. Kleinrock, L., and S. Lam, "Packet Switching in a Multi-access Broadcast Channel: Performance Evaluation," *IEEE Trans. Communications*, Vol. COM-23, April, 1975, pp. 410-423.
11. Metcalfe, R. M., "Steady-State Analysis of a Slotted and Controlled ALOHA System with Blocking," *Proc. 6th Hawaii Int. Conf. Syst. Sci.*, January 1973, pp. 375-380.
12. Carleial, A. B., and M. F. Hellman, "Bistable Behavior of ALOHA-Type Systems," *IEEE Trans. Communications*, Vol. COM-23, April, 1975, pp. 401-410.
13. Fayclie, G., E. Gelenbe and J. Labetoulle, "Stability and Control of Packet-Switching Broadcast Channels," *J. Assn. Comput. Mach.*, Vol. 24, July, 1977, pp. 387-396.
14. Garret, J., and S. Fraclick, "Technological Considerations for Packet Radio Networks," *Proc. National Computer Conference*, May 1975, pp. 233-243.
15. Burchfiel, J., R. Tomlinson and M. Beeler, "Functions and Structure of a Packet Radio Station," in *Proc. National Computer Conference*, May 1975, pp. 245-251.
16. Minoli, D., "An Approximate Analytical Model for Initialization of Single Hop Packet Radio Networks," *1978 Canad. Conf. on Comm. and Power*, Conference Record, pp. 110-118.
17. Minoli, D., et al., "Analytical Models for Initialization of Single-Hop Packet Radio Networks," Submitted to *IEEE Trans. Commun—Spec. Issue on Digital Radio*.
18. Minoli, D., "Closed Form Expressions for Initialization Time of Packet Radio Networks," *Frequenz*, May 1979.
19. Kahn, R. E., et al., "Advances in Packet Radio Technology," *Proc. IEEE*, November 1978, pp. 1468-1496.

APPENDIX A—INITIALIZATION MODEL

Transition probabilities

Because of the assumptions of complete interference and FIFO it is easy to compute the state transition matrix. For the complete interference case, a packet is successfully received by the station, if all other repeaters and the station are silent. Similarly, for a packet to be received successfully by a repeater it is required that all repeaters be silent.

We associate a status with each repeater during the initialization process, as in the second section. The state of the system is defined by the number of repeaters in each status. Given the present state of the chain $(m_1, m_2, m_3, m_4, m_5)$, where m_i is the number of repeater in the status i , the following list enumerates all possible transitions. We consider $m_3=0$ and $m_3=1$ separately.

1. No repeater received a label in the previous slot ($m_3=0$)
 - a. Successful ROP when station queue is not empty. This requires $m_1>0$, $0<m_2+m_4<b$. Transition to state $(m_1-1, m_2+1, 0, m_4, m_5)$ occurs with probability $(1-q)m_1p(1-p)^{m_1+m_2-1}=Z_1'$.
 - b. Successful ROP when station queue is empty. This requires $m_1>0$, $m_2+m_4=0$. Transition to state $(m_1-1, m_2+1, 0, m_4, m_5)$ with probability $m_1p(1-p)^{m_1+m_2-1}=Z_1''$. Since Z_1' and Z_1'' represent probabilities of mutually exclusive events, we will use one notation Z_1 which will denote Z_1' or Z_1'' , depending on the event.

- c. Successful label to a repeater with status 2. This cannot occur in the present model unless $m_4=0$. Transition to state $(m_1, m_2-1, 1, m_4, m_5)$ with probability $Z_2=(1-p)^{m_1+m_2}q$.
 - d. Successful label to the repeater in status 4. This requires $m_4>0$. Transition to state $(m_1, m_2, 1, m_4-1, m_5)$ with probability $(1-p)^{m_1+m_2}q=Z_3$.
 - e. Else remain in same state with probability $1-Z_1-Z_2-Z_3$.
2. Some repeater received a label in the previous slot ($m_3=1, m_4=0$)
 - a. Successful ETE Ack. Transition to state $(m_1, m_2, 0, m_4, m_5+1)$ occurs with probability $(1-p)^{m_1+m_2}(1-q)$.
 - b. Unsuccessful ETE Ack; transition to state $m_1, m_2, 0, m_4+1, m_5)$ with probability $1-(1-p)^{m_1+m_2}(1-q)$.
 3. If the present state is $(0, 0, 0, 0, m)$ go to state $(m, 0, 0, 0, 0)$ with probability 1.

It is elementary to verify that a unique stationary vector, $(\Pi_1, \Pi_2, \dots, \Pi_n)^T$, exists for this chain. Also, if the n th state is completely labeled state, $(0, 0, 0, 0, m)$, then the expected inter-arrival time between visits to state n is $1/\Pi_n-1$. Hence, the expected initialization time is $1/\Pi_n-1$.

APPENDIX B—DERIVATION OF SOLUTION FOR A TWO-BUFFER MODEL WITH FIFO SERVICE DISCIPLINE

By a cycle of a Markov chain we mean a section of the state transition diagram. We consider cycles C_i such that

1. The C_i s are disjoint.
2. Any state of the chain belongs to one cycle.
3. C_i and C_{i+1} , when considered as graphs, are isomorphic for all but possibly the first and last case.

The solution strategy involves computing the expected time to traverse one cycle, and then summing these over cycles $0, 1, \dots, m-1$.

General strategy

It can be observed that closed-form computations are relatively easy when the number of states by which one can enter a cycle is one. To achieve this, we define the following cycles:

- Cycle -1 Start with $(m, 0, 0, 0, 0)$
End with $(m-1, 1, 0, 0, 0)$
- Cycle i Start with $(m-i-1, 1, 0, 0, i)$
End with $(m-i-2, 1, 0, 0, i+1), i=0, 1, 2, \dots, m-2$
- Cycle $m-1$ Start with $(0, 1, 0, 0, m-1)$
End with $(0, 0, 0, 0, m)$

Note that Cycle -1 involves only the successful sending of a ROP to the station while the Cycle $m-1$ involves only the labeling of the last repeater after all others have been labeled, after this repeater has successfully sent a ROP.

Thus at the beginning of Cycle $i, i=0, 1, \dots, m-2$, there are $m-i-1$ repeaters with status 1, one repeater with status 2, and i repeaters with status 5.

Figure 6 illustrates the states for a typical cycle $0 \leq i \leq m-2$. SR, SL, SE, UE represent successful ROP, successful label, successful ETE ACK, and unsuccessful ETE ACK, respectively; notice that now there are three separate classes of routes which the chain can take to traverse the cycle. Each cycle starts with 1 ROP in the buffer. Either one of the following classes of routes can be taken:

1. Another ROP is received by the station before a label is received by a repeater (follow the lower route passing through states $4_i, 5_i$, and possibly 6_i).
2. The label is received and acknowledged by the repeater before another ROP is received (upper route without cross over passing through states $2_i, 0_i$, and possibly 3_i).
3. The label is dispatched several times but before it is successfully acknowledged another ROP is received (upper route with cross over passing through states $2_i, 3_i, 6_i$, and 5_i).

Our calculations involve examining the expected time to traverse each of these classes of routes, and deciding with what probability each is chosen.

Let $E[T_i]$ be the expected time to go through cycle i ; then the total expected initialization time is the sum of the expected times required to traverse each cycle:

$$I = \sum_{i=-1}^{m-1} E[T_i] \tag{B.1}$$

Since Cycle -1 , and $m-1$ are different, we address these first. Throughout the analysis we use the fact that the geometrical distribution is memoryless; that is, if τ is a geometrically-distributed random variable; then

$$Prob(\tau > i + j | \tau > i) = Prob(\tau > j)$$

for i, j integers.

In our case τ is the time spent in a specific state of the Markov chain; the time to leave any state is geometrically distributed.

Consider the following particular scenario. The chain is in state A . If event E occurs transition to state B occurs; else if \bar{E} (its complement) occurs go to state C . Remain in state C until some other event F occurs then return to state A . See Figure 7. Let $E[T_{AB}]$ and $E[T_{CA}]$ be the expected time to go from A to B and C to A , respectively. Because of the abovementioned memoryless property we obtain

$$E[T_{AB}] = \frac{1 + (1 - p(E))E[T_{CA}]}{p(E)}$$

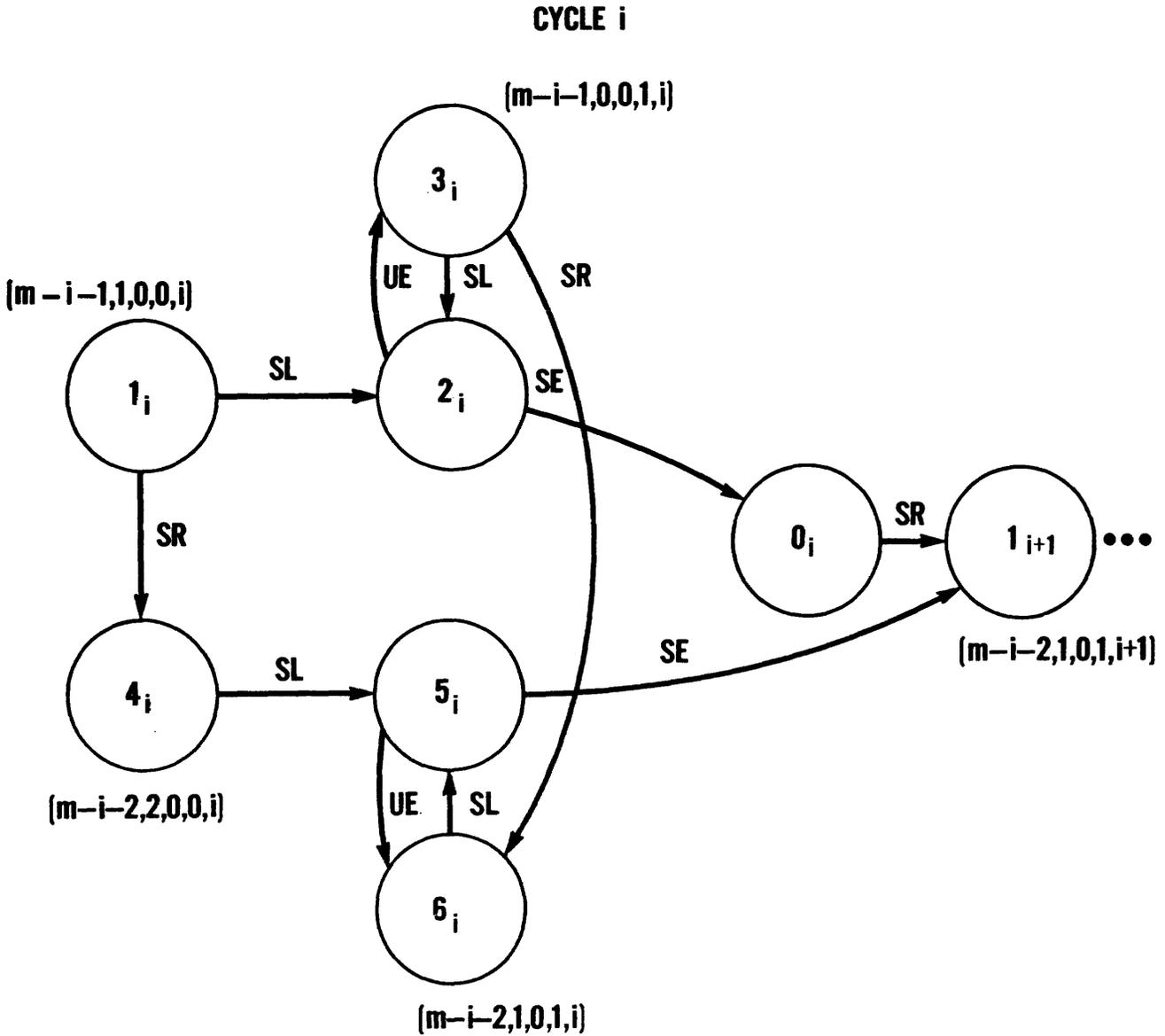


Figure 6—Cycle $i, i=0, 1, \dots, m-2$.

Cycles -1 and m-1

Figure 8 depicts that section of the state transition diagram for cycle -1 and $m-1$.

Evaluation of $E[T_{-1}]$

This involves a SR, given that all m repeaters are uninitialized; this requires one repeater to transmit and all others to remain silent. Thus the expected time of Cycle -1 is:

$$E[T_{-1}] = \frac{1}{P_{0,-1,0}} = \frac{1}{mp(1-p)^{m-1}} \quad (B.2)$$

Evaluation of $E[T_{m-1}]$

Observe that, except for the number of unlabeled repeaters, this section of the state diagram is the same as a cycle

for a one-buffer problem. As in Reference 18 we obtain

$$\begin{aligned} E[T_{m-1}] &= E[T_{1_{m-1}2_{m-1}}] + E[T_{2_{m-1}0_{m-1}}] \\ &= \frac{1}{q(1-p)} \\ &\quad + \frac{1}{P_{2_{m-1}0_{m-1}}} \\ &\quad \{1 + E[T_{3_{m-1}2_{m-1}}](1 - P_{2_{m-1}0_{m-1}})\} \end{aligned} \quad (B.3)$$

where $P_{2_{m-1}0_{m-1}} = Prob$ (ETE Ack is successful.)

Since every repeater is silent at this stage, we only require that the station be silent; thus, $P_{2_{m-1}0_{m-1}} = (1-q)$, giving the expected time for Cycle $m-1$:

$$E[T_{m-1}] = \frac{1}{q(1-p)} + \frac{2}{1-q} \quad (B.4)$$

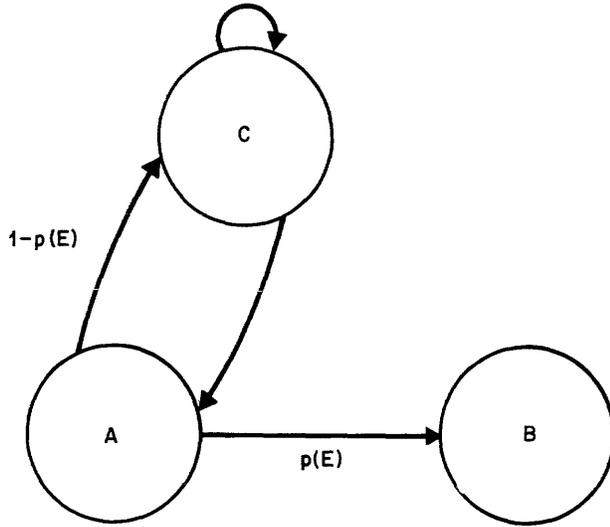


Figure 7—Transition time under memoryless distributions.

Evaluation of $E[T_i]$, $i=0, 1, \dots, m-2$

For a typical Cycle i , we are interested in quantifying the time needed to traverse it. Such time is made up of various components, indicated as follows:

Computation of $E[T_{1,i+1}]$

To compute this expression, assume that at the current slot the state of the chain is 1_i , then condition on the outcome of the next slot. It is easy to derive

$$P_{1,2i} = \text{Prob}(\text{go from } 1_i \text{ to } 2_i) = q(1-p)^{1+I(i)} \quad (B.5)$$

$$P_{1,4i} = \text{Prob}(\text{go from } 1_i \text{ to } 4_i) = (1-q)(1-p)^{m-i-1}p(m-i-1) \quad (B.6)$$

Then, referring to Figure 6, we see that

$$E \left[T_{1,i+1} \mid \begin{matrix} \text{outcome of} \\ \text{next slot} \end{matrix} \right] = \begin{cases} 1 + E[T_{2,i+1}] & \text{if a label is delivered} \\ 1 + E[T_{4,i+1}] & \text{if a ROP is received} \\ 1 + E[T_{1,i+1}] & \text{if neither of the above} \end{cases} \quad (B.7)$$

where we have made use of the memoryless property previously described. Unconditioning, we obtain:

$$E[T_{1,i+1}] = (1 + E[T_{2,i+1}])P_{1,2i} + (1 + E[T_{4,i+1}])P_{1,4i} + (1 + E[T_{1,i+1}])((1 - P_{1,2i} - P_{1,4i})) \quad (B.8)$$

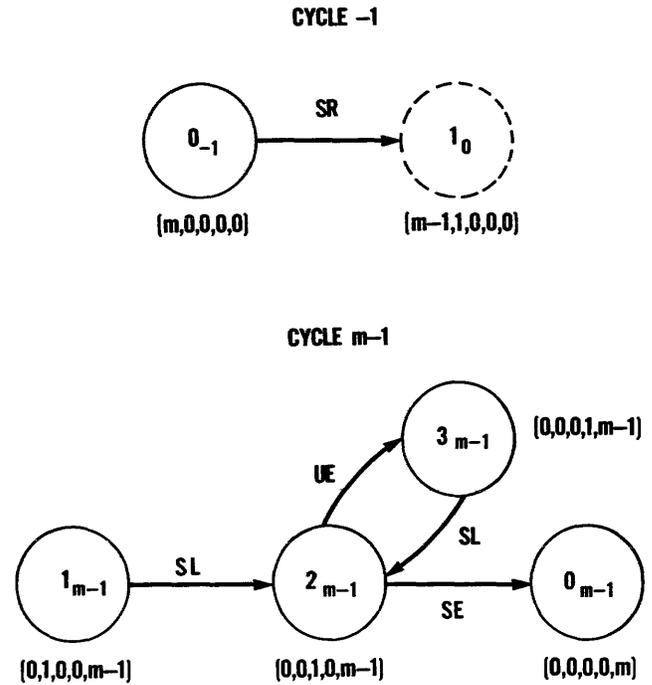


Figure 8—Cycles -1 and $m-1$.

Solving for $E[T_{1,i+1}]$, we obtain:

$$E[T_{1,i+1}] = \frac{1 + P_{1,2i}E[T_{2,i+1}] + P_{1,4i}E[T_{4,i+1}]}{P_{1,2i} + P_{1,4i}} \quad (B.9)$$

We notice that to proceed we need $E[T_{2,i+1}]$ and $E[T_{4,i+1}]$; we address $E[T_{4,i+1}]$ next.

Computation of $E[T_{4,i+1}]$

Referring to Figure 6, we see that

$$E[T_{4,i+1}] = E[T_{4,5i}] + E[T_{5,i+1}] \quad (B.10)$$

We now evaluate the expressions on the right-hand side of Equation B.10. It is quite simple to show that:

$$E[T_{4,5i}] = \frac{1}{P_{4,5i}} = \frac{1}{q(1-p)^{1+I(i)}} \quad (B.11)$$

since we require a successful label delivery from the state $(m-i-2, 2, 0, 0, i)$.

To obtain $E[T_{5,i+1}]$ we must condition on the outcome of the next slot, namely whether the ETE Ack is successful or not. Using the analysis applied to the one buffer case we find:

$$E[T_{5,i+1}] = \frac{1}{P_{5,i+1}} \{1 + E[T_{6,5i}](1 - P_{5,i+1})\} \quad (B.12)$$

Note that:

$$P_{5,i+1} = (1-p)^{m-i-1}(1-q) \quad (B.13)$$

since we are in state $(m-i-2, 1, 1, 0, i)$, and the station

and all other unlabeled repeaters must be silent for the ETE to be successful. Also

$$E[T_{6;5_i}] = \frac{1}{P_{6;5_i}} = \frac{1}{q(1-p)^{i(i)}} \quad (\text{B.14})$$

since one less repeater is now active after state 6_i is reached. Finally:

$$E[T_{4;1+i}] = \frac{1}{q(1-p)^{1+i(i)}} + \frac{1}{(1-p)^{m-i-1}(1-q)} \left\{ 1 + [1 - (1-p)^{m-i-1}(1-q)] \frac{1}{q(1-p)^{i(i)}} \right\} \quad (\text{B.15})$$

Evaluation of $E[T_{2;1+i}]$

Assuming the current state of the chain to be 2_i and conditioning on the event of the next slot, namely whether the ETE Ack is successful, we immediately see

$$E \left[T_{2;1+i} \mid \begin{array}{l} \text{event of} \\ \text{next slot} \end{array} \right] = \begin{cases} 1 + E[T_{0;1+i}] & \text{if ETE Ack} \\ & \text{successful} \\ 1 + E[T_{3;1+i}] & \text{if ETE Ack} \\ & \text{unsuccessful} \end{cases} \quad (\text{B.16})$$

Note that (see Figure 6):

$$P_{2;0_i} = \text{Prob (ETE Ack is successful from state } 2_i) \\ = (1-q)(1-p)^{m-i-1} \quad (\text{B.17})$$

Unconditioning, we get:

$$E[T_{2;1+i}] = P_{2;0_i}(1 + E[T_{0;1+i}]) \\ + (1 - P_{2;0_i})(1 + E[T_{3;1+i}]) \quad (\text{B.18})$$

Since

$$E[T_{0;1+i}] = \frac{1}{(m-i-1)p(1-p)^{m-i-2}} \quad (\text{B.19})$$

(due to the fact that the station is automatically silent when the label queue is empty) we obtain:

$$E[T_{2;1+i}] = 1 + P_{2;0_i} \frac{1}{(m-i-1)p(1-p)^{m-i-2}} \\ + (1 - P_{2;0_i})E[T_{3;1+i}] \quad (\text{B.20})$$

Thus, we need to find $E[T_{3;1+i}]$.

Evaluation of $E[T_{3;1+i}]$

Note this calculation is more complicated than the single-buffer case because it is possible to transfer from 3_i to 6_i ,

i.e.; another ROP is received before label delivery. Again, we condition on the next slot.

$$E \left[T_{3;1+i} \mid \begin{array}{l} \text{event of} \\ \text{next slot} \end{array} \right] = \begin{cases} 1 + E[T_{2;1+i}], & \text{if label successful} \\ 1 + E[T_{6;1+i}], & \text{if ROP successful} \\ 1 + E[T_{3;1+i}], & \text{if neither} \end{cases} \quad (\text{B.21})$$

Unconditioning:

$$E[T_{3;1+i}] = (1 + E[T_{2;1+i}])P_{3;2_i} + (1 + E[T_{6;1+i}])P_{3;6_i} \\ + (1 + E[T_{3;1+i}])(1 - P_{3;1+i} - P_{3;6_i}) \quad (\text{B.22})$$

where

$$P_{3;6_i} = \text{Prob (ROP is successful from state } 3_i) \\ = (m-i-1)p(1-p)^{m-i-2}(1-q) \quad (\text{B.23})$$

$$P_{3;2_i} = \text{Prob (label is successful from state } 3_i) \\ = q(1-p)^{i(i)} \quad (\text{B.24})$$

Solving, we obtain:

$$E[T_{3;1+i}] = \frac{1 + P_{3;2_i}E[T_{2;1+i}] + P_{3;6_i}E[T_{6;1+i}]}{P_{3;2_i} + P_{3;6_i}} \quad (\text{B.25})$$

$E[T_{6;1+i}]$ can be written as:

$$E[T_{6;1+i}] = E[T_{6;5_i}] + E[T_{5;1+i}] \quad (\text{B.26})$$

where:

$$E[T_{6;5_i}] = \frac{1}{q(1-p)^{i(i)}} \quad (\text{B.27})$$

and $E[T_{5;1+i}]$ was derived two sections ago. Hence, we obtained an equation for $E[T_{3;1+i}]$ in terms of $E[T_{2;1+i}]$. Together with the equation of the third subsection, we have a system which must be solved simultaneously.

Evaluation of $E[T_i]$ —The final result

We now pull things together; from the beginning of the third subsection we have:

$$E[T_{1;1+i}] = \frac{1}{P_{1;2_i} + P_{1;4_i}} + \frac{P_{1;2_i}}{P_{1;2_i} + P_{1;4_i}} E[T_{2;1+i}] \\ + \frac{P_{1;4_i}}{P_{1;2_i} + P_{1;4_i}} E[T_{4;1+i}] \quad (\text{B.28})$$

From the next subsection we have:

$$E[T_{4i+1}] = E[T_{4i}] + \frac{1}{P_{5i+1}} \{1 + E[T_{6i}](1 - P_{5i+1})\} \quad (B.29)$$

We only need $E[T_{2i+1}]$. From the next two subsections we have:

$$E[T_{2i+1}] = 1 + P_{2i}E[T_{0i+1}] + (1 - P_{2i})E[T_{3i+1}] \quad (B.30)$$

and

$$E[T_{3i+1}] = \frac{1}{P_{3i} + P_{3i+1}} + \frac{P_{3i+1}}{P_{3i} + P_{3i+1}} E[T_{2i+1}] + \frac{P_{3i}}{P_{3i} + P_{3i+1}} E[T_{6i}] + E[T_{5i+1}] \quad (B.31)$$

The following procedure is employed:

1. Substitute $E[T_{5i+1}]$ (which has just been computed) into Equation B.31.
2. Solve the system of Equations B.30 and B.31 for $E[T_{2i+1}]$.
3. The answer to 2 is in terms of $E[T_{6i}]$, which was just derived; substitute this expression.

We thus have $E[T_{2i+1}]$, namely:

$$E[T_{2i+1}] = \left\{ 1 + (1-q)(1-p)^{m-i-1} \frac{1}{(m-i-1)p(1-p)^{m-i-2}} + [1 - (1-q)(1-p)^{m-i-1}] \left[\frac{1}{(m-i-1)p(1-p)^{m-i-2}(1-q) + q(1-p)^{i(i)}} \right] + \frac{1}{(1-p)^{m-i-1}(1-q)} \cdot \frac{(m-i-1)p(1-p)^{m-i-2}(1-q)}{(m-i-1)p(1-p)^{m-i-2}(1-q) + q(1-p)^{i(i)}} \times \left[1 + \frac{1}{q(1-p)^{i(i)}} \right] \right\} \cdot \left\{ 1 - \frac{[1 - (1-q)(1-p)^{m-i-1}]q(1-p)^{i(i)}}{(m-i-1)p(1-p)^{m-i-2}(1-q) + q(1-p)^{i(i)}} \right\}^{-1} \quad (B.32)$$

Substitute Equations B.29 and B.32 into Equation B.28; we obtain Equation 3, that is,

$$E[T_{1i+1}] = E(T_i) = \text{Equation (3)}$$

Thus, we have accomplished our goal of determining the expected time to traverse a typical cycle. Summing over all cycles gives the total expected initialization time (Equation 2).

Fixing timeout intervals for lost packet detection in computer communication networks

by ROBERT J. T. MORRIS

Bell Laboratories
Holmdel, N.J.

INTRODUCTION

In a packet-switched data communication network which provides internal packet accountability via an end-to-end positive acknowledgment protocol, it is necessary to include a mechanism for detection and retransmission of missing packets. In particular, a decision must be made as to how long a sending element should reasonably wait before declaring an unacknowledged packet lost and initiating a recovery action. This waiting time is one of a class of system parameters which have come to be known as timeout intervals. The choice of a timeout interval is a delicate problem.^{1,2} If it is too short, network capacity is wasted by frequent unnecessary actions for packets which are not lost but merely delayed. If the timeout interval is too long, lost packets cause needlessly prolonged delays before recovery is initiated.

Several recent papers have considered this problem. Sunshine² considered a retransmission scheme where a packet P is retransmitted every T seconds, until an acknowledgment of P is received. He considered the quantities mean delay including retransmission and mean number of transmissions. He illustrated the analysis by considering an Erlangian delay distribution with a defect representing packet loss. As T is varied, the curve mean delay versus mean number of transmissions exhibits a definite "knee."

Fayolle, Gelenbe and Pujolle³ conducted an analysis of a related protocol. A packet P is retransmitted every T seconds until an acknowledgment of the last retransmission of P is received. They derive individually optimal values of T for three separate performance measures: buffer throughput (maximized), response time (minimized) and loss rate due to buffer overflow (minimized).

Buttò, Colombo, Taggiasco and Tonietti⁴ conducted an analytic and simulation study of a nontrivial network and demonstrated that overall network performance could be severely degraded by employing too short a retransmission timeout interval. They identified the following interesting network phenomenon: if a network reaches a critically busy state, delays may be sufficiently long that timeouts continually occur, resulting in a flooding of the network with retransmission traffic which in turn increases delays, causing unstable traffic growth until network saturation occurs. We

will discuss ways of averting this type of behavior in a later section.

McQuillan and Cerf⁵ gave the rule of thumb: the timeout interval should be set equal to the round-trip delay plus a "fudge factor" to account for variance.

The treatment we give here is distinguished by the following features:

1. We begin by identifying what appear to be the performance objectives of major importance in normal network operation. We use a simple model to produce a design method for the timeout interval which *simultaneously* satisfies each objective as well as possible.
2. We are particularly interested in networks using fixed routes or virtual calls. In contrast to References 2 and 3, we do not assume independently distributed packet round-trip times, since a packet and any retransmissions will normally follow the same network path causing highly correlated round-trip times.
3. Despite the inevitably subjective nature of the problem, we defend our design rules by demonstrating the presence of an inherent generality in the approach.
4. We consider the practical implications and *caveats* of the results. Of particular importance are the additional considerations which arise when we allow the possibility of abnormal network conditions caused by faults or congestion. It is shown that in order to obtain satisfactory operation, it is highly desirable that the timeout mechanism be made adaptive in an autonomous manner to the network and connection condition.

A SIMPLE MODEL

In this section we construct a simple model comprised of a network path characteristic and a rudimentary retransmit protocol. The model serves to identify some important performance issues which should be taken into account when fixing the timeout interval T in normal network operation.

Consider a path between two nodes A and B of a packet-switched network. Packets are transmitted from A to B , and B returns individual acknowledgment packets to A . Suppose the probability that a packet is not acknowledged (forward

packet or acknowledgment lost) is L and if the packet is not lost, its round-trip delay (transmission to receipt of acknowledgment) is governed by a continuous distribution F with finite mean μ and complement $F^c=1-F$.

Next, consider a retransmit protocol wherein a packet P is retransmitted every T seconds until an (any) acknowledgment of P is received.

The main performance issues in choosing T are as follows:

1. T should not be so short that frequent "false alarms" occur, i.e., frequent retransmissions of packets which are not lost but merely delayed.
2. T should not be so long that a lost packet causes an unnecessarily prolonged delay in system recovery. A long recovery delay affects system performance in two ways. Firstly, delayed recovery causes prolonged retention of unacknowledged packets in the retransmit buffer, and consequently a wastage of shared buffer space. Secondly, the delayed eventual packet receipt at node B caused by a packet loss can cause a delay which is directly apparent to an application such as terminal to host computer interaction.

Other performance issues related to abnormal network conditions are discussed later.

We now quantify the above considerations. Rather than assume that the round-trip delays of a packet and its successive retransmissions are independent random variables, we adopt the following assumption which is appropriate for fixed-route or virtual call paths.

Assumption A:

No retransmitted packet will pass an earlier version of the same packet.

Now let

D =time from packet first sent until packet acknowledgment first received, i.e., effective round-trip delay.

N =number of transmissions required to successfully deliver one packet.

F' =maximum round-trip delay of a packet which is not lost= $\sup\{t: F(t)<1\}$

X_i denote the i^{th} retransmission of packet X_0 , $i=1, 2, \dots$

Then

$$\begin{aligned}
 E(N) &= \sum_{i=1}^{\infty} i \cdot \Pr\{X_0, \dots, X_i \text{ only sent}\} \\
 &= \sum_{i=1}^{\infty} i \cdot \Pr \left[\bigvee_{j=0}^{i-1} \{X_0, \dots, X_{j-1} \text{ lost}; \right. \\
 &\quad \left. X_j \text{ acknowledged in } [(i-1)T, iT)\} \right], \\
 &\hspace{15em} \text{by Assumption A} \\
 &= \frac{1}{1-L} + \sum_{i=1}^{\infty} F^c(iT) \tag{2.1}
 \end{aligned}$$

and

$$\begin{aligned}
 E(D) &= \sum_{i=0}^{\infty} (\mu + iT) \Pr\{X_i \text{ first to be acknowledged}\} \\
 &= \sum_{i=0}^{\infty} (\mu + iT) \Pr\{X_0, \dots, X_{i-1} \text{ lost}; \\
 &\quad X_i \text{ acknowledged}\}, \\
 &\hspace{15em} \text{by Assumption A} \\
 &= \sum_{i=0}^{\infty} (\mu + iT)L^i(1-L) = \mu + TL/(1-L) \tag{2.2}
 \end{aligned}$$

We now go on to show how $E(N)$ and $E(D)$ are related to considerations 1 and 2, stated previously.

Let us define *transmission efficiency* η as the reciprocal of the number of packets transmitted per successfully delivered packet. A lowering of η causes an increase in effective network load which in turn leads to a depletion of the capacity of the network to do useful work. Consideration 1 is embodied in the requirement that $\eta = E(N)^{-1}$ be kept high. Note that the maximum network "throughput" allowable for a given delay figure will be proportional to η . Thus our first stated objective is to *choose T to keep $E(N)$ small*. According to (2.1), T should be chosen large.

We now turn our attention to consideration 2. First note that $E(D)$ describes the mean holding time in the sender's retransmit buffer. Assuming that the path throughput is constant, then it follows from Little's results that mean retransmit buffer occupancy is proportional to $E(D)$. Attention to mean buffer occupancy is reasonable since buffer space at the sender is shared amongst many independent calls. Next assume (although this can be relaxed) that both packet round-trip delay and loss are approximately equally shared between forward and return paths and that L is small. Then the mean time to successful receipt of a forward packet is approximately $E(D)/2$. Thus both aspects of consideration 2 are taken into account by requiring that $E(D)$ be kept small. Hence our second stated objective is to *choose T to keep $E(D)$ small*. According to (2.2), T should be chosen small, in conflict with our earlier requirement.

In order to study this demonstrated conflict between the requirements of avoiding false alarms yet retaining short delays in the presence of loss, we examine more closely expressions (2.1) and (2.2).

Assume for simplicity that L is non-zero and $F(t)$ is strictly increasing for $t \leq F'$. Then $E(N)$ and $E(D)$ are shown diagrammatically as T varies in Figures 1a,b. The "tradeoff curve" $E(N)$ versus $E(D)$ is shown in Figure 1c.

Considering the multiobjective optimization problem of minimizing both $E(N)$ and $E(D)$ by choice of T , we see immediately that any choice of $T \leq F'$ is pareto-optimal (also known as efficient, noninferior, etc.). The specification of an "optimal" value for T requires precise knowledge about the relative costs of $E(N)$ and $E(D)$. In the absence of any such knowledge, one way of identifying the knee of the tradeoff curve, Figure 1c, is to demand that the operating point be such that the relative degradations of both objec-

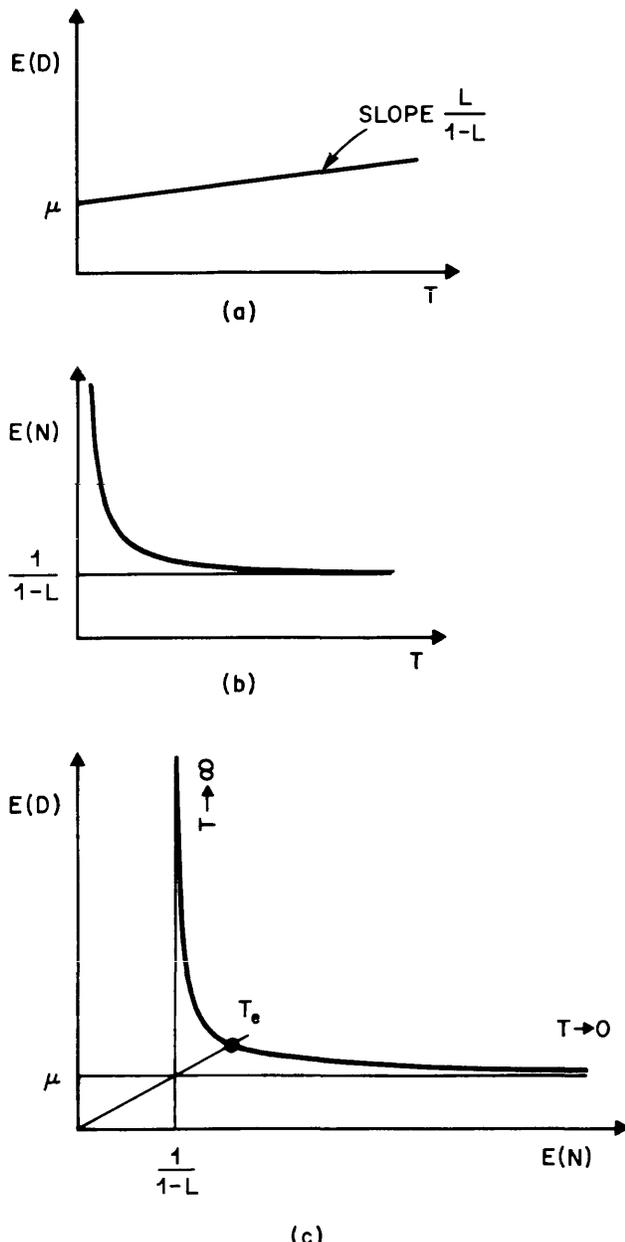


Figure 1

tives from their respective individual optima are equal. Hence we define an equitable value of T denoted T_e , by

$$\frac{E(N)_{T_e} - \frac{1}{1-L}}{\frac{1}{1-L}} = \frac{E(D)_{T_e} - \mu}{\mu} \tag{2.3}$$

Substituting from (2.1) and (2.2) yields

$$\frac{L}{1-L} \frac{T_e}{\mu} = (1-L) \sum_{i=1}^{\infty} F^c(iT_e) \tag{2.4}$$

The value of T_e is shown in Figure 1c and always exists by our assumption that F is continuous and L is non-zero.

We will be particularly interested in the case where L is small, so it is of interest to see what happens to T_e as L approaches zero.

Proposition I:

$$\lim_{L \rightarrow 0} T_e(L) = F'$$

Proof:

Assume first that $F' < \infty$. Then $T_e \leq F'$ or else (2.4) would be contradicted. Letting $L \rightarrow 0$, the lefthand side of (2.4) approaches zero. The only way for the righthand side of (2.4) to approach zero is for T_e to approach F' .

Now consider the case $F' = \infty$. Pick $L_0 \in (0, 1)$. Then for $L < L_0$, $T_e(L)$ must be unbounded, for if it were bounded the righthand side of (2.4) would be bounded below, but the lefthand side of (2.4) cannot be bounded below unless $T_e(L)$ is unbounded. The observation that $T_e(L)$ increases as $L \rightarrow 0$ completes the proof. \square

The effectiveness of the design decision $T = T_e$ naturally depends on the relative importance of the two objectives $E(N)$ and $E(D)$ and in turn the considerations 1 and 2. However, if the value of equation (2.4) is small then little improvement of either objective is possible, and then only at the expense of a worsening of the other objective. Thus when the value of equation (2.4) is small it would be difficult to find grounds on which to criticize the choice $T = T_e$ and for this reason the design can be considered sound.

Hence it is of interest to note that for small loss probability L , the value of equation (2.4) is also small.

Proposition II:

$$\lim_{L \rightarrow 0} \frac{L}{1-L} \frac{T_e(L)}{\mu} = 0$$

Proof:

Assume first that $F' < \infty$. By Proposition I as $L \rightarrow 0$, $T_e(L) \rightarrow F'$ and the result follows.

If $F' = \infty$, then as $L \rightarrow 0$, $T_e(L) \rightarrow \infty$. But

$$\sum_{i=1}^{\infty} F^c(iT_e) \leq \frac{1}{T_e} \int_0^{\infty} F^c(t) dt = \frac{\mu}{T_e}$$

which also approaches zero. \square

A natural question which now arises concerns the sensitivity of the performance objectives $E(D)$ and $E(N)$ to a variation of T away from T_e . We shall answer this question briefly for the case where L is small. By observation of Figures 1a and 1b, it is seen that if T is increased above T_e , the degradation of $E(D)$ will be negligible (since L is small) and the improvement of $E(N)$ slight. On the other hand, if T is decreased below T_e , there is a possibility of a serious worsening of $E(N)$ caused by excessively many false alarms. We consider this possibility further in a later section. An observation of some general value is that for systems with low loss probability, if there is any uncertainty, T should be made longer rather than shorter.

ALTERNATIVE APPROACH AND COMPARISON

Given a delay distribution and loss probability, we have given a technique for choosing a timeout interval. This technique was based on an examination of overall system objectives. We now develop an alternative approach.

We adopt the same model as in the previous section and continue to employ Assumption A.

Consider the first transmission of a packet. Denote by $q(t)$ the conditional probability that the packet is lost given that it has not been acknowledged by time t . Thus

$$q(t) = \frac{L}{L + (1-L)F^c(t)}$$

The curves $q(t)$ and $1 - q(t)$ are plotted in Figure 2. The significance of the point T_d shown on the figure is that

$$\begin{aligned} &Pr\{packet\ is\ lost\ | packet\ not\ acknowledged\ by\ T_d\} \\ &= Pr\{packet\ is\ not\ lost\ | packet\ not\ acknowledged\ by\ T_d\} \end{aligned}$$

and for $t > T_d$, presently available information implies that the packet is more likely lost than delayed. Assumption A now implies that if the packet is delayed but not lost, there is nothing to be gained by retransmission.

Thus, in the absence of further information, an appropriate time to declare the packet lost is at T_d . Note that T_d satisfies

$$q(T_d) = 1 - q(T_d) \tag{3.1}$$

or

$$F^c(T_d) = L / (1 - L) \tag{3.2}$$

Equation 3.2 always has a solution if $L \leq \frac{1}{2}$ and F is continuous and it is clear that $\lim_{L \rightarrow 0} T_d(L) = F'$.

The choice $T = T_d$ is, of course, inherently no less or more arbitrary than the choice $T = T_e$. What is of interest is that, as we now show, there is a degree of consistency between the values T_e and T_d .

We begin with a numerical example.

Example

Consider the case where F is an Erlangian distribution with 16 phases and mean μ . The values T_e and T_d obtained from equations (2.3) and (3.1) are shown as a function of L in Figure 3.

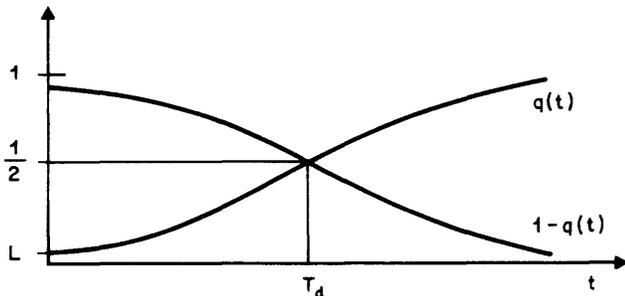


Figure 2

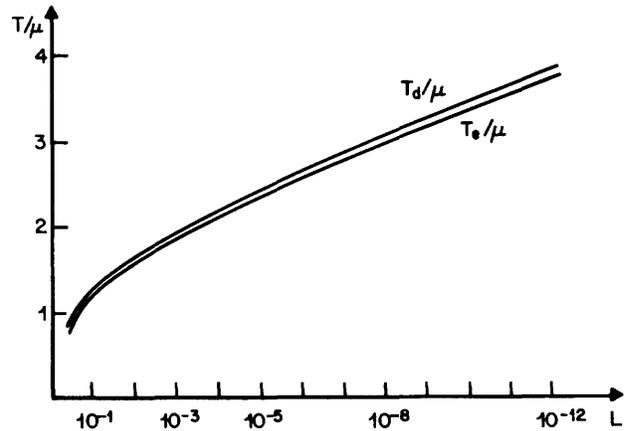


Figure 3

A reasonable agreement between T_e and T_d is observed. Note that for loss probabilities smaller than one percent, the value $LT_e / (\mu(1-L))$ is sufficiently small that it may be argued that T_e provides a sound design, in that little improvement of $E(D)$ or $E(N)$ is possible with a different decision.

For many delay distributions of interest (e.g., Erlangian, hyperexponential, exponential), it can be shown that

$$\lim_{L \rightarrow 0} \frac{T_e(L)}{T_d(L)} = 1 \tag{3.3}$$

Sufficient conditions for (3.3) to be true are that

- a. F has density f satisfying $\liminf_{t \rightarrow \infty} f(t) / F^c(t) > 0$.
- b. F^c is log-asymptotic to a log-concave complementary distribution.⁶

Thus, as well as the reasonable agreement evidenced in the example, it is observed that there is an intrinsic consistency between the two approaches.

It is of interest to observe that if Equation 2.4 is replaced by

$$\frac{1}{1-L} \frac{T_e}{\mu} = a(1-L) \sum_{i=1}^{\infty} F^c(iT_e)$$

where a is arbitrarily positive, then under the above conditions (3.3) remains true. This implies that the asymptotic (small L) behavior of T_e is independent of the relative costs attributed to the delay and efficiency objectives.

Similarly, if Equation 3.1 is replaced by

$$q(T_d) = b[1 - q(T_d)] \tag{3.4}$$

where b is positive and allows a solution to (3.4), then under the above conditions, (3.3) also remains true.

ABNORMAL NETWORK CONDITIONS

In the two previous sections we have considered that a network path is characterized by a fixed delay distribution

F and a fixed loss probability L . While this assumption is reasonable under normal network operation, F and L may be considerably altered during periods of network congestion or partial failure. A poor choice of timeout intervals can even contribute to a change in F and L .

In the introduction we described an effect where too short a timeout interval T led to continual retransmission of a packet causing increased traffic load and in turn increased delay. Such an effect can lead to artificially-induced network saturation and it is necessary to incorporate some adaptation into the timeout mechanism to prevent this hazard. The measurement of packet round-trip times by the sender for the purposes of detecting timeouts provides valuable information on the current network condition and T can be set using the method of the two previous sections, but taking into account recent observations of round-trip delay. However, this does not provide complete protection, since a sudden change in path or network condition could result in a sudden increase in mean round-trip time. A transmitter should interpret the occurrence of a timeout as meaning that a packet has been lost, or, quite possibly, that an increase in round-trip delay has occurred. Thus the transmitter's response to a timeout should include a lengthening of the timeout interval as well as retransmission. This latter mechanism provides an autonomous adaptation to a network condition that does not rely on the receipt of recent round-trip delay information. Automatic deflation of T occurs when the network problem subsides and round-trip times resume normal values. Similar comments to the above apply in the case where a change in network condition causes an increase in L (perhaps to unity!); the appropriate action is a lengthening of T .

Thus we can identify three classes of inputs to an adaptive timeout setting algorithm:

1. *Setup parameters*—Number and type of nodes in path; call priority.
2. *Observational corrections*—Observed round-trip delays; congestion indicators.
3. *Autonomous corrections*—Number of recent timeouts.

One issue which has not been discussed concerns the relationship between timeouts and fault detection strategies. If the occurrence of timeouts is used as an input to a fault detection/diagnosis mechanism, this will provide another reason for not choosing T too large.

SUMMARY AND CONCLUSION

In this paper we have tried to isolate some of the most important underlying issues which arise when choosing a timeout interval.

We have observed in the second section the presence of several conflicting goals and that the solution we have proposed should be viewed as a pleasing compromise. It was demonstrated in the next section that our solution yields a result similar to that of a quite different design technique and in fact, that for small loss probabilities the solution is insensitive to the parameter representing the compromise. In the fourth section we have considered how the basic design should be modified to take into account the possibility of abnormal network conditions and to ensure that timeouts themselves cannot act as a source of network problems.

It is likely that any real network and protocol will possess its own features and idiosyncrasies which require additional factors to be included in the timeout strategy. In the author's application,⁶ a number of additional objectives and protocol details have been taken into account. For example, an efficient protocol will acknowledge groups of packets rather than individual packets. As a general network strategy, when a timeout occurs it may be desirable for the transmitter to send an inquiry requesting status of the receiver rather than simply retransmitting the original packet. In this case the essence of the model remains valid since these inquiries need to be repeated periodically so that recovery from a short duration fault (e.g., trunk fade) is accomplished automatically. Another modification which can be accommodated in the above analysis arises from the consideration of block retransmit as opposed to the selective retransmit scheme we assume here.

ACKNOWLEDGMENT

The author is grateful to Jack Holtzman of Bell Laboratories for suggesting a study of this problem and providing many suggestions and comments.

REFERENCES

1. Belsnes, D., "Flow Control in Packet-Switching Networks," *Online Conference on Communication Networks, Europ. Comp. Conf. on Comm. Netw.*, London, September 1975, pp. 349-361.
2. Sunshine, C. A., "Efficiency of Interprocess Communication Protocols for Computer Networks," *IEEE Trans. on Comm.*, Vol. COM-25, No. 2, February 1977, pp. 287-293.
3. Fayolle, G., E. Gelenbe and G. Pujolle, "An Analytic Evaluation of the Performance of the 'Send and Wait' Protocol," *IEEE Trans. on Comm.*, Vol. COM-26, No. 3, March 1978, pp. 313-319.
4. Buttò, M., G. Colombo, G. Taggiasco and A. Tonietti, "Models for the Performance Evaluation of a Packet-Switching Network with Retransmission Time-out," *IEEE Nat. Tel. Conf.*, Los Angeles, 1977.
5. McQuillan, J. M., and V. G. Cerf, "A Practical View of Computer Communications Protocols," *IEEE Computer Society Tutorial Series Publication*, 1978.
6. Morris, R. J. T., "Considerations in Fixing Timeout Intervals," *Bell Laboratories Memorandum 3451-780815.01MF*, 1978.

Comparison of some end-to-end flow control policies in a packet-switching network

by G. PUJOLLE

IRIA-LABORIA
Le Chesnay, France

INTRODUCTION

Various techniques may be considered when it comes to setting up a communications system between computers. The "packet-switching" technique described by Davies¹ seems to be one of the best of existing approaches. In the following we consider only such a technique—users of a computer network communicate with each other by the intermediate of a store-and-forward packet-switching network.

The term host has been introduced in the Arpanet literature. It has been used widely, although not always in the very same sense as in Arpanet. We use it here in some loose sense—a host is a source and a sink of packets. A subscriber is an entity which provides to the host the data to be transmitted through the packet-switching network (PSN). The set of subscribers attached to host i asks on the average the transmission of λ_i packets per second.

It is well known in a system where resources are shared that when the load increases, it is necessary to have a congestion tool to avoid a degradation of performance. Such a phenomenon has been pointed out in PSN.^{2,3} Thus tools are necessary to prevent this degradation. They are flow control methods, namely procedures whereby the receiver allocates a potential transmission credit to the sender, no matter what the form may be to specify this credit.

In the second section, we describe several types of specifications in order to compare them in the fifth section. To do this, we shall introduce in the fourth section an unified mathematical model. This model will use a single source destination path taking into account intermediate interarrivals. Two different node transmission procedures, introduced in the third section, will be used in the model.

The main contribution of our paper is that we explicitly take into account most of the elements which characterize a packet-switching network—node-to-node and host-to-host protocol, retransmission policy, finite buffer size in nodes. Three flow controls are examined and their performance compared in detail under several working assumptions.

We show that the maximum throughput allowed by these three types of flow controls are very different and the higher the throughput, the more important it is to control ade-

quately the parameters of the system to avoid a thrashing phenomenon.

FLOW-CONTROL TECHNIQUES

Window flow control

One of the best known techniques is the isarithmic scheme.^{1,2} Under this control, there are a fixed number of credits circulating in the network. A packet is admitted into the network only if it can get hold of a free credit. The packet travels through the network accompanied by its credit. The credit is again free when the packet reaches its destination. Several policies can be followed to re-distribute the free credits. For example, they can be host-dedicated, namely, they return to the originating source when they are released. If they are host-to-host dedicated, the scheme corresponds to a window flow control and the number of credits used for an host-to-host communication corresponds to the value of the window width. Our first flow control is exactly this last one—we shall name it WFC (*Window Flow Control*). We shall assume first a fixed window width, with the possibility to render it variable in a later phase of this study.

Rate flow control

An attractive scheme can work by a limitation of packets entering into the network in the following way. As long as a destination is able to cope with the outgoing packets, there is no need to choke the sending host-sources. However, if there is an excess of traffic, queues will start building up, and will eventually block the nodes. It is convenient to let each host receive from each node the information of the maximum amount of packets it can accept. According to this knowledge, hosts can then limit their transmissions to a "good" number of packets per unit of time. This is an idealized flow control because propagation delay is assumed infinite. However, it is possible to anticipate correctly the

variations of flows in a network by a system of control packets.

In a first step we shall assume a fixed threshold on the number of packets that can enter the network each unit of time. Then this number will vary with the state of the network. We name this technique *the rate flow control (RFC)*.

Flow control induced by the X25 recommendation

X25 flow control is negotiated between two subscribers by a virtual channel. At the host level X25 flow control is viewed as a superposition of virtual connections. The flow control for a virtual connection is based upon authorizations from the receiver. The main means for this control is the Receive Ready indication which essentially consists of the sequence number of the last well received packet. For each virtual connection a window size specifies how many packets may be transmitted from the sender host to the receiver host, related to the last correct Receive Ready indication. The difference with the window flow control is the existence of virtual circuits. Virtual circuits need establishment and release. The first packet opens the virtual circuit from the source to the destination and reserves special buffers at each node so that no possible overflow is possible. We shall name this flow control XFC (*X25 flow control*).

SPECIFICATIONS OF THE PACKET-SWITCHING NETWORK

Before we develop the mathematical model, it is necessary to define some specifications of the packet-switching network itself. We must define a strategy for dealing with packets rejected because of overflow due to finite buffer size at

the switching nodes. We shall take into account two types of technique:

- The most common technique, here called switch-retransmission (used for example in ARPA) in which if a packet cannot be accepted by a switch, it is retransmitted from a back up copy held in the preceding switch.
- Another technique which we call host-retransmission (used in the Cyclades network) in which the network drops a packet which arrives at a full switch, to be re-sent later by the source host.

In the modeling of store-and-forward communication lines we have to know the behavior of nodes. This behavior is very dependent on the node-to-node transmission procedure. Two types of procedure can be allowed:

1. Without look-ahead (window width=1)
2. With look-ahead (window width>1)

“Send and wait” procedure

The “send and wait” (SW) procedure⁴ belong to the first category. Before transmitting a new packet, the previous one must be acknowledged. We are going to analyze their behavior. On the time axis of Figure 1 we have represented the state of the sender and of the receiver during the transmission of a packet.

- ω_{00}, ω_{10} are due to the task switching and software work of the packet.
- ω_{01}, ω_{11} are the software delay of the writing on both systems.

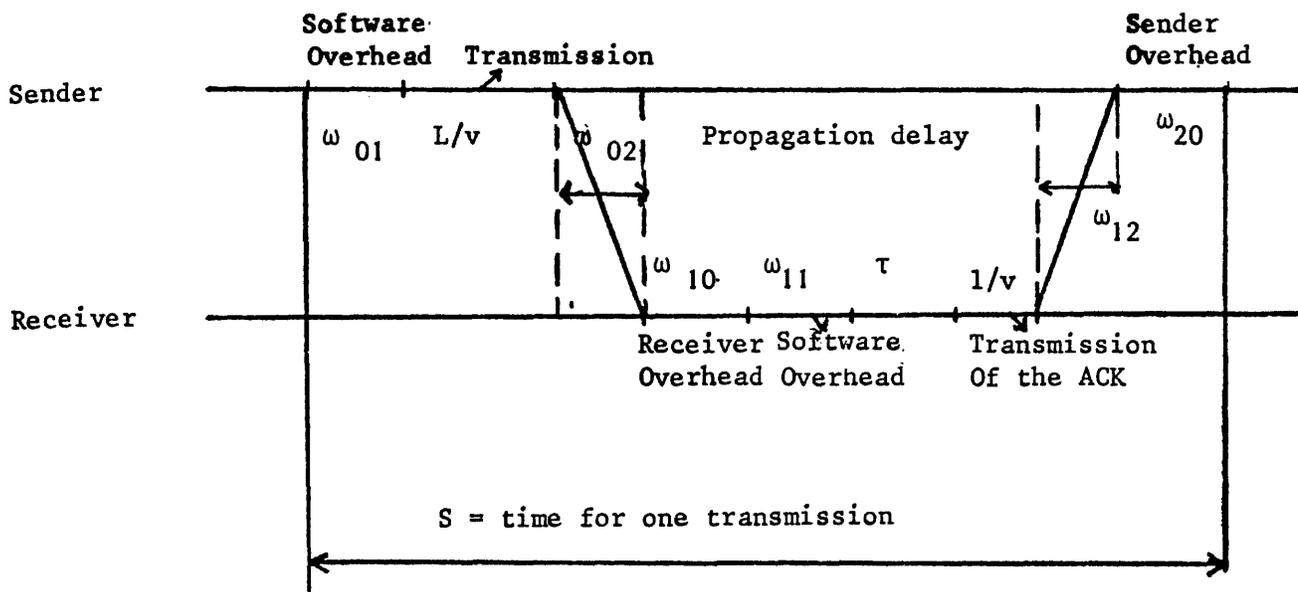


Figure 1—The “send and wait” behavior.

ω_{02}, ω_{12} are due to the propagation delay, and depend on the modems used and the line length (overhead due to modems are not negligible).

L is the mean total length of packets to be transmitted.

l is the length of the control packet which returns the acknowledgment.

v is the line capacity.

τ represents the time that the previous packet if any takes to finish its transmission.

We denote by S the total time necessary for the transmission of a packet. This mean service time is given by:

$$S = \omega_{00} + \omega_{01} + \omega_{02} + \frac{L}{v} + \omega_{10} + \omega_{11} + \omega_{12} - \frac{l}{v} + \tau$$

For the various overheads we will use the following values measured on the Cyclades network,⁵

$$\omega_{00} = \omega_{10} \approx 5\text{ms}, \quad \omega_{01} = \omega_{11} \approx 3\text{ms}$$

For a 500 km line length $\omega_{02} = \omega_{12} \approx 3\text{ms}$. We have to note the variation of τ according to the load on the lines. If the load is weak, $\tau = 0$ almost surely. In heavy traffic on the average τ is the transmission time of half a packet length.

In the sequel, we assume a symmetric traffic and we denote by ρ the load on a line. If $\rho = 0$ then the service S is minimum and if $\rho = 1$ the service time is maximum. We shall adopt a linear variation of the mean service time S between its maximum and its minimum.

Let $Ca = 1/v$ and $Cb = \omega_{00} + \omega_{10} + \omega_{01} + \omega_{11} + \omega_{02} + \omega_{12} + l/v$. We obtain the following simple expression for S :

$$S = CaL + Cb + Ca \frac{L}{2} \rho.$$

If the line speed is 48 Kb/s, we have for a 500 km line length:

$$S = 20.8 L + 26 + 10.4 L \rho$$

The quantity S we have defined is the time necessary to transmit successfully one packet. Now if an error occurs during the transmission or if the packet is rejected by an overflow in the receiver node, a backup copy has to be transmitted after a time-out. We have shown in a previous

paper⁶ that the performance of the node-to-node procedure is not sensitive to the probability of packets in error for usual values of this probability. Thus, we assume this probability negligible. Let p be the probability of overflow of the receiver node.

If we use the switch-retransmission (sr), a backup copy is retransmitted after a time-out T with the probability p . So the mean real time for one transmission is (without the retransmission if the packet is lost):

$$S_{sr}^{SW}(\rho) = (CaL + Cb + \frac{CaL}{2} \rho)(1-p) + Tp$$

We assume $T = 200$ ms for a 48 Kb/s line.

For the host-retransmission (hr) the overflow is detected after the acknowledgment is sent (the overflow is detected by the switch). So the mean time for one transmission is

$$S_{hr}^{SW}(\rho) = CaL + Cb + \frac{CaL}{2} \rho.$$

HDLC procedure

The HDLC procedure (High-level Data Link Control) which has been accepted as an international standard, belongs to "with look-ahead" node-to-node procedure. Its behavior is shown in Figure 2.

Due to the parallelism of the processes, the effective time required for a transmission is difficult to calculate; it depends on the window width. However, it is shown in Reference 6 that the throughput is only limited by transmission times if the window width is chosen adequately (the window width has to be sufficiently large so that no blocking occurs).

Thus we shall adopt for the mean effective transmission time the following values:

$$S_{sr}^{HDLC}(\rho) = CaL(1-p) + Tp, \text{ or } S_{hr}^{HDLC}(\rho) = CaL.$$

THE UNIFIED MATHEMATICAL MODEL

In this section, we consider a particular route inside the store-and-forward PSN. Such a route is modeled as a tan-

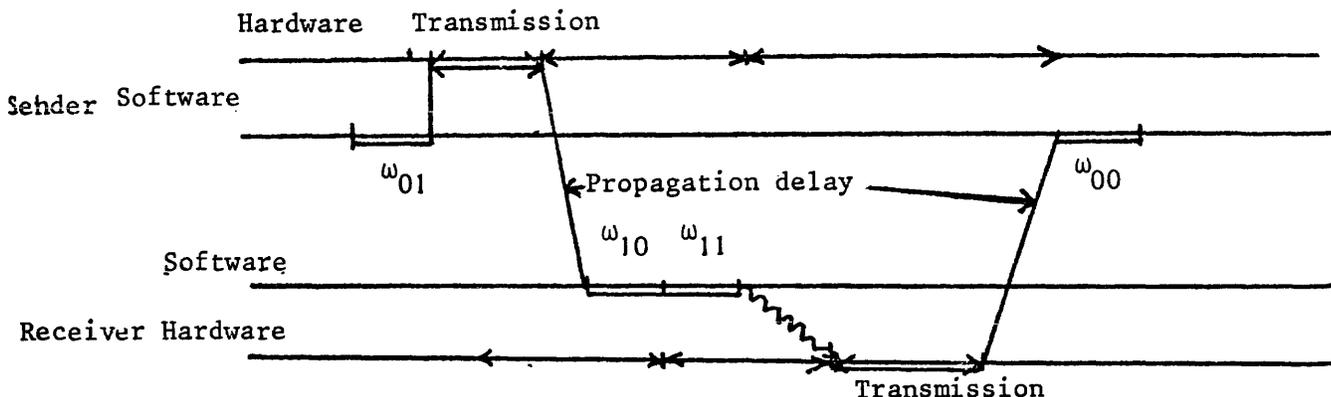


Figure 2—Behavior of the HDLC protocol.

dem queueing network. The system consists of $K+1$ queues. A customer (corresponding to a packet) goes through the system joining the $(i+1)$ -th queue after the i -th device for $0 \leq i \leq K-1$.

The first station 0, corresponding to the host, is assumed to have an infinite number of buffers. We assume that one packet and only one can be contained in one buffer. All the other stations have a finite number M_i of buffers (there is room for only M_i packets at station i). This finite size corresponds to the storage facility of the output queues of nodes of a route.

This mathematical model is shown in Figure 3. In Figure 3, customers of station C represent the credits. If its queue is empty the host must wait for a credit before transmitting a packet. The number of credits in the packet-switching network is denoted by N . When a packet leaves the network a credit comes towards the host. It passes through the station R which represents if necessary the return time of the credit (and the positive ACK). When a customer flows from station 0 to station 1 a credit disappears.

Our three flow control policies can be characterized as follows.

In the window flow control, the total number of credits circulating in the PSN represents the window width. We have to note that if this number N of credits is less than or equal to $\text{Min}(M_1, M_2, \dots, M_K)$, there is always a buffer available for a packet entering the network. This corresponds to the X25 flow control. If the total number of credits is greater than the sum $M_1 + M_2 + \dots + M_K$ and assuming station R does not exist, we have a network without flow control.

Finally, the rate flow control policy will be studied at the same time as the case without flow control because it corresponds to a threshold on the utilization of the server of the host.

In the applications, the mean service time will be chosen to be one of $S_{sr}^{SW}, S_{sr}^{HDLC}, S_{hr}^{SW}, S_{hr}^{HDLC}$ according to the node-to-node protocol and retransmission strategy chosen.

We shall denote by S this mean service time when the choice between several policies is possible.

Solution of the unified mathematical model

Several criteria can be used to evaluate the performance of the models. As we deal with flow control schemes we need an indicator of performance according to which the system is judged. This congestion measure can reasonably be chosen to be the throughput of the system versus the utilization of the server of the host. We have chosen this last parameter because it allows us to compare the different flow control schemes in a unified manner, and it is one the parameters used to control the traffic entering the packet-switching network.

The solution of the unified mathematical model will be carried out in two steps:

1. The model without station C and R.
2. The model with station C and R.

In order to solve this complex model some simplifying assumptions must be made; we describe them now.

In a real PSN each packet maintains its length as it travels from node to node, and service times are not independent. Here we will make the independence assumption of Kleinrock⁷ and a new independent packet length will be chosen at each station.

We assume that the distribution of service times of all the stations are identical and the average value is $S(\rho)$ where ρ is the utilization of the server of the host. This assumes that all stations have the same utilization rates. This is accurately verified in balanced networks.

Finally, our last assumption is to assume that a customer leaving a queue sees the system in an equilibrium state, namely the probability for a packet to be rejected is taken equal to the probability that the following queue is full. It has been shown⁸ that this assumption is quite accurate.

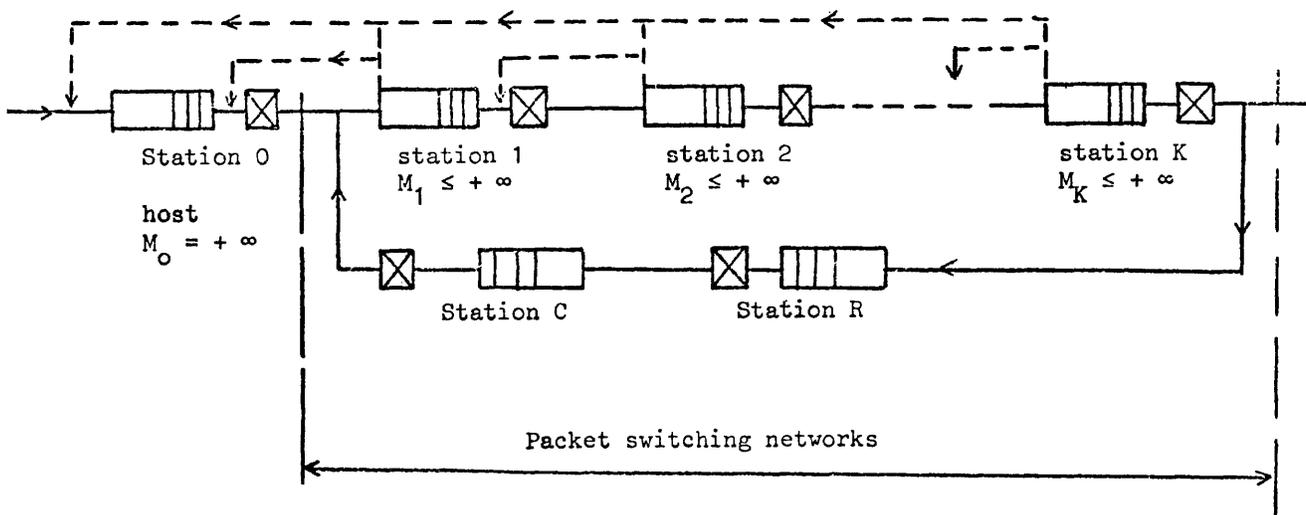


Figure 3—The unified mathematical model.

The explicit computations are introduced in the Appendix. We just give an idea here—from a given utilization of the host ρ , we compute the probability p that a customer is rejected by the PSN and comes back into the host. Then the mean transmission time is obtained from $S(\rho)$ whose value depends on the node-to-node protocol and the retransmission policy. Therefore we obtain the total arrival rate λ^* as $\lambda^* = \rho/S(\rho)$. This rate is the sum of external arrivals and recycling packets, therefore the throughput of the system will be: $\lambda = \lambda^*(1-p)$.

Though computation can be carried out with a different buffer size at each node, we shall assume $M_i = M, i=1, \dots, K$.

RESULTS AND COMPARISONS

For the SW procedure, we have compared the two retransmission policies. In Figure 4, some curves have been drawn representing both the situations of node-retransmission and host-retransmission, for a 48Kbits/sec. line and five or eight buffers at each output line, for six stations in series (one host and five nodes).

It is important to notice (this is true for all the following results) that the maximum throughput is reached for an utilization of the server of the host equal to 1. Namely, a

throughput greater than this value cannot be reached without flow control. This implies that the points corresponding to higher throughput than for $\rho=1$ are unstable points.

Thus, we see in Figure 4, that without flow control, host-retransmission leads to a better throughput than switch-retransmission. This can be easily explained—when we approach saturation ($\rho=1$) the switch-retransmission policy increases the congestion whereas the host-retransmission policy prevents congestion. This is even more explicit for the HDLC node-to-node procedure (see Figure 5).

We can notice by examining Figures 4 and 5 that if a flow control policy exists and allows us to obtain a throughput near the *optimal* point (the highest point of the curve) switch-retransmission is better than host-retransmission.

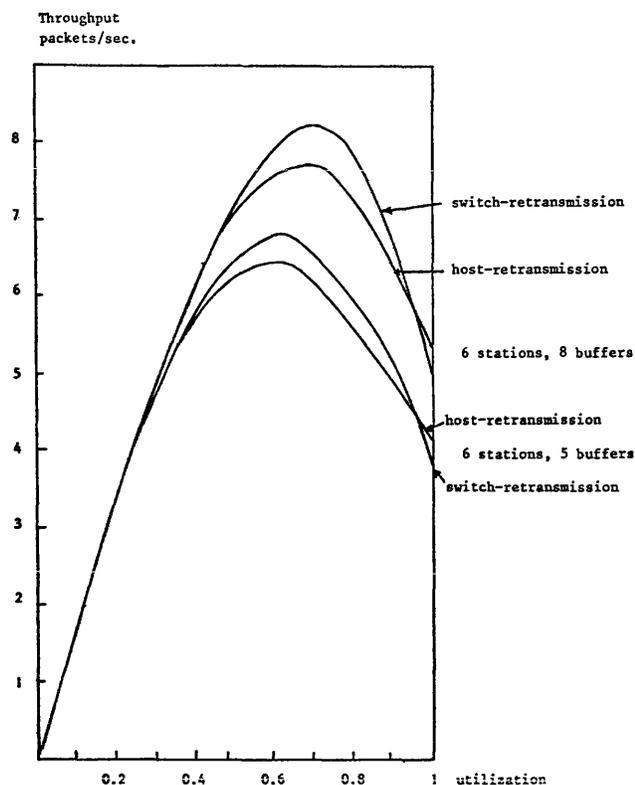


Figure 4—Throughput versus the utilization rate of the host. line capacity=48 Kbits/sec. mean packet length=1 Kbit packet length distribution=exponential procedure=SW

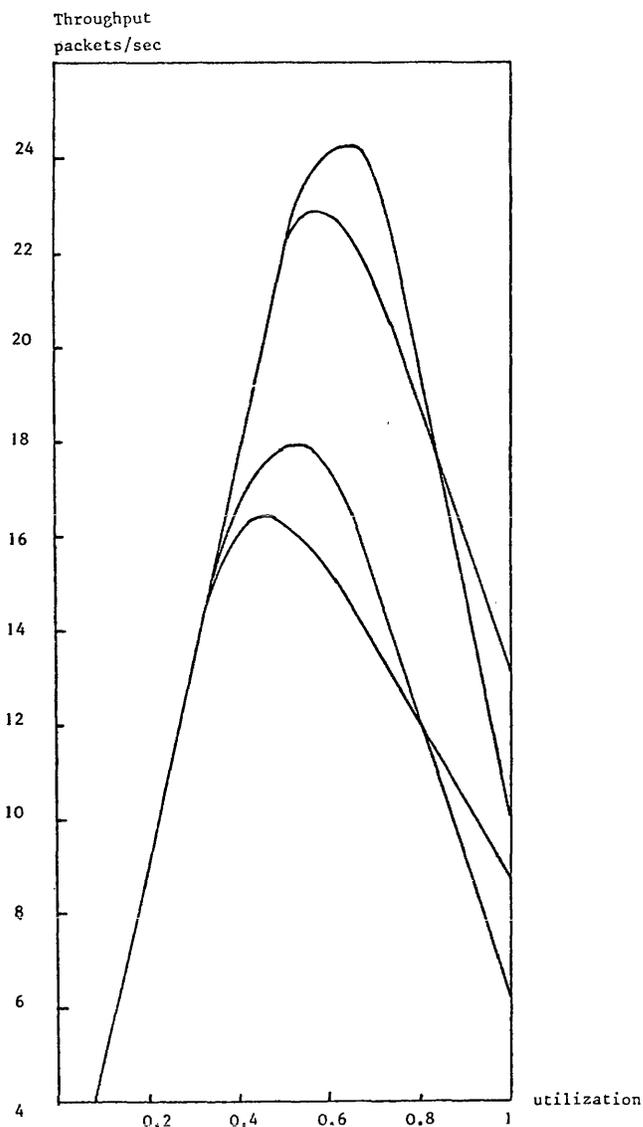


Figure 5—Throughput versus utilization rate of the host. capacity=48 Kbits/sec. mean packet length=1 Kbit packet length distribution=exponential procedure=HDLC

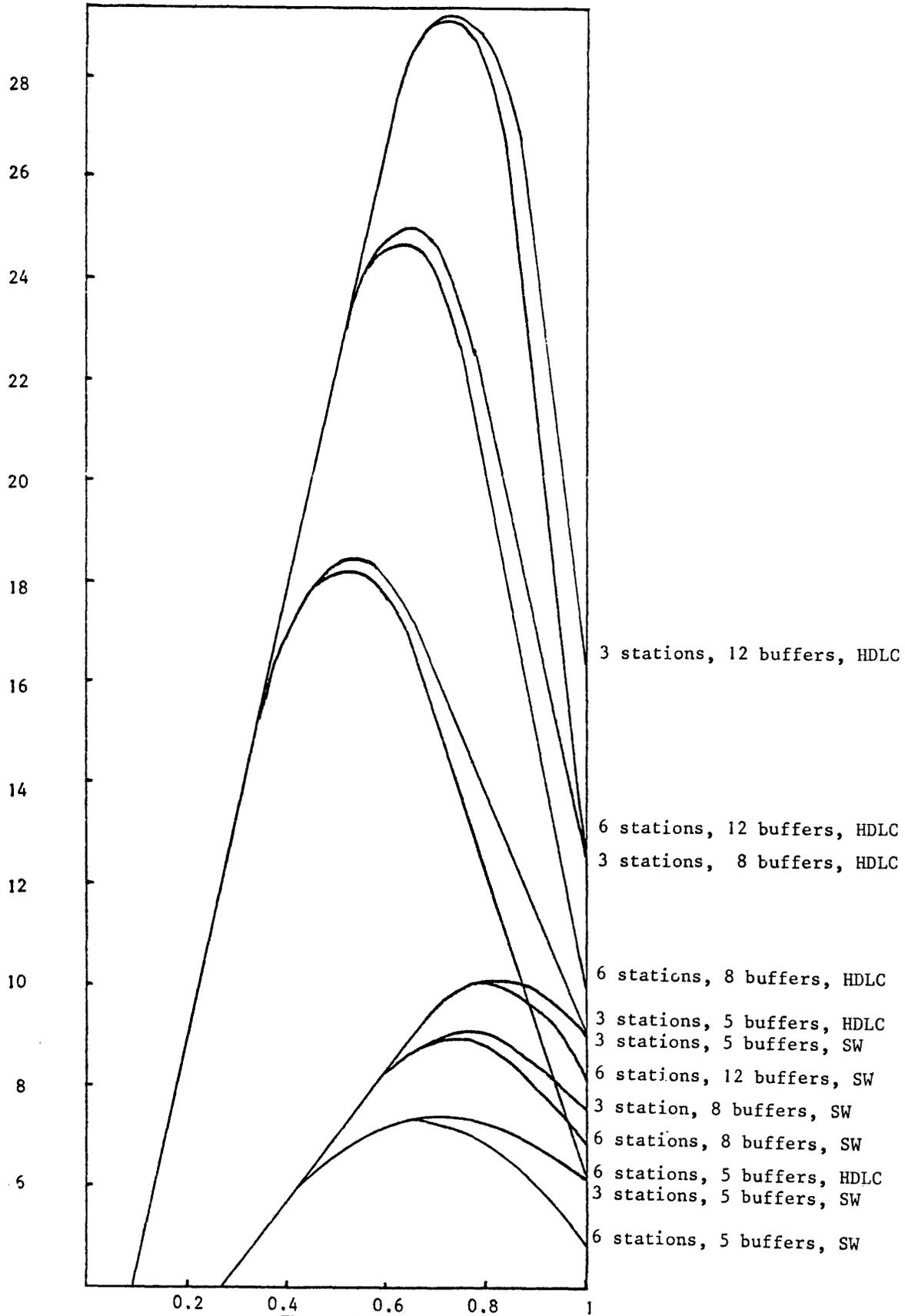


Figure 6—Throughput versus the utilization rate of the host.
 line capacity=48 Kbit/sec.
 mean packet length=1 Kbit
 packet length distribution=exponential

throughput
packets/sec

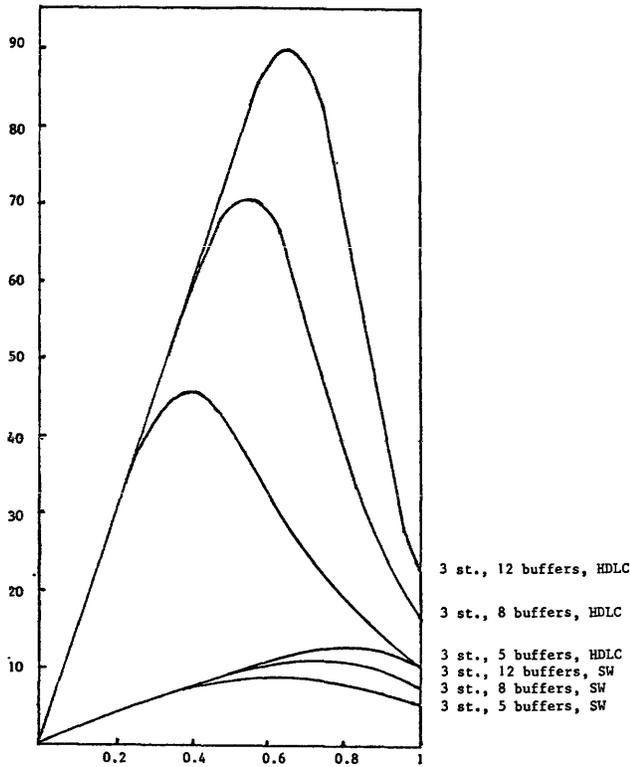


Figure 7—Throughput versus the utilization rate of the host.
line capacity=48 Kbits/sec.
mean packet length=300 bits
packet length distribution=exponential

This can also be easily explained—the optimal point is surely obtained when there are only few retransmissions but when the lines are utilized at the maximum. In this case to come back to the host is worse than to reset from the previous switch.

Since the purpose of this paper is to study and compare flow control methods, we limit ourselves to the switch-retransmission case, which is the best retransmission policy in this case.

Now we allow the packet length and the distribution of the service times to vary in Figures 6, 7 and 8. We have adopted a 48 Kbits/sec line and three or six stations in series, five, eight or 12 buffers per output line, and SW or HDLC protocols.

The degradation predicted is very clear and more important with HDLC than with the SW procedure. As long as there is no degradation (namely the probability of retransmission is negligible) three or six stations in series give the same throughput. Threshing is evidently stronger for six stations than for three stations in series.

The degradation is so much important when mean packet length is short. The degradation is less obvious with constant packet lengths. These three figures give an idea of throughput that can be obtained between two hosts of a PSN.

Rate flow control policy

The rate flow-control policy using a threshold on the number of transmissions per unit time can be studied with the previous results. This threshold corresponds to a value ρ^* of the activity of the server of the host. In Figures 8, 9 and 10 the maximum throughput is now obtained for the value of the throughput corresponding to ρ^* and the parts of the curves on the right of ρ^* cannot be reached.

It is obvious that the quality of this flow control policy depends on the choice of ρ^* . For the case studied in Figures 6, 7 and 8 it is sufficient to take for ρ^* the activity of the host corresponding to the optimal value of the throughput. However, in a general network, according to the destination of packets, the line capacities and the sizes of the pools of buffers, the optimal throughput does not correspond to the same limitation. Therefore, an efficient estimation and control system is necessary to adjust the threshold according to the state of the nodes of the PSN. This control leads to high throughput, but the risk of a strong degradation of performance exists as soon as the system is badly controlled. In Cyclades such a flow control is used and the rate defining the threshold varies according to the load of the lines of the system.

In Figures 9 and 10 both the two other flow controls are examined using the solution of the second part of the Ap-

Throughput (packets/sec)

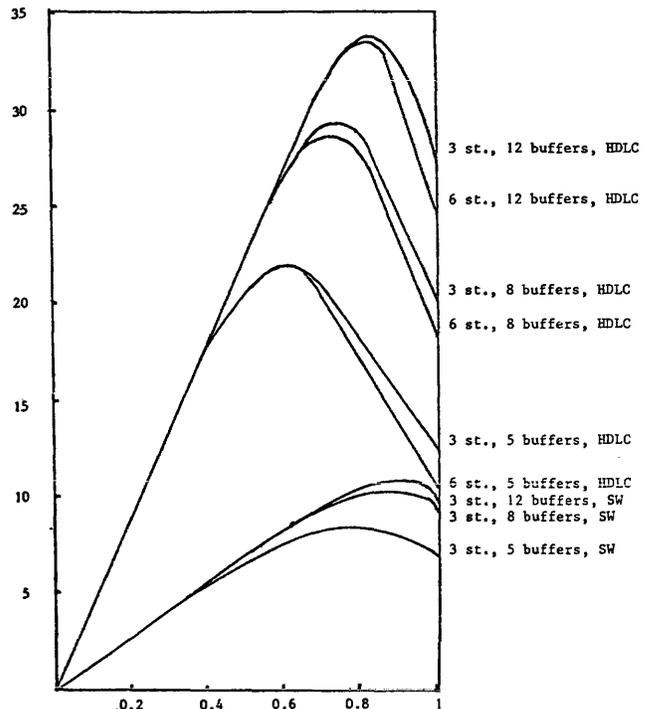


Figure 8—Throughput versus the utilization rate of the host.
line capacity=48 Kbits/sec.
mean packet length=1 Kbit
packet length distribution=Erlang 2

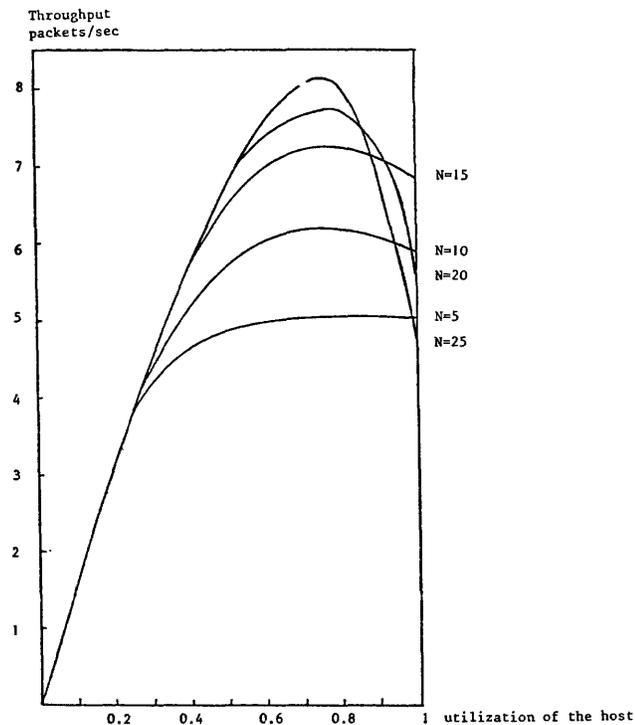


Figure 9—Throughput versus the utilization of the host for a number of credits N —six stations in series and five buffers per station. line capacity=48 Kbits/sec. mean packet length=1 Kbit procedure=SW packet length distribution=exponential

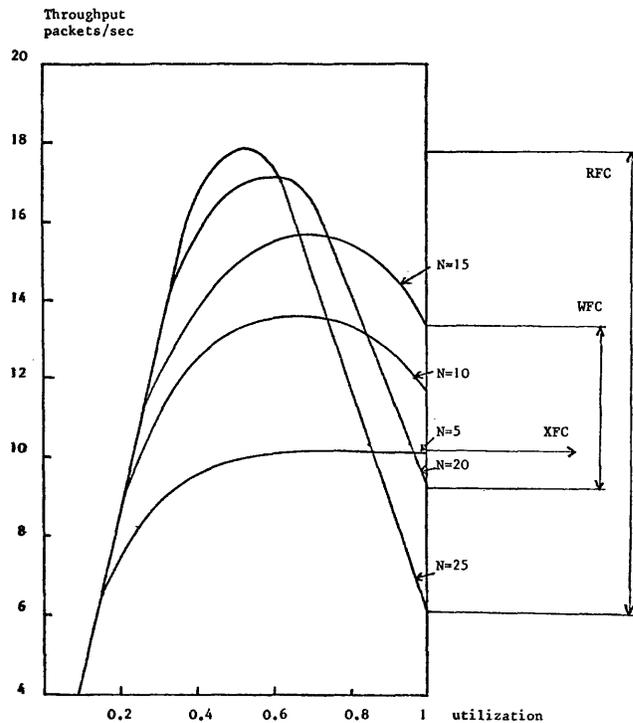


Figure 10—Throughput versus utilization of the host, for a number of credits N —six stations in series and five buffers per station. line capacity=48 Kbits/sec. mean packet length=1 Kbit procedure=HDLC packet length distribution=exponential

pendix to solve the unified mathematical model. We look at the case of 48 Kbits/sec line, six stations in series and five buffers per output line of nodes. Figure 9 corresponds to the SW node-to-node protocol and Figure 10 to HDLC. N is the number of credits. If $N=5$, we obtain the XFC. For example we have a superposition of five virtual circuits between the two hosts with a window width equal to 1. If $N=25$, we find again the case without flow control or with RFC if a ρ^* is given. Now if $5 < N < 25$ we obtain a WFC. We assume that the queue R of the mathematical model does not exist.

The advantage of the XFC is that no thrashing phenomenon exists. Above a certain value of the activity of the host, the throughput is practically constant. However, this constant value is far from the maximum value. This assurance is counterbalanced by a low throughput.

The best value for the window width (if it is to be fixed) seems to be $\bar{N} = (\sum_{i=1}^K M_i + \inf_i(M_i))/2$. In Figures 9 and 10 this value corresponds to $N=15$. In this case the maximum throughput (obtained for $\rho=1$) is intermediate between maximum throughput of RFC and XFC. Performance does not seem to be very sensitive to this parameter setting. Besides, means to regulate the value of the window width are easier than those used to throttle the rate for the RFC policy.

The view of Figures 9 and 10 gives an idea of an efficient dynamical WFC—the use of the upper envelope of the curves. For example on Figure 9, as soon as $\rho < 0.8$, $N=25$;

if $0.8 < \rho < 0.9$, $N=20$; if $\rho > 0.9$, $N=15$. When a threshold is exceeded no admittance is allowed until the number of packets in the PSN is above the associated window width.

As a conclusion of this comparison of flow control policies, we have written on the side of Figure 10 the throughput that can be reached by each of the three techniques described previously. We see that the zone corresponding to the RFC goes from the highest to the lowest point. WFC is intermediate. A very precise throughput is associated with XFC.

It has to be noted that the more we want to get high performance, the more the control of parameters must be sophisticated, otherwise the throughput will be very low.

CONCLUSIONS

First, we have shown that a flow control is necessary in PSNs. Without flow control schemes a thrashing phenomenon occurs when the traffic rate reaches a value close to 1. Several flow-control policies have been modeled and comparisons of the results can be interpreted as follows. The XFC technique allows one to obtain a certain throughput which does not decrease with increasing input traffic even when the system is saturated. We are sure that whatever the traffic conditions, a certain amount of service will always

be rendered. However, the maximum throughput is very low in relation with the throughput obtained by other flow-control policies.

The RFC (rate flow control) policy gives efficient results when it is easy to find accurate rate limitation corresponding to the optimal throughput. If the network is well balanced or very simple, such as a tandem queueing system, the value of the limitation can be obtained. But in a complex network, this rate limitation will have to change with the state of the network. A system of control packets must be created for the host to know the state of the system. The RFC policy will allow us to obtain large throughput; but a necessary condition is the need to have a sophisticated control system. Thrashing can appear here with bad management.

The WFC policy can be considered as an intermediate method. The maximum throughput is not as high as in the previous scheme, but there is no thrashing in saturation conditions. Moreover, the simplicity of this scheme can be a good reason for its implementation.

Finally, the question that can be raised is the accuracy of the previous results. Some validations by simulations of the unified mathematical model results have been done and are available in Reference 9. It is shown that even when the model parameters are somewhat different, the form of the curves is identical and the conclusions of the comparisons are similar.

A more widely available validation can be found in Reference 10 where the predictions of a mathematical model (with the parameters used here) are compared to the results of a measurement campaign on the Cyclades network. The model being a queueing system in tandem, the very good accuracy of the mathematical model prediction is established.

REFERENCES

1. Davies, D. W. "The control of congestion in packet-switched networks," *IEEE Trans. on Communications*, Vol. 20, No. 3, pp. 546-550, 1972.
2. Price, , "Simulation studies of an isarithmically controlled store and forward data communications network," *Proc. IFIP 74*, Stockholm, pp. 151-154, 1974.
3. Pouzin, L., "Cigale, the packet switching machine of the Cyclades computer network," *Proc. of IFIP 74*, Stockholm, 1974, pp. 155-159.
4. Fayolle G., E. Gelenbe and G. Pujolle, "An analytic evaluation of the performance of the 'send and wait' protocol," *IEEE Trans. on Communications*, Vol. 23, No. 3, 1978, pp. 313-320.
5. Grangé, J. L., and P. Mussard, "Performance measurements of line control protocols in the Cigale Network," *Computer Network*, August 1978.
6. Gelenbe, E., J. Labetoulle and G. Pujolle, "Performance evaluation of the protocol HDLC," *Computer Network*, August 1978.
7. Kleinrock, L., *Communication Nets—Stochastic Message Flow and Delay*, McGraw-Hill, New York, 1964.
8. Pujolle, G., "The influence of protocols on the stability conditions in packets-switching networks," *IEEE Trans. on Communications*, March 1979.
9. Pujolle, G., "Modélisation et évaluation de performances des réseaux à commutation de paquets," Thèse d'Etat, Université d'Orsay, 1978.
10. Eyries, F., and G. Pujolle, "Validation and prediction of performance in the Cigale network," *Proc. of ICPCI 78*, Gardone Riviera, Italy, 1978.
11. Gelenbe, E. and G. Pujolle, "Approximation to a single queue in a network," *Acta Informatica*, Vol. 7, 1976, pp. 123-136.

APPENDIX

Computation of the throughput as a function of the activity of the host

The model without station C and R

We recall the assumptions we have taken:

- The independence assumption
- The distributions of service times of all the stations are identical
- A customer leaving a queue sees the system at steady state.

Let μ^{-1} and K_s be the mean and the squared coefficients of variation (SCV) of the service time distribution of each server. We show in Figure A1 the model that we want to study (we have only depicted stations n and $n+1$). We denote by λ_n , K_{a_n} the rate and the SCV of interarrival distribution respectively to station n , (before rejection which occurs with probability p_n).

Analysis of the switch retransmission model

We will replace each station (for example station n) with its retransmission loop, by a simple queue with service time which represents the total time of the first retransmission and all retransmissions of the same packet. Let $\hat{\mu}_n$ and \hat{K}_{s_n} determine the distribution of the equivalent service time. Using a convolution product we obtain:

$$\begin{aligned}\hat{\mu}_n &= \mu_n(1-p_{n+}) \\ \hat{K}_{s_n} &= p_{n+1} + K_{s_n}(1-p_{y+})\end{aligned}$$

To compute the two first moments of the interarrival flow we must include all transmissions and retransmissions from station $n-1$; hence $\lambda_n = \lambda/(1-p_n)$ and $K_{a_n} = -1 + \rho_{n-1}^2 (K_{s_{n-1}} + 1) + (2\rho_{n-1} + 1 + K_{a_{n-1}})(1 - \rho_{n-1})$ where $\rho_n = \lambda_n/\hat{\mu}_n$. This last expression is a particular case of the SCV of the interarrival flow in a station computed in Reference 11 from a general network.

To summarize, each station n is treated as a $G/G/1/M_n$ system with a service time distribution determined by $\hat{\mu}_n$ and \hat{K}_{s_n} and an arrival time distribution by λ_n and K_{a_n} .

The probability that a packet is refused at this station is computed through a diffusion approximation:⁸ it is the probability at steady state that the queue is full.

$$p_n = \frac{\rho_n(1-\rho_n)}{e^{-\gamma_n(M_{n-1})-\rho_n^2}}$$

where

$$\check{\rho}_n = \lambda_n/\hat{\mu}_n$$

$$\gamma_n = 2b_n/a_n$$

$$a_n = \lambda_n K_{a_n} + \hat{\mu}_n \hat{K}_{s_n}$$

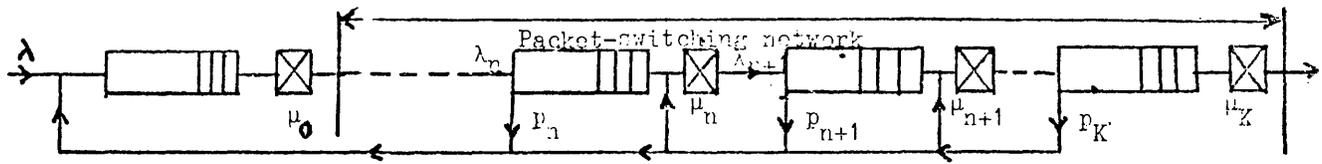


Figure A1—The tandem queueing model.

and

$$b_n = \lambda_n - \hat{\mu}_n.$$

Note that p_n depends on p_{n+1} by the intermediate of $\hat{\mu}_n$ and Ks_n . But if we begin to solve the equations for the last station first, we have $p_{K+1} = 0$ and so we can compute, one after another, the values of p_n for $n=K$ to $n=1$.

As we have assumed that the distributions of service times of all the stations are identical, we can suppose the mean service time equal to a time unit. We have to note also that p_i is an increasing function of λ the normalized arrival rate (mean service time = 1) such that the value of the activity of the host determines a unique value of the external arrival rate λ . Therefore, for a given utilization ρ of the host, by an iterative method we can compute λ and p_i (for a given λ we get step by step p_n , $n=K$ to $n=1$, thus p_i ; the exact value of λ is obtained when the equality $\rho = \lambda / (1 - p_1)$ holds). Now the value of $S(\rho)$ is derived as:

$$S_{sr}^{SW}(\rho) = (CaL + Cb + \frac{CaL}{2} \rho)(1 - p_1) + Tp_1$$

$$S_{nr}^{HDLC}(\rho) = CaL(1 - p_1) + Tp_1$$

following the retransmission policy.

Therefore, the total arrival rate is $\lambda^* = \rho / S(\rho)$. As this rate is the sum of external arrivals and recycling packets, the throughput of the system is

$$\lambda = \lambda^*(1 - p).$$

Analysis of the host-retransmission scheme

The arrival rate at station $n+1$ is $\lambda_{n+1} = \lambda_n(1 - p_n)$ for $n=1$ to $K-1$; if we conveniently denote by λ_{K+1} the departure rate from the last station, K , then the preceding equation is also valid for $n=K$.

As we use the host-retransmission only with exponentially distributed service times, we develop the solution only in this particular case. An extension to general service times can be done through a diffusion approximation.

The probability of refusal is obtained through the classical $M/M/1/M_n$ queue:

$$p_n = \rho_n^{M_n} \frac{1 - \rho_n}{1 - \rho_n^{M_n+1}} \text{ where } \rho_n = \lambda_n / \mu_n$$

We also know that the departure rate from the last station is equal to the external arrival rate into the network, i.e. $\lambda_{K+1} = \lambda$. Thus, beginning with the last station, we can compute (λ_n, p_n) , for $n=K$ to $n=1$. The host is a $M/M/1$ system with service time $S(\rho)$ and arrival rate $\lambda^* = \lambda + \sum_{i=1}^K p_i \lambda_i$.

The service time $S(\rho)$ depends on the node-to-node protocol:

$$S_{nr}^{SW}(\rho) = CaL + Cb + \frac{CaL}{2}$$

$$S_{nr}^{HDLC}(\rho) = CaL$$

Therefore, for a given utilization rate of the host we obtain the total arrival rate $\lambda^* = \rho / S(\rho)$, and the throughput of the system is obtained by an algorithmic method determining λ when the following equality holds:

$$\lambda = \lambda^* - \sum_{i=1}^K p_i \lambda_i.$$

The solution is unique because the p_i 's are increasing with λ .

The model with stations R and C

The model has been shown in Figure 3. We are interested only by the switch-retransmission policy.

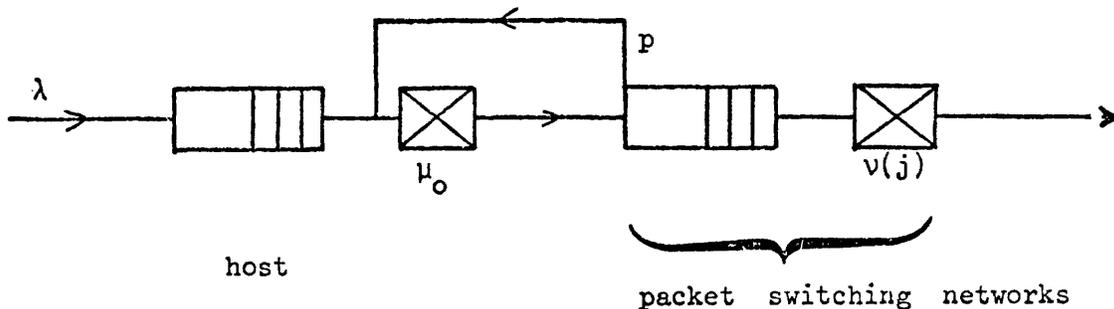


Figure A2—The equivalent unified model.

The solution we propose is to use an equivalent station. The closed network representing the model of the PSN itself can be replaced by a single queue with a state dependent rate $\nu(j)$, $j=1, \dots, N$ (N is the total number of credits, namely the total number of packets in the PSN plus the free credits). We have to study a closed queueing network with finite buffer size. As no analytical method is available to study such a queueing system, we adopt a simulation just to compute the utilization A_K^j of the server K , when there are j customers in the closed network. We have assumed in this simulation that the service time of the credit queue equals the host service time. The equivalent service rate is obtained as $\nu(j)=A_K^j$, assuming the mean service time of each station is a time unit.

The equivalent system is shown in Figure A2.

The rejection probability p for a customer is always taken as the probability the second queue is full, so:

$$p = \frac{\frac{\rho}{\nu(1)\nu(2)\dots\nu(N)}}{1 + \frac{\rho}{\nu(1)} + \dots + \frac{\rho^N}{\nu(1)\dots\nu(N)}}$$

where ρ is the utilization of the host.

Now the service time $S(\rho)$ can be computed and is equal to either S_{sr}^{sw} or S_{sr}^{HDCI} following the node-to-node protocol.

The total arrival rate is $\lambda^* = \rho/S(\rho)$, and the throughput of the system is

$$\lambda = \lambda^*(1-p).$$

Alternatives for providing highly reliable access to X.25 networks*

by RICHARD J. CHUNG

Teleglobe Canada
Montreal, Canada

and

A. M. RYBCZYNSKI

TransCanada Telephone System
Ottawa, Canada

INTRODUCTION

Public and private packet-switching networks (PSNs) are being developed or planned throughout the world—for example in Canada (DATAPAC), the U.S. (TELENET), France (TRANSPAC), the U.K. (EPSS), Japan (DDX), Spain, Netherlands and others.⁷ Such an explosive development of data communications services is one of the means for attaining the near-perfect availability required from future distributed information processing systems.⁹ The reasons for this requirement are the increasing size, complexity and geographic dispersion of information processing systems, and the increasing real-time dependence of users on these critical, distributed systems for the timely and effective operation of their businesses.

As a result, there is a sustained incentive towards improving the reliability and availability of PSNs¹¹ on the one hand and of information processing systems on the other. Both improvements use similar techniques, namely components of high reliability and redundancy of components. However, the reliability of the *access path* between the information system and the PSN is in some cases the weakest link in the overall reliability of distributed teleprocessing systems.¹⁰ Using the same reliability techniques for the access path (namely redundant, reliable physical circuit *configurations*) is not sufficient as the access path connects two dissimilar logical entities, and the *implementation* of the access protocol governing their inter-communication has to be considered.

This paper is concerned with the reliability of access from synchronous packet-mode Data Terminal Equipments (DTEs) through multiple physical circuits to X.25 packet-switching networks offering virtual circuit services. The techniques discussed in this paper are also generally appli-

cable to the reliable interconnection of X.25 networks via X.75. The reliability of access of asynchronous non-packet terminals using multiple dedicated physical circuits has not been identified as being cost-effective and can be adequately ensured by alternate dial-in facilities; it is not further discussed here.

Each subscriber DTE is connected to the PSN node by a local access circuit or local loop (LL) (see Figure 1). The local loop is typically configured through a number of telephone central offices and is constrained to existing inter-office facilities. User data and control information are exchanged across the interface between the DTE and the DCE (Data Circuit-Terminating Equipment, or the network side) according to the formats and procedures of the X.25 network access protocol, standardized by the International Telephone and Telegraph Consultative Committee (CCITT).¹⁴ The X.25 network interface has been specified as a hierarchical set of three separate functional protocol layers, namely a Level 1 (or the physical level protocol), a Level 2 (or the frame level or link access protocol, LAP), and a Level 3 (or the packet level protocol). It should be noted that X.25 is purely a local interface to the virtual circuit service provided by the network.¹³ A virtual circuit service is characterized by the establishment of a logical path and network association between two end-user DTEs for the purpose of exchanging data in the form of a network-maintained sequenced flow of packets. For the purpose of this discussion, no particular hardware or software implementation is assumed for the DTE.

NEED FOR MULTIPLE PHYSICAL CIRCUITS

Three common user reliability measures are Mean Time Between Failures (MTBF), Availability and Reliability.⁴ The Availability parameter (A) gives the probability that at any given time the access path is ready for use, but does not predict for how long it will stay operational. The Reliability

* This paper is based on work done by R. J. Chung while with the Computer Communications Group of the TransCanada Telephone System in partial fulfilment of the requirements for his M.Sc. (Computer Science) degree from the Université de Montréal.

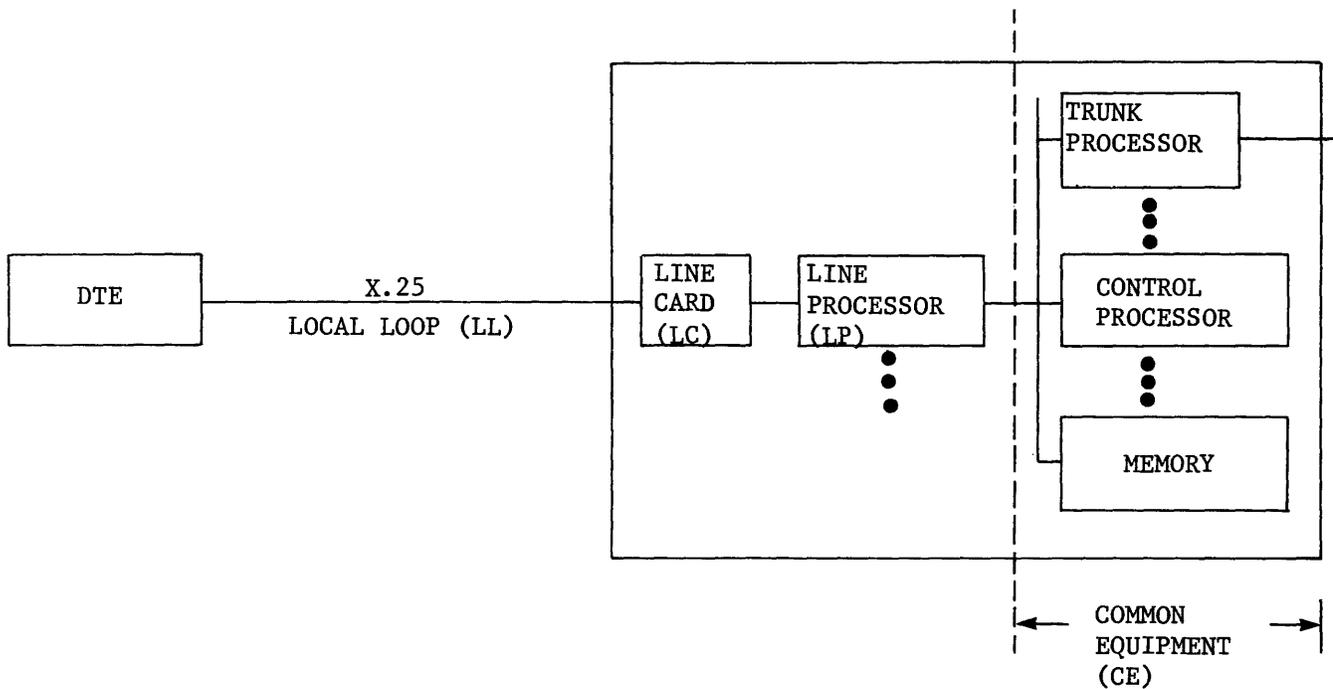


Figure 1—DTE X.25 single-circuit access to a typical packet-switching node.

parameter (R) gives the probability that the access path maintains its correct operation during a selected time period. We note that Availability applies when the access path is to be used (e.g. during X.25 call establishment phase) whereas Reliability applies while the access path is being used (e.g. during X.25 data transfer phase).

Since access circuit failures are not rare events and their probability of occurrence cannot be reduced to a negligible level, we define a third measure of reliability, namely Fault-tolerance, which may express future user concerns about the suitability of a PSN to meet their reliability requirements. The Fault-tolerance parameter gives the probability that the access path continues to provide virtual circuit services without clearing calls and without external assistance in the presence of circuit (also called line) or link failures. Access path Fault-tolerance consists of three components:

1. The probability of loss of virtual circuits during line failures. It is a function of the assignment of virtual circuits to lines, whether the allocation is fixed at subscription time, or dynamically assigned during call establishment, or dynamically switched (i.e. floating) during data transfer phase.
2. The probability of recovery of virtual circuits after line failures. This is a function of the availability of an alternative line and whether a call reconnect capability is implemented.
3. The probability of loss or duplication of data during the recovery of virtual circuits.

Thus, the Fault-tolerance of an access path can be char-

acterized by its capability to maintain or recover virtual circuits automatically, and by its robustness and adaptability against line failures. Whereas the Reliability measure ceases to apply at the first failure, Fault-tolerance recognizes that failures do occur and implies that built-in mechanisms exist in the system allowing it to continue to operate, though in degraded mode, by "tolerating" the fault. It also depends on the implementation of automatic recovery software which may be distributed in different hosts or nodes. The Fault-tolerance parameter is not discussed further here but will be reintroduced in the fifth section where the reliability of a number of implementations are compared.

For some applications, the failure of the telecommunications system during information transfer and the subsequent loss of connection is inconvenient, but acceptable if the user is allowed to reestablish his connection. Examples are time-shared computing, message and reservations systems. Hence, these users put a higher premium on Availability than on Reliability. They can live with short, infrequent disruptions of service provided alternate facilities are readily available. For other applications, the loss of connection or disruption of service during information transfer cannot be tolerated or may have catastrophic results, for example real-time systems like remote process control or pipeline control, or, less critically, bulk data transfer or remote job entry. For such users, Reliability and Fault-tolerance are significant parameters. Availability may not be as critical since the user can try to establish the connection at some other time or to some other place.

To meet user reliability requirements, the packet-switching node to which the X.25 DTE is connected is designed to perform all network functions (e.g. control, switching and

terminal support) while ensuring the highest degree of availability. It is typically a multi-processor (as in DATAPAC,³ TRANSPAC,⁵ TELENET¹²). The multi-processor organization of the DATAPAC SL-10 Network Processor is typical of current switch architectures designed for reliability, flexibility, reconfigurability, and expandability.³ This organization is illustrated in Figure 1 and is used as the basis for this paper. It consists of at most 15 functionally different and separate processing units, interconnected by a common bus and communicating through a common memory module. Each customer circuit terminates on an individual line interface card (LC). The line processor (LP) has sufficient memory and processing power to control a number of subscriber synchronous and/or asynchronous circuits. The assembly of common memory, control processors, trunk processors and other related hardware is referred to as Common Equipment (CE).

Communication networks and some computer systems are designed with built-in redundancy, bypass switches and fully duplexed components to guarantee an availability of at least 0.998. On the other hand, the access circuit connecting the

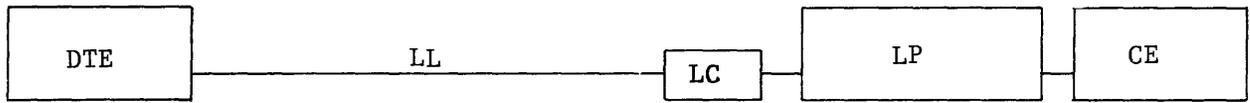
two has an average line availability of only 0.98 in the U.S.A.¹⁰ Typical analyses of leased line failures show that half the outages last for more than an hour. Consequently, multiple parallel physical circuits are used to make the reliability of the access path comparable to that of the network and computer system.

ACCESS PATH CONFIGURATIONS

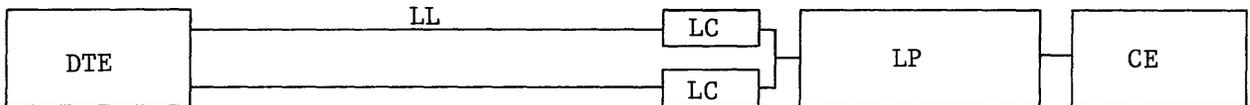
The current access to PSNs through the X.25 packet-mode interface is based on a single circuit using the High-Level Data Link Control (HDLC) procedures,² standardized by the International Organization for Standardization (ISO), as the link access procedure. For greater reliability, the *access path* of a packet-mode DTE may consist of a set of circuits. These can be configured as shown in Figure 2, subject to the protocol and implementation considerations discussed in the next section.

In order to develop a quantitative comparison of the reliability of these configurations, we assume a mathematical

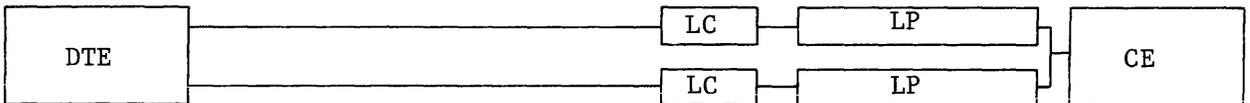
CONFIGURATION I: SINGLE CIRCUIT



CONFIGURATION II: TWO CIRCUIT



CONFIGURATION III: TWO-LINK, SINGLE NODE



CONFIGURATION IV: TWO-LINK, TWO NODE

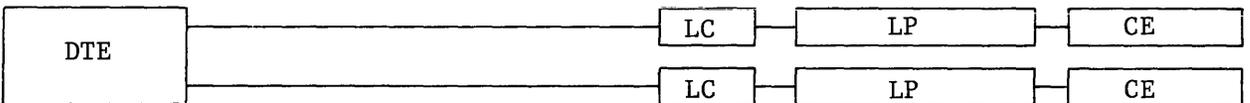


Figure 2—Access path configurations.

model in which physical components fail at random and the number of failures in a given time period depends only on its duration. That is, the probability of having failures in a given time period follows a Poisson distribution, and the probability that the times between failures are a given value follows an exponential distribution. A reliability analysis must consider, in addition to the local data loop (LL), the following components of a packet switch (see Figure 1): the line interface card (LC), the line processor (LP), and the remaining common equipment (CE). A failure of LP will lose all access circuits connected to it while a failure of CE will lose the circuits of all subscribers connected to that node.

As an example of the type of simple calculations which may be useful, we apply classical multiple component reliability theory⁶ to the model just outlined to derive the Availability, and the probability of failure-free operation during a given period, of each configuration in Figure 2. We assume all parallel components are identical and independent. The MTBF of LL has a wide variance as a result of the varying types of equipment used, e.g. analog or digital. We have therefore computed the Reliability of the different configurations over a month using a MTBF of 36, 24, 12, or 6 months respectively for the local data loop and typical MTBF figures for the other components. The results are given in Table 1.

The following conclusions can be drawn from analyzing the preceding model:

1. Two-circuits (II) increase the Reliability in a significant way (8-15 percent), especially when the local loop has the highest failure rate.
2. Two-links (III) have a 5 percent better Reliability than Two-circuits (II) independently of the local loop failure rate. This result follows from the fact that the LP has a failure rate comparable to that of LL and LC, and the multi-link only uses independent LPs. We note that connecting Two-links to independent nodes (IV) only increases the Reliability by 1 percent compared to connecting them on the same node (III).
3. The Availability of the local loop reduces the Availability of a single circuit significantly.

The quantitative approach just outlined is adequate for evaluating reliability in most cases; it cannot, however, be used for the design of data communications systems which require extreme reliability. This is because the assumption of identical and independent components, particularly for the local data loop, rarely holds for common carrier equipment.¹⁵ In fact, diversified routing of the access lines is very difficult and costly to obtain. The failures of different data communications lines are highly dependent because of the inherent sharing of the same multi-pair cable for local loops to the telephone central office and of short-haul or long-haul inter-office facilities. For techniques to handle these problems in computing communications system reliability, refer to Reference 15.

ACCESS PATH IMPLEMENTATIONS

This section outlines the different implementation alternatives possible in an X.25 network to support user access by multiple physical circuits. A DTE is said to be *multi-homed* if it is connected to multiple nodes by multiple autonomous links. A *link* is defined as a set of one or more physical circuits operating according to a single data link control procedure.

For reliability calculations, the only access path parameter of interest is the number of physical circuits available for data transfer. However, in implementing any X.25-based access path, other protocol factors must be considered, including the administration of network addresses for routing during call establishment, and of logical channels for relating packets locally to virtual circuits. Furthermore, the ordering property of the virtual circuit service must be maintained. This sequencing can be achieved by an appropriate switching function at either the physical level/frame level interface or the frame level/packet level interface of X.25. The problem can be avoided by using each physical circuit as an autonomous X.25 link. Thus, the following access path implementations (Figure 3) can be distinguished for the multiple circuit configurations shown in Figure 2:

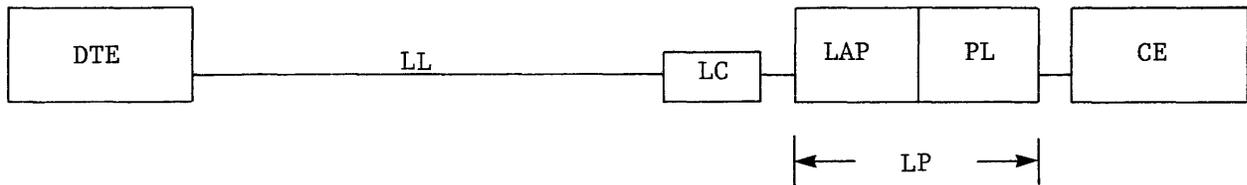
TABLE I—Reliability Results for Access Path Configurations*

LOCAL LOOP MTBF (months)	CONFIGURATION I			CONFIGURATION II			CONFIGURATION III			CONFIGURATION IV		
	MTBF (months)	R (month)	A (%)	MTBF (months)	R (month)	A (%)	MTBF (months)	R (month)	A (%)	MTBF (months)	R (month)	A (%)
36	4.74	.810	99.9575	9.01	.895	99.9932	19.49	.950	99.9989	27.11	.964	100
24	4.43	.798	99.9575	8.26	.886	99.9932	19.09	.949	99.9989	24.13	.959	100
12	3.77	.767	99.9575	8.11	.884	99.9932	13.77	.930	99.9989	17.75	.945	100
6	2.85	.704	99.9575	6.29	.853	99.9932	9.90	.904	99.9989	10.96	.913	100

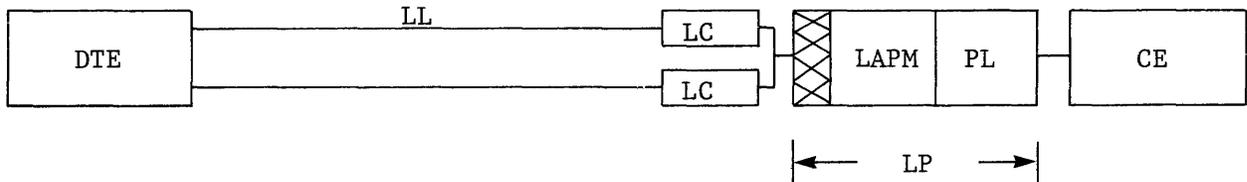
ASSUMPTIONS:		
COMPONENT	MTBF (months)	A (%)
LL	6-36	99.97
LC	12	99.9943
LP	12	99.9943
CE	60	99.9989

* See Figure 2.

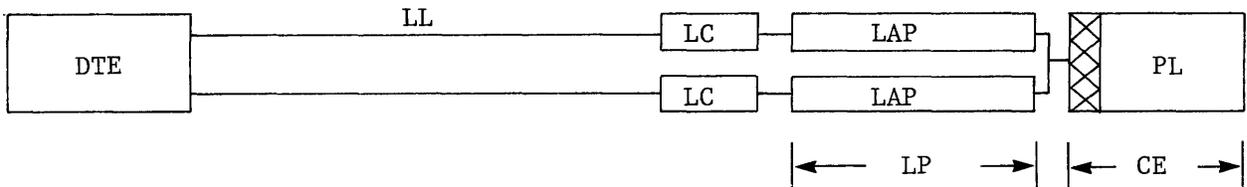
CONFIGURATION I: SINGLE CIRCUIT LINK



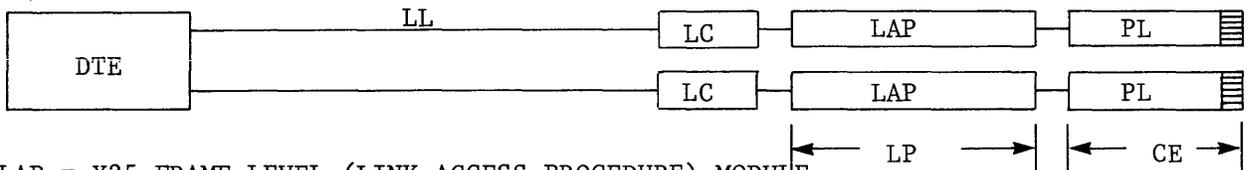
CONFIGURATION II: MULTI-CIRCUIT LINK



CONFIGURATION III: FLOATING MULTI-LINK



CONFIGURATION IV: AUTONOMOUS MULTI-LINK



LAP = X25 FRAME LEVEL (LINK ACCESS PROCEDURE) MODULE

LAMP= MULTI-CIRCUIT LAP MODULE

PL = PACKET-LEVEL MODULE

⊠ = SWITCHING FUNCTION

▮ = RECONNECT FUNCTION

Figure 3—Access path implementations.

1. A multi-circuit link (frame level) implementation of Configuration II.
2. A floating multi-link (packet level) implementation of Configuration III with automatic alternate routing.
3. An autonomous multi-link (packet level) implementation of Configuration IV with an optional reconnect procedure.

Each link of a multi-link implementation (i.e. III or IV in Figure 3) may be a uni-circuit or a multi-circuit link. A major concern of the multi-link packet level is failure recovery,

namely to maintain virtual circuits after a link failure. This function is automatic in the floating multi-link but requires a reconnect procedure in the autonomous multi-link.

Multi-circuit link implementation

A multi-circuit link implementation (Figure 3(II)) consists of parallel physical access circuits with the same network address and logical channel set statically assigned at subscription time to all of them. These circuits are attached to

uniquely different line interface cards on the same line processor in the same node. They are administered individually via a line level protocol and operate under a single multi-circuit link protocol.

A line level protocol is required for a multi-circuit link to establish the actual configuration of the link at any time in terms of line status and characteristics. It performs the functions of connecting/opening, disconnecting/closing and supervising a physical circuit individually according to commands from a higher level. Thus, it permits circuits to be removed or added with no disruption to link operation. We note that the line level protocol does not concern itself with data integrity or sequencing. It only provides a mechanism below the X.25 frame level for switching frames onto multiple physical circuits.

The purpose of a multi-circuit link protocol is:

1. To make a set of physical circuits appear as a single full duplex data link for better reliability and efficiency (throughput, delay). Thus, the physical characteristics of the link configuration (such as number of lines, line speeds, propagation delay, transmission media) are transparent to the higher-level protocols; frames are sent on the first circuit that is operational and not busy.
2. To ensure the integrity of the exchange of packets without loss, duplication or out-of-sequence delivery.
3. To detect circuit failures and control the link configuration by a line level protocol discussed above. Circuit failures do not impact the integrity or reliability of the link but only cause a graceful degradation of efficiency.

A multi-circuit link protocol has been used in the experimental French RCP network since 1974, i.e. before the standardization of HDLC by ISO, and is planned for TRAN-SPAC.^{5,8} A second possibility would be the use of HDLC over multiple circuits. Both these are currently called multiline procedures. We note that a third possibility, currently being studied, is the addition of a sequencing sublevel on top of a multiple circuit configuration where each circuit is operating under its own data link control procedure.

A multi-circuit link has the properties of variable delay, loss, duplication and out-of-sequence delivery of data. This is the result of the disordering phenomenon inherent in the simultaneous transmission of variable size frames on multiple physical circuits having variable line speeds, propagation delays, and multiplexing delays. Therefore, the main problem is how to maintain a sequenced data flow and its relationship to control, supervisory or out-of-band signals on each virtual circuit¹³ multiplexed over the physical circuits. A solution is to use a sequence-preserving link level.

Possible extensions and amendments necessary to the X.25 LAP or HDLC elements of procedure² to accommodate multiple physical circuits are outlined below:

1. I-frames (information frames carrying packets) received out-of-sequence should not automatically initiate REJ recovery. Instead, the receiver should buffer out-of-sequence frames and define a new secondary

timer TS (see Figure 4(A)), started on the reception of the first out-of-sequence I-frame. REJ recovery is initiated only at the expiry of TS.

2. The use of the REJ frame is inefficient (Figure 4(B)). A selective retransmission optional facility through a SREJ frame to request retransmission of a single I-frame is more efficient since the secondary has already buffered out-of-sequence frames.
3. To recover from lost I-frames, the primary retransmits its last unacknowledged I-frame after the expiry of primary T1 timer. The most conservative T1 value is twice the transmission time of the longest I-frame on the slowest line plus the transmission time of the longest I-frame on the fastest line. Such a long T1 value would ensure a high throughput but increases the delay in recovering from lost frames. However, this delay is minimized by the use of TS and SREJ at the secondary.
4. When the secondary receives a P-bit set to 1 from the primary (after T1 timeout), it should withhold sending an F-bit set to 1 until its own TS timer expires. Otherwise, after accepting the retransmitted I-frame with P-bit set, the original I-frame may arrive in the next numbering cycle and become an undetected duplicate.
5. The receipt of an invalid NR field should not initiate the resetting procedure because it may be in a late frame.
6. A window size less than the modulus of the sequence numbers minus one may be necessary to avoid ambiguity as to the meaning of NR received from different numbering cycles.
7. An extended numbering system using a modulus of 128 is desirable to ensure high link throughput.
8. To preserve link integrity, when a SARM (or UA) is to be transmitted all I-frames (or supervisory frames) in the process of being transmitted must be aborted. The SARM (or UA) should be transmitted only after the propagation delay of the last transmitted, still unacknowledged I-frame (or supervisory frame). Otherwise, this may cause undetected loss and duplication of frames (Figure 4(C)).

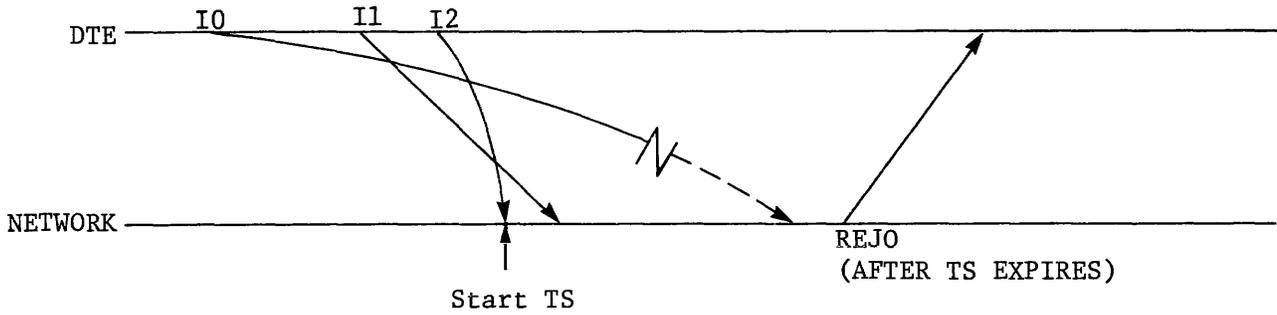
Finally, the following DTE implementation considerations must also be taken into account:

1. It is required that the multi-circuit link protocol be compatible with ISO HDLC procedures and X.25 LAP and LAPB.
2. The investment made by the customer in his development of the X.25 link control software must be protected by minimizing and localizing the impact of software changes.

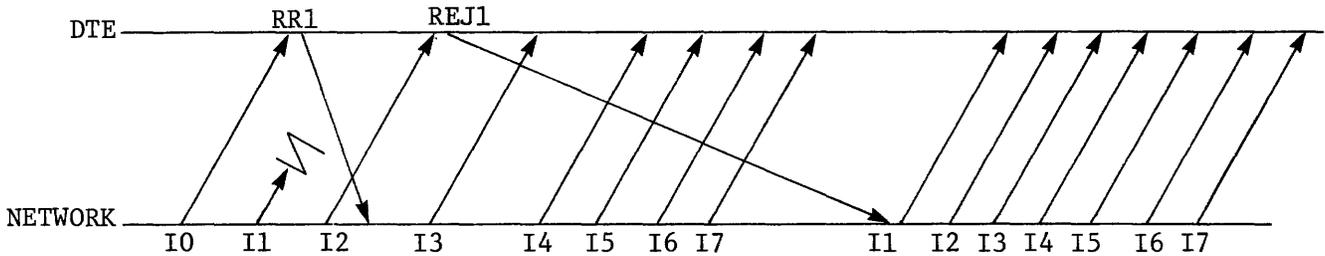
Floating multi-link implementation

A floating multi-link implementation (Figure 3(III)) consists of parallel links attached to different line processors on a node. At subscription time, all the links are assigned the same set of network addresses and logical channels. How-

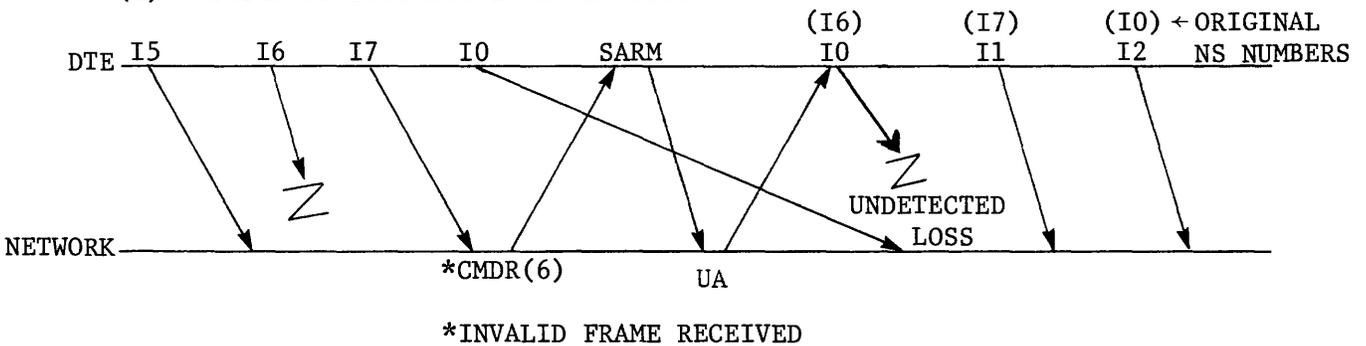
(A) DISORDERING EFFECT OF MULTIPLE CIRCUITS



(B) INEFFICIENCY DUE TO USE OF REJECT



(C) UNDETECTED LOSS DUE TO LINK RESET



LEGEND: FRAMES ARE REPRESENTED AS XY
 WHERE: X = I (INFORMATION FRAMES)
 = RR (RECEIVE READY FRAME)
 = REJ (REJECT FRAME)
 = SARM (RESET FRAME)
 Y = NS OR NR VALUE (0-7)

Figure 4—Some problems of multi-circuit HDLC.

ever, the implementation may dynamically switch the logical channels between the links in a disciplined manner. This depends on considerations of virtual circuit integrity, link availability and load balancing. Each link is independent and may use a different data link control procedure.

Logical channels are not statically fixed to links but float over all the links. To transmit a packet, it is queued to the link module on which there is an unacknowledged I-frame containing an outstanding packet for the same logical channel; if no such link exists, then any link can be used. This simple technique maintains virtual circuit integrity, while avoiding packet level resequencing, by using the sequencing property of data link control procedures such as HDLC.

A switching function is required between the packet level module and the various link level modules. Typically, it maintains a table giving the number of outstanding packets for each particular logical channel and the associated link module. This table is incremented when a packet is queued and decremented when an I-frame is acknowledged at the link level. When a link fails, all its unacknowledged I-frames are switched to another link module for initial transmission or retransmission, in the latter case creating duplicate packets.

Thus, after a link failure, the floating multi-link provides automatic alternate routing of virtual circuits but relies on the X.25 error procedures for handling duplicate packets. The maximum number of logical channels which may have to handle duplicate packets is equal to the window size (K) at the link level. There is no packet loss or requirement for extra resynchronization signals and procedures at the packet level.

However, the effects of duplicate packets in X.25 need further study. For example, they can give rise to conflicting situations in which the DTE and the DCE do not agree as to the state of the interface for a particular logical channel.¹ The possibility of deadlock due to out-of-the-blue duplicate packets (when the DTE/DCE interface is in an ambiguous state) is to be considered. These problems do not arise if extensions to HDLC or higher-level protocol procedures (e.g. using logical channel zero) would make it possible to exchange the HDLC receive state variable, VR, values between the DTE and the DCE. An alternative approach is for the transmitter to number every I-frame, independently of the link, to enable the receiver to eliminate duplicates. This would introduce an overhead on every packet compared to an overhead only incurred during recovery in the former approach.

The floating multi-link introduces an additional interaction between the packet level and link level (i.e. the implementation has to keep track of the mapping of packets to links).

Autonomous multi-link reconnect procedure implementation

An autonomous multi-link implementation (Figure 3(IV)) consists of several autonomous X.25 modules. At subscription time, each X.25 link is assigned its own node, and set of addresses and logical channels.

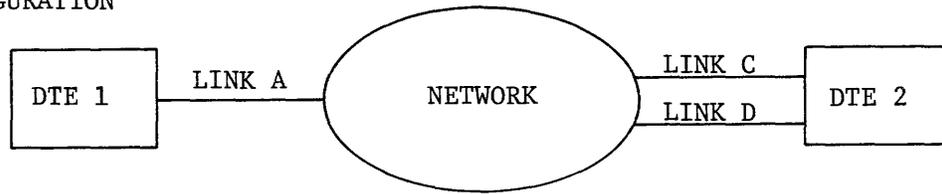
A link failure in an autonomous multi-link implementation loses its associated virtual circuits. The simplest recovery is by the user establishing new virtual circuits on an alternate link to maintain his end-to-end connections. A potential alternative is a network-provided DTE-to-DTE reconnect procedure proposed to improve the fault-tolerance of an autonomous multi-link by implementing an alternate routing capability over the multiply-connected access path.¹⁶ It refers to the capability to reconnect one end of a virtual circuit on an alternate link on any node. It then resynchronizes both ends of the access path and of the virtual circuit to recover from a link failure.

We note that a reconnect capability only protects against the loss of a virtual circuit (VC) when its associated link fails; it does not guarantee against packet loss or duplication. The recovery process is left to the user as it is a function of the reliability requirements of his application. Hence, PSN administrations may offer the reconnect capability as an optional user facility aimed at increasing the VC-recoverability against access link and node failures.

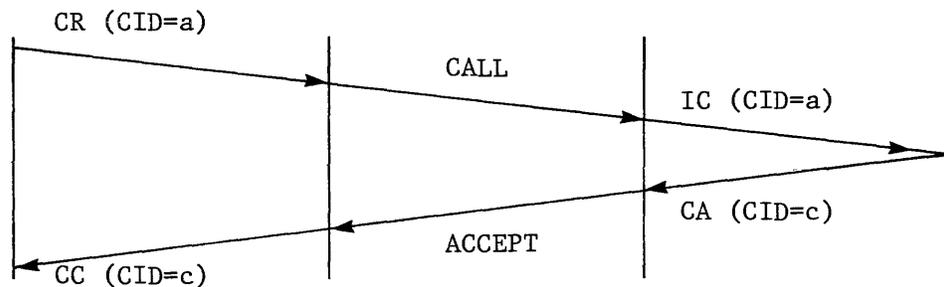
Virtual circuit reconnection can be accomplished by using a reset or a call packet with a facility field containing a parameter specifying it as a reconnection packet and another parameter identifying the virtual circuit to be reconnected. To identify a virtual circuit, it is sufficient for the ends to exchange their unique identification of the connection, namely a numeric connection identification (CID), when the call is created. Such a CID could consist of the (network address, logical channel identifier) pair which is the network-wide naming space at the DTE/DCE interface for identifying one end of a VC. Thus, Call Request and Call Accepted packets may contain a CID parameter field for the DTE to identify the local end of the virtual circuit. Call Connected, Incoming Call and Reset Indication packets may contain a CID parameter field for the DCE to identify the remote end of the virtual circuit. The CID field of a Call or Reset Request "reconnection" packet will contain a CID for the remote end, and optionally another CID for the DTE to identify the new local end to which the remote end is reconnected. We note that this requires extension of the Call Accepted and Call Connected packet formats of X.25. The simplification of the procedure to use a single CID for both ends of the VC is a subject for further study.

An example of the call "reconnection" procedure in a national call is shown in Figure 5. DTE 1 is connected to the network by link A whilst DTE 2 is connected by autonomous links C and D. A virtual circuit is established through links A and C originally. When DTE 2 detects link C is down, it selects a free logical channel on link D and reestablishes the call by sending a Call Request packet with reconnection facility (REC). When the node servicing link A receives this packet, it checks the facility field. As it refers to the reconnection of a previously established call, the node does not generate an Incoming Call but resets its end of the virtual circuit instead. On completion of the reset, the node will initiate an abort of the process originally serving the remote end (old logical channel). It also returns a confirmation that reconnection is completed to the new process (new logical channel) at the remote end. If the net-

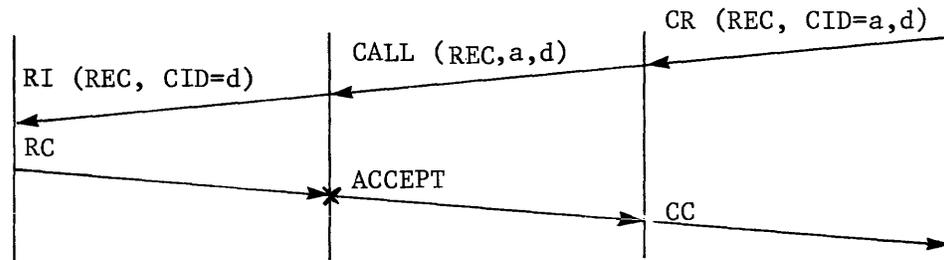
(A) CONFIGURATION



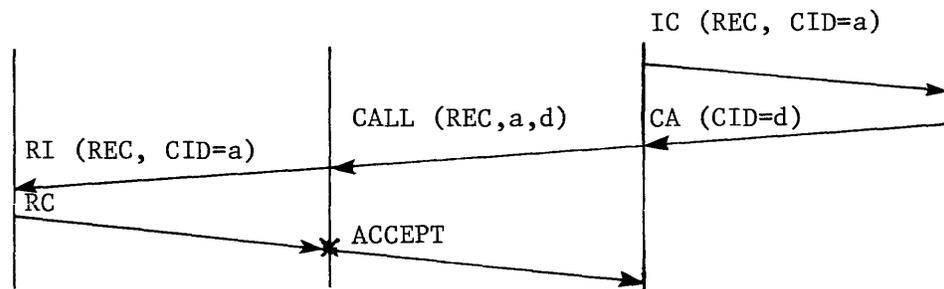
(B) CALL SET UP FROM DTE 1 (LINK A) TO DTE 2 (LINK C)



(C) DTE-INITIATED RECONNECTION BY DTE 2 AFTER LINK C FAILURE



(D) NETWORK-INITIATED RECONNECTION AFTER LINK C FAILURE



LEGEND: CR= CALL REQUEST
 IC= INCOMING CALL
 CA= CALL ACCEPTED
 CC= CALL CONNECTED
 RI= RESET INDICATION
 RC= RESET CONFIRMATION
 CID= CONNECTION I.D.
 REC= RECONNECT PARAMETER
 *= INITIATE ABORT OF PROCESS FOR CID=c ON LINK C

Figure 5—Illustrations of possible reconnect procedures.

work detects that link C is down and knows that there is an alternate link D, it may automatically reconnect as shown in Figure 5(D) on link D.

The reconnect capability may be useful in the following application areas:

1. Whenever fast re-establishment of calls is required. This is essential for real-time traffic or for commercial real-time systems, e.g. digitized voice, airline reservations.
2. Whenever the context of the communication of the end users is to be kept. One example is in facilitating access control to data base systems.
3. Whenever a specified quality of service (e.g. probability that a virtual circuit is not disconnected) is to be maintained. This is particularly relevant to permanent virtual circuits.

Software and addressing considerations

Having outlined the implementation alternatives, it is important to note that the node software organization and the internal operation of a PSN may impose constraints on the implementation of user access paths. In particular, multi-homing can be easily achieved only by the autonomous multi-link implementation since the other implementations need to coordinate the transfer of packets on the various physical circuits and links, and are restricted to a single node.

Typically, the node control software of a PSN may be designed as a set of high-level subsystems consisting of hierarchically cooperating, functionally separate, and modularly decomposed components. The distribution of the software components to the various physical processing units (i.e. LC, LP, CE) determines the circuit configurations which may be implemented. The line processor (LP) performs at least all link control functions, and its reliability would restrict the reliability of a multi-circuit link.

A floating multi-link implies that packets for the virtual circuits are dynamically switched between links. If both the link and packet levels of X.25 are implemented in the LP, then a floating multi-link must have all its links connected to the same line processor and can be considered as an alternative for a multi-circuit link. However, if only the link level is implemented in the LP, then the links can be connected to different line processors and use the common memory for communication and coordination. Such a software organization permits a floating multi-link with a still better reliability.

An autonomous multi-link may not be constrained by node software structure. If the links are controlled by different LPs, the coordination and resynchronization (i.e. the reconnect procedure) may be accomplished by the common equipment. However, for a multi-homed DTE, the addressing of access points on the network may impose constraints on the implementation.

A network address is a unique identification for an access

point in the network, and is used for routing calls during call establishment and for administrative purposes such as accounting, error reporting and directory listing. A multi-homed DTE is supported by assigning to it an autonomous multi-link with different addresses for the links. The subscriber usually prefers one or more addresses for his DTE independently of the number of links connecting it to the network. The subscriber may request an autonomous multi-link configuration in which the logical channel identifiers (other than zero) are partitioned over all the links for identifying individual conversations uniquely. To the subscriber, a multi-homed DTE with a single address set is a useful reliable configuration. Because of network routing, tariff and accounting problems, the PSN administration may find it difficult to offer such a single address, multi-homed configuration.

In conclusion, each PSN administration will determine which access path implementations to offer depending on the software constraints of the particular switch architecture (i.e. single, dual, multi-processor) used in the network, and constraints due to the subnetwork system design (i.e. addressing, routing, . . .).

COMPARISON OF ALTERNATIVES

This section contains a comparison of the access path configurations and implementations, described in previous sections, in terms of user reliability parameters, protocol features and implementation requirements. The three alternatives identified in this paper are: 1) multi-circuit link, 2) floating multi-link, and 3) autonomous multi-link reconnect procedure.

Reliability considerations

The Reliability and Availability of the access path implementations have already been quantitatively compared by referring to their associated configurations. However, we note that the Reliability of any associated virtual circuit depends on the characteristics of the protocol implemented on the configuration in a similar manner as does the Fault-tolerance.

The multi-circuit link implementation has the best Fault-tolerance because its protocol is defined to consider the different physical circuits as a single transmission facility such that the failure of any one physical circuit does not affect any virtual circuit. Virtual circuits are aborted only when all the physical circuits fail. The floating multi-link implementation has comparable Fault-tolerance in that it does not lose virtual circuits after a physical circuit or link failure. However, it requires a local recovery procedure which guarantees the integrity of the information transfer after a link failure.

On the other hand, a link failure in an autonomous multi-link implementation will lose the virtual circuits associated to it. These virtual circuits can be recovered by re-estab-

lishing them on an alternate link. Hence, the autonomous multi-link implementation has an acceptable Fault-tolerance only if the end-to-end reconnect procedure is implemented to resynchronize the virtual circuits in the event of a link failure.

For the highest reliability achievable by multi-homing, the PSN administration may find it advantageous to offer an autonomous multi-link with multiple address sets. It should be stressed that a multi-link, implemented on independent line processors of a single node, with a single address set has slightly worse Reliability than a multi-homed multi-link. Furthermore, the Availability of the single node multi-link is better than that of the network. Thus, multi-homing does not bring any significant gain in overall reliability since the node is usually a multi-processor system. The high correlation between the failures of the different communication lines, connected to the same DTE, and the wide variability of their failure rates (3-4 orders of magnitude are common) would reduce the reliability gain still further. A more reliable and available access to an X.25 network may be better achieved by ensuring that the lines are really diversely routed, and as identical as possible, than by multi-homing.

Protocol and implementation considerations

To maintain compatibility with ISO, a multi-circuit HDLC protocol may be a desirable alternative for an X.25 interface. Major points left for further study in a multi-circuit HDLC protocol are semantics of SREJ, duration of time-out values and performance considerations.

Two areas to be considered to make the floating multi-link a viable proposal are:

1. Link failure recovery procedures which avoid duplicate packets.
2. Specification of the interaction between the frame and packet levels.

The multi-circuit link and floating multi-link have the advantage of having only local implications, e.g. multi-circuit requires the implementation of a new frame level protocol in both the DTE and the node. On the other hand, multi-link reconnect implies an end-to-end protocol affecting Level 3 of the DTE/DCE interface. This necessitates some implementation changes in the DTE and complex internal resynchronization mechanisms in the network. All these alternatives require the development of appropriate standards.

Multi-homing can be achieved by an autonomous multi-link configuration. Hence, the autonomous multi-link has potentially better flexibility than the multi-circuit or floating multi-link, but it is more attractive only with the provision of a reconnect capability. It is simpler to implement reconnect for recovery from a link or a line processor failure than from a node failure. The implementation of reconnect by a PSN is constrained by its routing scheme and its assignment of network addresses to the multi-link.

CONCLUSIONS

Users have different reliability requirements depending on their applications. A fault-tolerant data communications service is becoming more economically feasible because of the availability of cheap, reliable hardware components and new programming techniques to master software complexity. There are different multiple circuit configurations to meet these reliability requirements for the access paths between the DTE and the PSN. The implementation of these configurations must take into account existing constraints in both the DTE and the PSN. It is recommended that PSN administrations offer some form of multiple circuit access for DTEs which hides the network operational constraints from the users, e.g. hunt groups or preferably a multi-circuit link. The definition of a multi-circuit capability within HDLC by ISO and CCITT is urgently required. Finally, packet level procedures providing higher virtual circuit reliability should be studied within CCITT.

ACKNOWLEDGMENTS

R. J. Chung would like to express gratitude to his supervisor at Université de Montréal, Professor Gregor V. Bochmann, for having introduced him to teleprocessing and protocols, and his useful comments on this paper. Also, Professor J. Gecsei was especially helpful with useful advice on the original report. Furthermore, he would like to thank his former colleagues at Computer Communications Group and Bell Northern Research for having provided such a stimulating work environment.

REFERENCES

1. Belsnes, D. and E. Lynning, "Some Problems with the X.25 Packet Level Protocol," *Computer Communications Review ACM SIGCOMM*, Vol. 7, No. 4, October 1977, pp. 41-52.
2. Bochmann, G. V., and R. J. Chung, "A Formalized Specification of HDLC Classes of Procedures," *Proceedings of the National Telecommunications Conference*, Los Angeles, December 1977, pp. 03A: 2-1 to 03A: 2-11.
3. Clipsham, W. W., P. E. Glave and M. L. Narraway, "DATAPAC Network Overview," *Proceedings of the ICCO*, Toronto, August 1976, pp. 131-136.
4. Cotton, I. W., and D. S. Grubb, "Criteria for the Evaluation of Data Communications Services," *Proceedings of IEEE 1976 Computer Networks—Trends and Applications*, Maryland, November 1976, pp. 71-78.
5. Danet, A., R. Despres, A. Lerest, G. Pichon and S. Ritzenthaler, "The French Public Packet-Switching Service: The Transpac Network," *Proceedings of the ICCO*, Toronto, August 1976, pp. 251-260.
6. Doll, D., "How to Calculate Network Reliability," *Data Communications*, Vol. 4, No. 1, Jan./Feb. 1975, pp. 43-49.
7. Kirstein, P. T., "Planned New Public Data Networks," *Computer Networks*, Vol. 1, No. 2, September 1976, pp. 79-94.
8. Mainguenaud, G., and B. Jamet, "A Multi-line Data Link Control Procedure," *Proceedings of the ICCO*, Kyoto, Japan, September 1978, pp. 289-294.
9. Marcus, M. J., "On Attaining the Availability Required in Future Information Processing Systems," *Information Processing '74, Proceedings of IFIP Congress '74*, August 1974, Stockholm, Sweden, pp. 141-146.
10. Martin, J., *Systems Analysis for Data Transmission*, Prentice-Hall, Inc., New Jersey, 1972.
11. Morgan, D. E., D. J. Taylor and G. Custeau, "A Survey of Methods for

- Improving Computer Network Reliability and Availability," *IEEE Computer*, Vol. 10, No. 11, November 1977, pp. 42-50.
12. Roberts, L. G., "Packet Network Design—The Third Generation," *Information Processing 77, Proceedings of IFIP Congress 1977*, Toronto, pp. 541-546.
 13. Rybczynski, A. and D. Weir, "DATAPAC X.25 Service Characteristics," *Proceedings of the Fifth Data Communications Symposium*, Snowbird, Utah, September 1977, pp. 4.50-4.57.
 14. Rybczynski, A., *et al.*, "A new communication protocol for accessing data networks—the international packet-mode interface," *AFIPS Conference Proceedings of the National Computer Conference*, New York, June 1976, Vol. 45, pp. 477-482.
 15. Spragins, J., "Reliability Problems in Data Communications Systems," *Proceedings of the Fifth Data Communications Symposium*, Snowbird, Utah, September 1977, pp. 3.9-3.13.
 16. Wessler, B., Private memorandum on "Reconnect Procedure."

A fail-safe distributed local network for data communication*

by JANE W. S. LIU, IZUMI SUWA, ROBERT STEPP and SERGIO M. HINOJOSA

University of Illinois
Urbana, Illinois

and

TSUTOMA UTSUQI

Nippon Electric Company
Tokyo, Japan

INTRODUCTION

We are concerned here with the design of an interconnection network for communication between terminals and computers in local distributed computer systems. By a local system, we mean one in which computers and terminals are concentrated in a relatively small area (e.g., with distance between terminals < 100 Km) and interconnected by communication links with relatively large bandwidth (e.g., up to 10 Mb/sec.). By a distributed network, we mean one in which the control of communication network is distributed among the computers and terminals. For our discussions, there is no need to distinguish terminals from computers. Hereafter, we shall refer to them simply as stations.

In recent years, several local networks have been designed and implemented. Among the better known ones are DCS (Distributed Computer System), Mitrix, Spider and Ethernet.¹⁻⁴ These networks, designed for different objectives (e.g., different intended applications), illustrate the different approaches taken to provide fast and reliable communication. For example, in both DCS and Ethernet, reliable communication is achieved through the distribution of network access control to the stations in the network. In Ethernet, all stations are interconnected via a passive medium (a high bandwidth low loss bus called Ether.) Broadcast packet switching discipline is used. The DCS is a ring network. A message from a source station addressed to a process residing in another station is placed on the ring and is transmitted in one direction to all stations connected to the ring. A station copies a message only if its interface finds that the destination process name contained in the message matches the name of a process residing in the station. The message is allowed to travel along the ring back to the source station and, there, is removed from the

ring. Spider is a ring network containing several rings. A central switch is used to route packets destined for stations not in the same ring with the source stations. Similar to Spider, Mitrix is also a centralized network in which switching and bandwidth allocation functions are performed by centralized computers.

We consider here a fault tolerant communication network in which multiple-link and subnetwork failures may occur due to natural or intentional damages. Application of fault tolerant networks can be found in shipboard communication systems, and in traffic control and navigation systems where the growing use of digital sensors and computers requires large local fail-safe communication systems. Other application examples include networks linking process control microprocessors, data acquisition terminals, central computers and other facilities in large manufacturing plants, and networks connecting intelligent terminals serving individual users for the purposes of text editing, managing personal data bases, providing controlled access to shared data bases, exchanging messages, etc. In these examples, an interconnection network must provide continuous communication linkage between stations.

Assumptions on the overall local distributed network and our design objectives are summarized in the next section. In the third section, a routing algorithm, which guarantees finding a route from any source and destination pair as long as routes between them exist, is described. A network access protocol is described in the fourth section. The relatively small round-trip propagation delays and wide bandwidth of the links are used in the design of this protocol.

GENERAL ASSUMPTIONS AND DESIGN OBJECTIVES

In the interconnection network considered here, the communication links are full duplex lines with bandwidths

* This work was partially supported by Office of Naval Research No. N00014-75-C-0982.

identical to or larger than that required by any communication station. Connecting to the intersections of two or more communication links are switch nodes. The switch nodes resemble outlets in power supply lines. Any station with the appropriate interface may be plugged into any of the switch nodes. Some of the switch nodes, therefore, are not connected to any station. Without a station attached to it, a switch node serves simply as a routing device.

To assure that the stations remain connected with high probability, redundant connections are provided. We consider here the special case when the interconnection network forms a homogeneous array. Examples of such networks are shown in Figure 1. The design of network topology for a given survivability criteria is a well known problem and will not be addressed here. We note that when successful communication between all pairs of communication stations are equally important and switch nodes and links are identical, homogeneous networks are maximally reliable⁵ (both in terms of being maximally connected networks and having maximal survivability.) Simulation studies⁶ of large arrays show that in terms of the survivability (average fraction of stations which remain connected after damage occurred), a network in which each switch node is

connected to four or more of its nearest neighbors is almost optimal when all routes can be used.

The communication links, switch nodes and stations may fail independently and at random. Moreover, multiply links and switch nodes may fail together due to natural and intentional damages. Changes in the network topology may also be caused by additions to and modifications of the network. However, links in a local network, not being exposed to conditions which affect long distance cables or radio links, are less susceptible to intermittent failures. Hence, we assume here that changes in network topology occur relatively infrequently. In particular, it can be considered as fixed during the routing of a particular message.

It is our objective to design a message routing algorithm so that the delivery of message from a source station to a destination station is guaranteed as long as there exist some paths from the source to the destination. If no such path exists, this fact is detected sooner or later. Because of the relative shortness and wide bandwidth of the communication links, it is not essential that messages be delivered via the shortest routes. We assume that each switch node is aware of the status of the links connecting to it and the status of the adjacent switch nodes. Only such local information on the condition of the network is used by the routing algorithm.

As in the case of store-and-forward networks, a message being transmitted is error-checked by each of the switch nodes enroute. However, to minimize the complexity of the switch nodes, the messages themselves are not stored by the switch nodes. The requirement on the processing capability of these switch nodes is further reduced by carrying out the tasks such as buffering for flow control, message error control, duplication detection, etc., in the network interface inside the stations. (The description of this interface is in the fourth section.)

ROUTING ALGORITHM IN A HOMOGENEOUS NETWORK

The routing algorithm designed here guarantees the delivery of messages between any pair of stations as long as there exist some paths between them. For the sake of concreteness, we confine our discussions to the case of the endless matrix network shown in Figure 1a. Generalization to other similar homogeneous network topologies is discussed in Reference 7.

Basic concepts

A natural set of addresses for the switch nodes in the endless matrix is the set of ordered pairs (i,j) , $i=0, 1, \dots, m-1$ and $j=0, 1, \dots, n-1$. Because of the relative shortness of communication links between switch nodes, propagation delay severed by a message can be reasonably measured by the number of switch nodes along the route between its source and destination. The number of switch nodes along a route connecting two nodes is referred to as the distance

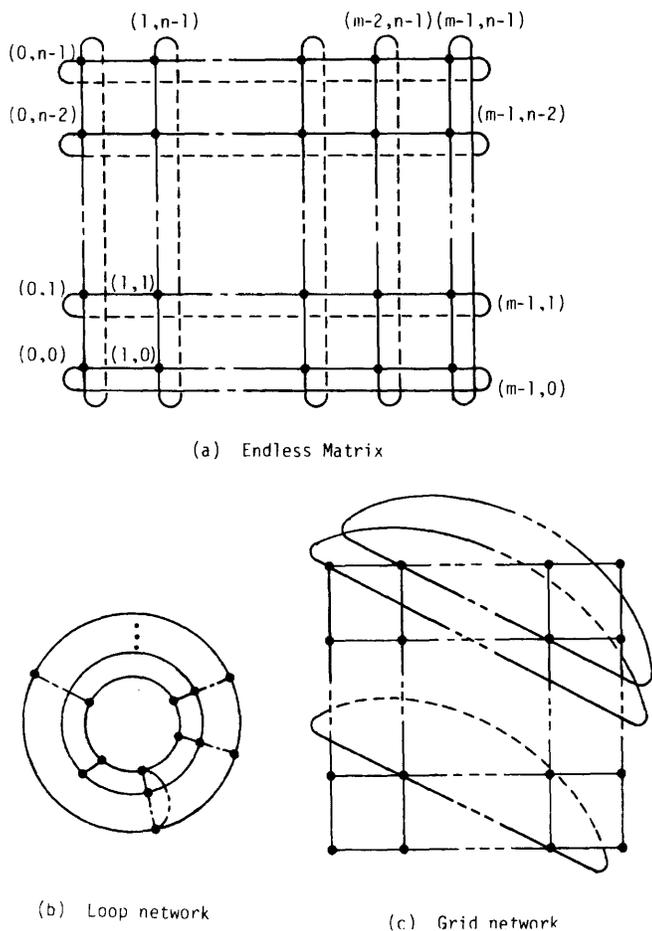


Figure 1—Examples of almost homogeneous arrays.

between the two nodes. For an m by n network, modulo m arithmetic for the x direction and modulo n arithmetic for the y direction allow the calculation of distances between switch nodes.

When every switch node and link is operational, a routing algorithm that finds the shortest path can be obtained easily. For example, the shortest route from the switch node (i, j) to the node (i_d, j_d) shown in Figure 2 can be found by calculating the distances from row j to row j_d along column i in both directions: $|j_d - j|$ or $n - |j - j_d|$ and selecting to send the message in the shorter direction along column i to node (i, j_d) . From node (i, j_d) , the message is forwarded to column i_d along j_d in the shorter one of the two directions. We referred to this simple shortest route algorithm as Algorithm P.

However, Algorithm P does not work when some switch nodes and links are down. If there is a switch node or link along the route chosen by Algorithm P fails, a detour around it must be found. To carry out this simple statement for all possible patterns of multiple link and node failures is not as simple as it first seems since switch nodes have only local information. Examples in Figure 3 illustrate some of the difficulties. The bold lines show schematically the links and/or switch nodes that are down, thus forming a "wall." Suppose that the wall forms a trap as shown in Figure 3a. Moreover, suppose that the entrance to the trap is only one row wide. According to Algorithm P, a message from S to D is sent from S to A and onward to the right. Clearly, the destination D cannot be reached in this way. As a matter of fact, in order to reach D , the message must be sent out of the trap, again through A . That is, the switch node A must relay the message to its right neighbor the first time but to its left neighbor the second time. Note that in both cases, the local information about the condition of the network for node A is the same. This fact implies that some

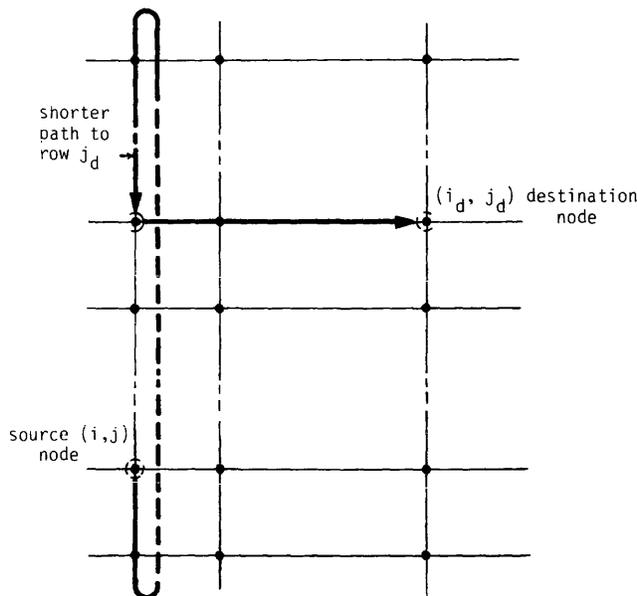
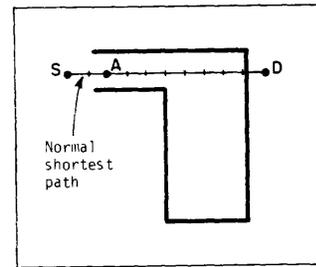
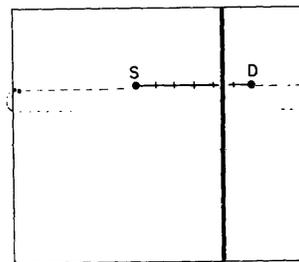


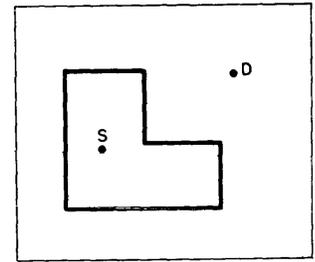
Figure 2—Illustration for the shortest route.



(a)



(b)



(c)

Figure 3—Examples of traps formed by failed lines and nodes.

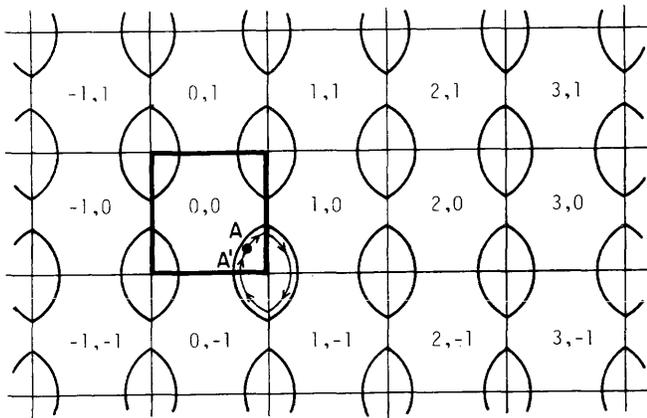
information in addition to the conditions of its neighboring nodes and links is necessary.

Let us consider also the configuration in Figure 3b. Suppose that a message is sent along the shortest route from S to D . When the wall is reached, we may attempt to search for an opening in the wall to reach column i_d . The search is obviously fruitless since the wall forms a closed loop. (Hereafter, this type of loop is referred to as a type I loop.) However, column i_d can be reached via the alternate route shown in the figure. On the other hand, the wall may form a loop (referred to as a Type II loop) as shown in Figure 3c. In this case, since there is no way to reach the destination D , this fact must somehow be detected.

Wall-touching principle

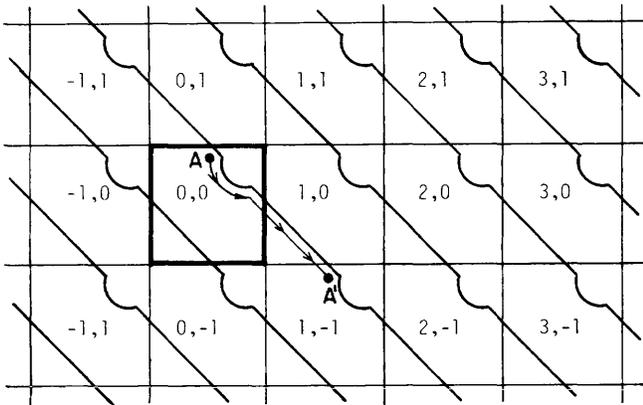
To get around the walls, we may use a version of the wall-touching procedure commonly used to find a way out of a maze. That is, one traverses along the wall always keeping the wall to the right (or left). Unless the wall forms a loop, a more desirable position will eventually be reached.

In order to detect the existence of a loop, we need to check if a switch node of reference (say A) has been reached more than once. (Selection of the node of reference will be discussed later.) However, checking the position of the switch node reached is not sufficient as illustrated by the example in Figure 4. In this case, the switch node A is reached twice although the wall does not form a loop. However, it is clear from this example that a loop can be unmistakably detected if we also check the direction in which the message is sent from the switch node A each



Expanded network address for node A: $[0,0](i,j)$
 Expanded network address for node A': $[0,0](i,j)$

(a) Type II loop



Expanded network address for node A: $[0,0](i,j)$
 Expanded network address for node A': $[1,-1](i,j)$

(b) Type I loop

Figure 6—Examples of Type I and Type II loops. A→→→→A' shows a closed path traced during wall touching.

destination that can be reached in each case. ("Left" means the cell to the left of the current position. "Up" means the cell on top of the current cell, etc.).

Routing algorithm

The routing algorithm is repeatedly carried out by the switch node enroute until the destination node is reached or the fact that there is no path connecting the source and the destination is ascertained. The routing algorithm is based on the minimal distance algorithm, to be described.

Minimal distance algorithm

When no loops are encountered, distance between the current node to destination node is calculated as in Algo-

rithm P. When a Type I loop is encountered, the distance between nodes $[p,q](i,j)$ and $[p',q'](i',j')$ in two different cells is given by $d=(p'-p)m+(i'-i)|+(q'-q)n+(j'-j)|$.

Let d_{cl}^{**} denote the shortest distance to the destination from all switch nodes already reached so far. The node j whose distance to the destination is equal to d_{cl} is referred to as the closest node. It serves as the reference in detection of loops.

Minimal Distance Algorithm

1. Calculate the distance from the current node to the destination, d_c , and compare with the current value of d_{cl} .
2. If $d_c \leq d_{cl}$, move to the neighboring node that is on the shortest route to the destination determined so far, if possible. Update the record for d_{cl} and the address of the closest node.

** When there is more than one node with distance to destination equal to d_{cl} , the closest node is one reached last during routing.

TABLE I—Reachable Virtual Destination

p	q	Basic pattern	Direction for message delivery
0	0		no way
1	0		up
-1	0		down
0	1		left
0	-1		right
$ p = q $			
1	1		left or up
1	-1		up or right
-1	1		left or down
-1	-1		down or right
$ p > q $			
1	1		left
1	-1		right
-1	1		left
-1	-1		right
$ p < q $			
1	1		up
1	-1		up
-1	1		down
-1	-1		down

p=No. of horizontal boundaries crossed.
 q=No. of vertical boundaries crossed.

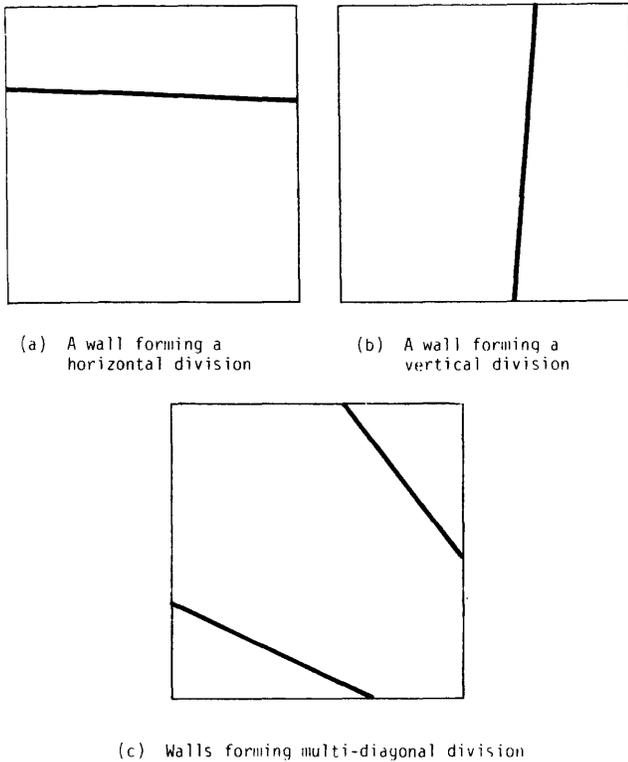


Figure 7—Division of the network into different regions by the walls.

3. If $d_c > d_{c1}$ or if $d_c \leq d_{c1}$ but impossible to reach the neighboring node chosen so far, move according to the wall touching principle. A loop is detected whenever a node is reached twice while traversing in the same direction.

Description of the routing algorithm

Starting in Phase 1.

Phase 1

1. Repeat the minimal distance algorithm until the destination is reached or a loop is detected.
2. If the destination is reached, or Type II loop is found, terminate.
3. If Type I loop is detected, determine the appropriate virtual destination in the expanded network as given in Table I and go to Phase 2.

Phase 2

4. Apply the minimal distance algorithm repeatedly, using the modified distance calculation for destination node until a loop is detected or the destination is reached.
5. If the destination is reached, terminate.
6. If a loop is detected (this loop will always be Type I if the network remains unchanged), change the wall

touching parameter (from left to right or vice versa) and go to (4). If both left and right had been tried, terminate.***

The algorithm is illustrated by the example shown in Figure 8. It is written in detail in [7] in a modified version of PASCAL language. Table II lists the information required for routing a message from the source node to the destination node. This information is carried in the message and is updated by the switch nodes enroute. The node address and virtual network address of the closest node is used in the minimal distance algorithm, and also as reference for loop detection. A current virtual address is needed for distinguishing the types of loops and also to calculate the distance in Phase 2. The entry in touch hand specifies whether left wall-touching or right wall-touching is being carried out. Sending direction specifies the direction in which the message was sent when the node is reached. This is used when testing for the existence of a loop. The total number of bits in the case of $m=n=16$ is 50 bits.

*** This case occurs when there are two or more Type I loops exist isolating the source from destination.

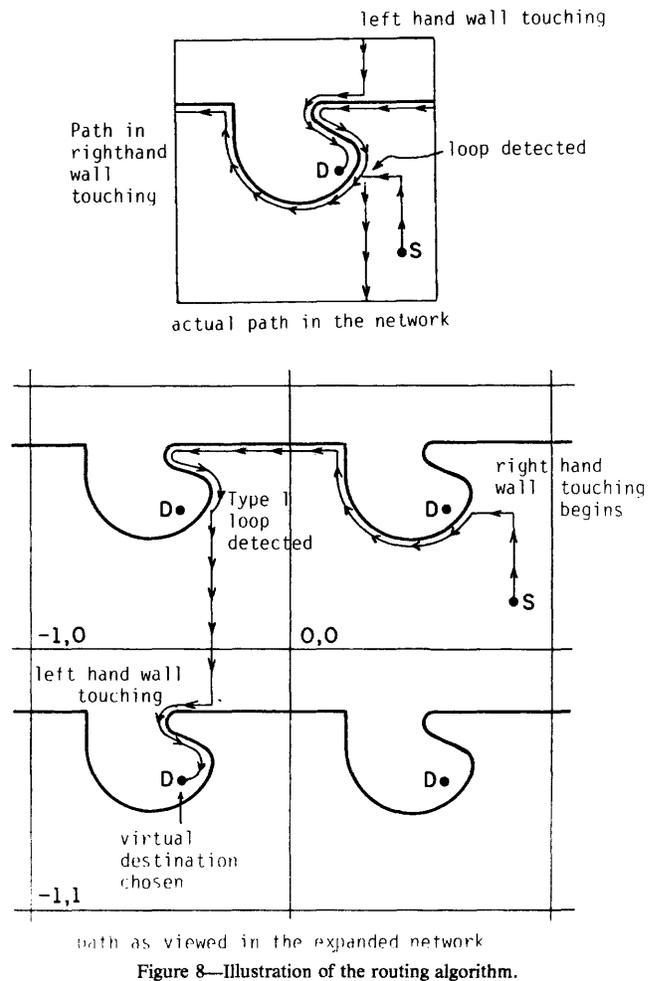


TABLE II

	length in bits ($x = \lceil \log_2 m \rceil + \lceil \log_2 n \rceil$)
Address of destination ((0,0) (m,n))	x
Virtual network address ((-m, -n) to (m,n))	x+2
Node address of closest node	x
Virtual network address of closest node	x+2
Current virtual network address	x+2
Touch-hand (right, left)	1
Sending direction a closest node (U, L, LEFT, RIGHT)	2
Phase (1,2)	1

NETWORK PROTOCOL AND INTERFACE

In the design of network protocol, we make use of the relatively small round-trip propagation delays and large bandwidth of the data links in a local network. Since changes in network topology occur relatively infrequently, a route from a source to a destination, once found, can be used until either it is no longer needed or corruption or disruption of data transmission is detected. Within the network, links which consistently corrupt data are said to be unoperational. A version of an ARQ (Automatic-Repeat-Request) scheme is used for error control. We assume that source stations in the network either contain data buffers or are able to regenerate data as needed.

The switching discipline used here is referred to as call-switching discipline. It resembles the scheme used in the telephone network where a path for audio signals is set up by sending a control signal during the call establishment phase. In our network, whenever a source station wants to communicate with a destination station, a network control message is sent from the source switch node for the purpose of establishing a route from the source to the destination. When a route is found, the destination switch node sends a response message to the source to acknowledge the success of call establishment. Hereafter, messages containing data from this source station to the destination station are sent along this route.

From the viewpoint of the source station, the transmission of a message proceeds according to Figure 9. To establish a route, the source station assembles an addressing control message containing the destination address (switch node address plus device number) and transmits it to the switch node which serves the source device. Without loss of generality, the formats of communication messages are assumed to be as shown in Figure 10. For each addressing control message sent, a response message is received in return. The source station compares the address in the response message with the address of the destination node. When they match, this response is interpreted as an acknowledgment that a working route to the destination is

found. The source station may begin to send data (using the data variant of the communication message format). Again, for each message sent, a response message is received. As long as the address in the response messages remains that of the destination node, data transmission proceeds. A response message containing a node address other than the destination node address is interpreted as a negative acknowledgment. When such a response is received, transmission is restarted beginning with the addressing phase, by the source. To terminate the call, one or more terminator control messages are exchanged. Upon detecting the terminator control messages, switch nodes along the established route may delete routing information used during the call.

The operations of the switch node are a little more complicated. Only the general operation of a node will be explained here. A switch node services many links to other nodes plus zero, one or more (up to d) ports to devices and is capable of receiving messages on all device ports and link ports simultaneously. Upon receiving a message, the switch node either forwards the message toward the destination within the time limit defined for receiver timeouts or returns to the sender a response message containing the switch node's own node address (plus a zero device code). This response message is used either when the node is too busy to issue a proper response in time, or when concurrent traffic makes it impossible to react to an addressing request.

Clearly not all requests for attention of the switch node can be serviced. The addressing mechanism resolves the switch node resource allocation problem. All contentions occur within the addressing phase. When an addressing control message arrives at a switch node where the resources to process it are not available, the node responds to the requestor with a response message containing the

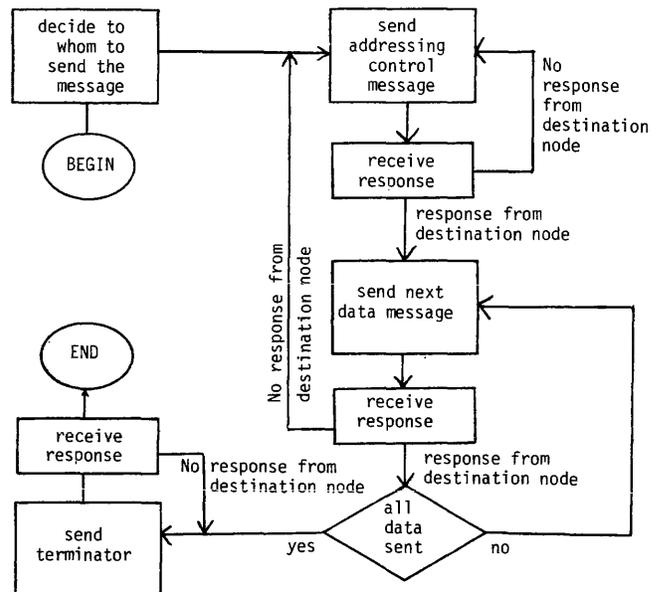
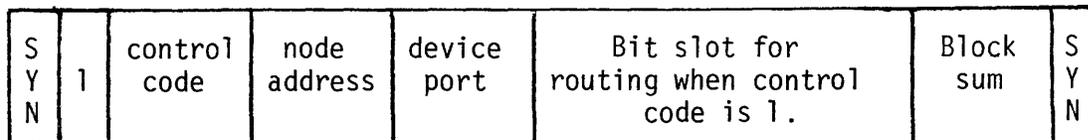
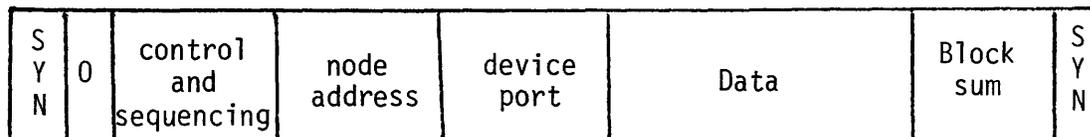


Figure 9



Network control message format



Data message format

Control code for network control message format

- 00 - addressing control: emitted by the source node as part of the call establishment signalling. The address of the destination node is contained in the word.
- 01 - addressing response: generated by switch nodes and returned to the source node as a response to each addressing control message processed and forwarded. The node address and device port form the address of the destination device address when addressing control message is correctly received and forwarded. The node address contains the address of the switch node when no response is received, or the response is erroneous, or when the switch node or its outgoing links are busy.
- 10 - terminator: emitted by the source node to terminate the call. Node and device address in the word is the address of the destination.

Figure 10—Communication message for stations in the network.

address of the node. The requestor then reissues his addressing request. This process repeats until a working route is found.

After a route to the destination is found, the switch nodes enroute keep track of the outgoing links on which messages to the destination are sent. A switch node acts like a relay point of messages in the network. It passes any messages received directly to outgoing links if the block-sum-check by the switch node does not find the message erroneous. For each message sent out over a link, the switch node expects a response message from the neighboring node in return. No erroneous messages are relayed further. When an erroneous message is received, it is discarded without eliciting a response. If no response arrives, or if the received response is erroneous, the link is deemed unoperational. The switch node informs the source node this fact by sending a response message containing its

own address. Thus, alternate route selection is invoked to find another working route. During addressing phase, when a response message is properly returned to a switch node, it is passed back to the incoming line and is the response which the source device will ultimately see. Thus the source device may monitor the progress made by the addressing message at any time. The whole operation is analogous to remote controlled switching units, where each switch node sets up a connection to relay messages as it processes an addressing message and the connection from source to destination is built up one link at a time.

The communication scheme presented above is distinctly different from packet-switching, and seems capable of supporting the demands of local communication networks. Future work includes (a) The determination of whether the network will behave deterministically. (That is the establishment of one route does not lockout activities over

another.) (b) Performance evaluation to determine the effective bandwidth of the network. (c) The design of addressing scheme and protocol for broadcasting mode. (d) The design of diagnostic aids needed to maintain/repair the network.

REFERENCES

1. Metcalfe, R. M., and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Comm. of ACM*, Vol. 19, No. 7, pp. 395-404, 1976.
2. Farber, D. J., "A Ring Network," *Datamation*, Vol. 21, pp. 44-46, February 1976.
3. Fraser, A. G., "A Virtual Channel Network," *Datamation*, Vol. 21, pp. 51-53, February 1975.
4. Willard, D. G., "A Sophisticated Digital Cable Communications System," *Proc. National Telecommunications Conference*, 1973.
5. Wilkov, R. S., "Analysis and Design of Reliable Computer Network," *IEEE Transactions on Communications*, Vol. Com. 20, pp. 660-678, June 1972.
6. Frank, H., and I. T. Frisch, "Analysis and Design of Survivable Networks," *IEEE Trans. on Communication Technology*, Vol. COM 18, pp. 501-519, 1970.
7. Liu, J. W. S., I. Suwa and R. Stepp, "End-to-End Protocols and Suboptimal Routing Algorithms in Distributed Local Fail-Safe Communication Network," report in preparation.

An analysis of a distributed switching network with integrated voice and data in support of command and control

by DANIEL SCHUTZER

Navalex

Arlington, Virginia

INTRODUCTION

The potential for increased transmission efficiency of military tactical command and control data links through voice/data integration has increased in importance in recent years. With the introduction of even more sophisticated and automated weapons systems, the requirement for on-line exchange of data among tactically-embedded military computer systems has risen dramatically. Today we are faced with a proliferation of data link requirements for innumerable military systems that span all the services. An already densely populated electromagnetic environment is faced with still greater demands for its scarce bandwidth resources. Adding to this situation is the fact that these very resources that are in such heavy demand are quite fragile in the face of military electronic warfare measures. And, with few exceptions, the means for countering these electronic warfare measures place still greater demands on the already scarce bandwidth resources. All of the above provide the motivation for exploring new avenues and techniques that achieve greater efficiency of transmission resources. Economically, the revolution in digital componentry makes more viable the consideration of a greater degree of voice/data integration over tactical data links than would have been possible just a few years ago. Namely, competing signal processing techniques—Digital LSI, CCDs and SAWs are pushing the boundaries of technologies and making economically attractive spread spectrum, time division multiple access data link systems and internal multiplexed data distribution techniques which would simplify the integration of voice and data. Finally, with the increased sophistication of military weapon systems, the interrelationships and employment of voice and data in the conduct of war becomes even more intertwined and requires in many cases careful re-evaluation and re-enumeration.

Although there has been much analysis and study in recent years concerning the advantages and disadvantages of integrating voice and data over commercial and military strategic switching networks and long-haul data transmission systems, there has not been much analysis to date that has studied similar trade-offs for tactical data links. Tactical data

links can be thought of as a distributed form of switching where multiple users share a common transmission facility and where the associated multiplexing (time and frequency sharing), addressing and routing functions, rather than being implemented on separate multiplexers, concentrators and switches, are implemented directly in the data link terminal design. Two quite different examples of such a data link are the JTIDS¹ and the Fleet SATCOM TSCIXS links.⁴

Interestingly enough, recent advances in the interconnection of loosely coupled heterogeneous intra-nodal computer systems (systems co-located on a single platform or site) have resulted in bussing and networking schemes (such as the Farber ring, the experimental distributed processor (XDP), the Shipboard Data Multiplex System (SDMS), the Xerox Ethernet, the LCS net at MIT, and the Shipboard Integrated Processor and Display System (SHINPADS)) which can also exploit the advantages of voice/data integration.

In fact, there are distinct advantages to designing the intra-nodal and the inter-nodal networking protocols to be as similar as possible. This is discussed in more detail in the following section.

VOICE/DATA INTERRELATIONSHIPS

All command and control information is communicated either in the form of voice, data, narrative message or graphics. From these communications, data is extracted and information derived. In the past, messages and the data extracted from these messages were filed in storage cabinets, clipboards, etc. for later retrieval and for message accountability. They were filed based upon content and were often cross-indexed on more than one subject. They were associated with one another in terms of the categories they were filed under and in terms of the data extracted and the information derived from the message. The data was retrieved, summarized and in some cases plotted. Although the media and the techniques have changed, the basic processes remain the same even in the computer-to-computer case.

Fundamentally, command and control involves the ex-

change, storage, retrieval and manipulation of information to support command decision-making with respect to military plans, objectives, resource allocations and assignment, conflict resolution and combat direction and management. Consider the following two examples.

At the National Command Authority level and the Fleet Command, or component command level, messages summarizing such information as the combat readiness, local weather and hostile units are prepared by the responsible organizations for electronic transmission, generally in formatted message form, to the appropriate Command authorities. There the messages are received and validated by computers. If validated, the entire message and/or the pertinent data is logged, indexed and filed in digital computers by categories such as date-time-group and unit identity. Likewise, orders, plans and warnings are also prepared in formatted message form for electronic transmittal to the responsible authorities. The stored data and formatted messages are later retrieved, sometimes with supporting computations based upon the retrieved data (information), to aid in situation assessment and command decision-making. On the other end of the spectrum, sensors (generally analog) are automatically processed and reduced to digitally formatted sensor contact reports which are then, where possible, automatically associated with earlier sensor reports to identify targets with sufficient location accuracy for assignment to an appropriate weapon system. These target assessments are transmitted directly to the appropriate weapon direction computer systems and there input automatically to the appropriate fire control solution. Human interaction is restricted to ambiguity resolution, validation and override types of activities. In all of these cases we find that the process can still be modelled by a basic message handling (message preparation, distribution and routing, validation, storage, retrieval and manipulation and presentation) operation/system.

In fact, all command and control inter-process data can be modelled as an n -ary relation, R , which has the properties:

1. Each row represents an n -tuple of R .
2. The ordering of the rows is immaterial.
3. All rows are distinct.
4. The ordering of the columns is significant—it corresponds to the ordering S_1, S_2, \dots, S_n , of the domains on which R is defined.
5. The significance of each column is partially conveyed by labelling it with the name of the corresponding domains. The example in Figure 1 illustrates a relation of degree 4, called "position report," which reflects the location and status of a unit in a task force commander.

POSITION REPORT	(Identification)	Location in LAT-LONG	Status	Time of Report
	1	63-50	10	0900
	5	67-71	2	1400

Figure 1—A relation of degree 4.

A relation whose domains are all simple can be represented in storage by a two-dimensional column-homogeneous array of the kind just discussed. Some more complicated data structure is necessary for a relation with one or more non-simple domains.

Consider, for example, the collection of relations exhibited in Figure 2. "Signature" is a non-simple domain of the relation "sensor report." The tree in Figure 2 shows these interrelationships of the non-simple domains.

These more complicated relationships can, however, be normalized to a multiple relation over simple domains. Figure 3 is an illustration of a normalized form of Figure 2.

The simplicity of the array representation which becomes feasible when all relations are cast in normalized format is not only an advantage for storage purposes but also for communication of data between systems which use widely different representations of the data. The communication form would be a suitable compressed version of the array representation and would have the following advantages:

1. It would be devoid of pointers (address-valued or displacement-valued).
2. It would avoid all dependence on hash addressing schemes.
3. It would contain no indices on ordering lists.

All inter-process communications could then be placed in either an informal narrative message format, or a formatted message in a normalized relational form. Similarly, all storage retrieval, computations and correlation of the received data can be shown to be defined in terms of operations on these relations and their domains.

For example, correlation involves a matching operation on the sensor report relations ID, signature and location domains. Thus, the basic C^2 processes of data dissemination, analysis, storage/retrieval and presentation can all be expressed in terms of message handling.

There are advantages to viewing the command control process in this manner. They are briefly summarized below:

1. The message-handling process is fairly well understood with an extensive theoretical foundation^{2,3,5,6} and supporting analyses and simulations. Platforms and nodes can be quantitatively characterized in terms of their message-handling capabilities.

SENSOR REPORT			
Sensor name	Report ID	Location (LOC)	Signature
			Waveform Class
			Pulse Repetition Interval (PRI)
Sensor report (Name, Report ID, Location, Signature)			
Signature (Waveform class, PRI)			

Figure 2—Unnormalized set.

Sensor report' (Name, Report ID, Loc)
Signature' (Report ID, Waveform class, PRI)

Figure 3—Normalized set.

2. The message distribution and routing process forms a *natural partitioning* upon which the interface and communication among command and control processes can be based.
3. Requiring the command and control process to communicate exclusively by means of man-readable message exchanges allows for *ease of human intervention and take-over* of any command and control process. Of course, a degradation in processing time would be experienced under manual take-over.
4. Inter-connecting command and control processes exclusively by means of asynchronous variable length message exchanges simplifies the interfacing among processes, allowing for a generalized approach to process interfacing which does not distinguish whether the interfacing processes are program modules on the same processor, on different processors in the same local node, or are processors remote from one another. This results in a *design flexibility* which can easily accommodate the addition and deletion of processors, new functions and capabilities, and which permits the dynamic remoting of functions. This enhances the practicality of implementing a survivable nodeless reconfigurable system. It also serves to isolate and, thus, insulate the main portion of the application design from the particulars of any specific message format or structure.
5. The asynchronous, variable length nature of the message exchanges allows the system to work independently of the reporting rates and the available communications bandwidth. This allows one to reduce the reporting rate and/or the message content in response to decreases in available system processing capacity and in available communications capacity. Thus, a loss or denial of facilities (processing or communications) and/or a manual override can be accommodated with a *graceful system* degradation rather than an abrupt loss of capability.
6. Restricting the communication between processes in a distributed processor network to a message exchange format should *ease the control and scheduling* of these processes. Several systems have already been built and operate on these principles, the ARPANET and its associated TENEX services being a notable example. Recent research trends in distributed processor network operating systems also appear to support these notions.⁷
7. By couching the command and control process in message-handling terms, there is the *potential for capitalization* of the software and concepts available from such rapidly growing *commercial application areas* as word processing, office automation and electronic mail.

On the negative side, there will be, of course, a loss of computer efficiency resulting from the requirement that all process-to-process communication be in man-readable message form and not through such mechanisms as common addressable blocks of memory or other such more tightly coupled program module inter-communication. But available computer processing capacity and memory is now a relatively abundant commodity, and to achieve computer efficiency at the sacrifice of other now scarcer resources such as manpower is the last thing we should do.

It should be noted, however, that the restriction of process-to-process communications via man-readable message format should not be misconstrued as prohibiting the compression and compaction of this message prior to its physical transmission over the communications media and its subsequent decoding and decompaction upon its reception at the other end. This operation is transparent to the process involved, is conservative of communications bandwidth, which is a relatively scarce resource, and is an acceptable, indeed, a preferred approach.

Such a model of the command and control process encourages the ease with which voice and data can be integrated. In this model, voice and data are recognized as merely two different medias to support Command and Control message (information) exchange and manipulation. Whereas voice requires greater bandwidth and exhibits less efficient information compression, it has the advantages of allowing for a rather informal, highly interactive and natural means for human communication. Data, on the other hand, exhibits more efficient use of communications bandwidth, and is a more natural medium for computer entry and for message storage, retrieval and manipulation. Both media are needed, and in fact, the employment of one over the other is highly dependent upon the situation. If the situation calls for complex team problem-solving, or is highly manual, informal voice is preferred. If the situation stresses conservation of bandwidth or communication with a highly automated weapon system, then data is preferred. Since the military situation is highly dynamic where systems fail, are jammed and destroyed, the ability for easy and effective control and reallocation of the voice/data mix is desirable. A distributed switching network over which voice and data are integrated would greatly facilitate and enhance such a capability. In addition, there is a definite advantage performance-wise in such an integration of voice and data. This is addressed in the following sections.

PROBLEM FORMULATION

The basic issue to be addressed can be reduced to the following question:

Given a fixed available bandwidth, is it preferable (e.g. more efficient) to (1) Allocate this bandwidth among two or more radios on a relatively fixed basis with voice traffic restricted to some pre-selected portion of the total number of available radios and data traffic restricted to the remaining portion? (2) Share the entire available spectrum among both

types of services, voice and data, on a totally demand basis? or (3) Allocate along some hybrid scheme (partial dedicated, partial shared on a demand basis)?

Furthermore, for the demand assignment model, is it preferable to allocate on a voice silence, call-by-call, message-by-message basis or on a less dynamic reservation type of basis?

The problem can be analyzed by studying the two statistical models illustrated in Figure 4 where λ_1 equals the average number of voice calls per second (the voice arrival rate), λ_2 equals the average number of data messages or packets sent per second, and μ_1, μ_2 represent the average service rates associated with a single voice-equivalent channel, and n and m refer to the number of voice-equivalent channels available for voice and data respectively. The arrivals and servicing of the calls and data are assumed to be probabilistic (random variables) and not to be fixed constants. The service rate for voice is inversely proportional to the average length of a phone conversation, or speaking period (depends on integration approach). The service rate for data is directly related to the bandwidth (capacity) of the voice channel and inversely related to the message length.

Voice calls and data messages arriving at a faster rate than they can be serviced will experience occasions when there is no available channel. In the case of voice, the attempted call is assumed aborted (or interrupted) and the caller is required to try again (or repeat). In the case of data, the data message is assumed to be stored or queued in a waiting state until a channel that can service it is freed up.

Case 1 is rather self-explanatory and straightforward. Voice traffic of average arrival rate λ_1 and service rate ($1/\text{average holding time}$) μ_1 is allocated n voice-equivalent channels.

Accordingly, the grade of service (GOS) or the probability of a lost call is given by the Erlang B distribution; equation 1 of Figure 4 where:

$$B\left(\frac{n, \lambda_1}{\mu_1}\right) = \frac{\left(\frac{\lambda_1}{\mu_1}\right)^n}{\sum_{x=0}^n \frac{\left(\frac{\lambda_1}{\mu_1}\right)^x}{x!}} \quad (5)$$

Data traffic of average arrival rate λ_2 is allocated m voice-equivalent channels. The service rate is equal to the channel bandwidth (measured in bits-per-second) divided by the average message length (measured in bits). Accordingly, the average delay (seconds) is given by Equation 2 in Figure 4.

Case 2 models an integrated voice/data system as a single server system which provides real-time, uninterrupted transmission service for voice, and data transmission service with a lower preemptable priority. Figure 4 illustrates this model. Assumptions for this model are:

1. Voice signals and data packets arrive in a Poisson fashion with mean arrival rates λ_1 , and λ_2 respectively.
2. No queueing is permitted for voice because of its real-time nature (i.e., as soon as a voice signal arrives, it

preempts the use of a server for its immediate transmission).

3. A data packet which was being served for its transmission and is preempted by the arrival of the voice signal will stand temporarily in front of the queue station of data packets until it is re-served (the whole packet is retransmitted). An exponentially distributed service time with a mean service time of $1/\mu_2$ is assumed.
4. Service time for the voice signal is also exponentially distributed with mean service time of $1/\mu_1$ seconds. In this case, the mean service time is not necessarily equal to the mean call holding time. It can be considerably less if a call is broken-up into talking and silent periods and the data packets sent during silent periods.

This model was analyzed in Reference 2 and the results are shown in Figure 4 with Equations 3 for the voice grade of service, and 4 for the average data packet delay. Because voice is treated as a preemptive priority class of arrivals, it is assumed to be represented by the same Erlang B distribution, Equation 5, used for Case 1.

Using Equations 1 through 5, Tables I-IV were computed.

RESULTS

From Tables I-IV it can be seen that it is generally more efficient for voice and data traffic to share voice-equivalent channels on a statistical contention basis than for voice and data to operate over separate dedicated data link voice-equivalent channels.

More specifically, for example, two voice-equivalent channels available for dynamic sharing by both voice and data on a statistical contention basis (with voice served on a preemptive priority basis) results in the following performance enhancements:

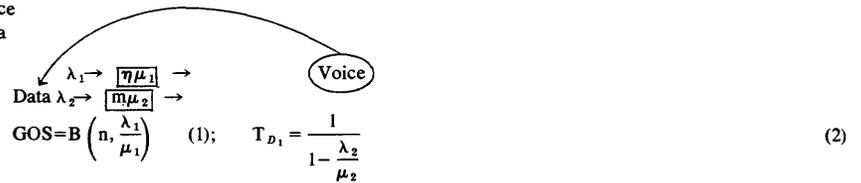
For a voice traffic intensity equal to or less than .1 second of voiced periods for every available second of channel time (.1 erlangs), the "grade of service" improves from .09 to .005, and the data traffic responsiveness is enhanced over the dedicated or non-integrated channel scheme up to data traffic utilizations (throughputs) in excess of 70 percent.

If the voice traffic density increases to $\frac{1}{2}$ second of voiced periods for every available second of channel time (.5 erlangs), an improvement in voice "grade of service" (probability of a blocked call) from .34 to .08 results from this

TABLE I.—Data Packet Delay—Separate Channel

$T_{D_1} = \frac{1}{1 - \sigma_2} = \text{Average Time Delay} \quad (6)$					
T_{D_1}	$\rho_2 = \frac{\lambda_2}{\mu_2}$				
	.1	.3	.5	.7	.9
1.11	1.43	2.0	3.33	10	

Case 1 - Separate radio channels: one dedicated to voice
one dedicated to data



Case 2 - Single radio channel servicing both voice and data on demand

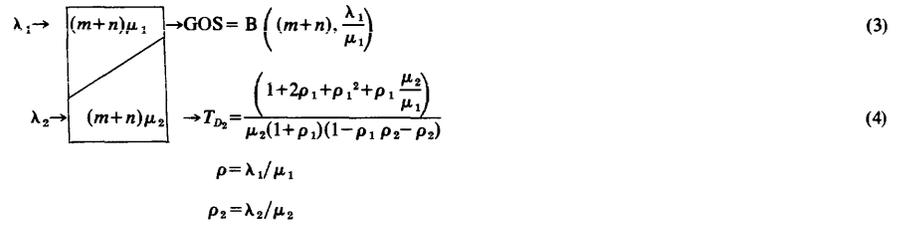


Figure 4—Models of voice/data integration.

TABLE II.—Data Packet Delay-Shared Channel

$$T_{D2} = \frac{1 + 2\rho_1 + \rho_1^2 + \rho_1 \frac{\mu_2}{\mu_1}}{\mu_2(1 + \rho_1)(1 - \rho_1\rho_2 - \rho_2)} \quad (7)$$

2 sec average call holding time 200 bits average data message length 16 kbps channel capacity ($\mu_2=80$) ($\mu_1=.5$)					2 sec average call holding time 2000 bits average data message length 16 kbps channel capacity ($\mu_2=8$) ($\mu_1=.5$)						
ρ_1	ρ_2	.1	.3	.5	.7	ρ_1	ρ_2	.1	.3	.5	.7
.1	.1	.22	.55	.81	1.02	.1	.1	.36	.72	1.00	1.25
.1	.3	.29	.78	1.25	1.72	.1	.3	.48	1.02	1.55	2.11
.1	.5	.43	1.37	2.74	5.63	.1	.5	.71	1.78	3.42	6.91
.1	.7	.85	5.31	-	-	.1	.7	1.39	6.93	-	-
2 sec average call holding time 200 bits average data message length 2.4 kbps channel capacity ($\mu_2=12$) ($\mu_1=.5$)					2 sec average call holding time 2000 bits average message length 2.4 kbps channel capacity ($\mu_2=1.2$) ($\mu_1=.5$)						
ρ_1	ρ_2	.1	.3	.5	.7	ρ_1	ρ_2	.1	.3	.5	.7
.1	.1	.31	.66	.93	1.16	.1	.1	1.23	1.78	2.25	2.70
.1	.3	.41	.93	1.44	1.97	.1	.3	1.64	2.53	3.48	4.57
.1	.5	.61	1.63	3.17	6.43	.1	.5	2.44	4.41	7.67	14.93
.1	.7	1.19	6.33	-	-	.1	.7	4.78	17.17	-	-

TABLE III.—Data Packet Delay-Shared Channel

2 min. call holding time 200 bits message length 16 kbps channel capacity					2 min. call holding time 200 bits message length 16 kbps channel capacity						
$(\mu_2=80)$ $(\mu_1=.009)$					$(\mu_2=8)$ $(\mu_1=.009)$						
ρ_1	ρ_2	.1	.3	.5	.7	ρ_1	ρ_2	.1	.3	.5	.7
.1		11.36	29.49	43.60	55.15	.1		11.50	29.66	43.79	55.38
.3		15.10	42.06	67.37	93.41	.3		15.28	42.30	67.68	93.80
.5		22.48	73.31	148.23	305.15	.5		22.75	73.72	148.90	306.43
.7		43.98	285.08	-	-	.7		44.52	286.71	-	-

2 min. call holding time 200 bits message length 2400 bps channel capacity					2 min. call holding time 2000 bits message length 2400 bps channel capacity						
$(\mu_2=12)$ $(\mu_1=.009)$					$(\mu_2=1.2)$ $(\mu_1=.009)$						
ρ_1	ρ_2	.1	.3	.5	.7	ρ_1	ρ_2	.1	.3	.5	.7
.1		11.45	29.60	43.72	55.29	.1		12.38	30.72	45.04	56.83
.3		15.21	42.21	67.57	93.66	.3		16.44	43.81	69.61	96.26
.5		22.65	73.57	148.65	305.96	.5		24.48	76.36	158.15	314.46
.7		44.32	286.10	-	-	.7		47.90	296.94	-	-

voice/data integration and an accompanying increase (enhancement) in data traffic responsiveness is still experienced if the data traffic throughput (for this example) does not exceed 30 percent channel utilization, or:

- a. 24 message packets/sec per channel for 200-bit average message length, and 16 kbps voice-equivalent channel.
- b. 1.2 message packets/sec per channel for 200-bit average message length and 2.4 kbps voice-equivalent channel.

TABLE IV.—Probability of Blocked Voice Call

$$B(n, \rho_1) = \frac{\frac{\rho_1^n}{n!}}{\sum_{x=0}^n \frac{\rho_1^x}{x!}} \quad (8)$$

ρ_1 = traffic intensity in erlangs
 n = # trunks

# Trunks n	ρ_1				
	.1	.3	.5	.7	.9
1	.09	.24	.34	.41	.49
2	.005	.03	.08	.13	.17

The contention scheme just analyzed requires the capability for data to be interleaved with voice by transmitting only over quiet (unvoiced) periods in a voice conversation. If the interleaving is only done on a completed call basis, then Table III applies, call-holding times on the order of 1/2 hour result, and the resulting data packet delay becomes prohibitively large.

The sensitivity of achievable data traffic throughput to data packet length is illustrated by noting that there is a tremendous loss in the efficiency (throughput) of the data traffic as one increases the length of the packet to be interleaved with voice signals over a common channel. Specifically in Table II (2 sec average voice holding time) for a voice signal of .5 erlang traffic intensity over 16 kbps voice-equivalent channel, the data traffic throughput is 2.4 messages/sec for 200-bit packets, and only .8 messages/sec for 2000-bit packets. This represents a 300 percent improvement in data throughput efficiency when one interleaves on a 200-bit basis as opposed to a 2000-bit basis.

Based on this sensitivity of data traffic throughput to data packet length, it is clear that if voice/data integration is desirable (which it is) then word or short-packet (200 bit or less) interleaving is preferred and this interleaving should be done on a voice silent period basis rather than on a completed call basis.

The above analysis was based upon a scheme which as-

sumed that if a voice signal occurs during a data transmission, the data transmission will be preempted by the voice signal and queued for later retransmittal. This was required because of the real-time nature of voice; voice signals cannot be randomly stored and forwarded without degrading its intelligibility.

For some military applications, it is conceivable that there will be classes of message traffic for which it is preferable to interrupt a voice conversation and to allow for a real-time message delivery (no or negligible queue time). Such a capability can be easily implemented. Although its impact on total system performance is not completely analyzed in this paper, it is clear that if this preferred class of messages appears sufficiently infrequently, this special message class can be accommodated with preferential preemption treatment, and performance advantages will still result over a wide practical range of traffic mixes for both voice and for lower-priority message traffic. For example, the impact on the performance of voice and lower priority messages, when practicing preferential preemption treatment for a select class of messages that represent less than one percent of the total channel capacity, can be approximated by increasing the traffic intensity, ρ_1 , in Equations 4 and 5 by 1 percent, i.e., .1 worth of traffic will look like $\rho_1 = .11$ worth of traffic.

If one wishes, a hybrid scheme could be adopted where a thin-line capability is reserved for a special class of data and/or voice and where the remainder supports an integrated voice-data operation. The reserved channels can be dynamically increased or decreased in response to channel fluctuations in such a way as to guarantee a minimum acceptable threshold of performance. One such scheme, analyzed in Reference 3 for a circuit-switched network, is reformulated in the context of the tactical data link situation, and is now summarized.

RESERVATION SCHEME

It is desired to guarantee that for a select class of data messages, their delivery time will not exceed a maximum allowable value. One way this guaranteed performance can be achieved is by the originating data terminal transmitting this select class of data messages as packets, over pseudo-dedicated channels, previously reserved and connected to the destination data terminal processor. The originating data terminal may reject messages/packets over a critical length as the pseudo-dedicated line utilization reaches a pre-determined threshold. Upon reaching this threshold, the data terminal must reserve additional voice-equivalent channels on a multi-channel basis if the guaranteed performance is to be maintained.

Let us now determine how one may compute the value of the threshold at which the data terminal must reserve additional voice-equivalent channels in order to keep the maximum message delay below some allowable value. This threshold is related to the maximum acceptable delivery time as expressed by Equation 9.

$$T_D = (Q_I + 1 + Q_0)t_{tr} \quad (9)$$

where T_D = maximum acceptable delivery time
 Q_I = input queue to originating data terminal expressed in packets
 Q_0 = output queue to the destination data terminal expressed in packets
 t_{tr} = packet transmission time in seconds/packet
 Equation 9 can be rewritten as

$$-1 + \frac{T_D}{t_{tr}} = Q_C \quad (10)$$

where

$$Q_C = Q_I + Q_0. \quad (11)$$

If the average packet length is ℓ bits, and a single voice-equivalent channel represents b_0 kbps, and if we reserve n voice-equivalent channels at a time, and if m pseudo-dedicated lines have already been reserved between the originating and destination data terminals, then the average packet transmission time t_{tr} , is at that instant of time

$$t_{tr} = \frac{\ell}{mnb_0} \quad (12)$$

Substituting Equations 11 and 12 into Equation 9, we obtain the relationship that

$$\frac{mnb_0 T_D}{\ell} - 1 = Q_C. \quad (13)$$

This tells us that if:

$$Q_T = Q_C - t_c(\lambda_0 - \mu_0) \quad (14)$$

where λ_0 = maximum arrival rate
 μ_0 = maximum service rate
 Q_T = threshold queue
 t_c = connect time

then combining equations 13 and 14, we obtain

$$Q_T = \frac{mnb_0}{\ell} T_D - t_c(\lambda_0 - \mu_0) \quad (15)$$

The maximum resulting queue for which buffer space must be provided is Q_{MAX} , where

$$Q_T + t_c(\lambda_0 - \mu_0) \leq Q_{MAX} \quad (16)$$

The basic theory underlying the hybrid network then is to reserve an additional n voice-equivalent channels when the queue approaches some critical value, Q_T .

As an example, consider that the output and input queues are equal.

$$Q_0 = Q_I,$$

then from Equation 9

$$T_D = (2Q_0 + 1)t_{tr} \quad (17)$$

Then for the maximum capacity queue, $Q_0 = Q_{MAX}$, and

$$\frac{T_D/t_{tr}}{2} - 1 = Q_{MAX} \quad (18)$$

From Equation 12, $t_{tr} = 0.125$, for $\ell = 2000$ bits average message length, $b_0 = 16$ kbps, $n = 1$, and $m = 1$.

From Equation 18, a maximum queue of seven packets ($Q_{MAX} \leq 7$ packets) results for a $t_{tr} = 0.125$ and a maximum allowable T_D of 2 seconds. This technique for maintaining a maximum allowable T_D is to monitor the queues, Q_0 and Q_I . As $Q_0 + Q_I$ approaches Q_T , one reserves more channels, thus increasing the capacity and reducing T_D .

The general conclusions to be made from the analysis presented in this section is that this reservation scheme is practical over a wide range of realistic values. More specifically, for messages of length 2000 bits or less, it is practical to maintain a delivery time under a pre-established allowable maximum of two seconds by adding more channel capacity on a reserve basis when a queue threshold less than seven packets, or 14,000 bits, is reached.

REFERENCES

1. Fawcette, James, "Mystic Link Revealed," *MSN*, Sept. 1977, pp. 81-94.
2. Wang, Jin-Tuu, and Ming T. Liu, "Analysis and Simulation of the Mixed Voice/Data Transmission System (MVD) for Computer Communication," *NTC 1976*, pp. 42.3-1 - 42.3-5.
3. Ricci, Fred, and Daniel Schutzer, "Design and Implementation of PAC-SWITCH Network," *NTC 1976*, pp. 42.6-1 - 42.6-5.
4. *Channel Simulation of Tactical Support Center Information Exchange System (TSCIXS)*, NOSC Publication #FSCS 207-7, July 15, 1977.
5. Cooper, Robert B., *Introduction to Queuing Systems*, MacMillan Co., New York, 1972.
6. Lee, C. Y., Analysis of Switching Networks, *BSTJ*, Vol. 34, No. 6, Nov. 1955, pp. 1287-1315.
7. May, M. D., R. J. B. Taylor and C. Whitby-Stevens, "EPL—An Experimental Language for Distributed Computing," *Trends and Applications: 1978, Distributed Processing*, pp. 69-71.

The exploratory system control model multi-loop network

by DANIEL J. PAULISH

Burroughs Corporation
Paoli, Pennsylvania

PURPOSE

The Exploratory Systems Control Model (ESM) Multiloop Network consists of the original three-loop ESM network delivered in 1977 and the fourth Exploratory Systems Control Model Development (ESMD) loop delivered in 1978. The ESM also includes a fifth loop supplied under the Modular System Control Development Model (MSCDM) project. The ESM provides a flexible tool for simulating and comparing a wide range of system control architectures and their related procedures and protocols. The ESM has been designed to model the class of the system control architectures that have the characteristics of decentralized operation, modularity, easy modification and upgrade capability, high reliability, high survivability and fail-soft operation.

BACKGROUND

The Defense Communications System (DCS) is a global, multiple-user system composed of leased and government-owned transmission media, relay stations and switching centers deployed in support of the National Command Authorities and the services, including command and control, intelligence and early-warning, as well as administrative and logistical communications. In order to increase the reliability and availability of these DCS services, it is essential to improve the responsiveness and robustness of the System Control (SYSCON) process as much as possible. This requirement demands a DCS SYSCON subsystem possessing such design features as modularity and "fail-soft" operation. Modularity implies a subsystem that is capable of being upgraded, modified and reconfigured easily, and "fail-soft" implies a subsystem that tolerates partial failures, yet is relatively immune to total collapse. To afford these capabilities, the future DCS SYSCON subsystem is expected to consist of many semi-autonomous, mutually-supportive, geographically-dispersed control centers.

During FY 75, Burroughs Corporation began development of an Exploratory System Control Model (ESM) which capitalized upon the inherent flexibility of multiple, interconnected data transmission rings and microprocessor-based host/ring interface nodes to provide an initial capability for experimental validation of various candidate SYSCON subsystem architectures characterized by distributed control

and graceful degradation under stress. This capability to model apparently dissimilar architectures is a consequence of the universal physical connectivity provided by the ring structure coupled with flexible protocols that permit definition of different logical connectivities through selective routing of transmitted data.

In the broader context of the DCS SYSCON Program, the longer-term joint purpose of this effort and the separate-but-related "Modular System Control Architecture Study and Feasibility Development Model" is to provide DCEC with the necessary integrated means to evaluate through hybrid simulation a variety of candidate SYSCON subsystem the architecture(s) thereby identified as being suitable for implementation. The technical and performance information obtained from the unified hybrid simulation model will ultimately be used in the preparation of performance specifications for the future DCS SYSCON subsystem.

LOOP OR RING COMMUNICATIONS SYSTEMS

General operation

A communications loop is a closed, ring-connected set of nodes providing data flow unidirectionally from one node to the next. Each link between nodes is a single twisted pair of wires carrying a serial data stream in a self-clocking code. Full connectivity is achieved by associating a destination address with each packet of data carried on the loop. A node to whom a packet of data is not addressed acts simply as a "delayed repeater," having no effect on that data other than introducing some delay. The concept of a data exchange loop has been described extensively in the literature of computer communications by Reames,¹ Jafari² et al. Loops may be distributed such that each node contains its own power supply and cabinet and is located near the equipment it interfaces or locally such that all nodes are connected within a single cabinet with cable connections to interfaced equipment.

A functional block diagram of a communication loop node is given in Figure 1. The Loop Interface Unit (LIU) is responsible for reading data addressed to the node and writing data on the loop. The Control and Interface Processor (CIP) is a microcomputer that provides a data communications interface to the external device. The memory is used

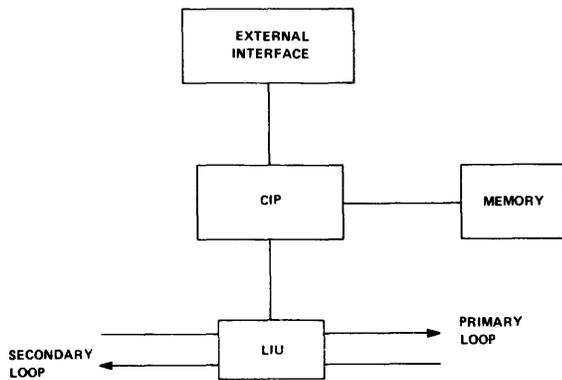


Figure 1—Communication loop node.

for program storage, routing tables and intransit queue storage. The external interface provides a hardware connection to the external equipment to be connected to the loop.

Modularity-adaptability features

The basic Burroughs loop node is a module made up of the LIU, microprocessor CIP, memory and external interface. The nodes are identical except for the external interface and the external device interface software used to handle the protocol between the microprocessor and the external device. In the ESM external devices include processors (PDP11/40, PDP11/70, B776), terminals (TD802, TD832), gateways (between the multiple rings) and data communication interfaces (SDLC, AUTODIN II, TCCF). The nodal external device interface software provides code conversion, flow control, intransit queueing, logical attachment capability and emulation of various communications protocols for the devices. The interfacing capability of the nodal modules provides communication capability between devices in the heterogeneous system.

When a module fails, the loop will recognize the failure and cut the failed module out of the system by forced loop-back from the module's nearest neighbors. The module may then be replaced and the loop will return to normal operation. In the meantime, the other modules will still be in operation so that degradation will be graceful in that only the operation of the failed module will have been lost.

Loop throughput capability

Loop throughput (total number of message units that can be sent over the loop per unit time without undue message delay to the receiving modules) is a function of line speed, loop discipline and the definition of "undue" message delay. Various loop disciplines have been developed and compared.

The Newhall loop which uses a special control packet called a Write Token can transmit only one message on the loop at a time and has the lowest throughput but has the advantage of simplicity and cannot be clogged by misdi-

rected messages. It also shows some advantage in ease of detection and deletion of certain types of faults.

The Pierce loop which uses fixed size slots in which data packets can be placed can transmit multiple messages, but the small fixed packet size causes greater overhead than in the DLCN loop.¹ The DLCN loop uses queues within each LIU that can expand or contract to hold upstream messages in temporary storage. This allows packets of variable size to be transmitted and allows multiple transmissions. Loop clogging is possible in both cases, however, and special means must be employed to "declog" the loops under certain error conditions. Loop clogging is the deadlock situation when packets cannot be written onto the loop until previously written packets are removed.

The Jafari loop² is a double loop, one used for control and the other for data. The data loop is segmented such that a switched point-to-point circuit is set up when requests for communication are issued on the control loop.

The ESM uses a Newhall protocol at a loop frequency of one mega-baud. Simulation studies and queueing analyses³ indicate that this loop can support a throughput in excess of 750K baud without undue delay. The Pierce, DLCN and Jafari throughputs can be even higher, due to simultaneous conversations.

Suppose the average node writes 15 packets of 2000 bits each per second for a total of 30,000 bits/second. At that rate a Newhall loop can support 25 nodes. The worst-case time that a node will have to wait for a write token is given by

$$T_{WT} = \frac{MP}{C_L} \quad (\text{Eq. 1})$$

where M is the number of nodes=25, P is the packet size—2000 bits, and C_L is the loop frequency of $1M$ bits/sec. Thus T_{WT} is 50 msec. The average wait for a write token is given by

$$T = \frac{\rho}{2} T_{WT} \quad (\text{Eq.2})$$

where ρ is the loop utilization=0.75 for our example, thus $T=19$ msec.

Multiple loops—addressing schemes

The ESM system has proved the capability for providing multiple loop systems and has acted as a vehicle for testing multiple loop addressing schemes. Figure 2 exemplifies a multiple loop system. Three loops are shown connected via gateway nodes. Gateway 2 of Loop 1 connects to Gateway 1 of Loop 2 via a hard-wire connection independent of the loops. Similarly, Loop 1 connects to Loop 3 and Loop 3 connects to Loop 2 via gateways. Each loop is independent of the other loops.

The small boxes are nodes and the numbers within the boxes represent the "functional address" of the node. The functional address (FAD) is the local address unique within the loop. In addition, each node has a "logical identifier" (LID) unique within the system.

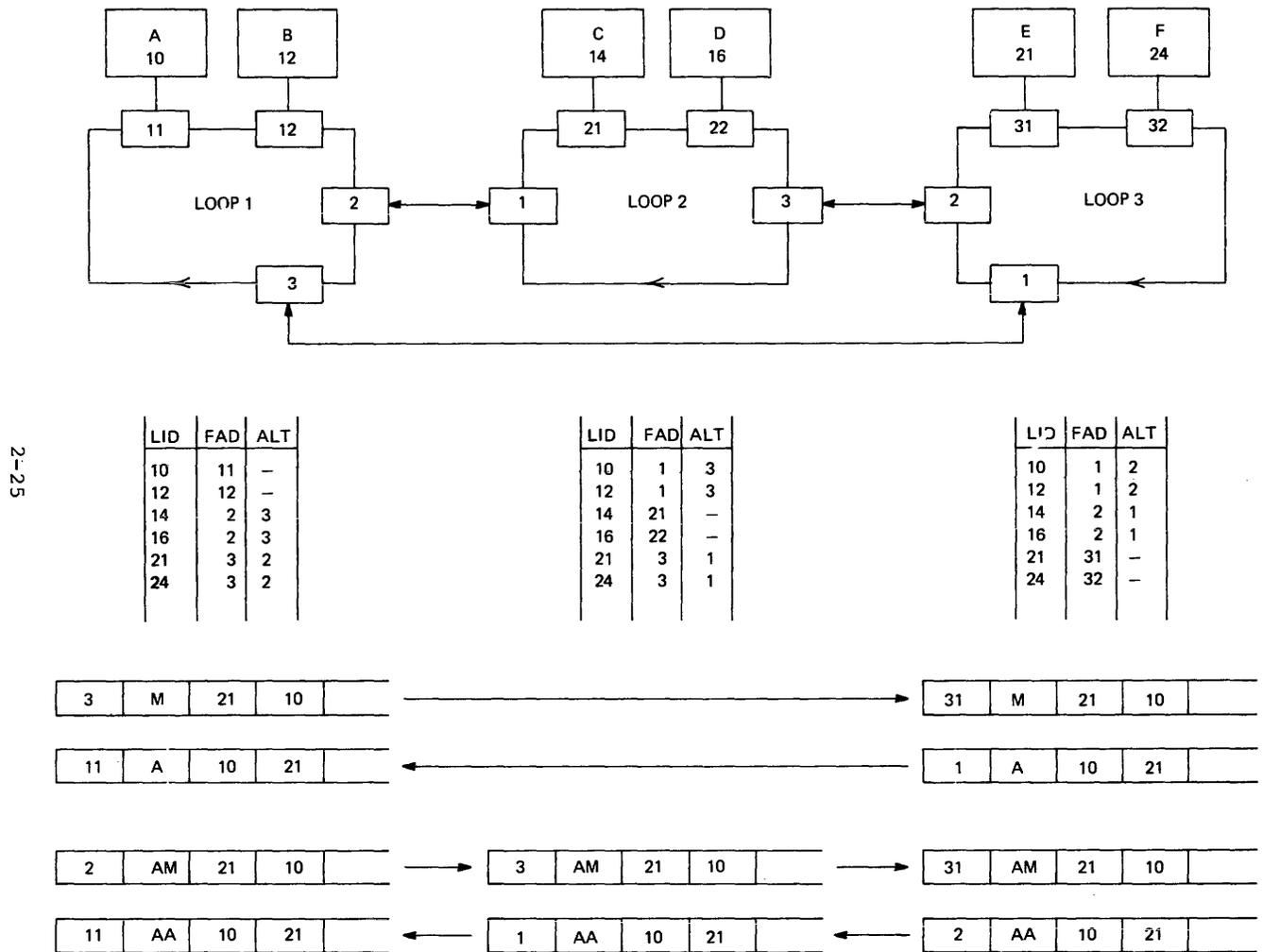


Figure 2—Indirect method of addressing.

An example of how alternate routing is implemented with a multi-loop architecture using indirect addressing is given in Figure 2. Let us assume that Host Processor A on Loop 1 wishes to send a message to System Process 21. Host A sends a packet to its CIP with 21 as the destination LID and 10 as its source LID. The CIP formats a packet using an FAD or loop address equal to 3. The packet is sent out onto the loop, bypasses Nodes 12 and 2 and is read by Gateway Node 3. Gateway Node 3 sends the information part of the packet across the 1-3 link. Gateway 1 in Loop 3 formats a packet having Loop Address 31. The packet bypasses Node 2 and Node 31 reads the packet. An ACK packet is sent out on the loop using LID 10, and the packet is linked to the input queue for deliverance to Host A.

If Node 11 had not received an ACK message after a specified number of retransmissions, it would utilize alternate routing. It would do this by marking the packet indicating that alternate routing was used and changing the loop read address (FAD) from 3 to 2. Gateway Node 2 in Loop 1 would read the packet and send it across the 1-2 link.

Gateway Node 1 in Loop 2 would use an FAD of 3. The packet would bypass Nodes 21 and 22 and be read by gateway Node 3. The packet would be sent across the 2-3 link and gateway Node 2 in Loop 3 would use an FAD of 31. The acknowledgment message would be sent via the alternate route.

Node 11 would also report to one or more network control processors who could remove the 1-3 link from service for repair. This would involve sending special broadcast control packets to Loops 1 and 3 so that Link 1-3 would not be used.

The above method of indirect addressing can be used for resource allocation such that processes could be moved around the network so that spare or less utilized processors can be utilized. For example, let us say that Host E is brought down for service and thus Process 21 is to be moved to another processor. Let us say that it is determined (possibly by some bid-quotation scheme) that Host D of Loop 2 is to handle Process 21. In order to move the process, control packets would be broadcast in each loop.

ESM/ESMD IMPLEMENTATION

System elements and connectivity

The ESM is a communications system used to interconnect devices (e.g., terminals, host processors, data communications lines) so that each device can interface with any other device for information transfer. To accomplish this, each ring is supplied with nodes that act as interfaces from ring to device and from ring to ring. The ring-to-ring nodes are called "gateway" nodes. Each node is the same physically as any other node except for a small amount of special separable hardware for each type of node. The major difference between nodes is in the software of the nodes. The nodes provide all the necessary communications functions of queueing, parity checking, ACKing, NAKing, retransmitting, alternate routing, etc. The hosts and terminals need only supply the data processing functions and need not be concerned with the communications functions.

The gateway node interchanges are via cables in the ESM configuration, but in principle can be via any communications medium such as telephone, microwave relay, optical transmission or satellite relay.

The terms "loop" and "ring" are interchangeable. Each loop is housed in a separate cabinet in this implementation, but this is not a necessity. A loop could, as easily, extend throughout a building or facility.

The ESM Multiloop Network is illustrated in Figure 3. Loops 1, 2 and 3 were delivered in 1977 as part of the ESM Contract.⁴ Loop 4 was delivered as part of the ESMD Contract in 1978.⁵ Loop 5 was delivered as part of the MSCDM Contract in 1979.

Features of the ESM

The ESM is designed to be transparent to the user. Regardless of the CRT used and the host on which a particular activity takes place, the activity will take place for the CRT that calls for it. When a message is transmitted from a CRT, suitable control bytes are added to the message by the CRT node and directed to the node of a nearby host. When the host receives the message, it will either handle the message completely if it can or it will pass it on to another host, via the ESM, for cooperative handling of the message. This is done under program control using the content of the CRT message and the added control bytes. The CRT will then receive a response from one of the hosts. A CRT can "ATTACH" itself to any node in the network via user command.⁶

Responses will generally be part of the "user language" which is designed to provide directions for further dialog as well as replies to previous messages. The language is designed to be modular so that it can be easily updated and

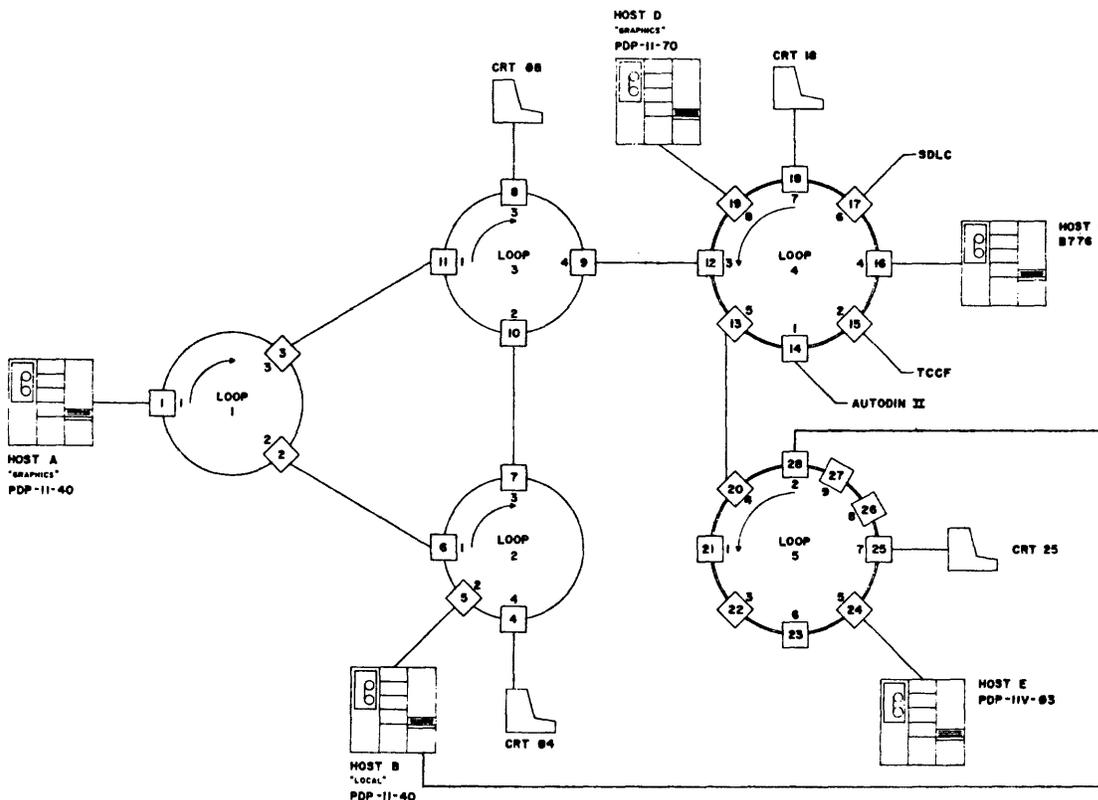


Figure 3—ESM multi-loop network.

enhanced. In addition CRTs can communicate directly with the operating system of the particular processor as if it were a local terminal.

Messages are sent in the form of packets of length not greater than 256 bytes. As each packet is sent, the sending node holds it for acknowledgment (ACK) from the receiving node. When an ACK is received by the sending node, it frees the packet space.

If a non-acknowledgment (NAK) is received, the message is resent or sent by an alternate route. Absence of an ACK or NAK after a timeout period is considered to be a NAK. After a suitable number of resends without an ACK, the message may be reported "not sent." Nodes automatically provide input and output queueing for the external device. Sufficient extra memory space is provided in each node to permit receipt of system control commands from the loop and to act on these commands. This is done to prevent a deadly embrace condition within the node. If the input queue (from the loop to the external device) is full, new input messages are rejected. Room always exists for the receipt of ACKs and NAKs and other control messages. These are acted upon with dispatch so that they do not reside in data memory for a long period. If the output queue is full, the external device is prevented from sending to the node.

The loop protocols are designed to be non-blocking and self-polling. Each node in the loop has its turn to write onto the loop and if any noise exists on the loop from prior transmissions, it is overwritten by the new transmission. Nodes share the polling activity and any loss of polling is restarted automatically.

ACKs and NAKs are generated by end-user nodes when they receive packets. Each packet is tested against cyclic redundancy bytes in the packet. A good check results in an ACK and a bad check results in a NAK.

Examples of use

The ESM Multiloop Network is part of DCA's Hybrid Simulation Facility (HSF). The ESM provides DCA with a System Control Simulation Facility. The uses of the system are to be outlined. The various applications are in different stages of development; some of the system uses are a direct result of the original implementation, others require additional modeling application and demonstration software.

User language

The User Language provides the human interface to the system and demonstrates many of its modeling capabilities. The User Language is an application program running on the various Host processors in the network. All loop connected terminals can communicate with the User Language on any processor.

The User Language consists of four major modes of operation. The first mode, CRT-to-CRT, provides users with the capability to send messages to each other. This simulates communication between System Controllers at different

sites who must talk to each other in order to isolate certain faults. Mode 2, System Inquiry, allows the user to examine the nodal configuration tables; the tables are monitored on disk on a host computer. Mode 3, System Control, allows the user to modify the nodal configuration tables; this feature provides the capability to model different network architectures. Mode 4 implements a distributed data base on the PDP 11/40s. The TOTAL Data Base Management System is used to distribute records of files on the two processors. The data base appears to the user to reside completely on one machine.

File transfer

A file transfer utility has been written to transfer files between host processors. The program allows peripheral sharing by providing a capability to send files to another machine's disk, printer, or tape. Files can be obtained from another machine's disk. In addition, terminals can be AT-TACHed to the various host processors in the system.

Fail-soft operation

The system is designed to tolerate partial failures. LIU failures result in automatic loop-back performed to remove the failure from the network. The failure is detected by the nodes and reported to a System Control Monitor node. Alternate routing is automatically performed when a node in another loop fails to ACK a packet after a specified number of retries. Failure to ACK a packet is reported to the Monitor node. Queue overflows are also reported to the Monitor; a queue overflow results from the failure of an external device to respond to the node.

Security

A demonstration of the use of a Security Monitor node is performed using Loop 4. A CRT-B776 conversation is monitored by a PDP 11/70 to detect an invalid password. The node connected to the PDP 11/70 is commanded such that it does a non-destructive read on packets addressed to the B776. Thus the data sent by the CRT is read by both the B776 and the PDP 11/70. The PDP 11/70 monitors the CRT-B776 conversation. When a bad password is detected, the Security node sends control packets to the nodes directly upstream and downstream from the CRT node to perform loop-around, resulting in the CRT node being removed from the network.

Network architecture

The ESM can be used to study System Control network architectures. Since the logical connectivity of the network is maintained by the modifiable nodal tables, the system can be used to model other network architectures. Network

control problems such as automatic channel reconfiguration can be studied.

Response time

Since each loop frequency is independently modifiable, response-time studies can be done on the system. The loop rates are modifiable via switches on the clock generator cards. In addition for Loops 4 and 5, an external clock generator can be connected to drive the loop.

Software development

The ESM can be used as a general software development facility. Since each loop-connected terminal can ATTACH to any processor, software can be written on a variety of machines with different operating systems and different language compilers. In addition, since files may be transferred between machines via the network, duplicate copies of files can be kept on different machines. Processors without certain resources (e.g. line printers) can utilize the resources of other machines via the network.

APPLICATION TO SYSTEM CONTROL

The ESM Multiloop Network will be used as a DCS System Control Simulation Facility. DCS System Control must accomplish the following functions⁷:

- *Network control*—Transmission and switched network configuration control, which includes network and extension supervision, reconstitution, restoral and satellite configuration control.
- *Traffic control*—Control of traffic routing and traffic flow.

- *Performance assessment* of the DCS and status monitoring of the DCS resources.
- *Technical control*—Includes quality assurance and monitoring, patching, testing, coordinating, restoring and reporting functions necessary for effective technical supervision and control over trunks and circuits traversing or terminating in a facility.

ACKNOWLEDGMENT

I wish to acknowledge that this work was supported in part by the Defense Communications Agency under Contracts DCA100-75-C-0054 and DCA100-76-0081. I also wish to acknowledge the contributions of personnel associated with the project at the System Control Group at the Defense Communications Engineering Center and the Advanced Development Organization of the Burroughs Corporation—in particular, the guidance given by D. Schutzer, O. Halpeny, J. Lynch and A. Meyerhoff.

REFERENCES

1. Reames, C. C., "System Design for the Distributed Loop Computer Network," Ph.D. Dissertation, The Ohio State Univ., Columbus, Ohio, March 1976.
2. Jafari, H., "A New Loop Structure for Distributed Microcomputing Systems," Ph.D. Dissertation, Oregon State Univ., December 1977.
3. Meyerhoff, A., D. Paulish, C. Tomlinson and T. Woodward, "Loop Communications Systems," *Burroughs Corp. Tech. Rept. TR77-3*, Paoli, PA, November 1977.
4. Burroughs Corp., "ESM User Manual," Doc. 66143-1, Paoli, PA, March 1977.
5. Burroughs Corp., "ESMD User Manual," Doc. 66146, Paoli, PA, March 1978.
6. Burroughs Corp., "Final Report for the ESMD," Doc. 64297, Paoli, PA, January 1978.
7. Fluk, M., "Technical Overview of the System Control Improvement Program," *DCEC Tech. Rept. TR4-78*, Reston, VA., May 1978.

Software reliability measures applied to system engineering

by JOHN D. MUSA

Bell Laboratories
Whippany, New Jersey

INTRODUCTION

Boehm, Brown, and Lipow¹ have characterized the multi-dimensional nature of software quality in terms of a hierarchy of attributes. One of the high-level attributes is reliability, which they define qualitatively as the satisfactory performance of intended functions. This definition may be refined to the quantitative statement "probability of failure-free operation in a specified environment for a specified time." A "failure" is an unacceptable departure of program operation from program requirements, where, as in the case of hardware, "unacceptable" must ultimately be defined by the user. The term "fault" will be used to indicate the program defect that causes the failure.

Several trends have recently combined to escalate the importance of quantitative software reliability measures:

1. The large and growing number of real-time and interactive systems has increased the operational and cost impacts of failures.
2. The increasing number, size, and complexity of computer networks and distributed processing systems have multiplied the risk and effects of failure.
3. The explosive growth of personal computing has created a demand for relatively foolproof software for unsophisticated users.

Measurement is seen to be important as soon as one recognizes that in software as in hardware there can be too much as well as too little reliability. Improvement of reliability, of course, costs money, and usually impacts development schedules and system performance (in the case of software, through increased memory, processing time, and peripherals requirements). The system engineer and the manager have to make design tradeoffs among the foregoing factors and it is best that this be done in quantitative terms. The need for a quantitative reliability measure continues throughout the development process, particularly during test, since reliability is a valuable indicator of system status. Finally, reliability or mean-time-to-failure (MTTF) is a useful metric for characterizing system operation and for controlling change during the maintenance phase. This paper will focus on the system engineering application, but it will

also touch on monitoring the system test phase and controlling change during maintenance.

EXECUTION TIME THEORY OF SOFTWARE RELIABILITY

Software failures are caused by design or coding faults, while the hardware failures dealt with by hardware reliability theory are caused by physical deterioration. However, software and hardware reliabilities are mathematically very similar. Thus they may be manipulated in similar fashion and they may be combined to yield system reliability.

The basic concept of the execution time theory^{2,3} is that execution (processor or CPU) time is the best practical measure for characterizing the failure-inducing stress placed on software. Execution time and calendar time can be related because the relationship between the two is characteristically paced at any given time by one of the resources failure identification (test team) personnel, failure correction (debugging) personnel, or computer time.

The execution time theory is based on assumptions that appear to be satisfactorily met by most executable programs and most development projects, assuming that testing is representative of the operational environment and that failures are observed.

A number of fundamental equations relating failures experienced, present MTTF, cumulative execution time, objective MTTF, failures to be experienced to reach the MTTF objective established for the project, and execution and calendar times required to meet the objective have been derived² and summarized.³

The equations mentioned above depend on four classes of parameters (described in detail in Reference 3, Section VI): program, planned, debug environment, and test environment. The two program parameters, the total failures expected during the maintained life of the software and the mean time to failure at the start of test, can be statistically reestimated (Reference 3, p. 436) as testing progresses.

A portable FORTRAN program^{4,5} has been developed to re-estimate the program parameters and compute status quantities significant to the manager. The program requires as input the execution time intervals between failures experienced, the MTTF objective, and the test environment

parameter. The debug environment and planned parameters are required only if it is desired to predict the test completion date.

The program computes confidence intervals for the quantities it estimates. The 75 percent confidence interval has been found to be the most useful; it represents a good compromise between high confidence and a narrow range of estimation. A sample report generated for an actual project is shown in Figure 1. The lower and upper confidence bounds are sandwiched around the "most likely" (maximum likelihood) estimates. Note that "999999" indicates "no upper limit." For the project illustrated on 10/6/77, the most likely present MTTF is 66.7 hr and the 75 percent confidence interval for present MTTF is 36.3 hr or greater. The most likely date on which the 5000 hr MTTF objective will be reached is 11/22/77 and the 75 percent confidence interval for reaching the objective is sometime between now (10/6/77) and 12/23/77.

SYSTEM ENGINEERING

The use of the execution time theory in system engineering can be illustrated by looking at the tradeoffs that can be made between MTTF, cost, and schedules. These tradeoffs are made for the system test phase of the project. It is assumed that MTTF improvement is obtained by more extensive testing, which of course affects costs and schedules. Costs and schedules for other phases are assumed to be constant. This assumption is reasonable; reliability improvement techniques such as structured programming, design reviews, etc. are commonly implemented on a "yes-no" basis dependent on their cost effectiveness. One does not ordinarily trade off the degree to which structured programming is employed with MTTF.

The procedures and formulas used for computing system test duration and cost will be described, since these two computations are central to the system studies to which the execution time theory of software reliability can be applied. An example of a system study will then be presented to suggest some of the kinds of questions that can be answered.

System test duration

Estimates of system test duration (excluding the test planning effort) may be made *before testing begins* as follows. The total inherent faults N_0 is estimated from data on faults per source instruction at the start of system test. The total expected failures M_0 may be equated to N_0 unless the ratio of faults corrected to failures detected departs appreciably from 1 (in this case, a correction must be made—see Reference 3, pp. 447-8). Data taken by the author and by Akiyama⁶ and Endres⁷ give a range of 3.36 to 7.98 faults per thousand source instructions for assembly language programs at the start of system test; the weighted (by number of instructions) mean is 5.43 faults per thousand instructions. It is likely that these numbers are also applicable to higher-order languages, although the author does not currently have data on this.

The initial MTTF T_0 is estimated from

$$T_0 = \frac{1}{fKN_0} \tag{1}$$

where f is the linear execution frequency of the program (the average object instruction execution rate divided by the number of object instructions in the program), K is a fault exposure ratio, and N_0 is the total number of faults in the program. The fault exposure ratio relates fault exposure frequency to "fault velocity." The fault velocity is the rate at which faults in the program would pass by if the program were executed linearly. The fault exposure ratio accounts for:

- Code is not "straight line" but has many loops and branches, except in very trivial cases, and
- The machine state varies, and hence the fault associated with an instruction may or may not be exposed at one particular execution of the instruction.

At present, K must be determined from a similar program. It may be possible in the future to relate K to program structure in some way. On six projects for which data is available, K ranges from 1.54×10^{-7} to 2.99×10^{-7} .

The calendar time interval t consists of the sum of one to three periods. In each period, a different resource (indicated by the value of the index k : C—computer time, F—failure correction personnel, I—failure identification personnel) is *limiting* or produces the maximum ratio of calendar time to execution time for that period. Thus the duration of each period is computed separately based on its limiting resource, and then the durations are summed. We have

$$t = \sum_k \frac{\Delta X_k}{P_k \rho_k} \tag{2}$$

where ΔX_k is the limiting resource requirement for the *period* (e.g., 100 person days), P_k is the limiting resource quantity *available* (e.g., 5 persons), and ρ_k is the limiting resource utilization factor. The resource utilization factor reflects the possibility [particularly for failure correction personnel (see Reference 3, pp. 432-3)] that all of an available resource cannot be usefully employed.

SOFTWARE RELIABILITY PREDICTION PROJECT 40

BASED ON SAMPLE OF 70 TEST FAILURES		EXECUTION TIME IS 813.76 HRS		MTTF OBJECTIVE IS 5000.00 HOURS		CALENDAR TIME TO DATE IS 149 DAYS		PRESENT DATE: 10/6/77	
	95%	CONF. LIMITS 90% 75%	50% MOST LIKELY	50%	CONF. LIMITS 75% 90%	95%			
TOTAL FAILURES	70	70	70	74	80	84	96	138	
INITIAL MTTF (HR)	1.80	2.40	2.94	3.21	3.86	4.51	4.78	5.32	5.92
PRESENT MTTF (HR)	999999	999999	999999	999999	66.7	43.0	36.3	25.4	16.7
PERCENT OF OBJ.	100.0	100.0	100.0	100.0	1.33	0.861	0.726	0.507	0.321
*** ADDITIONAL REQUIREMENTS TO MEET MTTF OBJECTIVE ***									
FAILURES	C	C	C	C	4	6	11	20	51
EXEC. TIME (HR)	0	0	0	0	1232	1715	1977	2752	4400
CAL. TIME (DAYS)	0	0	0	0	46.7	65.8	77.1	113.6	215.7
COMPLETION DATE	100677	100677	100677	100677	112277	121177	122777	129778	514078

Figure 1—Sample project status report.

Each resource requirement during its limiting *period* is given by

$$\Delta\chi_k = \theta_k \Delta\tau_k + \mu_k \Delta m_k, \quad (3)$$

where θ_k is the resource expenditure per unit execution time, μ_k is the resource expenditure per error, $\Delta\tau_k$ is the execution time interval for the *period*, and Δm_k is the number of failures experienced in the *period*.

The number of failures experienced in the period is given by

$$\Delta m_k = M_0 T_0 \left[\frac{1}{T_{k_1}} - \frac{1}{T_{k_2}} \right]. \quad (4)$$

The execution time interval may be determined from

$$\Delta\tau_k = \frac{M_0 T_0}{C} \ln \frac{T_{k_2}}{T_{k_1}}, \quad (5)$$

where C is the testing compression factor* and T_{k_1} and T_{k_2} represent the MTTFs at the boundaries of the limiting resource period.

The boundaries of the different resource-limited periods T_{k_1} and T_{k_2} are the present and objective MTTFs and the transition points

$$T_{kk'} = \frac{C(P_k \mu_{k'} \rho_k - P_{k'} \mu_k \rho_{k'})}{P_{k'} \rho_{k'} \theta_{k'} - P_k \rho_k \theta_k} \quad (6)$$

that lie within that range, where (k, k') have the values (C, F) , (F, I) , and (I, C) . One must determine which resource-limited periods actually occur from an examination of the boundaries and a determination of the maximum calendar time/execution time ratio for each period. The maximum calendar time/execution time ratio for each period is given by

$$\left. \frac{dt}{d\tau} \right|_{\max} = \max \left[\frac{\theta_k T + C \mu_k}{P_k \rho_k T} \right], \quad (7)$$

where T is any MTTF in the range $[T_{k_1}, T_{k_2}]$.

System test cost

Estimates of system test cost are made as follows. Determine the number of failures

$$m = M_0 \left(1 - \frac{T_0}{T_F} \right) \quad (8)$$

that must be experienced and the associated execution time

$$\tau = \frac{M_0 T_0}{C} \ln \left(\frac{T_F}{T_0} \right) \quad (9)$$

* The testing compression factor C depends on the test environment and its relationship to the actual operating environment of the program. More specifically, its value is related to the extent to which redundancy due to runs being made under identical conditions during the operation of the program is removed during the test phase. There is some indication that C is reasonably stable across similar test environments, but data from more projects is needed to verify this. The quantity C may be computed for a project after a sufficient period of operation has occurred following the test phase so that the operational phase initial MTTF can be accurately estimated. If there is no good way of estimating C for a particular project, it is probably best to be conservative and take $C=1$.

to increase the MTTF from T_0 to the MTTF objective T_F . Each of the three *total* resource expenditures χ_j is given by

$$\chi_j = \theta_j \tau + \mu_j m, \quad (10)$$

where j has the values $C, F,$ and I . The χ_j are multiplied by cost rates and the results totaled to yield overall cost.

The foregoing approach implicitly assumes that idle time for all resources during the project can be profitably employed in other activities and should not be charged as a cost. If this is not true for any resource, the cost for that resource should be determined by multiplying t from (2) by the total number of personnel (or dedicated computers) and the resource rate.

Sensitivity of results to parameter accuracy

The reader may be concerned about the accuracy with which parameters on a particular project can be estimated. If this is a problem, one should note that inaccuracies usually affect absolute rather than relative values. Many and perhaps most system engineering decisions are concerned with relative values of alternatives. In any case, calculations can be performed with different values of a parameter to determine the sensitivity of a decision to a parameter inaccuracy. As experience is gained and more data is available, it should be possible to determine parameters more accurately and hence improve the *absolute* accuracy with which costs, schedules, etc. can be estimated.

Example

Consider a cost optimization problem to illustrate the application of the foregoing concepts. An online system is being planned to process orders received by a business, generate bills, break down the work involved into tasks and write work orders on those tasks, order materials, etc. It is desired to establish the mean-time-to-failure (MTTF) objective for the system that will minimize total system cost over an estimated lifetime of two years. Faults are not to be corrected in the field for this system; they will be fixed at the next release. Assume for simplicity that the hardware components of the system are much more reliable than the software and hence may be neglected in this analysis. Also, for simplicity, assume that the entire system test period is failure-correction-personnel limited. The system is expected to operate 250 days/yr, 8 hr/day. The average total cost impact of a failure (in terms of reduced efficiency, extra supervisory time and other work required to "straighten out the mess," etc.) is \$10,000. The software consists of 100,000 source (400,000 object) instructions. Programmer loaded salary is \$30/hr and computer (CPU) time is \$1000/hr. The system test team has eight members and there are 40 program designers available for debugging. The utilization factor for failure correction personnel is 0.138 (computed from Reference 3, Equation (10), using a probability of 0.9 and a queue length of 3). Average instruction execution rate is one million instructions per second. On similar projects, a value

of fault exposure ratio $K=2.4 \times 10^{-7}$ has been experienced and the average fault rate has been 6.25 faults per thousand instructions. Data taken in similar environments indicates that six person hours are required for failure correction per failure and that this effort is independent of amount of execution time. Similarly, four person hours of system test team effort and one hour of chargeable computer time are required per hour of execution time and two person hours of system test team effort and one-half hour of computer time are required per failure. Assume a testing compression factor C of one.

We compute a value of $M_o=N_o=625$, using the program size and average fault rate. The linear execution frequency, determined by dividing object instruction execution rate by number of object instructions is 2.5 sec^{-1} or 9000 hr^{-1} . Hence from (1) we obtain $T_o=0.741 \text{ hr}$. Using (8) and (9) we find

$$m=625 - \frac{463}{T_F} \quad (11)$$

and

$$\tau=463 \ln \frac{T_F}{0.741}. \quad (12)$$

Substituting the resource expenditure rates and (11) and (12) in (10) for each resource we obtain

$$\chi_c=313 - \frac{232}{T_F} + 463 \ln \frac{T_F}{0.741}, \quad (13)$$

$$\chi_f=3750 - \frac{2778}{T_F}, \quad \text{and} \quad (14)$$

$$\chi_i=1250 - \frac{926}{T_F} + 1852 \ln \frac{T_F}{0.741}. \quad (15)$$

The cost of system test will be

$$\begin{aligned} \$1000 \chi_c + \$30(\chi_f + \chi_i) &= \$463,000 - \frac{\$343,000}{T_F} \\ &+ \$519,000 \ln \frac{T_F}{0.741}. \end{aligned} \quad (16)$$

The number of failures during operation will be the total operating lifetime divided by MTTF, or $\frac{4000}{T_F}$. Hence the cost of failures will be $\frac{\$40,000,000}{T_F}$. The expression for the sum of these costs,

$$\$463,000 + \frac{\$39,700,000}{T_F} + \$519,000 \ln \frac{T_F}{0.741}, \quad (17)$$

is of the form

$$a + \frac{b}{T_F} + c \ln \frac{T_F}{T_o}. \quad (18)$$

A simple minimization using calculus yields

$$T_{F|MIN} = \frac{b}{c}. \quad (19)$$

Thus we obtain a value of 76.5 hr for the MTTF objective

that minimizes system life cycle costs. The cost of system test and operational failures for this value is \$3,389,000

To determine the duration of system test, note that since there is only one limiting resource period, (2) becomes

$$t = \frac{\chi_f}{P_F \rho_F}, \quad (20)$$

where χ_f is the resource expenditure for the entire system test period and is given by (10). Hence, using (14) in (20), along with the number of failure correction personnel and their utilization factor, we obtain

$$t=679 - \frac{503}{T_F}. \quad (21)$$

At the minimum cost point, the system test period will require 672 hr or 84 eight-hr days.

Sensitivity analyses can be conducted for each of the quantities involved in the calculation. For example, let K be 1.6×10^{-7} rather than 2.4×10^{-7} . Repeating the calculations outlined above yields a value of 50.8 hr for the MTTF objective that minimizes system life cycle costs.

Another useful technique is to vary parameters that are under the control of the manager to determine the effects on schedules and costs. For example, suppose that the length of the system test period is unsatisfactory. Let us examine the effect of staffing with 60 rather than 40 programmers for debugging. This will reduce the failure correction personnel utilization factor to 0.121 (from Reference 3, Equation (10), using a probability of 0.9 and a queue length of 3). Equation (21) now becomes

$$t=517 - \frac{383}{T_F}. \quad (22)$$

The costs will remain unchanged, since the total resource expenditure required to reach a given MTTF remains the same. Thus minimum cost will still occur at the same value of T_F . However, the *time* required for system test is reduced to 512 hr (64 days).

The reader might suggest increasing the debugging staff still further to improve schedules. In actuality, possible improvement is restricted. We have oversimplified the example for explanatory purposes by dealing with only one limiting resource period. The other periods would come into play and limit further reduction in system test duration.

The reliability of the software for one day (eight hours of operation), assuming the MTTF objective of 76.5 hr is attained, is given by

$$R = \exp\left(-\frac{\tau}{T}\right) = 0.90, \quad (23)$$

where τ is the period of operation and T is the MTTF. This figure can be combined with reliabilities of hardware components to give overall system reliability (Reference 3, pp. 439-442).

MONITORING TEST PROGRESS AND RE-ENGINEERING THE SYSTEM

Recall that status estimates like those shown in Figure 1 can be obtained throughout the system test period. One can therefore continually track present MTTF and its confidence bounds. The number of failures, execution time, and calendar time required to reach the MTTF objective are also computed. By use of (10) and cost rates, remaining system test cost can be computed. The estimates are usually better than those made before test, and they generally improve in quality as testing proceeds.

The effects of changing the MTTF objective or various resources can be investigated if schedule or costs are unsatisfactory.⁸ Consequently the manager can not only determine the present status of his system test effort in terms of a parameter (MTTF) directly related to operational requirements, but he can explore alternatives for accomplishing his objective or the effects of altering it. Thus the techniques we have discussed can be decision-making aids.⁸ Many of the decisions represent a system re-engineering.

A plot of MTTF history for an actual project is given in Figure 2. The dot-dashed center curve is the maximum likelihood estimate and the solid outer curves delineate the bounds of the 75 percent confidence interval. Note the general upward progress with some downward swings. Although there may be some statistical variation, the downward swings have usually been found to be correlated with

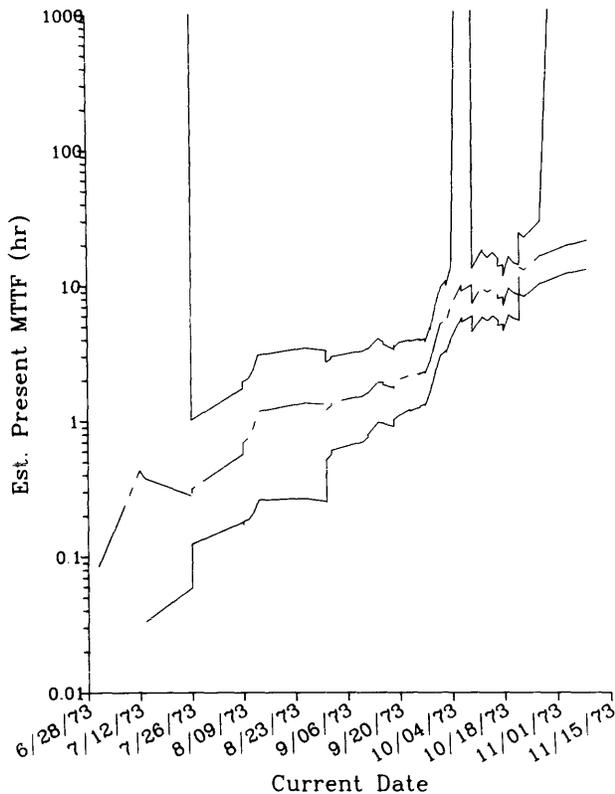


Figure 2—Present MTTF history for Project 1.

design changes or the introduction of new code. The present MTTF is very sensitive to remaining errors when only a few remain; hence its upper confidence limit can be noisy.

SOFTWARE MAINTENANCE

Failures continue to occur (and usually, be corrected) for virtually all software systems of any size during the operational phase. They may occur at increasing intervals, but in many cases, the MTTF of a system exhibits a general stability about some value over the long term, although there are many swings about this value. This behavior is generally due to the periodic installation of design changes (with a resultant drop in MTTF) followed by periods of error removal, during which the MTTF improves. It is particularly true of operating systems and other software provided in computation centers;⁹ this is illustrated in Figure 3.

If the MTTF can be tracked and plotted as illustrated and if service objectives can be set for the system, a quantitatively-based mechanism for change control can be established. When MTTF falls below the service objective, the system is frozen until improvement occurs. The manager may use the amount of margin above the service objective as a guide to the size of the change he will permit at any given time.

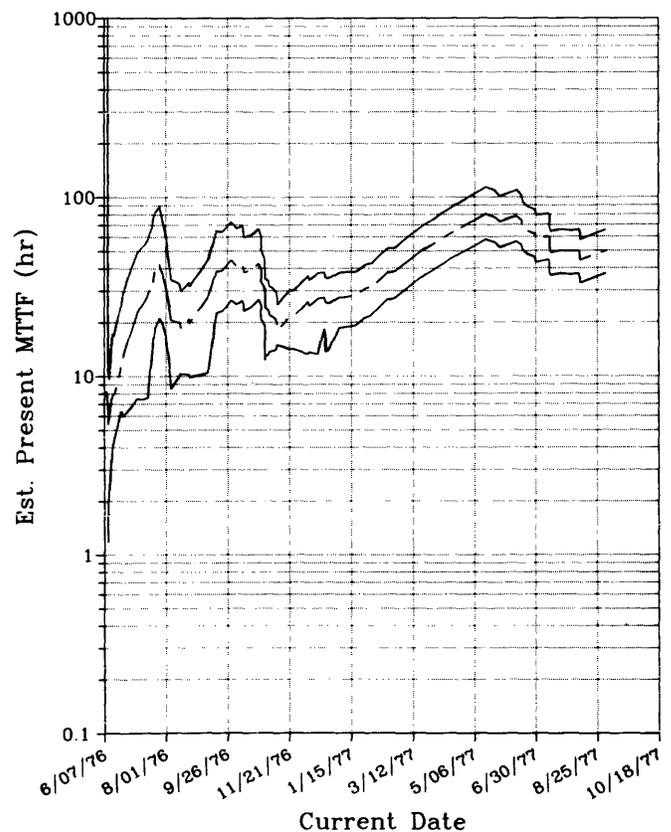


Figure 3—Typical computation center software in operational phase.

CONCLUSIONS

The theory outlined in this paper has proved to be a good framework for understanding, measuring, and predicting the reliability of computer programs. It constitutes an approach that is compatible with hardware reliability theory as to combination of components and thus permits reliability analysis of hardware-software systems. The theory has been applied to several software development projects of different kinds and several operational systems.³⁻⁹ Substantial experience has been gained in its use. It can be used in system engineering, test monitoring, and change control of operational software. As more data is collected from various projects, it should be possible to improve the estimates of some of the parameters, due to added insight into how they vary with program environment factors. This would result in further improvement in the estimation of status quantities, particularly completion date. Experience in application of the theory should lead to its further refinement and broadening, resulting in greater accuracy and wider utility.

ACKNOWLEDGMENT

The author is indebted to R. E. Archer, Jr., D. T. Chai, J. P. Collins, and T. R. Peters for their helpful comments and suggestions.

REFERENCES

1. Boehm, B. W., J. R. Brown and M. Lipow, "Quantitative Evaluation of Software Quality," in *Proceedings of 2nd International Conference on Software Engineering*, San Francisco, Calif., Oct. 13-15, 1976, pp. 592-605.
2. Musa, J. D., "A theory of software reliability and its application," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 3, pp. 312-327, September 1975.
3. Musa, J. D., "Software reliability measurement," in *Software Phenomenology: Working Papers of the Software Life Cycle Management Workshop*, Airlie, Va., August 21-23, 1977, pp. 427-451.
4. Musa, J. D., *Program for software reliability and system test schedule estimation—user's guide*, IEEE Computer Society Repository, Ref. No. R77-244.
5. Musa, J. D., and P. A. Hamilton, *Program for software reliability and system test schedule estimation—program documentation*, IEEE Computer Society Repository, Ref. No. R277-243.
6. Akiyama, F., "An Example of Software System Debugging," in *Proc. 1971 AFIPS Conf.*, p. 359.
7. Endres, A., "An Analysis of Errors and their Causes in System Programs," in *Proc. 1975 Int. Conf. Reliable Software*, Los Angeles, Calif., Apr. 21-23, 1975, pp. 328-329.
8. Musa, J. D., "The use of Software Reliability Measures in Project Management," in *Proc. COMPSAC 78*, Chicago, Ill., Nov. 14-16, 1978, pp. 493-498.
9. Hamilton, P. A., and J. D. Musa, "Measuring the Reliability of Computation Center Software," in *Proc. 3rd. Int. Conf. Soft. Eng.*, Atlanta, Ga., May 10-12, 1978, pp. 29-36.

Verification procedures supporting software systems development

by GRUIA-CATALIN ROMAN

Washington University
St. Louis, Missouri

INTRODUCTION

Particularly in the context of large software systems, prevention and early detection of errors during product development are critical factors in controlling cost and quality. Top-down design, structured programming, informal program verifications, and code inspection are some of the tools presently being used in order to reduce the probability of error. However, current methodologies fail to provide a systematic, comprehensive and well formalized error-detection strategy. In response to this need, a new software development methodology is proposed. The approach, described in the next section, incorporates a cohesive set of verification procedures that enables the validation of each development stage, from requirements definition through individual program implementation. The description of the verification procedures is part of the third section, while the fourth section deals with some managerial aspects related to the practical implementation of the advocated techniques in an industrial environment. A summary and conclusions are presented in the fifth section.

The approach described in this paper has already been adopted as standard practice by the MIS Department of the Monsanto Company of Saint Louis with the anticipation of considerable savings in software development and maintenance costs. Preliminary data already indicate widespread acceptance and significant qualitative improvements. However, quantitative data that would allow for a complete evaluation of the methodology will not be available for quite a while due to the considerable development time required by most systems currently being built at Monsanto. A detailed evaluation of the sociologic and economic impact of the method will be made public at a later date.

THE METHODOLOGY

The development of a software system typically has five stages:

- Requirements definition
- System architecture design
- Program design

- Program implementation and testing
- System integration and testing

The high level of convergence on the basic issues is reflected in a wide variety of approaches. Differences in managerial procedures, company standards, specification techniques, and design strategies make each methodology unique (e.g., References 6, 8, 10 and 11). The methodology proposed here distinguishes itself by the strong emphasis it places on verifiability, not on originality of design and specification techniques, which are described in the remainder of this section. The verification procedures will be presented separately in the next section.

The *requirements definition* stage establishes what functions are to be implemented by the target system. The functions reflect the user's needs and are the basis for the implementation. The requirements definition stage consists of a data-gathering phase (interviews with the user), a synthesis phase (formulation of a functional model) and an evaluation phase (study of feasibility, profits, user environment impact, resources, scheduling, etc.). The functional model supports the design process by providing a complete, precise, relevant and easy-to-access description of the problem at hand. Elements that relate to possible implementations rather than the problem description ought not to be included.

The formalism chosen to specify the functional model is a set of top-down functional diagrams. Each diagram, whose graphic symbols are explained in Figure 1, results from the decomposition of a single function present on an immediately higher level of abstraction. The "permanent record" symbol is used to indicate explicitly the memorization function while "external input" signifies an interaction with the outside environment. The only interdependence that can be expressed between functions is the relation "function F1 provides information I12 to function F2." It was found that this particular relation is the only relevant one since it suffices both for the purpose of designing the system and for establishing its correctness. Figure 2 contains a sample functional model. Although an English narrative is required to accompany each diagram and to explain its items in order, the narrative was omitted for the sake of brevity.

It is usually necessary to break down complex functional models into several parts describing related subsystems. The

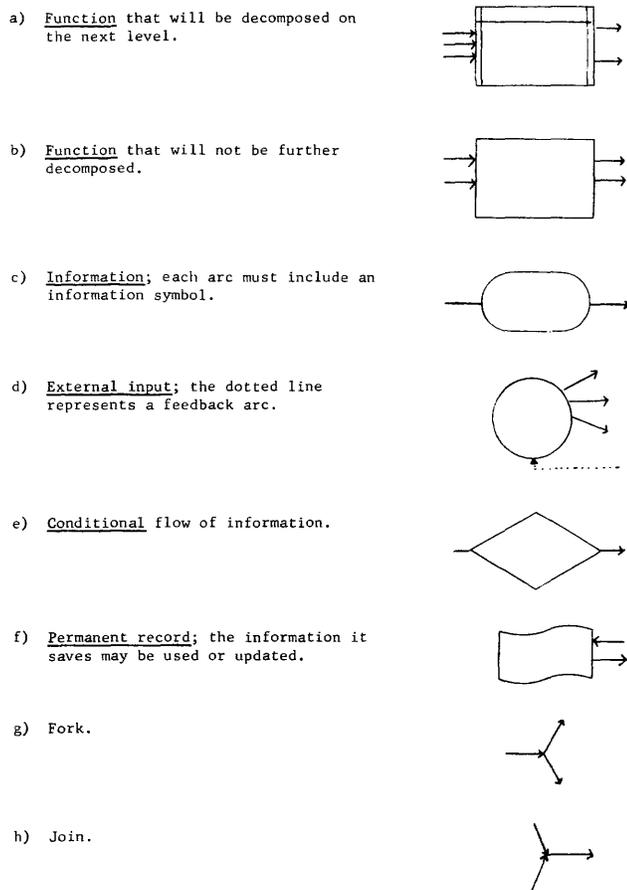


Figure 1—Functional diagrams symbolism.

criteria listed below can assist in the selection of groups of functions that are to establish the requirements for each subsystem:

- Parts of the functional model that are independent, having no connections, could represent separate subsystems.
- Functions that must be present concurrently on-line or satisfy other temporal and spacial relations should be part of the same subsystem.
- Logically-related functions should be associated together.
- Subsystems that are too large are difficult and costly to develop, while those that are too small may be wasteful.
- Few, simple and stable interfaces between subsystems assure a better chance for success.
- The selection of subsystems should assure that the potential new links between functions (due to changes in specifications) either are confined to one subsystem at a time or may be achieved by a minimum number of changes in, preferably, one interface.

The list given above is by no means complete. Furthermore, caution needs to be exercised in achieving a proper tradeoff among conflicting criteria.

During the *system architecture design* stage, programs, flow of control, input-output devices, and relations between programs and data are identified. The system architecture specifies the manner in which the functions considered in the requirements definition stage are to be implemented. Top-down stepwise refinement was selected in order to generate a set of top-down *flow diagrams* expressing the system architecture. The building blocks for the flow diagrams are shown in Figure 3, and examples of flow diagrams are provided in Figure 4. The flow diagrams and the narrative that is required to accompany them cannot completely specify the system architecture; a detailed design of the data structures needs to be included as a separate document. The relation between functional and flow diagrams will be identified better in the section to come; however, it must be pointed out that the top-down nature and labeling conventions of the functional diagrams assure quick access to requirements information, which is critical for a fast and error-free design of the system. As the need for more detailed flow diagrams arises during design, additional decomposition of the functional diagrams has to be carried out based on new interview data.

It is necessary to split the development of large and complex systems into several stages which can be implemented serially or in parallel. (The method is highly advisable for smaller systems as well.) There are two basic ways to carry out this multi-staged development:

1. *Parallel development* (also called system modularization). Parallel development is advisable primarily when working on a very tight timetable and should not be overused. The approach consists of dividing the system into several subsystems which are then developed in parallel. The key to success is to assure minimal need for communication and coordination between teams working on different subsystems. Therefore, in selecting the various subsystems one should assure that:

- The subsystems have very few interfaces.
- The interfaces are simple in nature and unlikely to change.
- The subsystems are conceptually independent and can be independently tested.

2. *Iterative development or system growth*. A more effective and safer approach to constructing large systems is the iterative method. This technique consists of developing an initial incomplete system which is later augmented by incorporating new features. Selecting the initial system, called the core, is the critical aspect of this method. Here are the basic guidelines to be used in choosing the core system:

- The core system must be general and highly flexible.
- The core system should allow for extensive growth at very low cost.
- Additions should not change the core (they may expand it) and should not affect its functions.

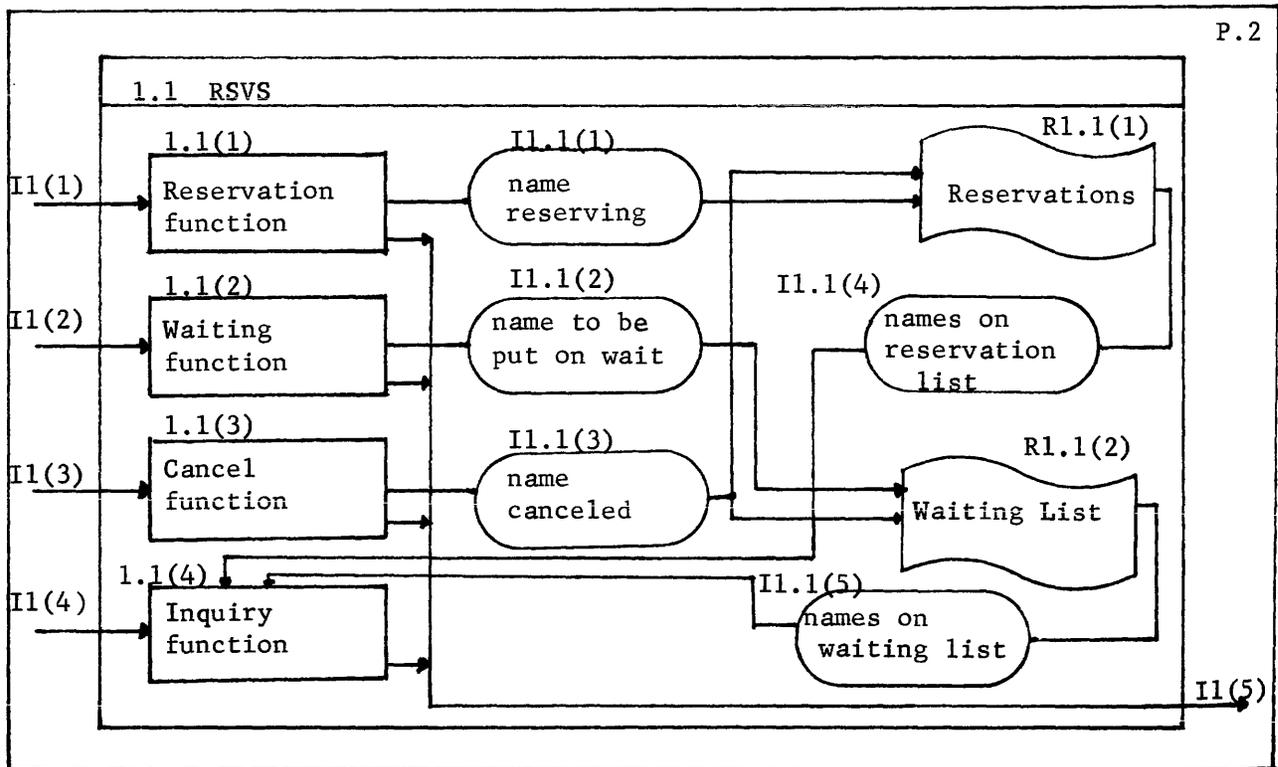
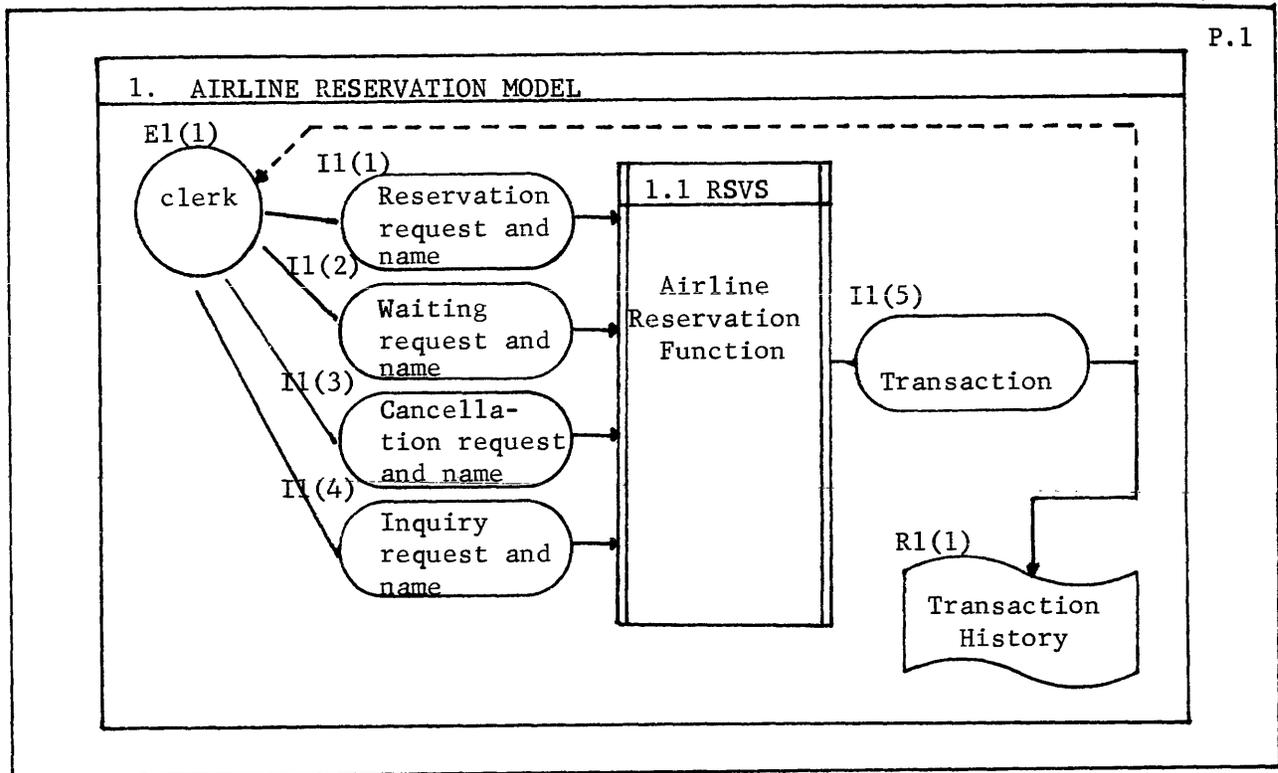
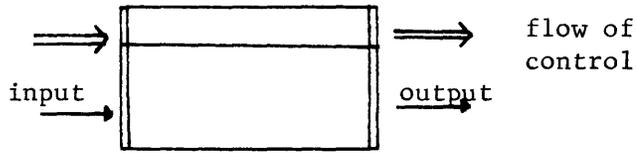
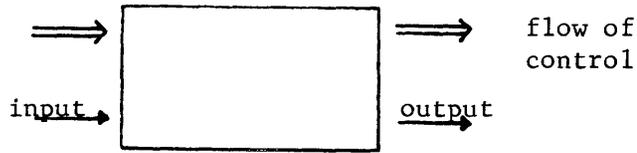


Figure 2—Requirements definition (function diagrams).

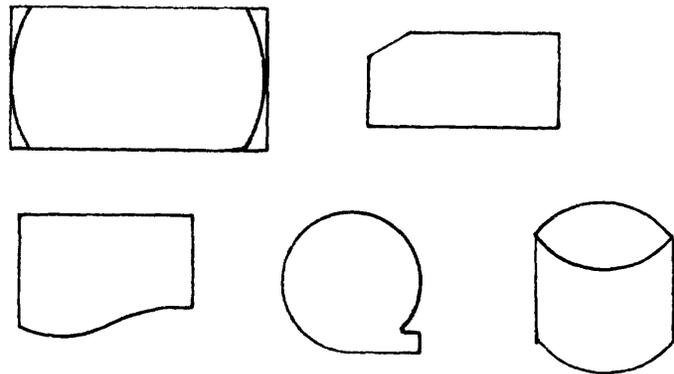
a) System component that needs to be further decomposed.



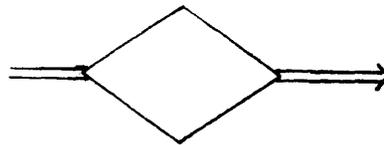
b) Program (will not be decomposed any further).



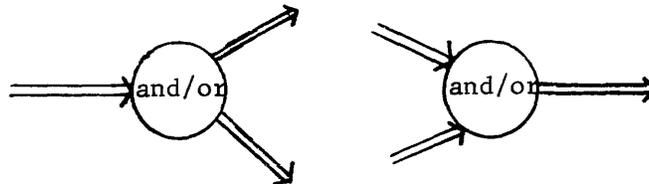
c) Devices (scope, card-reader, printer, tape, disk, etc.).



d) Human decision or clock signal enabling a certain flow of control.



e) Flow of control.



f) Program call.

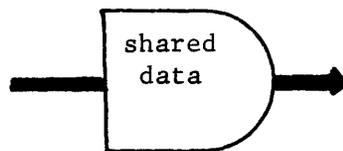


Figure 3—Flow diagrams symbolism.

- The growth should not severely impact the performance of the system.
- Growth should be achieved primarily by creating or implementing new *outgoing* interfaces.
- The number of "not yet implemented" incoming interfaces should be minimal.
- The core system should implement several key func-

tions which would allow for its early installation and/or convincing demonstrations.

When the *program design* stage is entered, the various programs that make up the system have already been identified and the detailed design of their inputs and outputs has been completed. During this stage the selection of the major

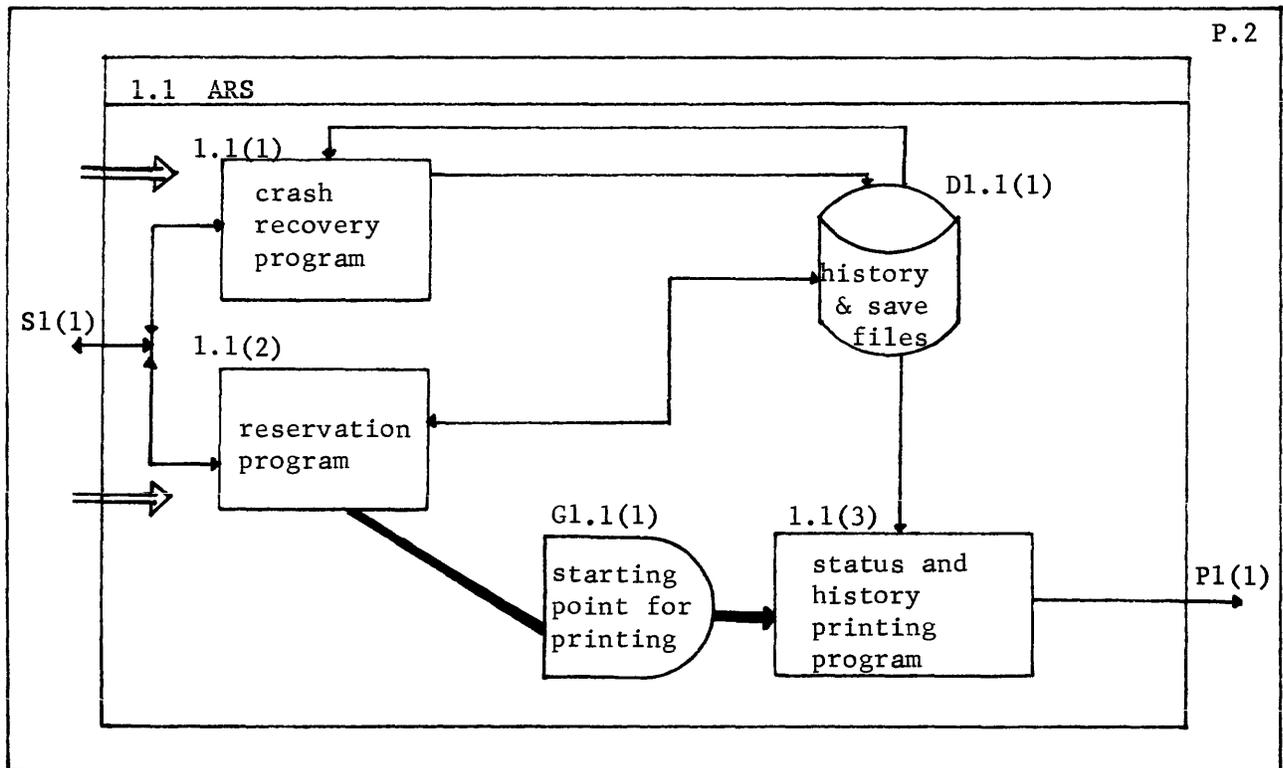
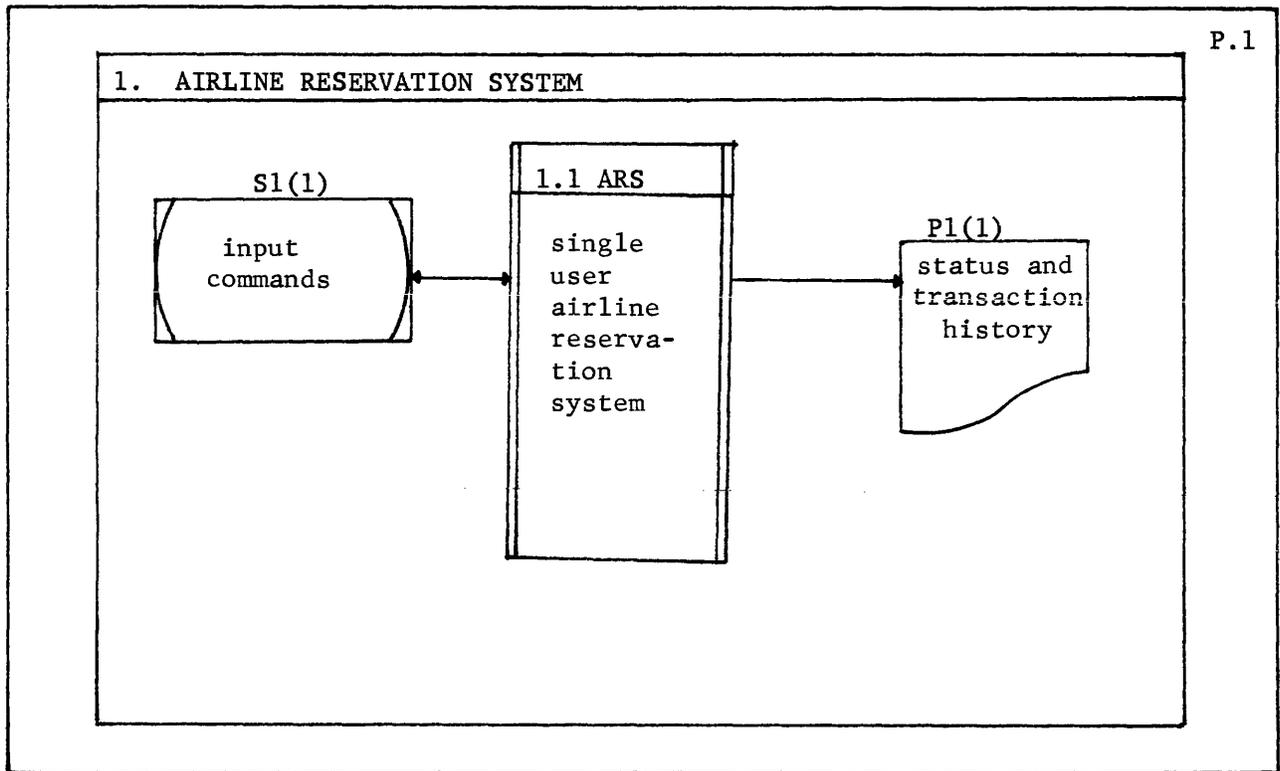


Figure 4—System architecture (flow diagram).

PROGRAM NAME: 1.1(2) RESERVE

MODULE NAME: 1. CONTROL

REMARKS: This is the top module of program RESERVE and is strictly an initialization and driver module.

ACCESS: COMMAND, WAITLIST, RSVLIST. (These data structures are initialized here.)

KEYWORDS: None.

INPUT ASSERTION: COMMAND, WAITLIST, AND RSVLIST are initially empty. The file "SAVE" indicates which names have reservations and which ones are waiting. The "HISTORY" file is empty.

Load WAITLIST and RSVLIST from the "SAVE" file.

ASSERTION: WAITLIST and RSVLIST have been restored to their last known values.

DO

Read next command into COMMAND and put it on the "HISTORY" file.

CASE

<u>WHEN</u> (print command)	<u>INVOKE</u> (1.1(3) PRINT).
<u>WHEN</u> (reserve command)	<u>CALL</u> (1.1) RSV to put name in RSVLIST.
<u>WHEN</u> (wait command)	<u>CALL</u> (1.2) WAIT to put name in WAITLIST.
<u>WHEN</u> (cancel command)	<u>CALL</u> (1.3) CANCEL to remove name from RSVLIST and WAITLIST.
<u>WHEN</u> (save command)	<u>CALL</u> (1.4) SAVE to save current RSVLIST and WAITLIST.
<u>WHEN</u> (inquire command)	<u>CALL</u> (1.5) INQ to print status of the name.
<u>WHEN</u> (halt command)	<u>BREAK</u>
<u>WHEN</u> ()	print error--command not recognized.

ENDCASE

ASSERTION: A single command was fully processed.

OD

ASSERTION: A halt command was detected.

CALL(1.4) SAVE to save the current RSVLIST and WAITLIST in file "SAVE" and to clear "HISTORY."

OUTPUT ASSERTION: The last state of the reservation and waiting lists has been saved on the "SAVE" file. The "HISTORY" file is empty.

Figure 5—Program design (pseudocode).

program data structures and the algorithms that act upon them takes place. The program design is done top-down following generally accepted structured programming practices. Pseudocode is used to formalize the flow of control. The data structures are described separately. A program is made up of several program modules or subroutines hier-

archically organized. The entry and exit points of each program module must include input and output assertions showing what is assumed to be true at the beginning of, and what should be true upon return from, the module. The assertions are important not only in understanding the algorithms but also in the process of establishing the correctness of the

design. For example, the reader is directed to Figure 5 which shows the design for the top module of a program identified in Figure 4.

Program implementation and testing is perceived as a single process that is carried out top-down in the tradition of structured programming. The program design is used as the basis for implementation and testing. If the program was carefully designed, if a good set of implementation standards was selected, and if the design was shown to be correct, then the actual implementation and testing should require little effort.

In its final form the program should exactly mirror the program design. One should be able to identify clearly the modules and the various levels of the top-down design. Each module should *at all times* be consistent with its design specifications. If, when coding, the need for changes in the design becomes apparent, the design should be altered and reverified. Only then should one proceed to recode or modify the module. Upon coding each module, one should verify that the module is in agreement with all implementation standards, and its correctness should be established by mental simulation before any testing takes place.

The *system integration* stage will not be discussed here as it is outside the scope of this paper.

VERIFICATION PROCEDURES

In the context of the methodology presented above, the term "verification" needs to be understood as meaning nothing more than a convincing demonstration that a certain formalization describes, implements, or computes the subject of that formalization, e.g., the agreement between the functional model and the user's needs. One would want to prove that the formalization is correct but, since such an approach is unrealistic at the present time, informal methods need to be used instead. The immediate objectives of the verification are to provide guidelines for the various stages of the development and to allow for effective and systematic reviews at preselected checkpoints. Ultimately, these procedures establish a unique design and error detection strategy capable of significant impact upon the overall system development productivity. This being the case, some methodological details that have been omitted in the previous section will become apparent as the verification techniques are discussed.

The number of checkpoints that one selects depends upon the personnel and the project involved. Nevertheless, there are four checkpoints that need be considered every time: (a) after the requirements definition is completed, (b) when the system architecture is entirely selected, (c) when the program design is finished, and last, (d) when the program is fully implemented. More checkpoints may be added in between, if the size of the project makes them necessary, without modification of the verification techniques. At every checkpoint one must verify that the formalization produced thus far is self-consistent and in accordance with the standards. Furthermore, one needs to establish the consistency between the formalization being reviewed (e.g., flow dia-

grams) and the formalization that preceded it (e.g., functional diagrams) which, in part, describes the correctness criteria. Let us next discuss each checkpoint in detail.

- **Requirements Definition Checkpoint**—At this checkpoint the verification is carried out in three distinct steps. First, the self-consistency of the functional diagrams is checked. This step includes both syntactic (form) and semantic (content) analysis. Second, one must verify the fact that the functional diagrams indeed correctly reflect the analyst's understanding of the business environment which is being modeled. Last, it is necessary to assure that the model meets the customer's approval.

Self-consistency check—It proceeds top-down starting at the root of the tree formed by the functional diagrams.

1. Start with the top level functional diagram.
2. Make sure that:
 - a. The diagram is syntactically correct.
 - b. One single function appears in the top diagram, call it F_o .
 - c. All external inputs to the model are present and necessary.
 - d. All externally visible permanent records generated by the model are present and necessary.
 - e. Given the information available on the incoming arcs, the function F_o can generate the information described on the outgoing arcs.
 - f. All information coming in from the external inputs is necessary.
3. Move one level down.
4. Consider the next not yet verified diagram on the current level. (Assume that (1) the diagram contains functions F_i with incoming information IN_{ip} and outgoing information OUT_{iq} and (2) the diagram is a refinement of the function F_k whose incoming and outgoing information are referred to as IN_{km} and OUT_{kn} , respectively.)
5. Make sure that:
 - a. The diagram is syntactically correct and its incoming and outgoing information are in agreement with the description of the function F_k which is being refined.
 - b. Given the corresponding incoming information, each function F_i can generate the corresponding outgoing information.
 - c. The information provided for each function F_i is indeed necessary.
 - d. Each permanent record introduced in the current diagram contains the information that will be required from it and nothing else.
 - e. The current diagram is correct with respect to F_k , i.e., it is a correct refinement of the function F_k .
6. If there is a not yet verified diagram on this level, go to 4.
7. If there is a next level, go to 3.

Completeness and accuracy check—The functional diagrams are constructed based upon the information that has been obtained from the user or customer during re-

peated interviewing sessions. Therefore, it is important to determine the fact that all that information, to the extent to which it can be incorporated into the functional diagrams, was included. This may be verified by selecting, one by one, the functions identified during the interviews and searching for their presence in the flow diagrams—the top-down organization considerably reduces the search effort.

User agreement—Regardless of the correctness of the requirements definition, its value is limited unless the user's agreement is secured. Since the user may not be sophisticated enough to follow and understand the functional diagrams, a simplified document, verified to be fully consistent with the functional diagrams, may need to be created and passed on to the user for approval.

- *Case Study for the Reader.*—Let us consider a user that needs a small airline reservation system. From interviews it becomes apparent that the system is to be accessed by a single reservation clerk that can make reservations, put passengers on "wait," cancel their names from the reservation or waiting list, and request information about either one of the two lists. Furthermore, the user specifies that a written record of all transactions needs to be saved. Based upon all these data the functional diagrams of Figure 2 may be conceived. As a simple exercise, the reader may want to verify the functional diagrams by employing the first two verification procedures.

Consider, for instance, Step 2 of the first procedure:

- a. Diagram 1 is correctly constructed (standard symbols are used, on each arc there is an information box, etc.).
- b. The diagram contains a single function, RSVS.
- c. A single external input was specified by the user, the reservation clerk.
- d. The only visible permanent record is the transaction history. The reservation and waiting lists are internal to the RSVS function and need not be introduced yet.
- e. Trivial.
- f. All information coming in is necessary, e.g., reservation request and name to be entered on the reservation list are needed in order to reserve.

Similarly, Step 5 of the first procedure may be used to verify Diagram 1.1.

In carrying out the second procedure, one establishes the "coverage" relation between the interview data and the functional diagrams: each function identified in the interviews is covered by a subset of the functional diagrams. For instance, the canceling function is covered by function 1.1(3) from Diagram 1.1 (in conjunction with the information boxes I1(3), I1.1(3) and I1(5)). In general, the coverage relation is not necessarily one to one, but the verification is easier if that is the case.

- *System Architecture Checkpoint*—The system architec-

ture poses for the designer a much more complex and varied set of problems. While the requirements definition involved mostly the formalization of amorphous information, the conception of the system architecture is a complex creative process which must result in a product that possesses a large set of attributes, often in conflict with each other. The result is a more complex set of verification procedures, each reflecting the concern with specific system architecture viewpoints. Thus, besides establishing self-consistency, correctness with respect to the requirements definition and user approval, one must also evaluate the system architecture from the point of view of fault tolerance, hardware compatibility and anticipated performance. However, discussion will be restricted to self-consistency and correctness, which form the main subject of this paper.

Self-consistency check.

1. Start with the top-level diagram.
2. Make sure that:
 - a. Only one system component is included.
 - b. Each external input from the top functional diagram is covered by one or more devices.
 - c. Each permanent record present in the top functional diagram is covered by some storage or output device.
 - d. Any other files present in the diagram (preferably on the lower side) are created and used by the system component and are important enough as to be identified at the top level.
 - e. All inputs are necessary.
 - f. All outputs are required and can be produced from the available inputs.

(NOTE: This step assures that the top-level flow diagram covers the top-level functional diagram. This relation is not necessarily true for any subsequent levels.)
3. Move one level down.
4. Consider the next not-yet-verified flow diagram from the current level. (Assume that the diagram contains the system components C_i with the inputs $INPUT_{ip}$ and outputs $OUTPUT_{iq}$ and is a refinement of component C_k with inputs $INPUT_{km}$ and outputs $OUTPUT_{kn}$.)
5. Make sure that:
 - a. The diagram is syntactically correct and its inputs and outputs are in agreement with the description of the component C_k which is being refined.
 - b. Given its inputs, each component C_i can compute its outputs.
 - c. No unnecessary inputs are provided.
 - d. Each file is created by some component and the data is all there before being used.
 - e. There are no unwanted loops in the flow of control.
 - f. Each component that is marked as not being further decomposed can be implemented as a single program, (based on previous experience with problems of similar complexity).
 - g. Every arc that indicates a call has associated with it a description of the global data being shared between the two programs.

- h. The diagram faithfully implements C_k .
- 6. If there is a not yet verified diagram on this level, go to 4.
- 7. If there is a next level, go to 3.

Correctness check—The system architecture is correct if it implements the model defined by the functional diagrams. The verification procedure will determine the coverage relation, thus indicating what was left out as unimplemented.

1. Show that all external inputs are covered by some input devices.
2. Show that each permanent record is covered by output devices, one or more files, or by being internal to some program.
3. Show that each function, regardless of its position in the functional diagrams, is covered.
4. Show that all key relationships between functions are preserved by the system architecture.
 - *Case Study for the Reader*—Figure 4 does not provide enough information so as to allow for a conclusive verification of the self-consistency and correctness of the system architecture. The missing data is to be found in the narrative that must accompany each diagram and explain each item in the diagram, and in the detailed design of program interfaces (D1.1(1) and G1.1(1)). The relatively simple nature of the system described in Figure 4 allows the reader to supply the missing details. It may also be an interesting exercise to see under which assumptions the verification is successful and when it is not.

However, before going any further, some interesting facts should be pointed out. First, when trying to establish the correctness of the system, the coverage relation is bound to point out that the crash recovery program does not cover any function appearing in the functional diagrams. The justification for introducing that program is not in the problem being solved but in the means by which it is solved, the architecture. If the program had not been included, the fault-tolerance studies would have signaled the fragility of the system. Secondly, the reservation program covers more than one function, actually four, and also two permanent records. Such complicated coverage relations are by no means unusual and further point out the great distinction between functional and flow diagrams.

Lastly, let us observe more closely the connection between self-consistency and correctness. Self-consistency is a conclusive demonstration of the fact that the architecture is a viable one, could be actually implemented, and carries out the basic intent expressed in the top level functional diagram, but is not necessarily a true realization of the requirements definition. In contrast, the correctness check tries to identify the level to which the intent of the requirements definition is reproduced under the assumption of self-consistency. Therefore, one can see that self-consistency is only a stepping stone, a weaker condition to be verified first.

- *Program Design Checkpoint*—It is intended that the program design checkpoint determines the correctness

of the program before the actual coding is started. The detailed design of its interfaces (accomplished during the system architecture design) in combination with a program abstract that indicates the input/output relation represent the criteria against which the program design is to be judged. The presence of assertions facilitates the verification and helps in the understanding. However, the informal (and incomplete) nature of the assertions prohibits one from carrying out a formal proof of correctness. Therefore, as in previous steps, the procedure employed is informal and does not represent any guarantee that all errors have been eliminated.

The verification ought to establish that:

1. The program design paper is in agreement with the program design standards (the pseudocode is correctly used, the format and organization standard, the level of detail adequate, etc.)
2. The data structures have been designed properly.
3. Each program module is correct with respect to its input and output assertions.
4. The interfaces between program modules have been correctly designed and provide all the data required by each module. Also, there is agreement in the communication signals.
5. The program always terminates (if termination is desired).
6. The proper relation between inputs and outputs is achieved.
7. Adequate performance is to be anticipated.
8. The overall design will result in a maintainable program.
- *Implementation Checkpoint*—In the verification scheme proposed here, the last check takes place at the time when the program is being coded and tested. This code inspection (the term seems to have caught on lately) is designed to investigate four distinct problems:
 1. Compliance with the implementation standards.
 2. Consistency between the code and the design paper.
 3. Correctness of data structures implementation.
 4. Reevaluation of the correctness problem to a new level of detail.
 5. Proper trade-off between efficiency and maintainability.

Particular mention needs to be made of the fact that the process of establishing program correctness could benefit significantly from already available theoretical results. While it is doubtful that the average programmer could be asked to formally verify his product, informal use of formal techniques is bound to prove itself highly effective. Furthermore, informal correctness proving should not be very difficult to teach.

IMPLEMENTING THE VERIFICATION PROCEDURES

The existence of a well defined and systematic set of verification procedures plays a fundamental role not only in

the discovery of errors but also in the avoidance of them in the first place. The verification may be conceived of as more than a post-factum investigation, which has its own merits. It may be carried out as an integral part of the design itself. Since most verification procedures involve a top-down analysis, it is only natural for the designer to employ them as the design progresses top-down. Furthermore, familiarity with the verification procedures and the prospect of an imminent check are bound to generate questions in the designer's mind that otherwise would have passed unattended.

Like any other methodology, the proposed verification scheme is worthless if it is not strongly enforced. The methodology as a whole was developed in such a way as to assure that auditing be economically feasible and humanly possible: a uniform approach to design including great emphasis on standardization makes the documentation readable, the consistent top-down approach assures fast access to information and a good organization of the material, and the verification procedures provide uniform and relatively precise evaluation criteria. Nevertheless, a very strong commitment of the management is absolutely necessary in order to overcome the initial resistance to such a novel approach. At the same time people need to be convinced that the audit is not going to be used as a means of evaluating personnel, but rather of evaluating or improving products. If this philosophy is truly implemented, supervisors should not be part of the team auditing the products of those under their authority.

The solution that was finally adopted during the first implementation of this methodology is an audit team including one of the authors of the document being reviewed, persons familiar with the project but not involved in that particular design aspect and complete outsiders to the project. Each member of the audit team is to carry out the verification independently and to reveal his finding during group discussions following the author's oral defense of the material. The task of the audit team could be simplified significantly by providing some adequate software support that would analyze the documentation, enforce the standards, ease the search for information and its retrieval, and perform some primitive self-consistency and correctness checks. Such a system is presently under consideration.

SUMMARY AND CONCLUSIONS

A relatively straightforward methodology for software systems development augmented by a systematic error discovery strategy, a set of verification procedures, was proposed and justified. It was also argued that the informal character of the verification procedures makes them both practical and effective. The discussion emphasized the role played by the verification procedures during software development.

At the present time one large company has adopted the approach as standard practice due to the realization that the cost of carrying out the verification (especially when projects tend to extend over three to five years) represents only

a very small financial investment compared with the significant benefits of early error detection. However, the reader is advised that more work needs to be done. First, the full impact of the methodology has to be established, which in turn should result in improvements to the verification procedures as well as the audit methods. Second, the development of software supporting the enforcement of the methodology and the verification process is needed to further increase their effectiveness while decreasing the time spent during auditing. Third, research needs to be directed toward the development of more formal methods that would support partially-automated consistency and correctness checks.

ACKNOWLEDGMENT

This research was supported through a contract between the Management Information and Systems (MIS) Department of the Monsanto Company and the Computer Science Department of Washington University. The author extends his thanks to David R. Wilson and Edgar J. Kline of Monsanto for reviewing the material and contributing ideas and for the efforts involved in introducing the approach to the MIS community.

REFERENCE LIST

1. Baker, F. T., "Structured Programming in a Production Programming Environment," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, June 1975, pp. 241-252.
2. Blazer, R., N. Goldman and D. Wile, "Informality in Program Specifications," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 2, March 1978, pp. 94-103.
3. Belford, P. C., A. F. Bond, D. G. Henderson and L. S. Sellers, "Specifications Key to Effective Software Development," *Proceedings of the 2nd Software Engineering Conference*, October 1976, pp. 71-79.
4. Enger, N. L., *Documentation Standards for Computer Systems*, The Technology Press, Inc., 1976.
5. Hamilton, M. and S. Zeldin, "Higher Order Software—A Methodology for Defining Software," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 1, March 1976, pp. 9-32.
6. Hammer, M., W. G. Howe, V. J. Kruskal and I. Wladawsky, "A Very High Level Programming Language for Data Processing Applications," *CACM*, Vol. 20, No. 11, November 1977, pp. 832-840.
7. Mills, H. D., "Syntax-Directed Documentation for PL360," *CACM*, Vol. 13, No. 4, April 1970, pp. 216-222.
8. Myers, G. J., *Software Reliability Principles and Practices*, John Wiley and Sons, 1976.
9. Roman, G.-C., "An Argument in Favor of Mechanized Software Production," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 6, November 1977, pp. 406-415.
10. Ross, D. T., "Structured Analysis(SA): A Language for Communicating Ideas," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January 1976, pp. 16-34.
11. Tausworthe, R. C., *Standardized Development of Computer Software*, Prentice-Hall, 1977.
12. Teichroew, D., and E. A. Hershey, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January 1977, pp. 41-48.
13. Willis, R. R., "DAS—An Automated System to Support Design Analysis," *Proceedings of the 3rd Software Engineering Conference*, May 1978, pp. 109-113.

A language for distributed processing

by RONALD J. PRICE

Perkin-Elmer Data Systems Group
Tinton Falls, New Jersey

INTRODUCTION

The main question being addressed here is, what is a good way to program a multiple processor system (whether tightly or loosely coupled) to accomplish an integral distributed processing application? Writing concurrent programs for a uniprocessor is tough enough, but writing programs which interact and operate simultaneously in parallel can be a most difficult and frustrating experience. Opportunities abound for operational failures due to race conditions, for time-dependent bugs and for deadlock situations.

Help is on the scene, though, in the form of new concurrent languages as typified by Concurrent Pascal.⁴ The new software technology embodied by these languages can be applied to multiple processor problems as a methodology regardless of the implementation mechanisms.^{23,25} Nevertheless, the utility of having an effective language is beyond question, even if only as a design tool.

A key feature of Concurrent Pascal is the *monitor* construct that protects critical data regions shared among co-operating sequential processes. With a mutual exclusion mechanism, only a single process is permitted to access the critical region at any given time. This notion was first suggested by Dijkstra,¹¹ formalized by Hoare,¹³ and implemented by Brinch Hansen in Concurrent Pascal. Monitors, or an equivalent construct or capability, have since been incorporated in many other languages.

Although different linguistic variations are possible, Concurrent Pascal was selected as a base for implementing distributed processing programs because of its track record and extensive documentation. The language has proved to be a powerful and effective tool in practice for building structured concurrent programs.⁵ Brinch Hansen recorded improvement in programmer productivity while building a complete operating system with his language,⁶ and the utility of the language has been tested for many diverse applications.²⁹

There has been some criticism of the language, however. For one thing, it depends on a run-time kernel facility that is invariant and built with a different language.²⁰ For another, critical system design decisions have been assumed by the language.²⁴ Researchers are also actively pursuing improved language constructs, most notably the *manager* concept,^{18,27} which ultimately may lead to simpler and even more reliable concurrent programming concepts.

The purpose of this report is to propose two fundamental modifications to Concurrent Pascal that not only will alleviate many of the above concerns, but more importantly, will extend the language's applicability to distributed system environments.

In many respects, the proposed changes are adaptations of principles incorporated in Wirth's real-time language Modula.³¹ As presented in the next two sections, they would enable the kernel and system control operators (i.e., the lowest levels of an operating system) to be written in the language itself and would enable partitions of a global, distributed multiprocessing program to be mapped to physical processors, but yet represented as an integral program.

The last section of the paper summarizes the proposed concepts and applies them as a methodology for constructing systems—from kernels, across processor boundaries, and up through application programs. As such, the extended language is a *systems description language* in that it can be employed to describe the algorithmic behavior of a multiple processor system (not to be confused with a hardware description language that prescribes physical circuits). It offers the systems designer a tool for:

- Synthesis
- Documentation
- Modeling
- Simulation
- Verification

and implementation if used directly as an implementation language.

Although the emphasis of this report is on distributed processing, the proposed extensions increase the power of the language for solving complex operating system problems irrespective of the multiprocessing issues. For example, the following problem areas are difficult under Concurrent Pascal as defined, but are quite amenable with the modified language:

- Data communications
- Process creation
- On-line system generation
- Dynamic software restructuring

The main intent of this paper is to justify and to explain

the benefits of the proposal, not to specify the language nor to suggest a method of implementation. Semantic details and the mechanics of integrating the new constructs within the language need further study and exposure to actual practice.

The level of presentation assumes the reader is familiar with Concurrent Pascal, but the definition of a few items might be useful. A *program* that can be described by the language is called a *concurrent program*. It consists of system (or program) *components* defined as *process* types and *monitor* types (and *class* types not mentioned here); redefinition of the monitor type and the definition of a *task* component as a partition of a concurrent program are described. A Concurrent Pascal program includes a programmable initial process that directs the initialization of the components in the program. The interpretation of *concurrency* is the execution of multiple processes overlapped in time, either by multiplexing periods of execution on a single machine or by simultaneous execution on multiple machines. When important, the latter connotation of true concurrency (i.e., parallelism) will be explicitly denoted in context; multiprocessing implies parallelism, for example.

CONCURRENT PASCAL WITH A PROGRAMMABLE KERNEL

As represented by Figure 1, Concurrent Pascal is based on a virtual machine kernel that implements process switch-

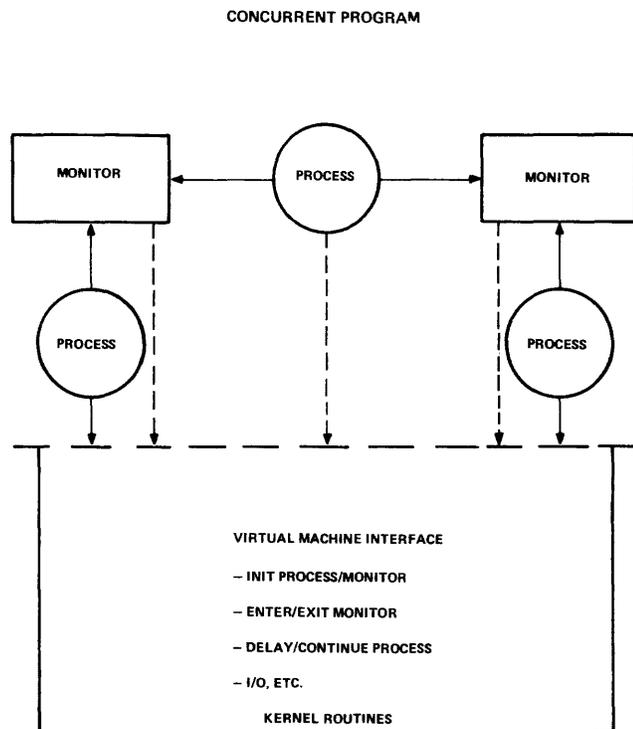


Figure 1—Concurrent Pascal system.

Note: The arrows in Figure 1 and in the following figures that depict a concurrent program represent access rights as defined in Concurrent Pascal, and not the flow of data. Further, circles represent processes and boxes represent monitors.

ing, mutual exclusion on access to monitors, and the various control operators (DELAY, CONTINUE, etc.). The definition of the virtual machine interface can be a problem for system builders interested in different kernel features and/or in multiple machine operations. The problem is that the virtual machine has been abstracted away from the systems programmer to the point of existing literally in another world as defined by its unique language (typically assembly). Moreover, the line between the real and virtual machine might not be optimum for a given application. There are simply too many variables, parameters, factors and extenuating circumstances to consider in general.

In some situations, the programmer of the concurrent program would like to have an influence on the design of one or more of the virtual machine modules, sometimes even to interact with the internal machine dynamically. A prime example of this is programming interrupt service routines. Interrupt handling (typically for I/O processing) is related more to an application than to central general purpose kernel routines; this is clearly so in dedicated systems.

The handling of interrupts has historically caused untold grief and frustration for system programmers. The interrupt is an indeterminate and irreproducible happening. Contemporary systems researchers recommend against using it as a synchronization mechanism and avoid preemption in general. The notion of an interrupt does not even exist in Concurrent Pascal. Instead, synchronizing primitives are provided (DELAY and CONTINUE) that allow system programs to be designed with so-called cooperating sequential processes.

Unfortunately, the processes embodied by most peripheral devices on even modern computers cannot be considered cooperative. Modula was designed to handle them.³² But even if we stopped using the interrupt as a synchronizing mechanism, we still need it as a signal with which to measure time and to build real-time functions.

So although we might want to hide the interrupt in some abstract way, we still have to deal with it. Today this is generally accomplished through the kernel. However, not only is the interrupt hidden by the kernel, it is also typically inaccessible to the high-level software in a direct manner. Brinch Hansen and Hoare point out that scheduling cannot rely solely on built-in abstractions and that high-level software should be in control of response times at the lowest level.³ Indeed, the interrupt is the simplest form of low-level scheduling for machines that can switch an instruction stream automatically upon recognizing an external signal. (Some machines provide multiple priority states where an interrupt level may be interrupted by yet another level, but for purposes of discussion, a single level is assumed here.)

The ability to dispatch programmable service routines in rapid response to external signals and to manage them in a disciplined manner could be afforded to Concurrent Pascal by extending the language with a new construct that allows procedures to be called with interrupts disabled. To allow controlled sharing of the uninterruptable procedures and their data structures, they could be treated much like the ordinary "virtual-time" (i.e., interruptible) monitors. This new construct could then take the form of another system

type in the language—a “real-time” monitor (for want of a better name). Generally speaking, the idea being presented here is to incorporate the real-time principles of Modula within the framework of Concurrent Pascal. Actually, we need not add a new system type to the language, but only have to redefine the monitor to include statements that execute in real-time.

The use of “real-time” monitors for interrupt handling is illustrated in Figure 2. Different delay (wait-on signal) and

continue (send-signal) operators would be needed that are consistent with the real-time environment. With appropriate entry and exit mechanisms, processes could communicate directly with interrupt service routines without going through pre-defined intermediary kernel routines; even interrupt service handlers could directly intercommunicate.

The “real-time” monitor construct would have far more application than just for programming interrupt handlers. For example, when multiprogramming a single machine,

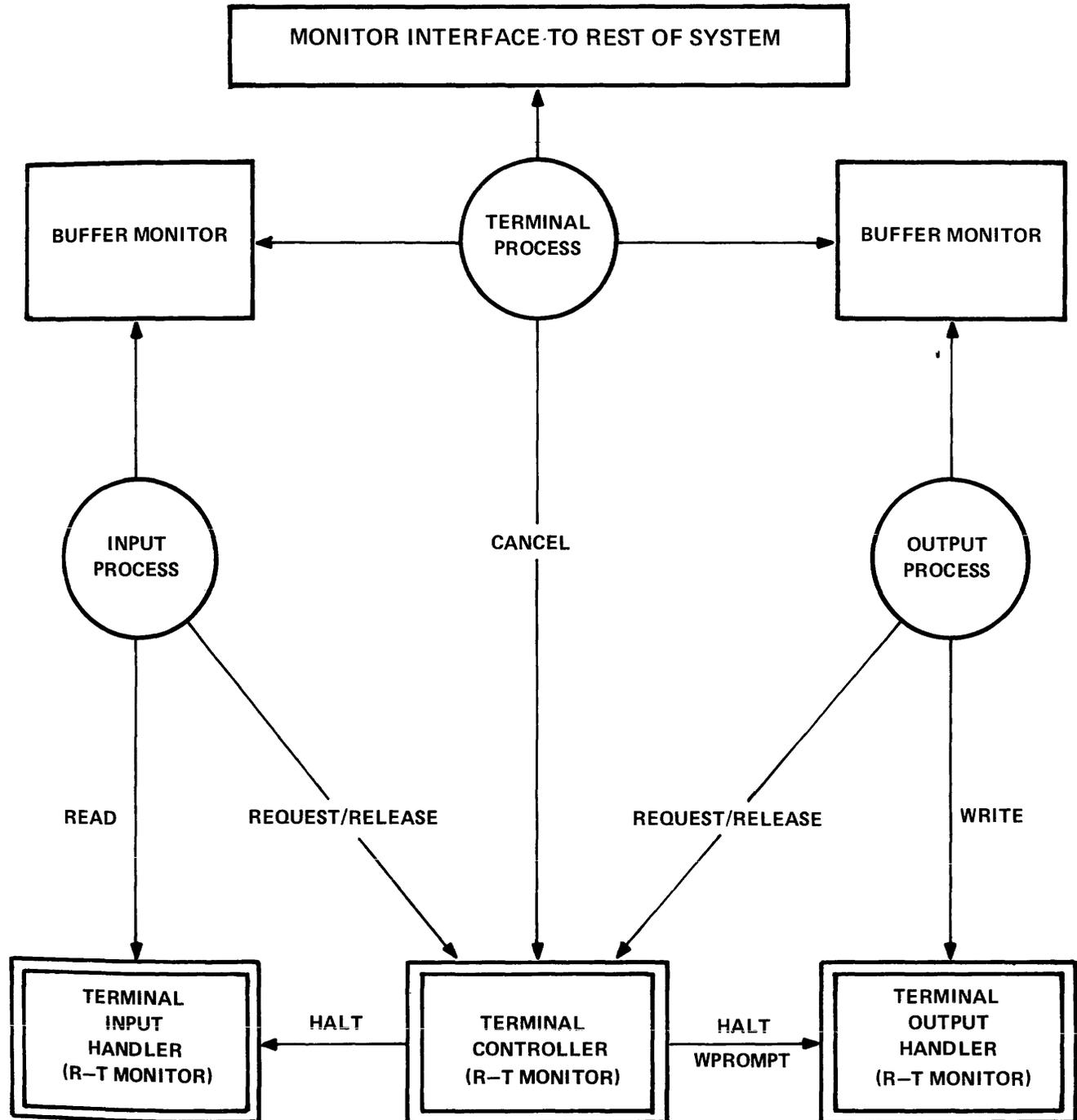


Figure 2—Concurrent program with real-time monitors for terminal I/O handling.

mutual exclusion on access to a monitor is assured simply by having interrupts disabled. In fact, there can be no busy queuing of processes on a "real-time" monitor. Consequently, they could be used in certain situations as a more efficient substitute for the ordinary "virtual-time" monitors in Concurrent Pascal.

Moreover, since the procedures in "real-time" monitors represent indivisible operations to their using program components, they can be employed to implement Concurrent Pascal's "virtual-time" monitors with the language itself. That is, in Concurrent Pascal a process does not directly call a monitor procedure. The call is actually intercepted by a kernel routine to perform mutual exclusion and busy queuing if necessary. This kernel intervention is installed by the compiler in a transparent manner to the programmer. Under the proposed language, this kernel routine would be programmed explicitly and not automatically installed by the compiler (except possibly by default as an implementation-dependent feature).

The various monitor operators and, for that matter, any kernel-like function the systems builder needs would also be programmed in a direct manner. Even conditional critical regions with different scheduling algorithms (guarded regions⁸) can be implemented with this "real-time" construct. In other words, the "real-time" monitor is a means for implementing explicit kernel routines, although the compiler could still support standard implicit kernel calls in a transparent manner.

Figure 3 is an extension of Figure 2 with kernel modules illustrated. An important aspect of this viewpoint is that the full power of the language can be brought to bear on the construction of the lower-level software when it is included as an integral part of the entire system. Such capability is important for embedded systems, process control environments, and data communications applications.

Kernel-like functions could be "hidden" through levels of abstraction, but this would be up to the systems builder and not a condition of the language. In fact, no run-time program, nor a pre-defined kernel definition, is required to support the proposed language.

The kernel can be treated as a concurrent program in its own right,¹⁹ and Figure 3 also illustrates this point. The Genesis process interacts with external processes in peripheral equipment through real-time monitors. It also performs system initialization and takes on the role of the initial process of a concurrent program as per Concurrent Pascal, including in this case the explicit creation of the high-level abstracted user processes. The Kernel Services real-time monitor in this example provides the standard *Enter*, *Exit*, *Delay*, *Continue*, etc., procedures and a *Dispatch* procedure for multiplexing processes. The kernel might control private devices as illustrated, but interrupt handling for the higher-level software would also be supported (typically with considerable hardware assist) for dispatching processes in real-time monitors in response to interrupt signals.

The Genesis process selects high-level processes to execute with the *Dispatch* procedure and executes them much like a subroutine with interrupts enabled. So the high-level abstracted processes are in reality still the Genesis process

in disguise. When the Genesis process recognizes an interrupt signal (presumably with hardware assist), it enters Kernel Services (with interrupts disabled) and takes appropriate action. In the event this action results in activating a waiting process, the Genesis process can decide whether to preempt (reschedule) the current running process or to schedule the waiting process. Typically, the action in response to an interrupt signal would be to dispatch the recipient process immediately in its real-time monitor which in turn would initiate scheduling actions as required.

Many of these low-level functions could be implemented in hardware or firmware. Nevertheless, they can be accurately represented and programmed with the "real-time" monitor construct.

Incorporating the real-time feature does not make the proposed language machine-dependent. From a language point of view, the new proposed construct simply represents the sequential state of the machine. However, escape mechanisms would have to be provided in the compiler for programming machine-dependent features in the low-level software modules; or provide machine-dependent statements as an adjunct to the high-level machine-independent language.

A MULTITASKING CONCURRENT PASCAL

The representation of a kernel as a concurrent program becomes more important when we consider a multiple processor system. Figure 4 is an example expansion on Figure 3 to illustrate kernels for a three-processor system; the surrounding higher-level software is not illustrated. The Inter-Kernel Communication (IKC) monitors are real-time monitors designed for exchanging information between kernels.

As should be evident from the previous discussion, the Genesis process together with the support modules in each kernel's partition is actually a sequential program running on a sequential machine; parallelism is just an illusion to the higher-levels of software. Even in Saxena's verification of the monitor concept,²⁶ he had to represent the idle state of multiple physical processors with an idle process for each processor, the equivalent of the Genesis process. Consequently, in order to represent true concurrency (i.e., parallelism) we need a mechanism for representing the multiple processors, or at least the actions of their kernels.

Even if we were to assume the prior existence of a collection of cooperating kernels on multiple machines that form a virtual multiple instruction, multiple data path machine on which we somehow apply the high-level concurrent program, we still could not take full advantage of the parallel machine with Concurrent Pascal as defined. Loading and initialization of the program, for example, must take place sequentially, either on a single processor or in sequential phases on multiple processors because the initial process of the concurrent program is really the Genesis process of a single kernel.

So we need a way of dividing the global concurrent program into logical partitions that can be delegated to separate processors for initiation and execution. Indeed, we have no viable alternative but to divide the program into physical

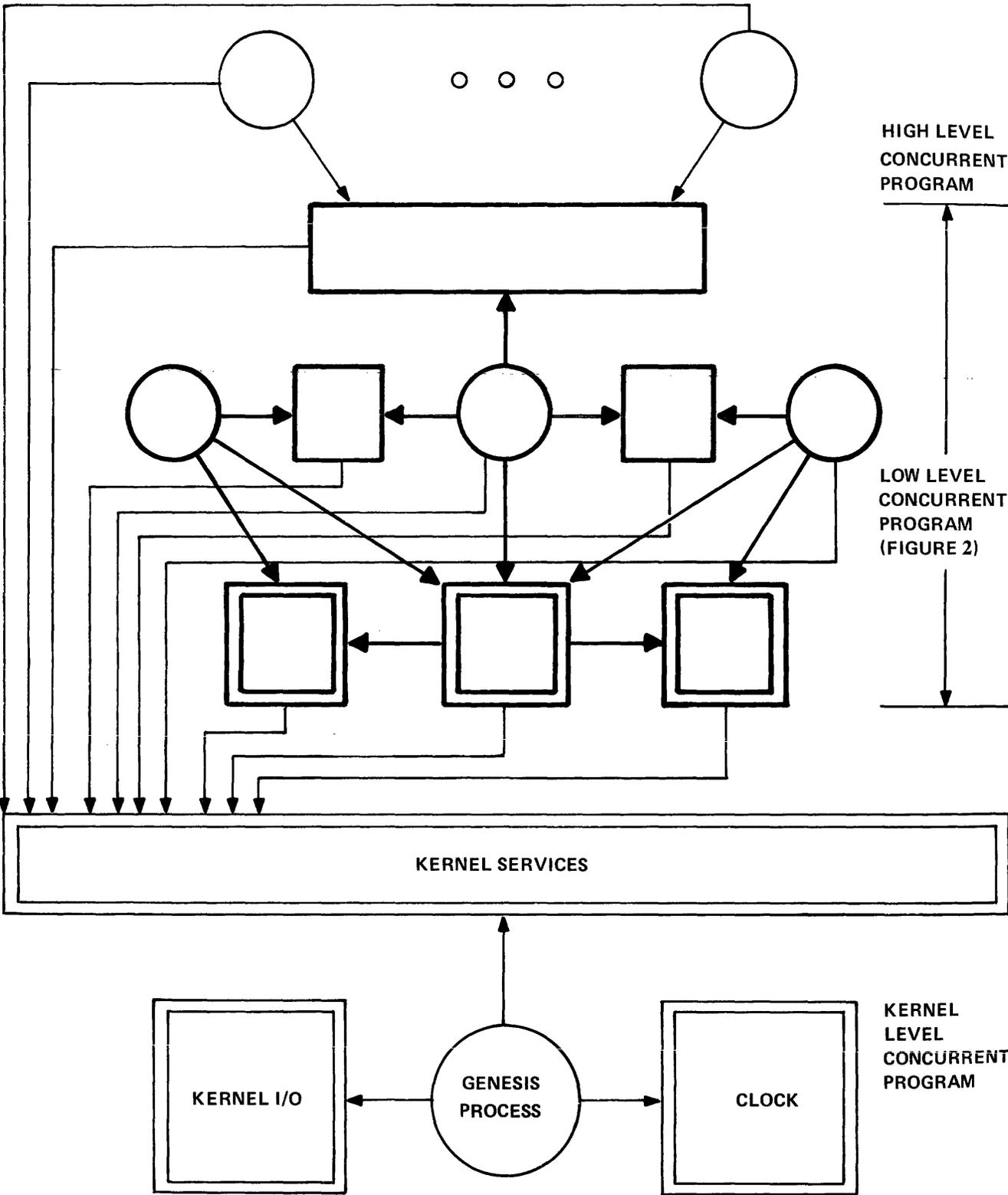


Figure 3—Multiple levels of a concurrent system program.

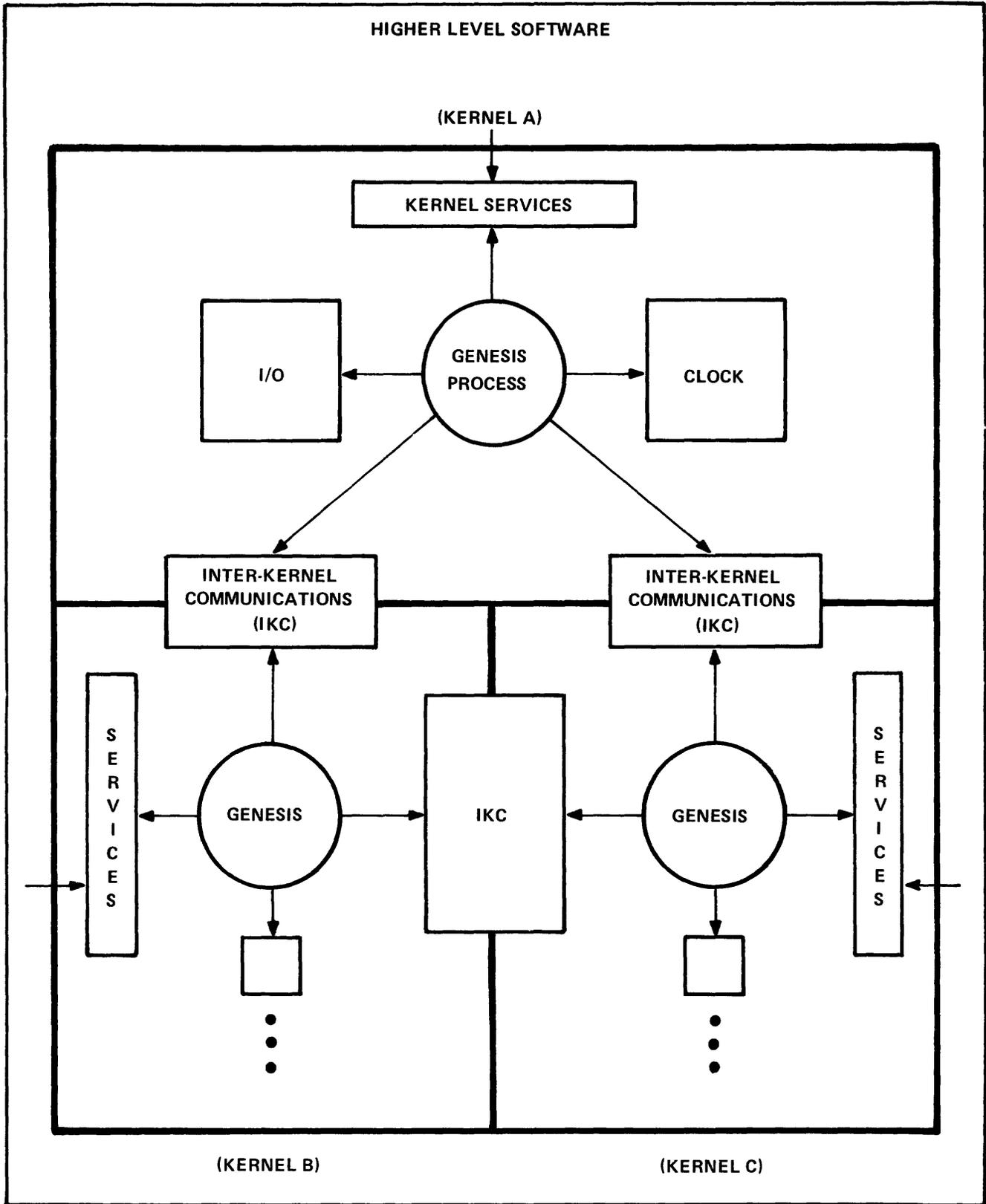


Figure 4—Multiprocessing kernels.

partitions if it is going to be run on a loosely-coupled configuration that does not share memory.

The partitioning mechanism proposed here is to extend Concurrent Pascal with a *task* block structure somewhat analogous to *module* in Modula. The task construct, however, defines a concurrent system component that contains a collection of processes and monitors. It specifically includes an initial process for initializing the task. And tasks cannot be nested. A multiple task program represents parallelism in that different tasks, via their initial processes can be dispatched and executed simultaneously by separate processors. In other words, a multiprocessing program can include multiple initial processes which represent abstracted extensions of multiple kernels.

Each kernel in Figure 4 would be represented by a separate task, and each would be dedicated to a specific processor. The higher-level software could be implemented as extensions of each kernel or as separate tasks. As one or more tasks on a tightly-coupled system, the high-level modules need not be dedicated to specific machines and could be dispatched by any of the three kernels.

Each task is, in essence, an independent concurrent program and can be compiled into a separate load module. Tasks are linked at run-time to form a global system.

The correctness of the system can be tested with an integral compilation where the tasks interact through monitors at the interface of the task boundaries. The compilation of any given task, however, need only include its predecessor tasks in the system and not any task outside its view of the system.

Regardless of the issue of being able to express parallelism in the language, the task construct is a tool for partitioning a multiprocessing system program. Access rights as implemented in Concurrent Pascal will assure a structured design.

We can divide a concurrent program into sections by taking advantage of the isolation property of monitors. That is, processes intercommunicate and synchronize their operations through monitors, and consequently, they need not know anything about each other—even their existence. For example, in Figure 5 the User_B process need not know of the presence of the User_A process when calling the Buff_2 monitor, nor for that matter, even if multiple job processes interface the Buff_2 monitor. Therefore, we can safely cut the program between monitors and processes as illustrated. The trick is to keep the access right arrows pointing in the same direction across the task boundary. (Whether task initialization is performed by a separate initial process or one of the application processes in each task is not relevant to this example.)

Note that the system structure and hierarchical order of the program components, as required by Concurrent Pascal, is preserved if we define and initiate Task A before Task B, even if one physical processor dispatches Task A and another processor dispatches Task B. This would not be the case, however, if the Job_3 process were included in the Task A partition because then each task would have access rights to each other in a cycle.

Sometimes the initial layout of a concurrent program does not lend itself to partitioning. For example, if we tried to

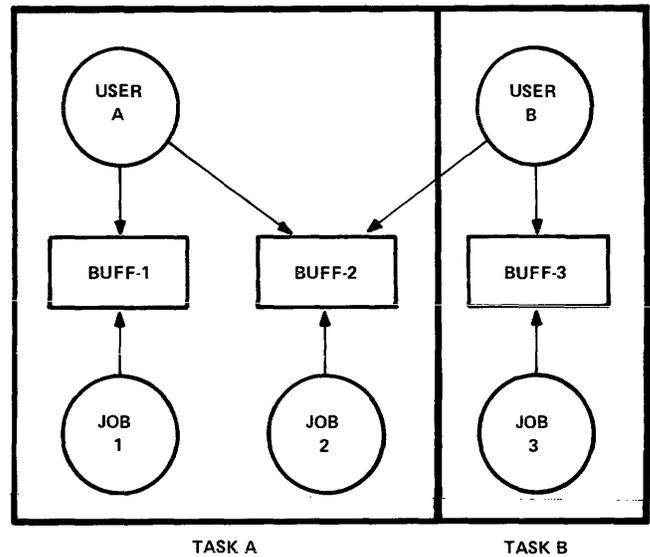


Figure 5—Partitioning a concurrent program.

apply the tasks in Figure 6 to two different machines, the multiprocessing program could easily crash when started (even if one task is initiated before the other) because the design does not guarantee that the monitors will be initialized before being called. But then the program might not crash; the problem is a time-dependent race condition.

Start-up is only part of the problem. We also need orderly ways of stopping a multiprocessing program, and more importantly, mechanisms for detecting error situations across processor boundaries and recovering from them. This is what partitioning is about.

Figure 7 shows how we can take advantage of the insertion property of monitors to resolve this task layout problem. Here, a message exchange monitor and server process are inserted in the User_A process access path. This gets the arrows pointing in the same direction across the task bound-

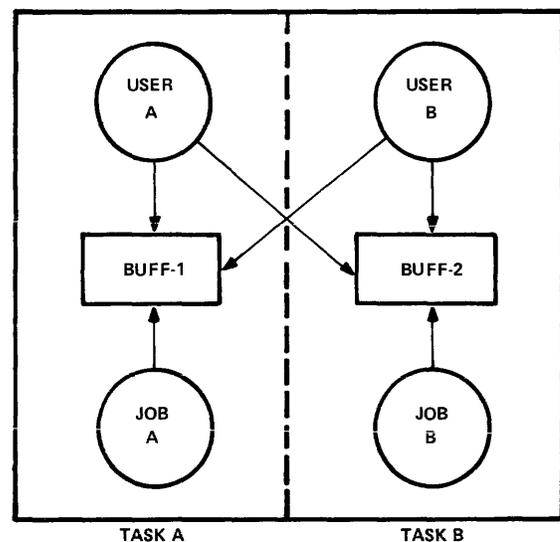


Figure 6—Invalid task partitioning.

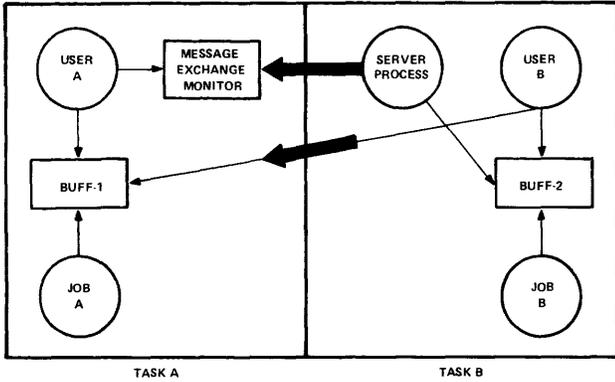


Figure 7—Partitioning with insertion.

ary. The server process acts in behalf of the User_A process in the Task B partition.

A correct way to partition a multiprocessing program is to group logically-related processes and monitors into separate tasks in such a manner that their access rights point in the same direction across the task boundaries and by arranging the tasks in a hierarchy such that tasks which access other tasks are ranked below their predecessors, as in Figure 8. This ranking assures an orderly initialization (and termination) and eliminates race conditions and deadlock situations that otherwise might occur with a cyclic control structure.

In some arrangements, tasks, such as Task C in Figure 8, can be literally removed and brought back on-line without disturbing the rest of the system. The status of Task A does have to be known to Tasks B and C, however. In essence, the kernel tasks (not illustrated) and Task A form a virtual machine for Tasks B and C. This capability allows a system program to be generated and restructured dynamically.

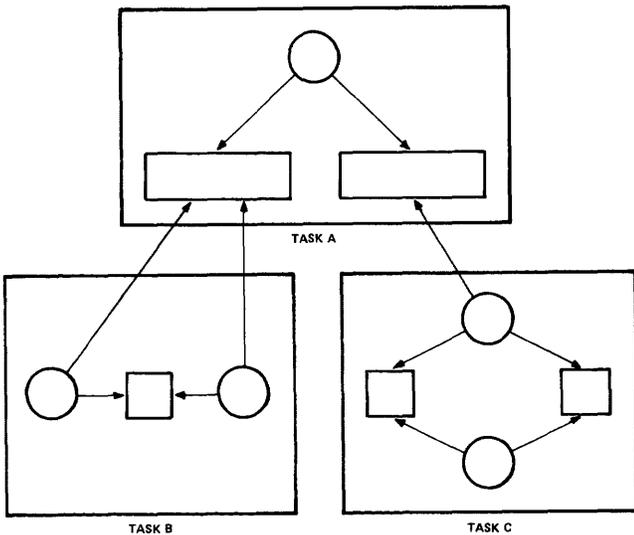


Figure 8—Hierarchical structuring of multiprocessing programs.

DISTRIBUTED PASCAL

A key feature of this proposal for implementing distributed programs is the ability to describe interface monitors between processors. The characteristics of a given interface can be programmed with the "real-time" monitor construct. Parallel kernels can then be described with the task block structure where the legality of their interface monitors is tested with an integral compilation. Higher-level tasks are built on top of the kernel tasks.

Interface monitors can be implemented in shared memory employing "thick-wire" communication techniques or in shared "thin-wire" I/O facilities. Mutual exclusion between machines is achieved by mutual cooperation in adhering to a protocol.

In the thick-wire case, permission to access the data structures is achieved by locking the monitor with a *read-modify-write* operation (e.g., Test and Set instruction) and then the data structures are manipulated in place. The logic for manipulating the data (i.e., the monitor's program code) can also be located along with the data if the hardware configuration allows code to be executed out of shared memory, or otherwise the logic can be replicated in the private memory of each processor.^{9,25}

In the thin-wire case, data are physically copied from one location to another. Although Concurrent Pascal's monitors cannot be directly supported across a thin-wire boundary, an abstracted user's environment illustrated by Figure 9a could be supported by an underlying message communications system as depicted by Figure 9b. This software message system is conceptually the same thing implemented in hardware to support shared memory; however, the flexibility of a thick-wire emulation in software has to be highly constrained because of the limited bandwidth and long response times of the communication facilities.

A good case can be made for adopting a standard thin-wire communication technique for multiple processor systems which is adaptable to networks as well as to tightly-coupled architectures.^{15,22,30} The overhead normally associated with a message-based system can be ameliorated by implementing message exchange facilities in hardware.^{16,28}

Special languages have been proposed for message systems,^{1,21} but Concurrent Pascal is a very suitable language

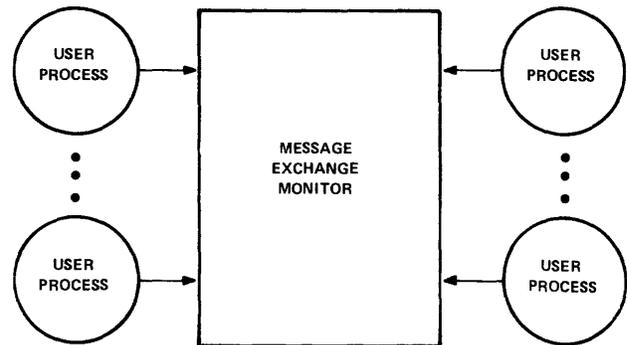


Figure 9a—Message-based concurrent program

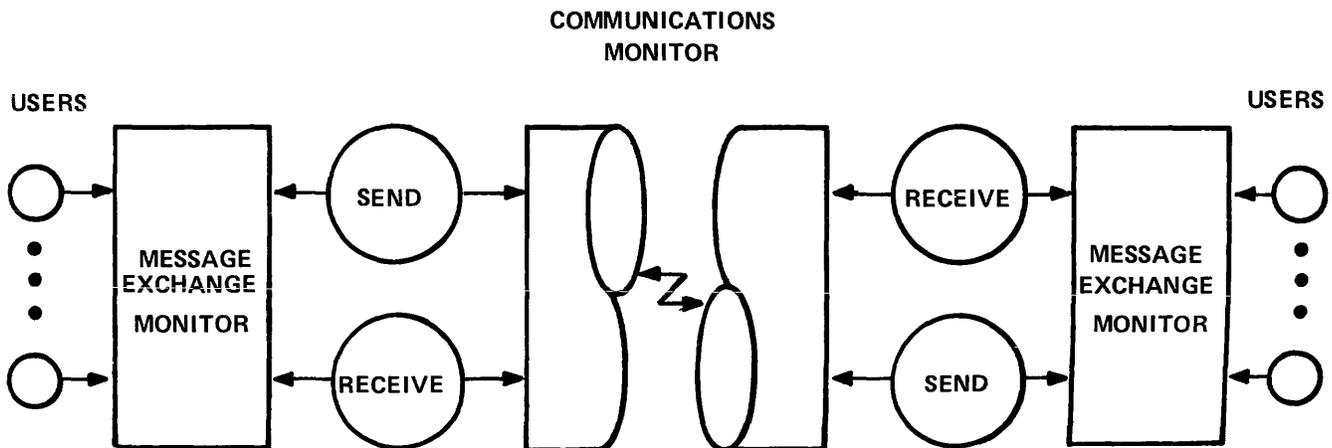


Figure 9b—System implementation of message communications.

for expressing networked systems,⁷ including the communications protocol.^{2,9} Moreover, the language offers the flexibility of general monitor designs where appropriate in addition to any built-in message exchange monitors of the communications system.

In any case, Concurrent Pascal as proposed to be modified is open-ended in the sense that both communication approaches can be accommodated. For example, if an operating system built with the language establishes a message-based inter-process communications protocol for conventional system use, the underlying implementation can still be based on thick-wire techniques where appropriate and more efficient.

Figure 10 depicts a multiple-task, multiple-processor system employing both thick-wire and thin-wire communications. The kernels dedicated to the processors in each tightly-coupled dual processor complex interface through multiple real-time monitors in shared memory, whereas the two complexes interface through a single real-time monitor over a communications channel. Kernels are represented by tasks and form the lower levels of the system. Higher-level tasks in the global system intercommunicate through monitors in a hierarchical fashion, as well. It is important to note that the different levels do not necessarily imply physical levels; that is, virtual kernels that emulate process switching, interrupts, etc., on top of real kernels is not a requirement to support high-level concurrent programs.

The point being made here is that this total system can be described with a single program (although part of it might be implemented in hardware). The program consists of a set of cohesive routines (program components) that implement the behavior of the global system. Indeed, the whole system operates as a harmonious confederation of cooperating sequential processes, some of which may run in parallel.

Even if the language is used only as a modeling tool, it can help us to design reliable systems by applying computer programming technology to their construction. This is so because Concurrent Pascal is based on proven software engineering techniques.

When building the "THE" multiprogramming system, Dijkstra suggested employing hierarchical levels of abstraction as a methodology for dealing with the complexity of operating systems;¹⁰ that is, modules are built on top of others with well defined interfaces and interactions. This technique, actually a formal method of structured programming, is an invaluable aid for proving program correctness and is an inherent capability of Concurrent Pascal.

The axiomatic definition of Pascal¹² and the treatment of critical regions (e.g., monitors of Concurrent Pascal) and other research efforts have led to many proofs of program correctness relevant to concurrent programming (e.g., References 8, 14, 17, 26). These principles are now being applied in attempts to discover simpler, more flexible and more reliable techniques for constructing monitors and by enlisting the aid of the compiler itself.

The fact that formal constructs can lead to provably-correct programs may sound academic in reality. However, they actually do in practice lead to rapid program synthesis and to program correctness by inspection. Testing becomes much more systematic and takes on more of a verification role than a debugging operation. Modification and maintenance are also assisted.

By maintaining the consanguinity of Concurrent Pascal as proposed, we can apply these formal constructs to the construction of distributed systems. The language serves as a synthesis aid by enabling the system designer to decompose a system in terms of task components which in turn are decomposed into logically-related processes and monitor components; this can be illustrated in diagrammatical form. Moreover, the language allows a system to be designed in incremental stages, and it can be used to simulate and to evaluate different implementation strategies. In particular, the system designer can describe proposed solutions as models that accurately represent the physical environment and that can be demonstrated to run correctly. The language can also serve as a vehicle for documentation and testing. Finally, it becomes a piece-part of the end product where it is employed as an implementation language.

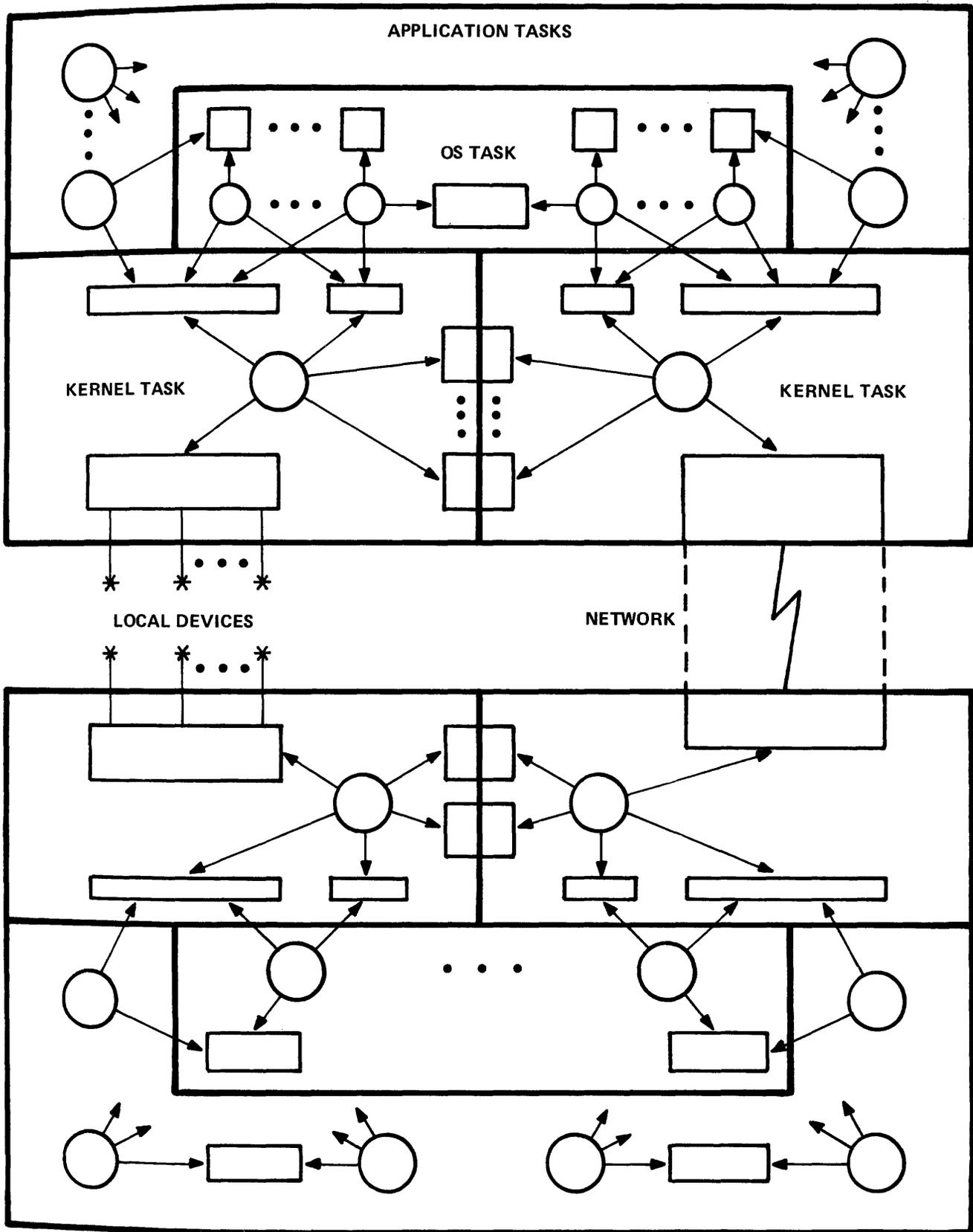


Figure 10—Distributed processing system.

CONCLUSION

Changes to the language Concurrent Pascal are proposed that enable it to be used to:

1. Describe the algorithmic behavior of the physical system.
2. Express the physical parallelism of a distributed multiprocessing program.

As such, the new language acquires the connotation of Distributed Pascal.

ACKNOWLEDGMENT

Mr. Gary Anderson and Mr. Tom Kibler are thanked for their helpful contributions and review.

REFERENCES

1. Ambler, A. L., et. al., "GYPSY: A Language for Specification and Implementation of Verifiable Programs," *ACM SIGPLAN Notices*, Vol. 12, No. 3, March 1977, pp. 1-10.
2. Bochmann, G. V., "Logical Verification and Implementation of Protocols," *Proceedings Fourth Data Communications Symposium*, October 1975, pp. 7-15 to 7-20.
3. Brinch Hansen, P., et. al., "Process Dispatching Techniques," *Operating Systems Techniques*, C.A.R. Hoare and R.H. Perrott (ed.), Academic Press, New York, 1972, pp. 201-207.
4. Brinch Hansen, P., "The Programming Language Concurrent Pascal," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, June 1975, pp. 199-207.
5. Brinch Hansen, P., *The Architecture of Concurrent Programs*, Prentice-Hall, Englewood Cliffs, New Jersey, 1977.
6. Brinch Hansen, P., "Experience with Modular Concurrent Programming," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 2, March 1977, pp. 156-159.
7. Brinch Hansen, P., "Network: A Multiprocessor Program," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 3, May 1978, pp. 194-199.
8. Brinch, Hansen, P., "Specification and Implementation of Mutual Exclusion," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 5, September 1978, pp. 365-370.
9. Cavers, J. K., "Implementation of X.25 on a Multiple Microprocessor System," *Proceedings 1978 International Conference on Communications*, 1978, pp. 24.6.1-24.6.6.
10. Dijkstra, E. W., "The Structure of the 'THE' Multiprogramming System," *Communications ACM*, Vol. 11, No. 5, May 1968, pp. 341-346.
11. Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," *Acta Informatica*, Vol. 1, 1971, pp. 115-138.
12. Hoare, C. A. R. and N. Wirth, "An Axiomatic Definition of the Programming Language PASCAL," *Acta Informatica*, Vol. 2, 1973, pp. 335-355.
13. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," *Communications ACM*, Vol. 17, No. 10, October 1974, pp. 549-557.
14. Howard, J. H., "Proving Monitors," *Communications ACM*, Vol. 19, No. 5, May 1976, pp. 273-279.
15. Jensen, E. D., "Distributed Processing in a Real-Time Environment," *Distributed Systems, Infotech State-of-the-Art Report*, Infotech International Ltd., Berkshire, England, 1976, pp. 303-318.
16. Jensen, E. D., "The Honeywell Experimental Distributed Processor—An Overview," *IEETC*, Vol. 11, No. 1, January 1978, pp. 23-38.
17. Karp, R. A., and D. C. Luckham, "Verification of Fairness in an Implementation of Monitors," *Proceedings of 2nd International Conference on Software Engineering*, October 1976, pp. 40-46.
18. Kiebertz, R. B., and A. Silberschatz, "Capability Managers," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 6, November 1978, pp. 467-477.
19. Lister, A. M., and P. J. Sayer, "Hierarchical Monitors," *IEEE Proceedings 1976 International Conference on Parallel Processing*, pp. 42-49.
20. Löhr, K., "Beyond Concurrent Pascal," *Proceedings of 6th ACM Symposium on Operating System Principles, ACM Operating Systems Review*, Vol. 11, No. 5, November 1977, pp. 173-180.
21. May, M. D. et. al., "EPL—An Experimental Language for Distributed Computing," *Proceedings NBS-IEEE Trends and Applications: Distributed Processing*, May 1978, pp. 69-71.
22. Metcalfe, R. M., "Strategies for Interprocess Communication in a Distributed Computing System," *Symposium on Computer-Communications Networks and Teletraffic*, Polytechnic Institute of Brooklyn, April 1972, pp. 519-526.
23. Paquet, J. L., et. al., "Concurrent High-Level-Language Machines and Kernels," *Proceedings of the IEEE International Symposium on Mini and Microcomputers*, November 1977, pp. 293-298.
24. Parnas, D. L., "The Non-Problem of Nested Monitor Calls," *ACM Operating System Review*, Vol. 12, No. 1, January 1978, pp. 12-14.
25. Price, R. J., "Multiprocessing Made Easy," *Proceedings National Computer Conference*, AFIPS, 1978, pp. 589-596.
26. Saxena, A. R., and T. H. Bredt, "Verification of a Monitor Specification," *Proceedings of 2nd International Conference on Software Engineering*, October 1976, pp. 53-59.
27. Silberschatz, A., et. al., "Extending Concurrent Pascal to Allow Dynamic Resource Management," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 3, May 1977, pp. 210-217.
28. Swan, R. J. et al., "Cm*—A Modular, Multi-microprocessor," *Proceedings National Computer Conference*, AFIPS, 1977, pp. 637-644.
29. Wallentine, V., and R. McBride, *Concurrent Pascal—A Tutorial*, Department of Computer Science, Kansas State University, November 1976.
30. Wecker, S., "A Design for a Multiple Processor Operating Environment," *Proceedings 7th Annual IEEE Computer Society International Conference*, COMPCON '73, February 1973, pp. 143-146.
31. Wirth, N., "Modula: A Language for Modular Multiprogramming," *Software Practices and Experiences*, Vol. 7, No. 1, January 1977, pp. 3-84.
32. Wirth, N., "Toward a Discipline of Real-Time Programming," *Communications ACM*, Vol. 20, No. 8, August 1977, pp. 577-583.

Automatic program transformations for virtual memory computers*

by W. ABU-SUFAH, D. KUCK and D. LAWRIE

University of Illinois at Urbana-Champaign
Urbana, Illinois

INTRODUCTION

Improving the behavior of virtual memory systems is a popular subject, as evidenced by the vast number of papers in the literature. Typically, attempts to improve behavior fall into two areas—those which accept existing locality properties of programs and attempt to modify system parameters (e.g., memory allocated, window size for the working set policy, etc.), and those which attempt to reorganize programs in some way. The first approach treats programs behavioristically, i.e., without any attempt to change the original behavior of the program. This type of research generally attempts to deal with space allocation policies and replacement algorithms in order to improve the performance of the system, given the original behavior of the programs. The work of Denning,^{12,13} Belady,⁸ Chu and Opderbeck,⁹ Smith,²¹ Trivedi²² and many others has contributed greatly to the evolution of operating systems and hardware for virtual memory systems.

Yet the second approach promises even more significant improvement. Early work in this area¹¹ indicated the importance of program reorganization, and more recent research (e.g., References 17 and 15) has borne out this promise. Our work belongs to this latter group, but with an important difference. We reorganize programs—automatically—by examining their structure at the source code level where more information about the program is available. The papers by Elshoff¹⁴ and Trivedi²³ describe some similar techniques but left the restructuring to the programmer.

In the next section of this paper we present a brief description of our program transformations. By detailed analysis of the program, we reorganize the loop structure of programs in an attempt to ensure that once a page is used, as much computation as possible is done on that page before it is discarded and replaced by a new page. The result of our transformations is a program whose locality is better controlled. Our presentation will be through examples. For a formal description of the transformations, implementation problems and theorems related to the correctness of the transformations see References 3 and 1.

* This work was supported in part by the National Science Foundation under Grant Nos. MCS76-81686 and MCS77-27910, and the Department of Computer Science, University of Illinois at Urbana-Champaign.

In the third section we present a summary of some preliminary experimental results which we obtained by applying our transformations to a collection of FORTRAN programs. As we will describe in that section, we have obtained good results so far in our work. We have seen more improvement in space-time product over standard paging from our source level transformations than we see in going from a nonpaging system to paging.

In the final section we will present some concluding remarks.

BRIEF DESCRIPTION OF THE TRANSFORMATION

Throughout this paper we will be concerned only with data paging. Moreover, we will ignore references to scalar variables. Similar assumptions were made by other researchers.^{6,5} The storage of each array will start on a page boundary. Moreover, we are primarily concerned with scientific programs; it is usually the programs with large arrays which cause serious problems for virtual memory computers.¹⁴

One of our principal transformations is *distribution* of DO-loop control. In order to see how this improves paging behavior, consider Program 1.

```
Program 1
DO 1 I=1, N
  A(I) = B(I) + C(I)
  X(I) = A(I) * X(I-1) + D(I)
1   E(I) = 2 * X(I) + F(I)
```

Consider first a non-paged versus a paged machine. If allotted seven data pages, this loop will run completely through each of seven array partitions between page fault bursts. If each array occupied a total of p pages, then about $7p$ page faults would be generated in total, whereas a nonvirtual memory machine would have to allot a total of $7pz$ words of memory to its execution (z is the page size, in words). Thus, the memory-saving due to paging is a factor of p . Generally, however, paging will increase the I/O activities over a non-paged system, because some pages may be re-fetched several times. Note that in programs containing sev-

eral loops, the same space can be used for each loop so the total space saved by standard paging is proportional to the product of p and the number of loops in the program (assuming that distinct loops reference distinct pages).

Program 2

```

DO 1 I=1, N
1   A(I)=B(I)+C(I)
DO 2 I=1, N
2   X(I)=A(I)*X(I-1)+D(I)
DO 3 I=1, N
3   E(I)=2*X(I)+F(I)

```

Suppose that Program 1 is rewritten as Program 2 by loop control distribution. This version can be run with an allotment of only three pages and still not generate page faults, except at the end of processing each set of pages. Note that in this case loop control can be distributed down to the level of individual statements.

Program 3

```

DO 1 I=1, N
   X(I)=Y(I-1)+A(I)
1   Y(I)=X(I)*Y(I-1)+B(I)

```

In general, loop distribution is possible down to the level of cyclic data dependence graphs (called π -blocks¹⁹) for individual loops. The nodes of a cyclic data dependence graph (each node representing a statement) are connected by directed arcs which form a cycle. An arc is drawn from one node to another if, at some instance, the statement represented by the first node must be executed before the statement of the second node. Thus, the loop of Program 3 cannot be further distributed since its two statements form a data dependence cycle and hence constitute a single π -block. Since such cycles seldom contain more than two statements in practice, the improvement in memory space of this method over standard paging is a factor proportional to the number of statements in the longest loop in a program. Note that in practice this may be comparable to the improvement obtained by standard paging over non-paged memory schemes, although I/O activities may increase in general.

In the past, a great deal of work has been done on the problem of extracting array operations from standard programs for purposes of compiling for high-speed array machines.¹⁸ Based on this, we have implemented a comprehensive FORTRAN program analyzer for speeding up FORTRAN programs, and have found that a very high percentage of FORTRAN loops can be broken into array expressions and linear recurrences by loop distribution (as in Program 2). An important key to our success has been in obtaining very accurate data dependence tests for subscripted variables inside loops. While a number of earlier attempts to solve this problem used only array names or simple subscript tests, we now use tests that in most cases are exact,⁷ i.e., we have necessary and sufficient tests for the data dependence of one subscripted variable on another, subject to a loop index set. This allows us to obtain a data dependence graph that has many fewer arcs (and more π -

blocks) than would be obtained by more naive tests and, in particular, allows the breaking of many false cycles. Thus, control in most graphs can be distributed to the level of individual assignment statements.

An outline of the complete transformation algorithm is shown in Figure 1. The first step, analysis, is done automatically by the FORTRAN program analyzer. During this step, data dependences are determined, and certain simplifying transformations are performed (e.g., DO-loop initial value and bound normalization).

Following analysis, statements are *clustered* depending on common data elements. Consider, for example, Program 4. Statements S_1 and S_3 are clustered together because array B is common to both statements. Clustering is done only within a loop however, so statement S_4 is not clustered with S_2 (yet) because they belong to different loops. Each cluster is called a *name partition* (NP). Loops are now distributed over NPs, as shown in Program 5. Notice that the loop could have been further distributed over the NP(S_1, S_3). However, while this would reduce the space requirement for each loop, it would increase the page faulting since each page of B would have to be fetched twice.

Following clustering, an attempt is made to *fuse* different loops together. Observe that the data in S_4 of Program 5 is a subset of the data of S_2 . Thus, these two loops can be fused (as shown in Program 6) without increasing the space required in the loop, but allowing a decrease in the total number of page faults.

Program 4

```

DO S3 I=1, N
S1   A(I)=B(I)+C(I)
S2   D(I)=E(I)+F(I)+X(I)
S3   G(I)=B(I)+H(I)
DO S4 J=1, N
S4   E(J)=D(J)*F(J)
      ↓ Clustering

```

Program 5

```

DO S3 I=1, N
S1   A(I)=B(I)+C(I)
S3   G(I)=B(I)+H(I)
DO S2 I=1, N
S2   D(I)=E(I)+F(I)+X(I)
DO S4 J=1, N
S4   E(J)=D(J)*F(J)
      ↓ Fusion

```

Program 6

```

DO S3 I=1, N
S1   A(I)=B(I)+C(I)
S3   G(I)=B(I)+H(I)
DO S4 I=1, N
S2   D(I)=E(I)+F(I)+X(I)
S4   E(I)=D(I)*F(I)

```

Since eventually we will distribute the loop control of an NP on its π -blocks, our next step is to simplify the data dependence graph of each NP by breaking any dependence

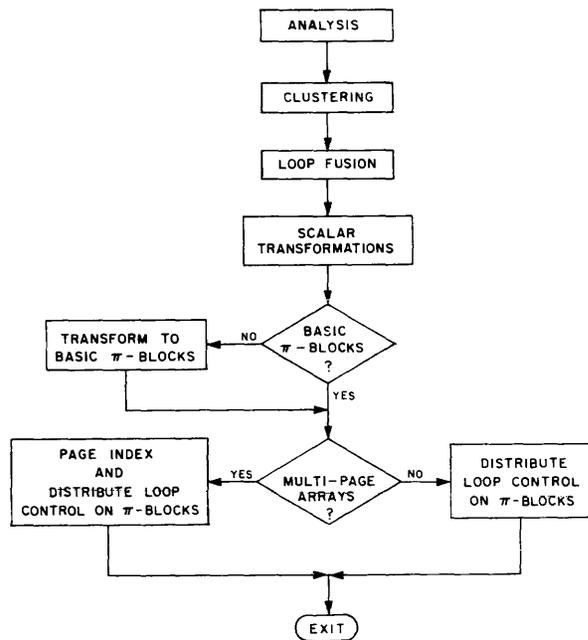


Figure 1—Flowchart of the transformation process.

arcs due to assignment statements to scalar variables. We use one of two techniques to handle such assignment statements—*forward-substitution* or *scalar expansion*. As an example, consider Program 7a.

Program 7a

```

DO S3 I=1, N
S1 T = (A(I)*C(I))/2
S2 D(I) = D(I)**2 - T**5
S3 F(I) = T*(A(I) - C(I)) + F(I)/C(I)

```

Because of S_1 , the data dependence graph of this NP is cyclic and there is one π -block. For this NP, the amount of memory allotment needed to obtain minimum I/O activity is four page frames. By substituting the right-hand side expression of S_1 in S_2 and S_3 , we can eliminate S_1 and we will have two π -blocks: $\pi_1 = S_2$ and $\pi_2 = S_3$. Note that the memory needed for each of these π -blocks is three page frames. Thus the space requirement of Program 7a can be dropped by a factor of $\frac{4}{3}$ by forward-substitution and then distributing the control on the π -blocks as in Program 7b.

Program 7b

```

DO S2 I=1, N
S2 D(I) = D(I)**2 - ((A(I)*C(I))/2)**5
DO S3 I=1, N
S3 F(I) = ((A(I)*C(I))/2)*(A(I) - C(I))
          + F(I)/C(I)

```

In other situations, forward-substitution might be impossible or undesirable (e.g., if it increases the space requirement of the program). In such cases, we use scalar expansion as shown in Program 8.

Program 8a

```

DO S3 I=1, N
S1 T = T + A(I)*E(I)
S2 A(I) = B(I)*C(I)
S3 B(I) = T + F(I)/D(I)

```

↓ Scalar expansion and loop distribution

Program 8b

```

DO S1 I=1, N
S1 T'(I) = T'(I-1) + A(I)*E(I)
DO S2 I=1, N
S2 A(I) = B(I)*C(I)
DO S3 I=1, N
S3 B(I) = T'(I) + F(I)/D(I)

```

Note that for Program 8a the memory requirement is six page frames while for Program 8b it is four (the maximum of the space requirement of the three π -blocks). Rules to be used in choosing between forward-substitution and scalar expansion (when both are possible) are discussed in Reference 3.

As mentioned earlier, distributing the control of an NP on its π -blocks will increase the page fault rate if the arrays referenced are multi-page arrays. To prevent this from happening, we apply the *page indexing* transformation to such loops. Program 8c shows the page-indexed distributed version of Program 8a. Basically, this transformation ensures that a page that is referenced in different π -blocks of an NP will not be removed from memory until it is used by all relevant π -blocks. (Additionally, by page-indexing we have reduced the size of the scalar expansion array in Program 8c to only z words instead of N .)

Program 8c

```

DO S3 IP=1, [N/Z]
  ILB = 1 + (IP-1)*Z
  IUB = MINIMUM(IP*Z, N)
DO S1 I=ILB, IUB
S1 T'(I MOD(Z)+1) = T'(I MOD(Z))
      + A(I)*E(I)
DO S2 I=ILB, IUB
S2 A(I) = B(I)*C(I)
DO S3 I=ILB, IUB
S3 B(I) = T'(I MOD(Z)+1) + F(I)/D(I)

```

Page indexing can be applied only to NPs which have basic π -blocks. A *basic π -block* is one with all of its statements at the same loop nesting depth. However, we have an algorithm for transforming non-basic π -blocks into basic π -blocks.³ After this is done, page-indexing is applied as before. Program 9a is a non-basic π -block (this is a Gaussian elimination program). In Program 9b, it is transformed to a basic π -block. Page-indexing is applied as shown in Program 9c. We use the same storage scheme for all multidimensional arrays of a program, the submatrix storage scheme.¹⁰ This is because this storage scheme has inherent advantages over the row or column-wise storage schemes as was shown in 10. We have developed tests to check the correctness of the

page-indexing transformation.³ Currently, we are looking into storing arrays using different schemes when the arrays are referenced in different uniform ways in the program.

Program 9a

```

DO S2   I1=1, N-1
DO S1   I2=(I1+1), N
S1     A(I2, I1) =A(I2, I1)/A(I1, I1)
DO S2   I3=(I1+1), N
DO S2   I4=(I1+1), N
S2     A(I4, I3) =A(I4, I3)
           -A(I4, I1)*A(I1, I3)
           ↓
           Non-basic to basic
           π-block

```

Program 9b

```

DO S2   I1=1, N-1
DO S2   I2=(I1+1), N
DO S2   I3=(I1+1), N
DO S2   I4=(I1+1), N
S1     IF (I3.EQ.(I1+1).AND.I4.EQ.(I1+1))
           A(I2, I1)=A(I2, I1)/A(I1, I1)
S2     IF (I2.EQ.N)
           A(I4, I3)=A(I4, I3)
           -A(I4, I1)*A(I1, I3)
           ↓
           Page indexing

```

Program 9c (Note we have substituted K for I_1 , J for I_3 , and I for I_2 . We assume RZ divides N .)

```

RZ=Z*.5
NP=N/RZ
DO S2 KP=1, NP
   KLB=1+(KP-1)*RZ
DO S2 JP=KP, NP
   JLB=1+(JP-1)*RZ
   JUB=JP*RZ
DO S2 IP=KP, NP
   ILB=1+(IP-1)*RZ
   IUB=IP*RZ
   IF (IP.EQ.KP) KUB=KP*RZ-1
   IF (IP.NE.KP) KUB=KP*RZ
DO S2 K=KLB, KUB
   IF (IP.EQ.KP) ILB=K+1
   IF (JP.EQ.KP) JLB=K+1
DO S2 J=JLB, JUB
   DO S2 I=ILB, IUB
       IF (J.EQ.JLB.AND.JP.EQ.KP)
           A(I, K)=A(I, K)/A(K, K)
S2     IF (J.LE.JUB) A(I, J)=A(I, J)
           -A(I, K)*A(K, J)

```

PRELIMINARY RESULTS

We define performance in terms of three criteria—space (m), time (measured in terms of page faults, $f(m)$), and space-time product ($f(m)m$). Notice that for simplicity we assume that computation time is negligible relative to the time for a page fault, so we can measure time in terms of

the number of page faults. In fact, our transformations do increase the CPU time for a program due to increased loop control overhead, redundancy (from forward-substitution of scalars), etc. However, much of this is invisible due to control-execution overlap, and the total increase in CPU time is generally negligible when compared to disk access time.

We define m_r to be the amount of memory required by a program in order to produce the minimum space-time product. More specifically, for the untransformed program this value is m_o , and for the transformed program it is m_t . Our choice of a memory allotment that minimizes space-time product is somewhat arbitrary, but is based on the intuitive idea that this will lead to maximum throughput and minimum turnaround in a multiprogrammed environment. In such a system, throughput and turnaround time are related by

$$\text{Throughput} = \frac{\text{average number of jobs present}}{\text{average job turnaround time}}$$

Roughly speaking, reducing page allotments as much as possible maximizes the average number of jobs present. Since reducing each job's page faults reduces the average job turnaround time, the transformations we carry out tend also to maximize throughput. Thus, for the fixed memory allotment case discussed above, our techniques improve both turnaround and throughput. In fact, however, m_r does not always lead to minimum page faulting as we shall see. Thus, in an I/O bound system, m_r may not produce optimal results. We shall discuss this problem in more detail later.

Regarding space, it is intuitively clear that if one can transform a program into a form that contains a set of π -block computations, then each of these π -block computations can be broken into a sequence of page-sized loops (using the page indexing transformation). The net effect would be to reduce the necessary page allotment for an entire transformed program to that required for the largest π -block computation in the program, m_t . The simplest π -blocks might be expected to contain one or two distinct arrays, while the maximum number per π -block in a program might be six or eight. Thus if one were successful, perhaps any program could be run with a data page allotment of at most eight pages.

Another important program characteristic is page faulting and how it would be affected by such transformations. For any given program loop, the number of page faults per iteration equals the number of arrays referenced in the loop, unless *all* pages are left in main memory for the entire loop execution, in which case page faulting occurs only after a number of iterations proportional to the page size. In this case the page allotment for the original program, m_o , would at best be equal to the largest number of arrays referenced in any loop in the program, in contrast to the m_t for the largest π -block obtained above. At worst, many more pages than array names would be required. For example, a matrix multiplication loop has three array names but needs an entire row and column of pages. Thus, one would expect that the number of required pages for any program to run well would be less for a transformed program, i.e., $m_t < m_o$.

By a combination of automatic and manual transformations, as well as a tracing program that handles FORTRAN programs compiled for the IBM/360, 370, we have obtained preliminary statistics for 17 FORTRAN programs, comprising a total of almost 1600 cards (excluding comments). These were selected from about 300 programs we have collected for various studies. The 17 programs contained a total of 200 DO-loops whose limits were supplied by users. The program generated over 1.4 million array element references. Most of the programs are numerical in nature but they are drawn from diverse application areas. One important criterion in our selection process was that the programs not contain too many statements, as our analysis procedures were rather time-consuming.

We have observed average values of $m_o=25$ and $m_t=4$. Thus, a space-saving of greater than six could be expected for these programs. However, since page faulting does increase somewhat for the transformed programs, the space-time product improvement is only about three or four (the mean is four and the median is three). Nevertheless, this implies a potential increase in throughput of a factor of three or four for a multiprogrammed system. Furthermore, for our programs the average space-time product improvement of standard paging over a non-paged main memory (one that allocates sufficient space for all arrays) was only a factor of about 2.5 (mean and median). Thus, the performance increase of our transformations over standard paging is comparable to (in fact, greater than) the performance increase of standard paging over a non-paged system.

It is important to realize that our m_o values were obtained by direct observation of the space-time product of our sample programs over a wide range of memory allotments. The difficulty of directly observing m_o (say, by compiler measurement) has been discussed in Reference 20, where possible correlation with the number of array names in a loop was rejected as impossible in practice. On the other hand, for our transformed programs there is a strong correlation between the number of array names and m_t ; the two are almost equal in most π -blocks.

In Figure 2 and Figure 3 we show typical page fault versus memory allotment and space-time product versus memory allotment curves, respectively. Note that both page fault curves approach approximately the same low level of page faulting. The transformed program approaches this low level much earlier than the original program. This, in turn, causes the space-time product of the transformed program to be substantially less than that of the original program over a wide range of memory allotments. Note that the transformed program's space-time product decreases monotonically to a minimum at m_t , then increases to meet the minimum for the original program at m_o and then both increase together as useless memory is allotted. However, the space-time products of original programs usually vary a great deal between one page and m_o , thus making the operating system's (OS) page allotment job nearly impossible unless m_o pages are allotted (the difficulty of knowing m_o was discussed above). Of course, allotting each program an efficient amount of space would also complicate the overall scheduling job of the OS. However, the transformed programs are much bet-

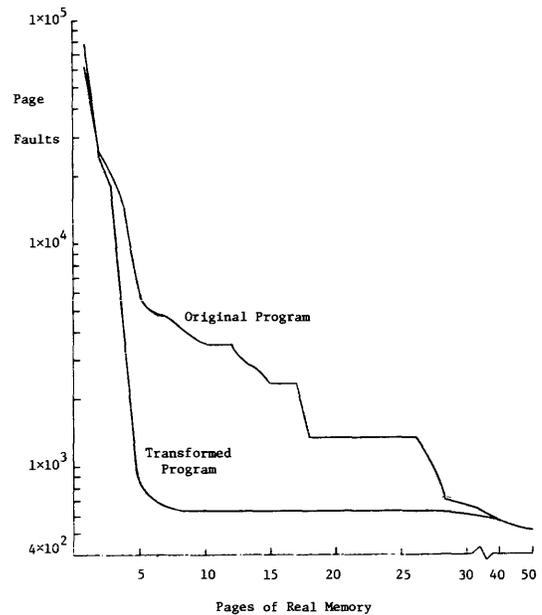


Figure 2—Page faults vs. memory allotment.

ter behaved, and any allotment in the region of m_t , say from four to eight pages will give a reasonably good space-time product for any of the programs we observed.

To compare averages using ideal memory allotments (m_o and m_t) is rather pointless because of the difficulty of achieving m_o allotments in practice. More realistic is a comparison of the performance of the original and transformed programs given fixed allotments, m_a , which are not necessarily optimal. This reflects the situation where space is allotted by an OS that does not know the optimal allocation. (Note, however, that it may be easier for a compiler to estimate m_t even though estimation of m_o is known to be difficult.) We have tabulated the ratio $m_a f_o(m_a) / 6 \cdot f_t(6)$, for $16 \leq m_a \leq 48$ with m_a increasing by increments of four. For $m_a=16$ the space-time product of transformed programs is improved over untransformed programs for all but one of the 17 programs (where it drops by a factor of .75), with an average improvement by a factor of 8.8. Fur-

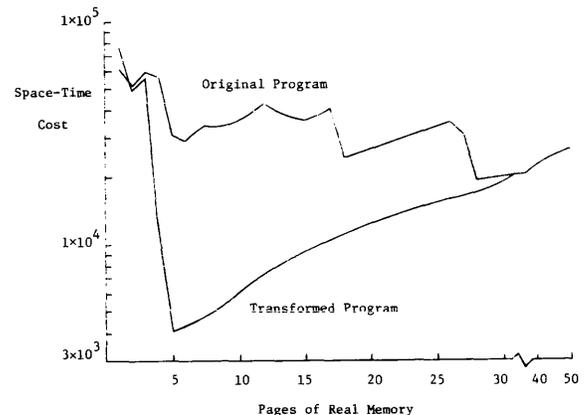


Figure 3—Space-time product vs. memory allotment.

thermore, in all but three cases, both space and page faults decrease (in two of these the page faults increase very slightly). As m_a increases, several other programs reach m_o and need no more pages, but averaging over those that need 28 pages (14 programs), the average space-time product improvement reaches a maximum of 12.9 with a median of seven. In fact, over the entire range of m_a and using four, six, or eight pages for the transformed programs, we achieve an average space-time product improvement over untransformed programs in the range of seven to over 12. Thus, we feel safe in concluding that by using our transformations, operating systems that use fixed memory allotments can achieve a decimal order of magnitude improvement in space-time product over standard paging techniques for the data of ordinary FORTRAN programs.

The above studies assumed a fixed memory allocation and a least-recently-used (LRU) page replacement algorithm. We have done similar studies assuming paging is done according to the working set policy, WS. For transformed programs, there was no difference between the cost of execution under LRU and WS. Moreover, several programs exhibited the working set anomalies.¹⁶ For more discussion on the results under WS, see References 3 and 1. For detailed measurements of the working set anomalies in FORTRAN programs, see Reference 2.

CONCLUSION

In this paper we presented an overview of compiler transformations which are aimed at the enhancement of the locality property of programs. Moreover, we presented a summary of preliminary experimental results which show that our techniques have good potential for achieving their goals. These results indicate that transformed programs are cheaper to execute, easier to manage, and simpler to model.⁴ For example, using simple and practical memory management policies, we observe a factor of 10 improvement in space-time cost over untransformed programs.

REFERENCES

1. Abu-Sufah, W., D. Kuck and D. Lawrie, "On the Performance Enhancement of Paging Systems through Program Analysis and Transformations," in preparation, 1979.
2. Abu-Sufah, W., and D. Padua-Haiek, "Measurements of the Working Set Anomalies in FORTRAN Programs," in preparation, 1979.
3. Abu-Sufah, W., "Improving the Performance of Virtual Memory Computers," Ph.D. thesis, Univ. of Ill. at Urb.-Champ., Dept. of Comput. Sci. Rpt. No. 78-945, November 1978.
4. Abu-Sufah, W., "Modeling the Behavior of FORTRAN-like Programs," in preparation, 1979.
5. Arvind, R. Y. Kain and E. Sadeh, "On Reference String Generation Processes," *Proc. Fourth ACM Symp. on Operating System Principles*, October 1973, pp. 80-87.
6. Batson, A. P., and A. W. Madison, "Characteristics of Program Localities," *Comm. of the ACM*, Vol. 9, No. 5, pp. 285-294, May 1976.
7. Banerjee, U., "Data Dependence in Ordinary Programs," M.S. thesis, Univ. of Ill. at Urb.-Champ., Dept. of Comput. Sci. Rpt. No. 76-837, November 1976.
8. Belady, L. A., "A Study of Replacement Algorithms for Virtual Storage Computers," *IBM Sys. J.*, Vol. 5, No. 2, 1966, pp. 78-101.
9. Chu, W. W., and H. Opderbeck, "The Page Fault Frequency Replacement Algorithm," *AFIPS Conf. Proc.*, 1972 FJCC, Vol. 41, 1972, pp. 547-609.
10. Coffman, E. G., and A. C. McKeller, "The Organization of Matrices and Matrix Operations in a Paged Multiprogramming Environment," *Comm. of the ACM*, Vol. 12, No. 3, March 1969, pp. 153-165.
11. Comeau, L. W., "A Study of the Effects of User Program Optimization in a Paging System," *Proc. ACM Symposium on Operating System Principles*, 1967.
12. Denning, P. J., "Thrashing and Its Cause and Prevention," *Proc. AFIPS Fall Joint Comput. Conf.*, Vol. 33, 1968, pp. 915-922.
13. Denning, P. J., "The Working Set Model for Program Behavior," *Comm. of the ACM*, Vol. 11, No. 5, May 1968, pp. 323-333.
14. Elshoff, J. L., "Some Programming Techniques for Processing Multidimensional Matrices in a Paging Environment," *Proc. of AFIPS Nat'l. Comput. Conf.*, 1974, pp. 185-193.
15. Ferrari, D., "The Improvement of Program Behavior," *IEEE Computer*, Vol. 9, No. 11, November 1976, pp. 39-47.
16. Franklin, M. A., and R. K. Gupta, "Working Set and Page Fault Frequency Paging Algorithms: A Performance Comparison," to appear in *IEEE Trans. on Comput.*, 1978.
17. Hatfield, D. J., and J. Gerald, "Program Restructuring for Virtual Memory," *IBM Sys. J.*, Vol. 10, No. 3, 1971, pp. 169-192.
18. Kuck, D. J., "A Survey of Parallel Machine Organization and Programming," *ACM Computing Surveys*, Vol. 9, No. 1, March 1977, pp. 29-59.
19. Kuck, D. J., Y. Muraoka and S. C. Chen, "On the Number of Operations Simultaneously Executable in FORTRAN-Like Programs and Their Resulting Speed-Up," *IEEE Trans. Comput.*, Vol. C-21, No. 12, December 1972, pp. 1293-1310.
20. Madison, A. W., and A. P. Batson, "Characterization of Program Localities," *Comm. of the ACM*, Vol. 19, No. 5, May 1976, pp. 285-294.
21. Smith, A. J., "Sequentiality and Prefetching in Data Base Systems," *IBM Res. Rpt. No. RJ 1743*, March 1976.
22. Trivedi, K. S., "Prepaging and Applications to Array Algorithms," *IEEE Trans. Comput.*, Vol. C-25, No. 9, Sept. 1976, pp. 915-921.
23. Trivedi, K. S., "On the Paging Performance of Array Algorithms," *IEEE Trans. Comput.*, Vol. C-26, No. 10, October 1977, pp. 938-947.

Analysis of data flow models using the SARA graph model of behavior*

by W. RUGGIERO**, G. ESTRIN, R. FENCHEL, R. RAZOUK, D. SCHWABE and M. VERNON

University of California at Los Angeles
Los Angeles, California

A number of investigators have continued to discuss application of asynchronous techniques to improve the computational power of computing systems.^{2,12,5,9} In fact the need for asynchronous design techniques arose in the earliest machines^{26,6,20} which introduced parallel handling of bits in a number and overlapping of independent operations. The concept of distributed autonomous concurrent processors was essential to the visionary architecture proposed by Holland¹⁶ to support "operating programs floating in a sea of hardware." The importance of such concurrent and asynchronous systems has increased recently because of the availability of entire processing units on one or a few chips and the potential cost reduction of those units. Moreover, it is expected that work on VLSI technology will permit even more complex systems to be reduced to silicon if their design and market analysis have been validated sufficiently to justify the cost. The need for validation prior to costly physical implementation has increased the value of methods and tools which support modeling and analysis.

SARA (System ARchitectures Apprentice) is such a supported methodology particularly applicable to multilevel modeling of concurrent systems. Dennis has proposed computation structures for concurrent system design which have aroused considerable interest. This work shows how SARA tools may be used to analyze data flow models utilizing Dennis' computation structure. This work refers to the latter as the Dennis Data Flow (DDF) Model.

In the first part of this paper we establish relationships between primitives of two models—the Dennis Data Flow Model and the SARA Graph Model of Behavior (GMB). The second part of the paper presents an example showing how SARA tools can be used to construct and simulate a data flow model. The discussion and the example should help to increase the reader's understanding of both models.

This opens the way for application of supported multilevel design methodologies, like that supported by SARA,⁸ to higher-level design of systems incorporating devices which implement data flow primitives.

* This research was supported by the U.S. Department of Energy, Contract No. EY-76-S-03-0034, PA214.

** Formerly UCLA Computer Science Dept., now at Laboratorio de Sistemas Digitais, Depart. de Eng. Electrica, Escola Politecnica-U.S.P. Caixa Postal 8174-S.P. 30 000 Brasil.

RELATIONSHIP BETWEEN DDF AND GMB

Token flow models

The Graph Model of Behavior (GMB)¹¹ and the Dennis Data Flow (DDF) Model⁵ can be considered token flow models which were invented to help synthesize asynchronous concurrent systems. A token flow model uses a directed graph composed of nodes and arcs to describe the static component of a system behavior. Tokens, which are initially placed on the arcs, flow through the graph, activating and deactivating nodes. In this way, selected dynamic behavior of a system can be modeled and observed. Such models are most suitable for describing the behavior of concurrent and asynchronous systems in which some events may occur concurrently, but those occurrences must be controlled to satisfy constraints, i.e. precedence and frequency.

The GMB was developed in a search for a natural, simple and powerful method for describing and analyzing the flow of data and control in systems. The GMB was derived from the bigraph model of computation (GMC)¹³ which formalized earlier work at UCLA.^{7,18} The GMB was influenced by the LOGOS work at Case Western Reserve.²⁴

A simple data flow model was studied in the basic work by Karp and Miller.¹⁷ Dennis and Rodriguez²³ developed program graphs which were later revised and analyzed as a form of parallel program schema. A chain of investigators in Dennis' information structure group has continued exploration of fundamental data flow primitives and architectures. In this paper we concentrate on the data flow model developed by Dennis. Other noteworthy data flow models have been reported in the work of Adams,¹ Arvind - Gostelow,¹⁵ Bahrs³ and Kosinski.¹⁹

*Dennis' data flow primitives*⁵

A data flow language is a machine language for expressing computations in which an instruction executes when and only when all operands needed for that instruction become available. The instructions, at whatever level they might exist, are purely functions and produce no side effects. Dennis' data flow language seeks to define a scheme of repre-

sensation that exposes concurrency while maintaining a guarantee of determinacy. This language considers only programs that compute a set of output values from a given set of input values and that define the functional dependence of output values on input values.

An elementary data flow program can be represented as a bipartite directed graph where the two types of nodes are called links and actors. The arcs of a data flow program should be regarded as channels through which tokens flow carrying data values. A data flow token is a primitive concept. It is better described as a pair $\langle e_r, v \rangle$; e_r is an enable flag and v is a data value. The enable flag signals the presence of the token which carries a data value to be used in the computation.

A DDF link cannot have more than one incident arc, whereas it may have more than one emanating arc. A distinguished input link is one that has no incident arcs; a distinguished output link has no emanating arcs. There are two kinds of link nodes—data link and control link. Figure 1 shows the kinds of actor nodes from which elementary data flow programs are constructed. The *T*-gate, *F*-gate and Merge actors are called control actors. The Or, And and Not actors are called boolean actors.

The execution of a data flow program is described by a sequence of snapshots; each snapshot shows the data flow

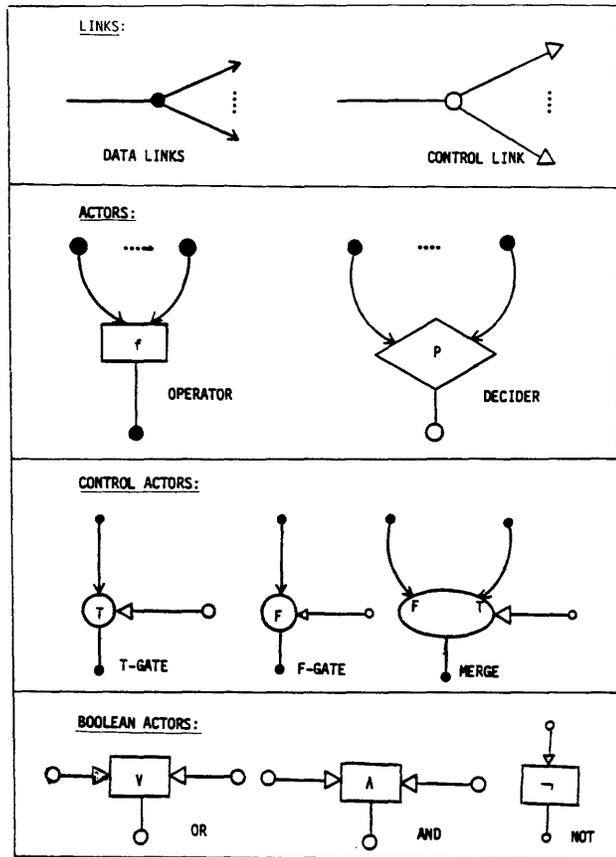


Figure 1—DDF nodes.

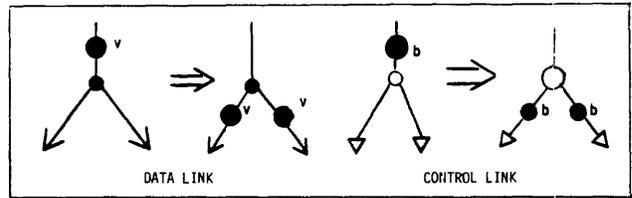


Figure 2—Firing rules for link nodes.

program with tokens and associated values placed on some arcs. In the case of control arcs, the associated values are of the type *truth* = {true, false}; for data arcs, the values are of the types *integer*, *real* or *string*. Execution of a data flow program advances from one snapshot to the next through the firing of some randomly selected link or actor that is enabled in the earlier snapshot. Except for the Merge actor, a node is enabled when all input arcs have a token and there is no token on any output arc. The Merge actor is enabled if there is no token on the output arc and either it has a token with value true on the input control arc and a token on the true arc input or it has a token with value false on the input control arc and a token on the false arc input. When a node fires it removes all enabling tokens from its input arcs and places tokens on its output arcs. An exception is made for the *T*-gate and *F*-gate. If the token value on the input control arc of a *T*-gate is False, the token on the input data arc is removed but no token will be placed on the output data arc. The same happens with an *F*-gate and a True value on the input control arc.

The value associated with an output token is a function of the values of the enabling input tokens. Firing an Operator applies the function denoted by the symbol written in the Operator to the set of values associated with the tokens on the input arcs and associates the resulting value with the

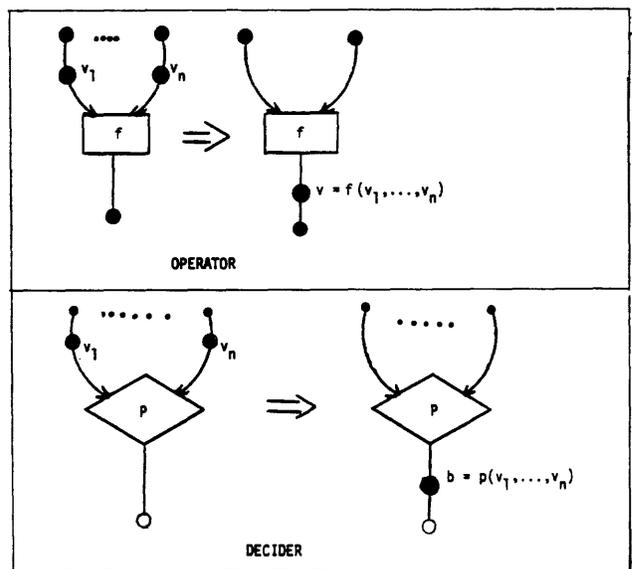


Figure 3—Firing rules for operators and deciders.

token placed on the output arc. Firing a Decider has a similar effect, but the symbol in the Decider denotes a predicate and a control value is associated with the output token. Figures 2, 3 and 4 show the effects of firing.

Relating the two models

In the most general terms, information processing systems can be considered as flow and transformation systems. Basically, there are two commodities which flow during a processing activity in those systems—one is the control over resources and data and the other is the data itself.

The Graph Model of Behavior (GMB)^{10,21,22} is a fundamental model incorporated in the SARA (System ARchitects Apprentice) methodology⁸ for multilevel design of concurrent systems. The GMB tools provide languages for modeling the behavior of a digital system in three related domains—control, data and interpretation.

The control domain is concerned only with the control flow aspects of the system. It utilizes a directed graph (called the control-graph) where nodes model steps in the computation and arcs model precedence. Tokens flowing through the control graph establish activation conditions for control nodes. A node is activated if a simple function (a combination of +, * and a weight) holds for the tokens on the incoming arcs. Activation amounts to removing the activat-

ing tokens and, upon completion of the activated process, adding tokens to the outgoing arcs according to a similar simple function.

The data domain describes flow of data and access capability constraints. A directed bipartite graph (called the data graph) is used to show how input data streams can flow through the system and where they are transformed (processed) in order to generate output data streams. The data graph describes the organization of data places (called data sets) and computation points (called processors). Data arcs describe allowable directed data paths between processors and data-sets.

The third domain, the interpretation, defines the format of data-sets, the format of data which flows along the data-arcs and the specific procedures to be carried out by activated processors. The current implementation of the GMB simulator uses an interpretation language which is an extension of PL/1 (called PLIP), but different interpretation languages could be introduced to enhance flexibility.

The three domains are associated as follows: Each control node is associated with at most one controlled processor in the data graph. Each controlled processor is associated with a unique (non-empty) set of control nodes in the control graph. The activation of a control node implies the non-preemptable activation of the associated controlled processor (if any). All concurrent activations of control nodes that are associated with the same controlled processor imply a sequence of activations of that controlled processor. The order of these activations is random. Segments of interpretation define the data structure associated with each data arc and the specific computation that is executed by each controlled or uncontrolled processor.

The GMB operates as follows: The "token machine" activates enabled nodes in the control graph. Control nodes activate controlled processors in the data graph while changes in the distinguished inputs activate uncontrolled processors in the data graph. The controlled and uncontrolled processors cause execution of code segments in the interpretation domain. The code segments cause data transfers (possibly with transformations) between data sets. Controlled processors may feed information back to the control domain to determine choice of output branching of the associated control nodes. The GMB can be used in a flexible way for the following purposes: a designer may choose to explicate more or less control flow; a designer may choose to vary the amount of data flow abstraction; a designer may choose to utilize different interpretation languages offering different powers of abstraction. Tables 1 and 2 tersely summarize properties of the current set of structural and behavioral modeling primitives. Table 3 summarizes properties of proposed extended primitives.²⁵ The SL1, GMB and PLIP languages (which support multi-level modeling) and a simulator (which supports analysis) are implemented at both UCLA and MIT-MULTICS.

In order to illustrate and make more meaningful comparison between the two models, let us first express the data flow primitives in terms of GMB primitives. Subsequently, some GMB constructs are expressed in terms of data flow primitives.

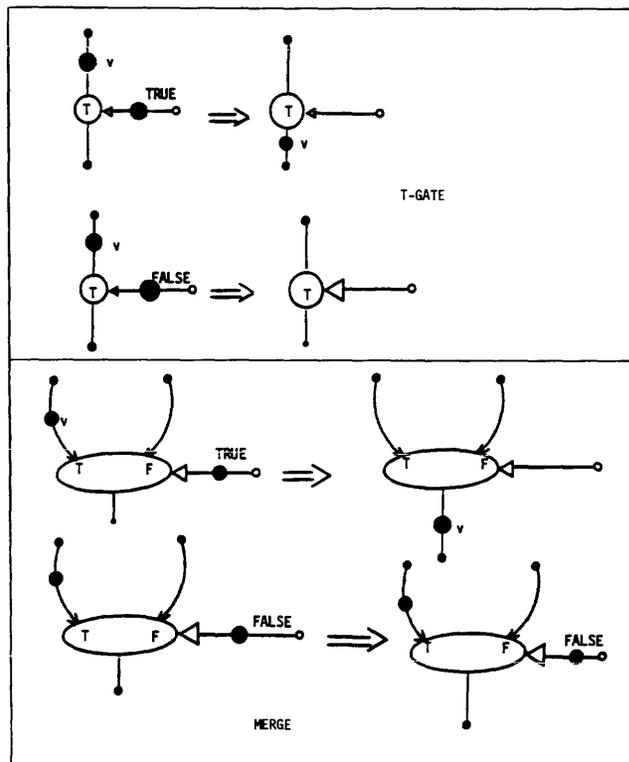


Figure 4—Firing rules for control actors.

TABLE I.—Modeling Primitives

STRUCTURAL PRIMITIVES	TYPE	GRAPHICAL	MACHINE PROCESSABLE
A NAMED MODULE REPRESENTS AN OBJECT WHOSE INTERNAL, FULLY-NESTED STRUCTURE IS HIDDEN FROM THE OUTSIDE. A MODULE'S ONLY POSSIBLE COMMUNICATION WITH THE OUTSIDE IS THROUGH A SOCKET. OTHERWISE, A MODULE'S NAME IS KNOWN ONLY TO THE STRUCTURE WITHIN WHICH IT IS NESTED.			NAME_C (A,B,C)
A NAMED SOCKET IS PART OF A MODULE BUT THE SOCKET'S NAME IS KNOWN BOTH INSIDE AND OUTSIDE OF ITS HOST MODULE.			NAME_C (A<@SA>, B<@SB>, C<@SC>)
A NAMED INTERCONNECTION IS A STRUCTURE WHICH CONNECTS TWO OR MORE SOCKETS.			NAME_C (A<@SA>, B<@SB>, C<@SC>, LINK : A@SA-B@SB-C@SC)

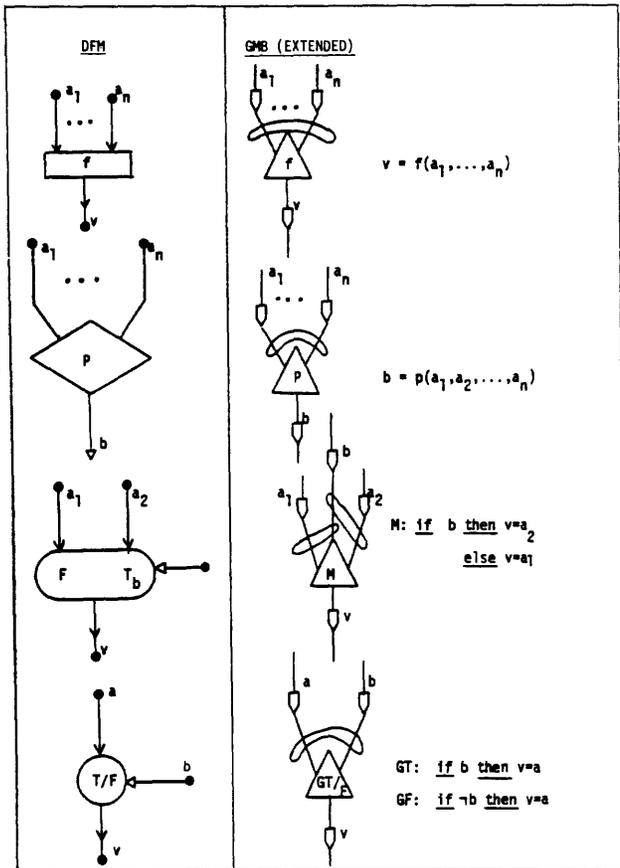


Figure 5—DDF to GMB using uncontrolled processors and data links.

Expressing data flow primitives in terms of GMB primitives

In general, any DDF can be abstracted into an uncontrolled processor where the interpretation of this processor

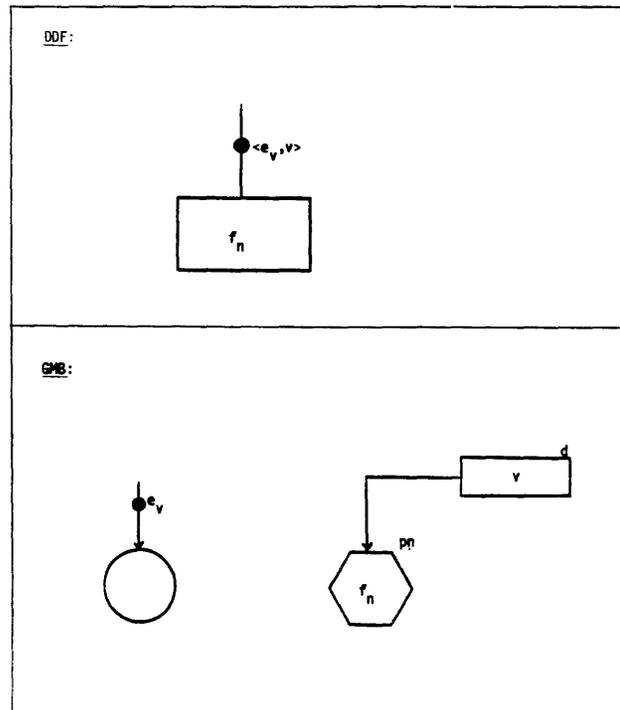
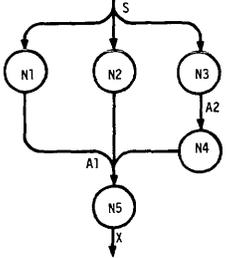
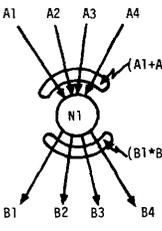
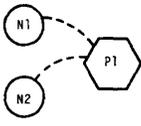
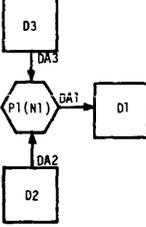


Figure 6—A GMB representation of a DDF token.

TABLE II.—Behavioral Modeling Primitives

BEHAVIORAL PRIMITIVES	TYPE	GRAPHICAL	MACHINE PROCESSABLE
<p>A NAMED CONTROL NODE REPRESENTS A STEP IN A PROCESS BEING MODELED. A CONTROLLED DATA PROCESSOR (SEE BELOW) MAY BE ASSOCIATED WITH A NODE TO PROVIDE INTERPRETATION OF THE PROCESS.</p> <p>EXAMPLE: A NODE N1 HAS A SINGLE ENTRY ARC S AND A SINGLE EXIT ARC X.</p>			<pre>@CONTROL_GRAPH; @NODES N1; @ARCS S,X; N1 (S:X); @END;</pre>
<p>A NAMED DIRECTED CONTROL ARC REPRESENTS NON-VOLATILE PRECEDENCE RELATIONS BETWEEN SETS OF NODES. IF THERE IS MORE THAN ONE SOURCE OR DESTINATION NODE THE ARC IS CALLED COMPLEX; OTHERWISE IT IS CALLED SIMPLE. AN ENABLING TOKEN IS PLACED ON AN ARC EITHER AS A STARTING STATE OR UPON TERMINATION OF ANY OF ITS SOURCE NODES. WHEN A NODE IS INITIATED, ITS ENABLING TOKENS ARE ABSORBED.</p> <p>EXAMPLE: A2 AND X ARE SIMPLE CONTROL ARCS. A1 IS A COMPLEX CONTROL ARC WHOSE SOURCE SET IS NODES N1, N2 AND N4 AND WHOSE DESTINATION SET IS N5. S IS AN INCOMING COMPLEX ARC WHOSE DESTINATION SET IS N1, N2, AND N3. IF THERE WERE AN INITIAL TOKEN ON S, THE TOKEN MACHINE MECHANISM WOULD NON-DETERMINISTICALLY ENABLE N1 OR N2 OR N3 AND THE TOKEN WOULD BE ABSORBED.</p>			<pre>@CONTROL_GRAPH; @NODES N1,N2,N3,N4,N5; @ARCS S,A1,A2,X; N1 (S:A1); N2 (S:A1); N3 (S:A2); N4 (A2:A1); N5 (A1:X); @END;</pre>
<p>INPUT CONTROL LOGIC</p> <p>A LOGICAL RELATION AMONG THE INPUT ARCS TO A NODE SPECIFIES THE PRECEDENCE CONDITIONS THAT MUST BE SATISFIED BY TOKEN STATES FOR THE NODE TO BE INITIATED. TOKENS FROM THE INITIATING ARCS WHICH SATISFY THE INPUT RELATIONS ARE ABSORBED BY THE TOKEN MACHINE. TOKENS ARE ABSORBED FROM ONE OF AN INITIATING ARC SET GOVERNED BY AN OR RELATION IN A MANNER ESTABLISHED IN THE TOKEN MACHINE AND FROM ALL MEMBERS OF AN INITIATING ARC SET GOVERNED BY AN AND RELATION.</p> <p>EXAMPLE: IF ENABLING TOKENS EXIST ON EITHER A1 OR A2 AND ON EITHER A3 OR A4 THEN N1 CAN BE INITIATED.</p> <p>OUTPUT CONTROL LOGIC</p> <p>A LOGICAL RELATION AMONG THE OUTPUT ARCS SPECIFIES WHICH ARCS HAVE TOKENS PLACED UPON THEM WHEN A CONTROL NODE IS TERMINATED. WHEN AN EXCLUSIVE OR OUTPUT RELATION HOLDS, A DATA PROCESSOR INTERPRETATION MUST DECIDE WHICH ARC RECEIVES A TOKEN. WHEN AN AND RELATION HOLDS ALL OUTPUT ARCS RECEIVE TOKENS.</p> <p>EXAMPLE: WHEN N1 TERMINATES, ITS ASSOCIATED CONTROLLED DATA PROCESSOR WILL HAVE DECIDED WHETHER TOKENS ARE TO BE PLACED ON B1 AND B2 OR ON B3 AND B4.</p>			<pre>INPUT: OUTPUT CONTROL LOGIC @CONTROL_GRAPH; @NODES N1; @ARCS A1,A2,A3,A4,B1,B2,B3,B4; N1((A1+A2) * (A3+A4); (B1+B2) * (B3+B4)); @END;</pre>
<p>A NAMED CONTROLLED DATA PROCESSOR REPRESENTS A DATA TRANSFORMATION OBJECT WHICH IS ACTIVATED WHEN AN ASSOCIATED CONTROL NODE IS INITIATED. E.G., PROCESSOR P1 IS INITIATED WHENEVER EITHER N1 OR N2 IS INITIATED. WHEN PROCESSOR P1 TERMINATES IT CAUSES TOKENS TO BE PLACED ON OUTPUT ARCS OF THE CONTROL NODE WHICH INITIATED IT. AN INTERPRETATION OF THE DATA TRANSFORMATION AND OTHER PARAMETERS SUCH AS THE DELAY OR RESOURCE REQUIREMENTS CAN BE ASSOCIATED WITH THE DATA PROCESSOR.</p> <p>EXAMPLE: PROCESSOR P1 HAS A RANDOM DELAY ASSOCIATED WITH IT. IRAND IS A BUILT-IN FUNCTION. THE CONTROL GRAPH CARRIES THE BURDEN OF GUARANTEEING THAT N1 AND N2 ARE ENABLED IN A DESIRED SEQUENCE. OTHERWISE THEY WILL BE ACTIVATED IN A NON-DETERMINISTIC ORDER AND THE SIMULATOR WILL SHOW POSSIBLE CONTENTION.</p>			<pre>@DATA_GRAPH; @PROCESSOR P1 (N1,N2); @END;</pre> <p>PLIP INTERPRETATION</p> <pre>@PROCESSOR P1; DCL IRAND ENTRY(FIXED BIN(31))RETURNS (FIXED BIN(31)); /*RANDOM # GENERATOR*/ DCL NUMBER FIXED BIN(31); NUMBER=IRAND(1,2) /*PICK AN INTEGER: 1 OR 2*/ IF NUMBER=1 THEN @OUTPUT_ARCS= 'B1,B2'; ELSE @OUTPUT_ARCS='B3,B4'; @DELAY=IRAND (10,100); /*PICK RANDOM DELAY FROM 10 TO 100*/ @ENDPROCESSOR;</pre>
<p>A NAMED UNCONTROLLED DATA PROCESSOR REPRESENTS A DATA TRANSFORMER WHICH PROVIDES, AT ITS OUTPUT, STATED FUNCTIONS OF ITS INPUTS INDEPENDENT OF CONTROL NODE STATES. IN THE DATA GRAPH AN UNCONTROLLED PROCESSOR IS IDENTIFIED BY PROVIDING AN EXPLICIT DECLARATION. AN INTERPRETATION OF THE DATA TRANSFORMATIONS AND OTHER PARAMETERS MAY BE ASSOCIATED WITH IT IN AN IDENTICAL MANNER TO THE CONTROLLED PROCESSOR.</p>			<pre>@DATA_GRAPH; @UNCONTROLLED_PROCESSORS U1; @END;</pre>
<p>A NAMED DATA SET REPRESENTS A PASSIVE COLLECTION OF DATA. DATA STRUCTURE MAY BE ASSOCIATED WITH A DATASET. ALL PL/1 DECLARATIONS NOT CONTAINING SCOPE OR STORAGE CLASS ATTRIBUTES ARE ACCEPTED AS DEFINITIONS OF DATA SETS. CHARACTER STRINGS CANNOT HAVE THE VARYING ATTRIBUTE.</p> <p>EXAMPLE: THE DATASET D1 IS A SIX-DECIMAL-DIGIT COMPLEX FLOATING POINT NUMBER.</p>			<pre>@DATA_GRAPH; @DATASETS D1; @END;</pre> <p>PLIP INTERPRETATION</p> <pre>@TEMPLATE (1) D1 COMPLEX FLOAT DECIMAL(6);</pre>
<p>A NAMED DATA ARC STATICALLY BINDS DATA PROCESSORS AND DATASETS. A DATA PROCESSOR HAS READ OR WRITE ACCESS TO A DATA SET IF THE ARROW POINTS TO OR FROM THE DATA PROCESSOR RESPECTIVELY.</p> <p>EXAMPLE: PROCESSOR P1 IS INITIATED BY CONTROL NODE N1. P1 READS DATA FROM DATASETS D2 AND D3 AND WRITES THEIR SUM INTO DATASET D1.</p>			<pre>@DATA_GRAPH; @PROCESSORS P1(N1); @DATASETS D1, D2, D3; @ARCS DA1, DA2, DA3; DA3 (D3:P1); DA2 (D2:P1); DA1 (P1:D1); @END;</pre> <p>PLIP INTERPRETATION</p> <pre>@TEMPLATE (DA1, DA2, DA3) T FIXED BIN(31); @PROCESSOR P1; @READ D3 @FROM DA3; @READ D2 @FROM DA2; D1 = D2+D3; @WRITE D1 @TO DA1 @AFTER 10; @ENDPROCESSOR;</pre>

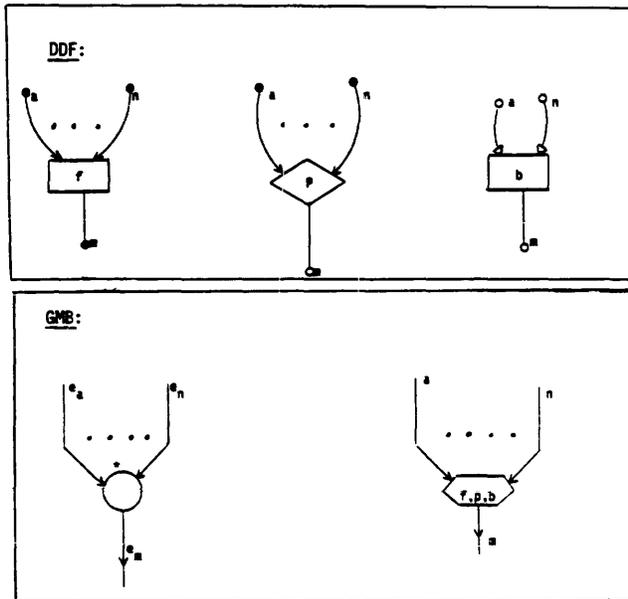


Figure 9—GMB representation of an elementary actor.

controlled processors and data links (see definitions in Tables 2, 3) as shown in Figure 5. The GMB data links have internal storage, are written into just before termination of a processor and are read from only immediately after initiation. Another approach is to describe the data flow behavior by separating the control and the data parts into a GMB CG and a GMB DG respectively. For the rest of this section we assume the latter approach because it gives more insight into the properties of the two models. For practical purposes, a data graph-oriented description of a DDF is usually preferable, (and is used in the demonstration example of Part B); however, the control-oriented description is more suitable for analysis purposes. In a data flow program the flow of control and the flow of data are shown together in the same graph.

As shown in Figure 6, the DDF enabling flag, e_r , is, in GMB terms, an input token for a node n in the control graph. It signals the presence of a data value v in a data-set d . A controlled processor pn associated with a control node n then has read access to the data-set d .

Let us consider, in a data flow program a link node l with one input arc and m output arcs (Figure 7). The equivalent

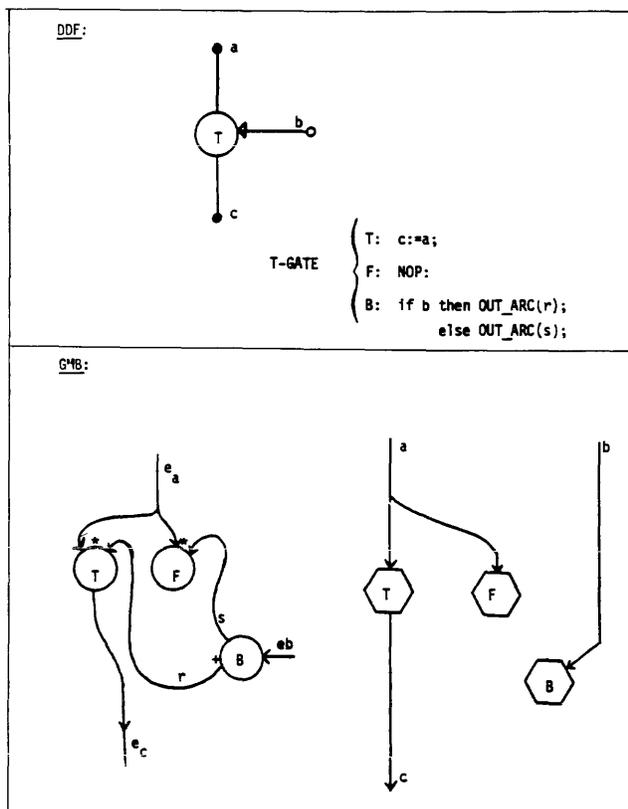


Figure 10—GMB representation of a GATE actor.

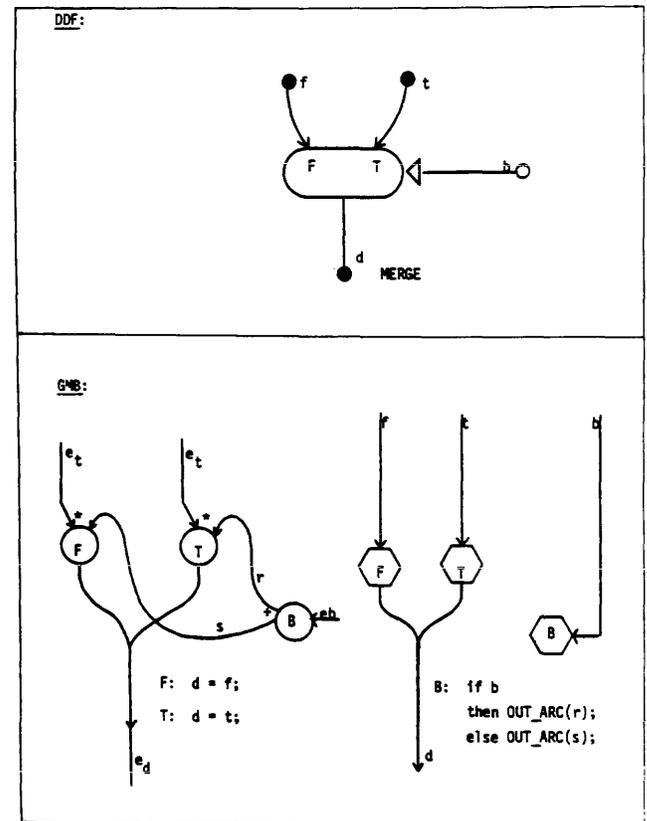


Figure 11—GMB representation of a MERGE actor.

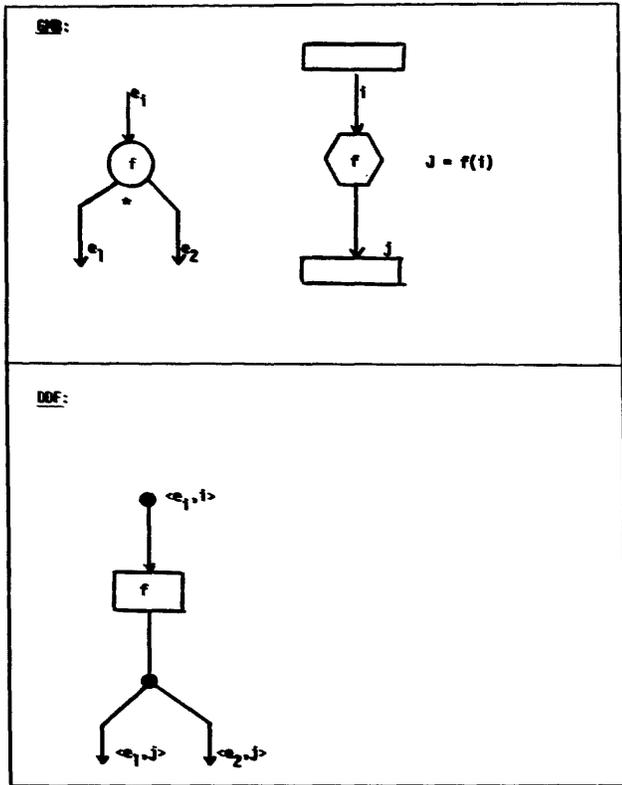


Figure 12—DDF representation of GMB fork construct.

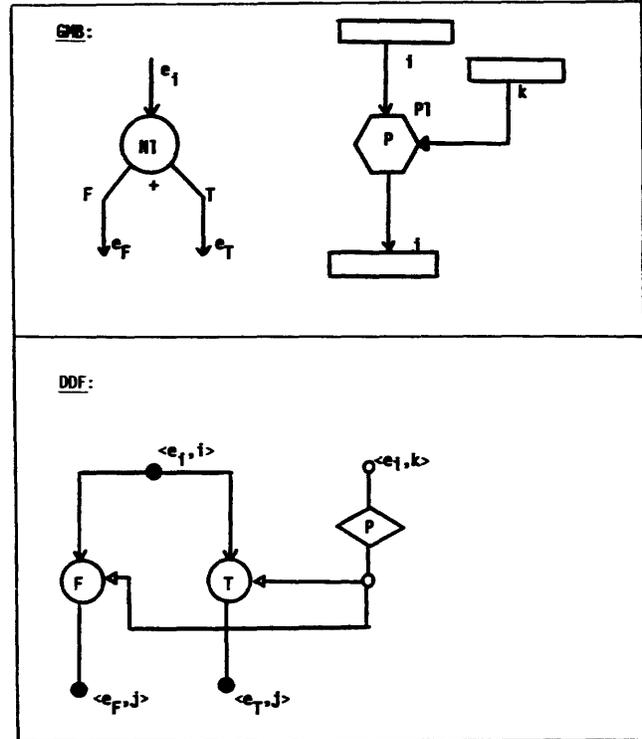


Figure 14—DDF representation of GMB switch construct.

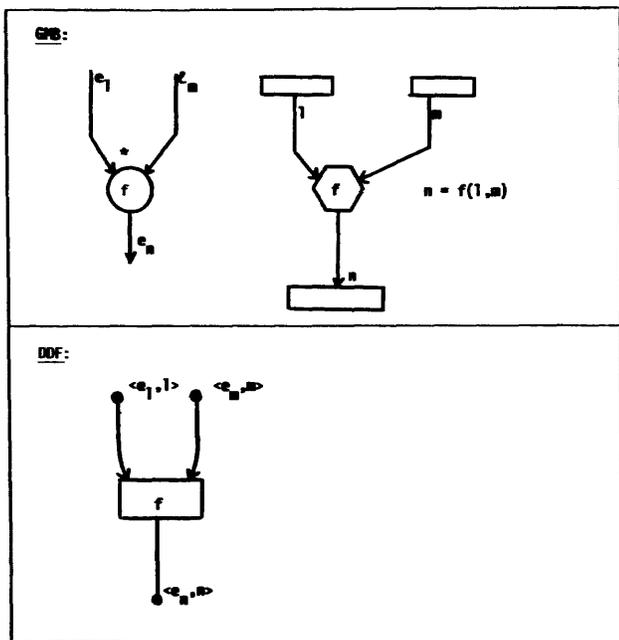


Figure 13—DDF representation of GMB join construct.

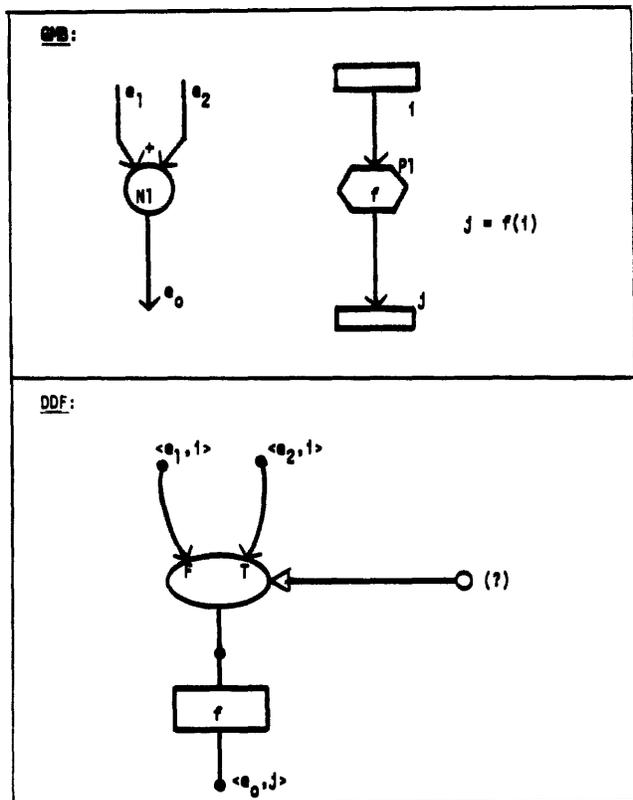


Figure 15—DDF representation of GMB union construct.

representation in GMB will have

- A data set d with one input data arc and m distinct output data arcs.
- A control node n with one input control arc and m output control arcs related by an "*" operator in the output logic expression of n .
- If m is equal to "1," node n does not need to exist. It is reduced to just a control arc.
- The control node n does not have any controlled processor associated with it in the data graph.

Except for control actors, all elementary DDF primitives can be represented in GMB using only one control node and one controlled processor. The DDF control actors represent more complex computations.

The DDF interpreter requires an elaborate operation to decide which node to activate next. It needs some kind of acknowledge token which flows internally in the interpreter. In order to effect the same behavior in GMB, the acknowledge tokens may be shown explicitly in the control graph (see Figure 8), may be included in every GMB processor interpretation or may be placed as a burden on the token

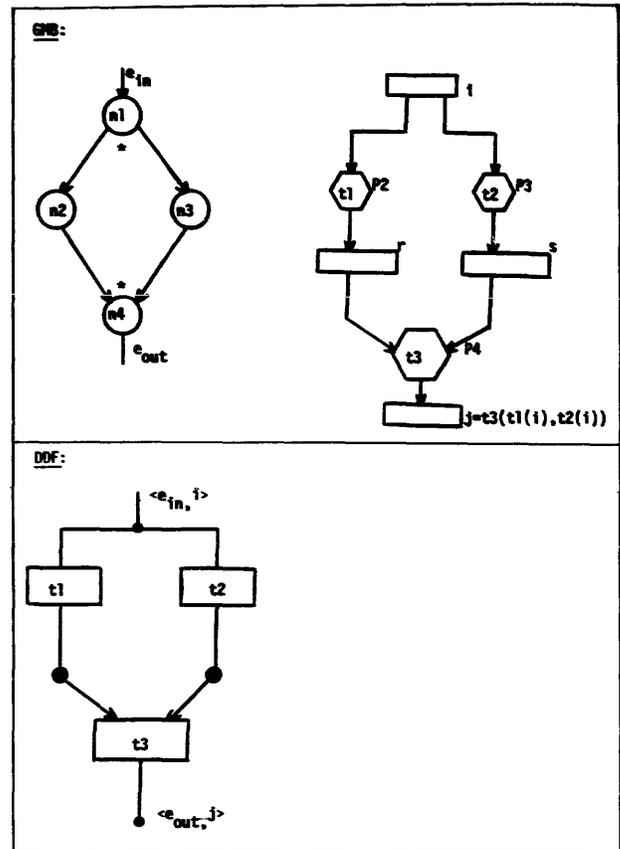


Figure 17—DDF representation of GMB Parbegin_Parend construct.

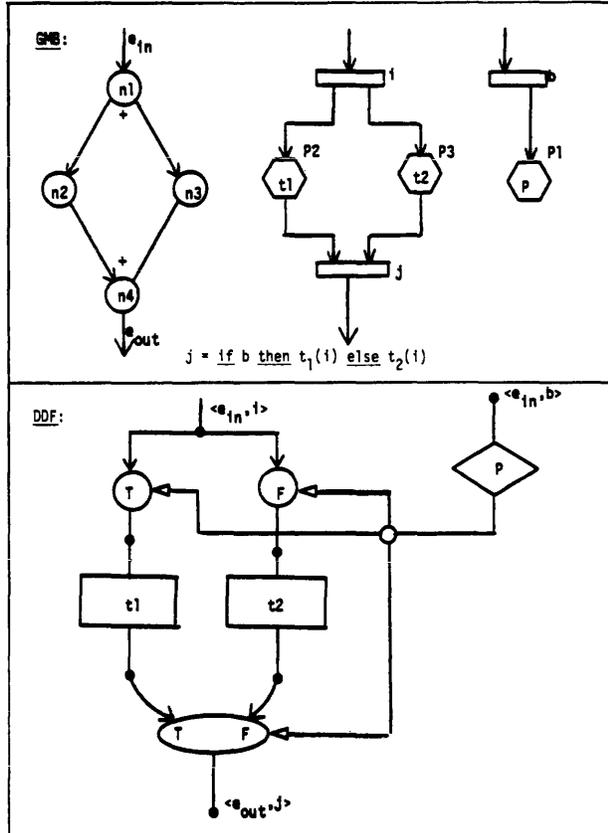


Figure 16—DDF representation of GMB If_Then_Else construct.

machine. The last approach would make the token machine equivalent to the DDF interpreter.

The use of explicit acknowledge tokens in a GMB makes the description much less readable. In the remainder of this section, we neglect the acknowledge tokens required in the GMB equivalent expression of the DDF primitives so as to focus attention on the relationship between the two models. Figure 9 shows how actors, except for control actors, are represented in GMB. Figures 10 and 11 show the representation of Gate and Merge control actors respectively.

The GMB representation of a Gate actor shows clearly (Figure 10) that this primitive has a control flow description which is not properly terminating.¹³ In GMB terms, the Gate actor is not considered "well behaved" and its presence may complicate the verification of a DDF model.

Finally, we should observe that there is no need to distinguish between data for different data types (boolean and numeric) in a GMB model.

Expressing some GMB constructs in terms of DDF primitives

We do not attempt the expression of non-deterministic GMB constructs in terms of DDF primitives because the set

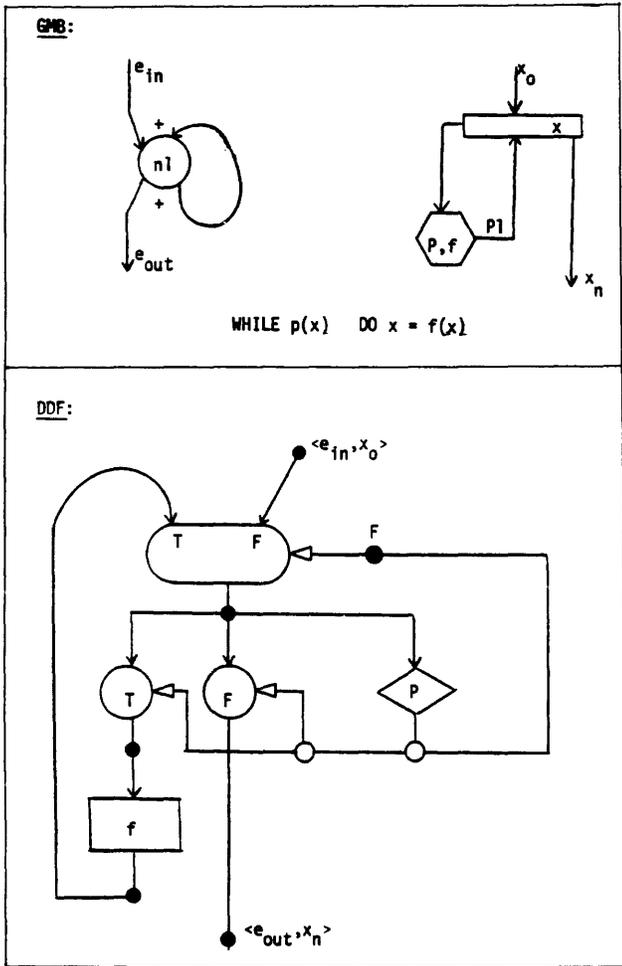


Figure 18—DDF representation of GMB Do_While construct.

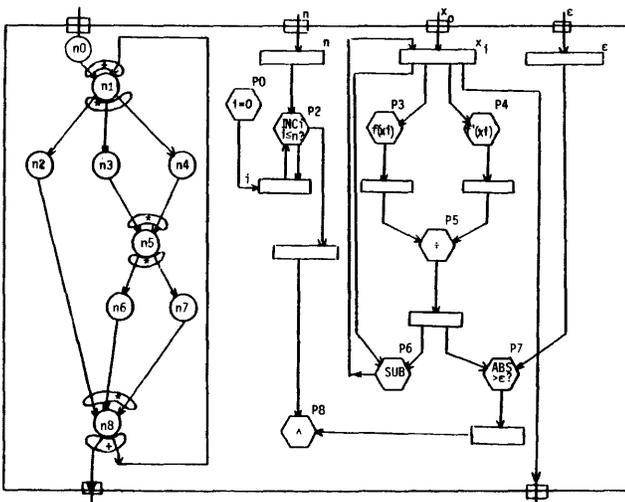


Figure 19—GMB solution of the root of f by Newton approximation, $x_{i+1} = x_i - f(x_i) / f'(x_i)$.

of programs intended to be expressed by the DDF language includes only deterministic programs. But, we should point out here, that the lack of non-determinism is one of the main weaknesses of DDF language when applied in a real-time programming environment. Some important problems in real-time processing such as mutual exclusion, synchronization and resource management, are in essence non-deterministic problems. The complex control arc is the general form of expressing non-determinism in a GMB program.

Now let us see the equivalent expression, using DDF primitives, of some deterministic GMB constructs. Figures 12-18 show respectively the Fork, Join, Switch, Union, If_Then_Else, Parbegin_Parend and Do_While constructs in terms of DDF primitives.

The equivalent expression of the Union construct (Figure 15) (which is another form of non-determinism in GMB) presented a problem when described in DDF terms—namely the control input of the Merge actor was not defined. This does not seem to be a problem in deterministic programs, because this construct should be used as shown in the If_Then_Else and Do_While constructs (Figures 16 and 18).

Looking at Figure 18, where a sequential construct is presented, we note flexibility of the GMB program in including all the sequential processing inside of a controlled processor. To describe this purely sequential construct in DDF we need the full DDF interpreter mechanism, which was designed to handle concurrent operations. The capability to embody segments of sequential code in elementary primitives, without involving the elaborated mechanisms of the token machine, represents a great advantage for GMB programs. The programmer may specify the use of an extra processor only when the problem really requires it, i.e., when there are some concurrent operations to be performed.

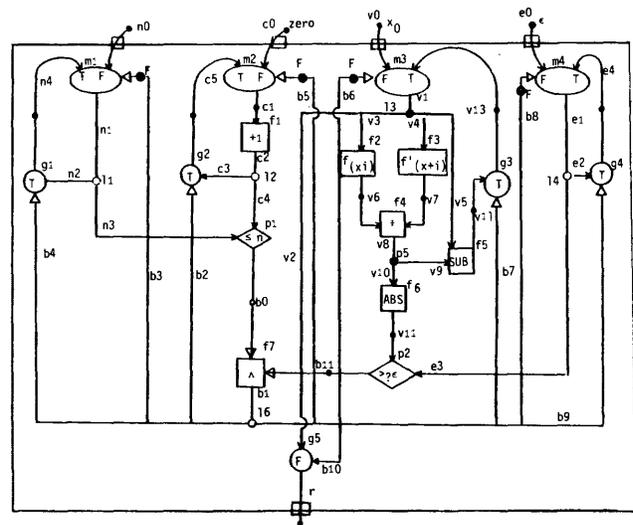


Figure 20—DDF solution of the root of f by Newton approximation, $x_{i+1} = x_i - f(x_i) / f'(x_i)$.


```

> /* Now we use PLIP (PL/I Preprocessor) to define the */
> /* attributes of the GMB dataarcs and the interpretations */
> /* for the GMB processors. The interpretations for the */
> /* processors were produced from templates which corres- */
> /* pond to the various data-flow primitives. Similarly, */
> /* the attributes for the dataarcs were produced from a */
> /* template which gives each dataset a "value" and a */
> /* "present" flag. The "present" flag is used to mark */
> /* the presence or absence of a data-flow token. The */
> /* initial state of the model is given by the initial */
> /* state of the dataarcs. */
> /* To save space only one example of each dataflow primitive is included in */
> /* this paper, and only two of the template declarations are shown. */

>@plip newton
SARA.Behavior.GMB.PLIP
GMB PL1 Preprocessor October 5, 1978

> /* As before, the source for PLIP resides in a file and */
> /* it is necessary to request that the input be echoed */
> /* to the terminal. */

>@output * +input_echo
output destination * for message warning error listing trace pro
mpt input_echo classes
>@input newton.plip
input source is >>>SARA>SARA_library>dataflow>newton.plip
@template (daon0)
1 n_init,
2 value fixed bin init(20),
2 present bit(1) init("1"b);

@template (daib0 daob0 daib1 daob1 daib2 daob2 daib4 daob4
daib7 daob7 daib9 daob9 daib10 daob10 daib11 daob11)
1 boolean,
2 value bit(1),
2 present bit(1) init("0"b);
.
.
.

@processor m1 ; /* merge node */

#include iotypes;
#include ioputs;
declare rc fixed bin (15);
@read n1 @from dain1; /* first, check the presence of tokens on */
if "n1.present /* the output arcs */
then do;
@read b3 @from daob3;
if b3.present /* check presence of inputs */
then do;
if b3.value /* b3 carries a boolean */
then do;
@read n4 @from daon4;
if n4.present
then do;
n1 = n4 ;
b3.present, n4.present = "0"b ; /* remove input tokens */
rc = pu_t ("merge m1 fired", i_otype.listing);
rc = pu_t (" n4 -> n1", i_otype.trace);
@write n1 @to dain1 @after T;
@write n4 @to daon4;
@write b3 @to daob3;
end ;
end ;
else do;
@read n0 @from daon0;
if n0.present
then do;
n1 = n0 ;
b3.present, n0.present = "0"b ; /* remove input token */
rc = pu_t ("merge m1 fired", i_otype.listing);
rc = pu_t (" n0 -> n1", i_otype.trace);
@write n1 @to dain1 @after T;
@write n0 @to daon0;
@write b3 @to daob3;
end ;
end ;
end ;
end ;
@endprocessor ;
@processor p1 ;

/* decider node */
/* decider node is identical to operator */
/* node with the operands of type boolean */

@endprocessor ;

@processor g1 ; /* gate node */

#include iotypes;
#include ioputs;
declare rc fixed bin (15);
@read n4 @from dain4;
if "n4.present
then do;
@read b4 @from daob4;
@read n2 @from daon2;
if (b4.present & n2.present) /* check presence of inputs */
then do;
if b4.value /* b4 carries a boolean token */
then n4 = n2 ;
n2.present, b4.present = "0"b ; /* remove input tokens */
rc = pu_t ("gate g1 fired", i_otype.listing);
rc = pu_t (" n2, b4 -> n", i_otype.trace);
if b4.value
then rc = pu_t (" n4", i_otype.trace);
else rc = pu_t ("", i_otype.trace);
@write n4 @to dain4 @after T;
@write n2 @to daon2; @write b4 @to daob4;
end ;
end ;
@endprocessor ;
@processor f1 ; /* operator node */

#include iotypes;
#include ioputs;
declare rc fixed bin (15);
@read c2 @from daic2;
if "c2.present /* check presence of token on output arc */
then do ;
@read c1 @from daoc1;
if c1.present then do; /* check presence of inputs */
c2.value = c1.value + 1 ;
c1.present = "0"b ;
c2.present = "1"b ;
rc = pu_t ("operator f1 fired", i_otype.listing);
rc = pu_t (" c1 -> c2", i_otype.trace);
@write c2 @to daic2 @after T;
@write c1 @to daoc1;
end ;
end ;
@endprocessor ;
@processor l1 ; /* link node */

#include iotypes;
#include ioputs;
declare rc fixed bin (15);
@read n2 @from dain2; /* check tokens on output arcs */
if "n2.present
then do;
@read n3 @from dain3;
if "n3.present
then do ;
@read n1 @from daon1;
if n1.present then do;
n2, n3 = n1 ; /* copy input onto output arcs */
n1.present = "0"b ; /* remove token from input arc */
rc = pu_t ("link l1 fired", i_otype.listing);
rc = pu_t (" n1 -> n2, n3", i_otype.trace);
@write n1 @to daon1;
@write n2 @to dain2; @write n3 @to dain3;
end ;
end ;
@endprocessor ;
>@output * -input_echo
output destination * for message warning error listing trace pro
mpt classes
>@end
*** 0 errors
*** 0 warnings
Do you want to compile the PLIP output? (y or n)>y
PL/I compilation in progress
PL/I
End of GMB PL1 Preprocessor
SARA.Behavior.GMB
> /* The model has been preprocessed by PLIP and then */
> /* compiled by the Multics PL/I compiler */

```

Figure 21 (continued)

the parallelism existent in the problem. Figure 19 presents the GMB model and Figure 20 represents the DDF model. Both models should be viewed as if they were part of a SARA closed design universe, i.e., the model of the system to be designed is enclosed by a module structure and can

interact with other modules only through specified sockets. It is assumed that outputs from other modules initialize the model and provide the four inputs at the top of the figure. It is assumed that the outputs at the bottom are received by other modules and possibly tested for error.

control graph. Templates for PLIP interpretations were used to make each uncontrolled processor behave like the corresponding DDF primitive. The resulting GMB description representing the DDF model was then translated and simulated by SARA at MIT-MULTICS via the ARPANET. The dialogue between user and SARA was captured and is incorporated in Figure 21 to complete the body of this section. In the interest of space, some details have been removed and comments have been added to notify the reader.

CONCLUSIONS

This paper has attempted to describe two token models which have been used to explore architectures taking advantage of concurrency. The primitives of the Dennis Data Flow Model and the SARA Graph Model of Behavior have been described and further understanding of their relationships was obtained by expressing one set of primitives in terms of the other. It was then demonstrated that data flow architectures could be explored making use of existing SARA tools. In particular, some investigators have found that developing a DDF model is very much like low-level design of a hardware system. The multi-level modeling supported in SARA gives some hope of expressing functions at a high level and refining them systematically.

REFERENCES

- Adams, D. A., "A Computational Model with data flow sequencing," Technical Report CS 117, Computer Science Dept., School of Humanities and Sciences, Stanford University, CA, December 1968.
- Arvind and K. P. Gostelow, "A computer capable of exchanging processing elements for time," *Information Processing 77*, B. Gilchrist (ed.), North Holland, New York, 1977.
- Bahrs, A., "Operations Patterns (an Extensible Model of an extensible language)," *Symposium on Theoretical Programming*, Novosibirsk, USSR, August 1972.
- Bussell, B., "Properties of a Variable Structure Computer System in the Solution of Parabolic Partial Differential Equations," Ph.D. Diss., UCLA, August 1962.
- Dennis, J. B., "First Version of a Data Flow Procedure Language," Massachusetts Institute of Technology, MIT/LCS/TM-61, May 1975.
- Estrin, G., "A Description of the Electronic Computer at the Institute for Advanced Study," *Proceedings of the ACM*, Sponsored by the Joint Computer Conference, September 1952, pp. 95-109.
- Estrin, G. and R. Turn, "Automatic Assignment of Computations in a Variable Structure Computer System," *IEEE Transactions on Electronic Computers*, Vol. EC-12, No. 5, December 1963.
- Estrin, G., "A Methodology for design of digital systems—supported by SARA at the age of one," AFIPS, *Proceedings of the National Computer Conference*, 1978.
- Faggin, F., "How VLSI Impacts Computer Architecture," *IEEE Spectrum*, May 1978, pp. 28-31.
- Gardner, R. I., "Multi Level Modeling," *Proceedings of Symposium on Design Automation and Microprocessors*, Palo Alto, CA, February 1977.
- Gardner, R. I., "A Methodology for Digital System Design Based on Structural and Functional Modeling," Technical Report, UCLA-ENG-7488, January 1975.
- Glushkov, V. M., M. B. Ignatyev, V. A. Myansnikov and V. A. Torgashev, "Recursive Machines and Computing Technology," *Information Processing 74*, North Holland Publishing Co., 1974.
- Gostelow, K., V. Cerf and G. Estrin, "Proper Termination of Flow of Control in Programs Involving Concurrent Processes," *Proceedings of the ACM*, Vol. 11, Boston, August 1962.
- Gostelow, K., and Arvind, "A new interpreter for data flow schemas and its implications for computer architecture," *Technical Report 72*, University of California at Irvine, October 1975.
- Gostelow, K., and Arvind, "A Computer Capable of Exchanging Processors for Time," *IFIP Congress 1977*, Toronto, Canada, August 1977.
- Holland, J., "A Universal Computer Capable of Executing an Arbitrary Number of Subprograms Simultaneously," *Proceedings EJCC*, 1959.
- Karp, R. M., and R. E. Miller, "Properties of a Model for Parallel Conventions: Determinacy, Termination, Queueing," *STAM J. of Applied Math.*, November 1966.
- Martin, D. F., and G. Estrin, "Path Length Computations on Graph Models of Computations," *IEEE Transactions on Computers*, Vol. EC-18, pp. 530-536, June 1969.
- Kosinski, P., "A Data Flow Programming Language," *Report RC4264*, IBM T.J. Watson Research Center, Yorktown, N.Y., 1973.
- Muller, D. E., "Asynchronous logics and application to information processing," *Switching Theory in Space Technology*, Stanford University Press, 1963.
- Overman, W., and G. Estrin, "Developing A SARA Building Block—The 8080," Computer Science Department, University of California, Los Angeles, CA, 1977.
- Razouk, R., and G. Estrin, "The Graph Model of Behavior Simulator," *Proceedings of Symposium on Design Automation and Microprocessors*, Palo Alto, CA, February 1977.
- Rodriguez, J. E., "A Graph Model for Parallel Computation," *Report MAC-TR-64*, Project MAC, MIT, Cambridge, Mass., September 1969.
- Rose, C. W., F. T. Bradshaw and S. W. Katzke, "The LOGOS representation system," *Proceedings Sixth Annual IEEE Computer Conference (CompCon 72)*, San Francisco, September 1962, pp. 187-190.
- Ruggiero, W., "A distributed data and control driven machine: programming and architecture," UCLA Computer Science Department, University of California, Los Angeles, CA, 1978.
- Ware, W. H., "Logical principles of a new kind of binary counter," *Proceedings of the IRE*, Vol. 41, October 1953, pp. 1429-1437.

Software metrics for aiding program development and debugging

by N. F. SCHNEIDEWIND

Naval Postgraduate School
Monterey, California

INTRODUCTION

Computer program graphs have proven very useful because they illuminate the structural characteristics of a program. Structural characteristics, as a representation of program complexity, have been shown to be strongly related to program development time, program quality and difficulty of debugging.¹⁻³ The use of graphs for these purposes is not widely known or understood in the data processing community. It is the aim of this paper to provide an introduction to graphs as they apply to program representation and to show examples of their use in program design and debugging.

GRAPH DEFINITIONS AND PROPERTIES DEFINITIONS

The following definitions are due to Chan:⁴

Graph

A graph is a set of line segments called edges (e_j) and points called vertices (v_i) which are the end-points of the edges, interconnecting in such a way that the edges are connected only to the vertices. A non-directed graph has no orientation of the edges; a directed graph does have edge orientation in the form of arrows. Figure 1a shows a directed graph G .

Degree of a vertex

The degree of a vertex is the number of edges incident with that vertex. The degree of v_6 of G is 3.

Sub-graph

A portion of a graph, containing a subset of the edges and vertices of the graph is called a sub-graph. Two sub-graphs of G , G_1 and G_2 , are shown in Figure 1b.

Path

If a set of edges e_1, e_2, \dots, e_i can be ordered in the form $e_1(v_1, v_2), e_2(v_2, v_3), \dots, e_i(v_i, v_{i+1})$, where v_1 and v_{i+1} are the terminal vertices and all vertices are distinct, then the set of edges forms a path. In G , the set of edges a, d, e forms a path. It is important to note that by this definition vertices may not be revisited. Thus, the sequence of edges a, b, c, d in G is not a path because v_2 appears twice in the edge sequence. In graph theory this sequence is called a walk. However, since iteration is an important characteristic of computer programs, we will modify the above definition of path to include walks in order to avoid using two terms when describing a program graph. When the edges of a path have consistent orientations, the path is directed. The above paths are directed.

Circuits

If the two terminal vertices of a path coincide and the remaining vertices are distinct, this path is a circuit. A directed circuit has all edges with the same (clockwise or counter-clockwise) orientation. Thus G_1 and G_2 are circuits but only G_1 is directed.

Connected graph

A graph is connected if there is at least one path between every pair of vertices. G , G_1 and G_2 are connected.

Tree

A tree T of a connected graph is a connected sub-graph that contains all vertices of the graph but no circuits. The edges contained in a tree are called branches. The complement set of edges T' ; that is the remaining edges of the graph, are called chords. One of the trees of G is shown in Figure 1c, where $T=(a, b, d, e, f, g, h, i, k)$ and $T'=(c,$

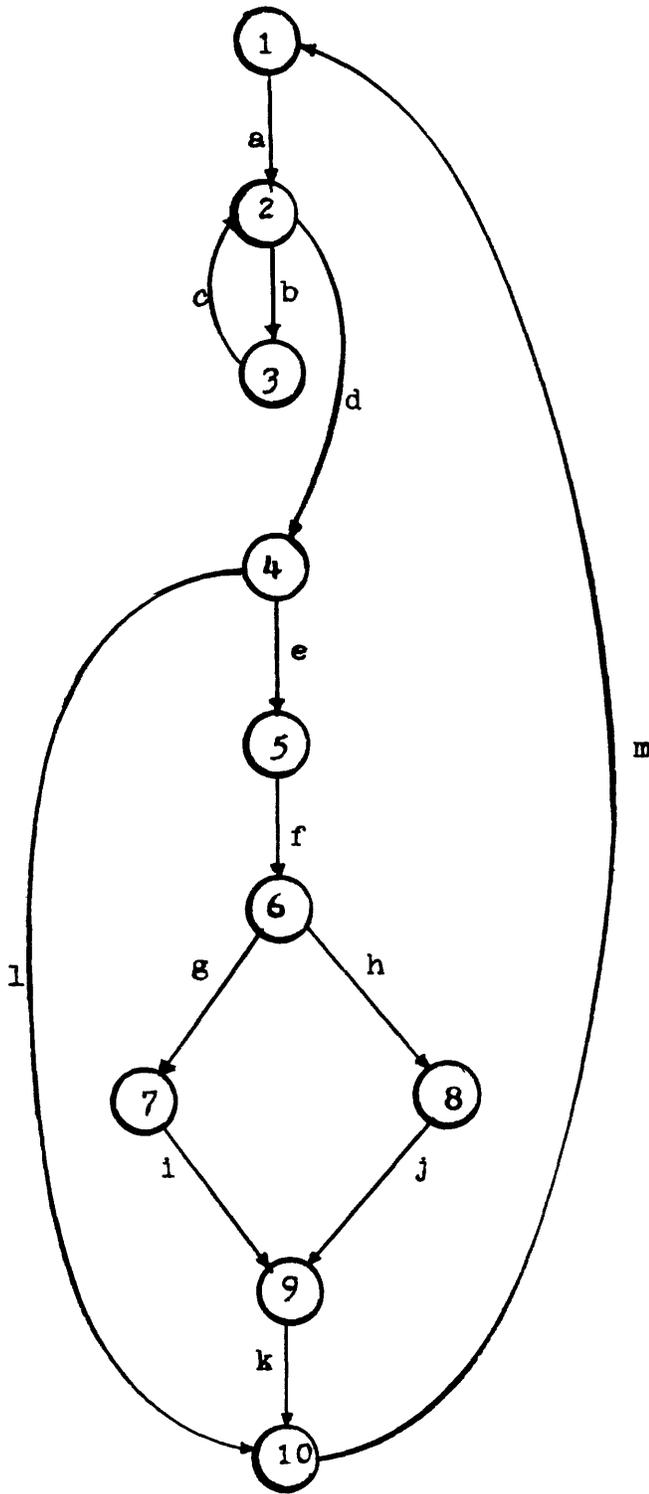


Figure 1a—Graph G.

j, l, m) are the branch set and chord set, respectively. If we let E and V represent the number of edges and number of vertices, respectively, the number of branches and chords is equal to $V-1$ and $E-V+1$, respectively.

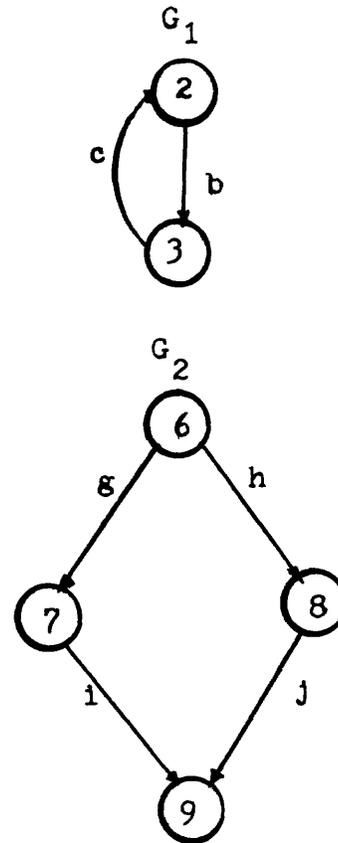


Figure 1b—Sub-graphs G_1 and G_2 .

MATRIX PROPERTIES

Adjacency matrix

The adjacency matrix $X=[x_{ij}]$ of a directed graph with V vertices is a $V \cdot V$ matrix consisting of 0, 1 elements, where $X_{ij}=1$ if there is an edge directed from v_i to v_j , and 0 otherwise.⁵ The r th power of the adjacency matrix X^r is equal to the number of directed paths of length r edges between each pair of vertices v_i and v_j . Shown in Figures 2a, 2b and 2c are X, X^2 and X^3 of G , respectively. Thus Figure 2b shows that there are two paths of length 2 from v_6 to v_9 ; these consist of edges g and i and h and j (see Figure 1a). A directed path of length 3 from v_1 to v_{10} is indicated in Figure 2c; this is a path from start to terminal vertices consisting of edges a, d and l . If v_i designates the terminal node of a graph and recognizing that the maximum possible path length is $r=E$ edges, the matrices X, \dots, X^E with non-zero entries in the $1, t$ cells will enumerate all of the paths which start at v_1 and terminate at v_t .

Fundamental circuit matrix

A fundamental circuit matrix,⁴ with respect to a tree T of a graph G of V vertices and E edges, is the matrix B_f of

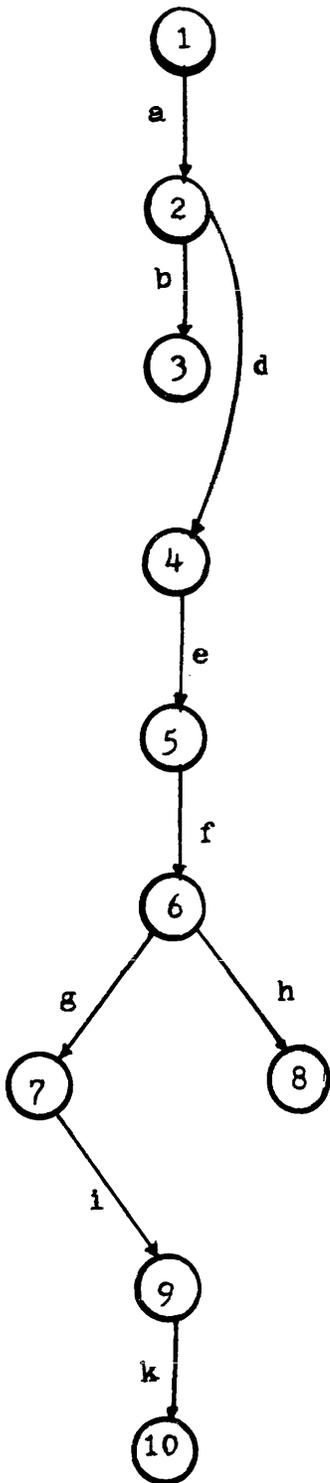


Figure 1c—Tree T .

order $(E - V + 1) \cdot E$ with each row identified by a fundamental circuit c_i (with respect to T) and each column by an edge e_j , where $b_{ij} = 1$ if e_j is in c_i and has the same orientation as chord in c_i , $b_{ij} = -1$ if e_j is in c_i and has opposite orientation of chord in c_i , $b_{ij} = 0$ if c_i is not in c_i . The B_f for

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	1
$X=5$	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	1	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1
10	1	0	0	0	0	0	0	0	0	0

Figure 2a—Adjacency matrix.

	1	2	3	4	5	6	7	8	9	10
1	0	0	1	1	0	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0	1
3	0	0	1	1	0	0	0	0	0	0
4	1	0	0	0	0	1	0	0	0	0
$X^2=5$	0	0	0	0	0	0	1	1	0	0
6	0	0	0	0	0	0	0	0	2	0
7	0	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	0	1
9	1	0	0	0	0	0	0	0	0	0
10	0	1	0	0	0	0	0	0	0	0

Figure 2b—Square of adjacency matrix.

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	1	0	0	0	0	1
2	1	0	1	1	0	1	0	0	0	0
3	0	1	0	0	1	0	0	0	0	1
$X^3=4$	0	1	0	0	0	0	1	1	0	0
5	0	0	0	0	0	0	0	0	2	0
6	0	0	0	0	0	0	0	0	0	2
7	1	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0
9	0	1	0	0	0	0	0	0	0	0
10	0	0	1	1	0	0	0	0	0	0

Figure 2c—Cube of adjacency matrix.

G is shown in Figure 3. The chord set $T' = (c, j, l, m)$ forms a unit matrix on the left. The branches of T are on the right. Circuits are formed by adding one chord at a time to T . Thus the circuits are: bc , $ghij$, $efgikl$ and $adefgikm$, corresponding to c_1 , c_2 , c_3 and c_4 , respectively, where c_1 and c_4 are directed circuits. Fundamental circuits have the property that no circuit in the set can be obtained by a linear combination of other circuits in the set. The number of fundamental circuits in a graph is given by $E - V + 1$, the number of chords. Once B_f has been determined, all circuits in a graph, comprising the circuit matrix B_a , can be generated by performing all possible sum (Exclusive OR) operations indicated by (\oplus) on the rows

	Chords	Branches	Program Construct
	c j l m	a b d e f g h i k	
c_1	1	0 0 0 0 1 0 0 0 0 0 0	While Do
c_2	0	1 0 0 0 0 0 0 0 -1 1 -1 0	If Then Else
$B_f=c_3$	0	0 1 0 0 0 0 -1 -1 -1 0 -1 -1	If Then
c_4	0	0 0 1 1 0 1 1 1 1 0 1 1	Main Line

Figure 3—Fundamental circuit matrix B_f of G .

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1	1
4	0	0	0	0	1	1	1	1	1	1
5	0	0	0	0	0	1	1	1	1	1
6	0	0	0	0	0	0	1	1	1	1
7	0	0	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0	1	1
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0

Figure 4—Reachability matrix R of G (with no edge from 10 to 1).

of B_f , if the negative signs in B_f are ignored. A ring sum operation on two rows of B_f will generate either another circuit in G , whose edges are either in one of the original circuits but not in both, or an edge disjoint union of circuits;⁵ the latter are ignored. Thus from Figure 3, $C_2 \oplus C_3$ will generate circuit $e f h j k l$ and $C_2 \oplus C_4$ will generate circuit $a d e f h j k m$. Program constructs, which will be described later, are shown on the diagram.

Reachability matrix

The reachability matrix $R=[r_{ij}]$ has a value $r_{ij}=1$ if a directed path exists between v_i and v_j and 0 otherwise [6]. The R matrix for G is shown in Figure 4. This matrix does not include the edge m , because this would result in R having all ones, a special case where each vertex can be reached from every other vertex (strongly connected graph).

APPLICATION OF GRAPHS TO PROGRAM DEVELOPMENT AND TESTING

Directed graph representation of computer programs

The use of a directed graph to represent a program will now be demonstrated. In fact, the connected graph G which has been discussed in the examples is the graph of the ALGOL procedure in Figure 5. The circled numbers in this figure correspond to the vertex numbers in Figure 1a; edges correspond to ALGOL statements between vertices. The program constructs (e.g. *If Then Else*) of this procedure are shown in Figure 6. The four constructs (*While Do*, *If Then Else*, *If Then* and *Main Line*) are connected sub-graphs. The part of the procedure corresponding to no iterations and the satisfaction of all true conditions is called the *Main Line*. Each of the constructs can be obtained from the tree in Figure 6 by adding a chord to the tree. These chords are c for *While Do*, j for *If Then Else*, l for *If Then* and m for *Main Line*. Each of the constructs is an independent circuit as previously defined. Edge m is an artificial edge which has been added to the graph for the purpose of obtaining the *Main Line* construct as an independent circuit; it is not part of the ALGOL procedure. Using *Main Line* allows edges a and d , which do not appear in the other three constructs, to be represented in the set of independent circuits. The independent circuits in matrix form (B_f) are shown in Figure 3. The extent of branching at a vertex is given by the degree of the vertex. For example the beginning of the *If Then Else* construct is at v_6 . This vertex has degree 3, corresponding

```

1  PROCEDURE TEST_CONDITIONS;
   COMMENT TEST ALL CONDITIONS FOR MEMBER IDENTIFIED BY CURRENT_NODE;
   COMMENT IF ALL CONDITIONS HOLD ADD MEMBER TO LINKED LIST;
   BEGIN
     INTEGER A, I;
     LOGICAL FAIR;
     FAIR:=TRUE;
     I:=1;
2  WHILE ((REQUEST(I) = "Q") AND ( FAIR = TRUE )) DO
     BEGIN
       FAIR:=MATCHING(I);
       I:=I+1;
3     END;
4   IF FAIR = TRUE THEN
5     BEGIN
6       A:=ALLOCATE1;
7       IF LIST_POINTER = NIL THEN LIST_POINTER:=A
8       ELSE SETCDR1(LAST,A);
9       LAST:=A;
10      SETCDR1(LAST,NIL);
        SETCAR1(LAST,CDR2(CURRENT_NODE+1));
        END;
   END TEST_CONDITIONS;

```

Figure 5—ALGOL procedure corresponding to graph of Figure 1a.

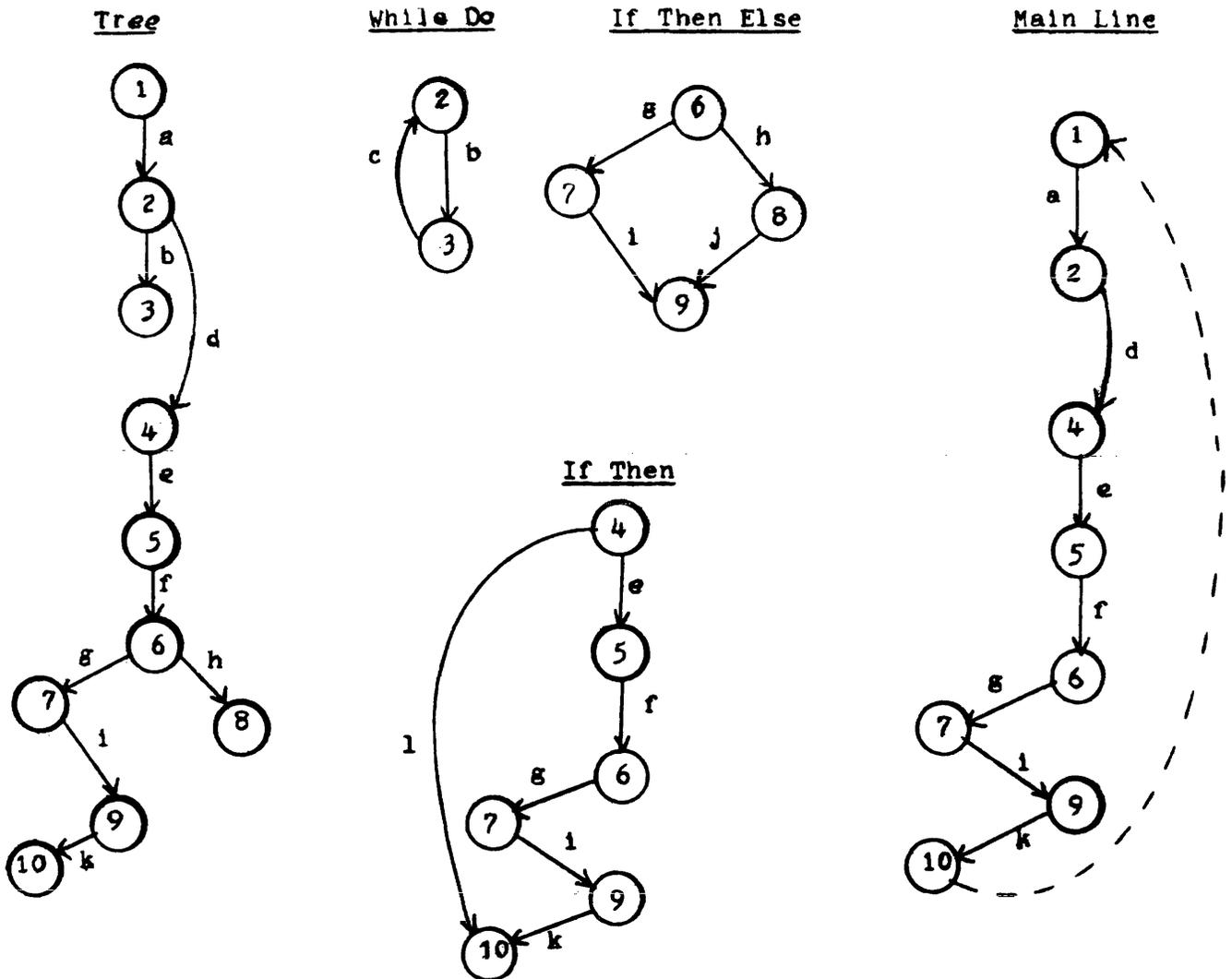


Figure 6—Decomposition of Figure 1a structure.

to the entry, *Then* and *Else* branches. Vertex degree is one indicator of program complexity.

Implications of graph properties for program development and testing

Program constructs

A program graph can be partitioned into its constructs by first identifying a tree and then adding a chord at a time as shown in Figure 6. Each construct is a basic unit of a program which must be tested. The number of constructs or independent circuits is called the cyclomatic number. This was previously given as $E - V + 1$. This quantity has been shown to be highly related to difficulty of debugging.^{1,3}

Program paths

The Adjacency Matrix and its derivatives provide program path information. This information can be used to identify

the various paths whose correct execution should be verified. Two elementary program paths are given by Figure 2b, where it is shown that there are two paths of length 2 from v_6 to v_9 ; these correspond to the *If Then* and *If Then Else* branches. It should be noted that path length as used in Figure 2 refers to number of edges and not to number of source statements.

Complete paths from v_1 to v_{10} can be obtained by performing ring sum operations on the independent circuits of matrix B_f , as explained previously. The six possible paths so obtained are

- a d e f g i k
- a d e f h j k
- a d l
- a b c d e f g i k
- a b c d e f h j k
- a b c d l

These are paths from start vertex to terminal vertex which should be tested.

Code reachability

The reachability matrix can be used to ascertain whether any program code is not used. This would be indicated by one or more zero rows in R . The relative importance of vertices can also be ascertained by examining R and noting the number of ways a given vertex can reach other vertices. A high number indicates that the vertex and the edges comprising the paths to the other vertices are relatively important for correct execution of the program and should be accorded corresponding emphasis in testing; v_2 is such a vertex in Figure 4.

Reachability may also be defined as the summation, over all vertices, of the number of ways in which a vertex can be reached. Average reachability can be obtained by dividing this figure by number of vertices. This is the way reachability was calculated in Table I, which will be described subsequently.

To make the use of directed graphs practical for program representation and complexity measurement, it is necessary to significantly automate the production of the various matrices and complexity measures from a definition of the program graph. Even the latter can be generated, if the problem has been put in the form of a decision table. Several automated tools exist for directed graph manipulation.

PROGRAM COMPLEXITY MEASURES OBTAINED FROM DIRECTED GRAPHS

The data in Table I show the results of an experiment conducted at the Naval Postgraduate School involving four ALGOL programming projects, where average values of four complexity measures were computed for programs with and without detected errors [3]. Three of the measures (cyclomatic number, number of paths and reachability) were obtained from directed graphs and were discussed previously in this paper. Complexity measure values were significantly lower for the no-error case, suggesting a set of quantitative measures for program quality control and error avoidance.

TABLE I—Software Error Experiment
Complexity Measure Comparison
Procedures With No Errors vs. Procedures With Errors

	No Errors		Errors	
	Mean Value	Number of Procedures	Mean Value	Number of Procedures
Cyclomatic Number	1.699	83	4.74	31
Number of Source Statements	9.361	83	27.23	31
Number of Paths	2.671	82	27.1	20
Reachability	10.1	82	120.3	20

SUMMARY

Several directed graph properties which are useful for program representation and complexity measurement were described. These were then applied to a small ALGOL program. Evidence was then presented suggesting that directed graph properties can provide quantitative measures of program quality.

REFERENCES

1. McCabe, T. J., "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, December 1976, pp. 308-20.
2. Green, T. F., N. F. Schneidewind, G. T. Howard and R. J. Pariseau, "Program Structures, Complexity and Error Characteristics," *Proceedings of the Symposium on Computer Software Engineering*, Polytechnic Press of the Polytechnic Institute of New York, Brooklyn, N. Y., April 1976, pp. 139-54.
3. Schneidewind, N. F., and M. H. Hoffmann, "An Experiment in Software Error Data Collection and Analysis," *Proceedings of the Sixth Texas Conference on Computing Systems*, University of Texas, Austin, Texas, November 1977, pp. 4A1-4A12.
4. Chan, Shu-Park, *Introductory Topological Analysis of Electrical Networks*, Holt, Rinehart and Winston, Inc., New York, 1969.
5. Deo, Narsingh, *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, Englewood Cliffs, N. J., 1974.
6. Stigall, Paul D., and Ömür Tasar, "Special Tutorial: A Review of Directed Graphs as Applied to Computers," *Computer*, Vol. 7, No. 10, October 1974, pp. 39-47.

A measure of software complexity

by NED CHAPIN

InfoSci Inc.
Menlo Park, California

IMPORTANCE OF COMPLEXITY

In recounting the copious reasons or excuses for our traditional problems in developing and maintaining computer software, many authorities have mentioned complexity (e.g., References 2, 8, 15). The complexity pointed to sometimes is the inherent complexity of the jobs the computer is to do; sometimes it is the complexity of the systems or programs that direct the computer to do the jobs.

Yet thoughtful observers have also long noticed that the inherent complexity of jobs may differ greatly from the apparent complexity of the software for those same jobs (e.g., References 26, 29). Anyone who has had the opportunity to study the diversity of approaches, designs, and codes resulting when different people independently produce software for the same job, keenly senses the difference between apparent software complexity and inherent job complexity (e.g., Reference 25).

Intuition and common sense generally agree that software which appears simple is superior to software that appears complex, whatever the inherent complexity of the job (e.g., Reference 20). This position is in fact incorporated in the appraisal guidelines of structured design as "simplicity."^{23,24} But applying intuition and common sense is not really sufficient to obtain consistently simple software. What is needed is objective, quantitative, reliable, valid and convenient ways of measuring either the complexity or the simplicity in software. To that end, a number of proposals have been advanced.^{1,7,11,12,17,19,21,22,26,32}

This paper proposes an alternative measure of software complexity. The background of the measure is briefly given, and its computational procedure described. Then it is applied to a given software design of a small modular structured program. Afterward, the measure is compared with other alternative measures and with programmer ratings of the program. The paper closes with a discussion of the validity of the proposed measure of software complexity.

BASIS OF MEASURE

The proposed measure of software complexity, Q , is an index of the difficulty people have in understanding the function implemented by the software. The proposed meas-

ure is quantitative, highly reliable, shows reasonable validity and is easily computed from system and program documentation. Since source code is not required, the measure can be applied to designs before writing the source code, as long as they display some modularization.

The theoretical basis for the proposed Q measure springs from the set theoretic definition of function. The function is what the software is to direct the computer to do. Briefly put, a function is a correspondence between sets of input data of specified domains, and sets of output data of specified ranges. Hence, any difference in functions must be expressed as differences in the input or in the output or both.

These differences may take two forms—differences in the membership in the sets, or differences in the domains and ranges. Changes in domains—that is, the legal, allowable values the input may take for which the function is defined—and in ranges, are usually of smaller impact in the specification of a function than are changes in the component sets of the input or output. Thus, expanding the domain of a function such as "find a square root" from a domain of one- and two-digit positive integers to a domain of one-, two- and three-digit positive integers is usually a minor matter. But changing the components of the input or output to include a new variable, such as the natural log of the root as an additional output, usually makes a non-trivial change in the function of the software. At the loss of some validity, the proposed measure of software complexity ignores domains and ranges, and concentrates on the components of the sets of input and output.

A listing of the sets of input and output define the function coarsely. A more refined specification is possible if the role of the data is recognized.⁵ Some input data are needed for processing (role "P" data)—that is, for the production of output data. The data changed, created, or modified (role "M" data) in value or identity by the performance of the function are the output data. The data used to select or decide which functions to perform serve in a controlling role (role "C" data). Or, data may pass through (role "T" data) a function unchanged in value and identity, when a function of the software is to communicate data from one part of the software to another part, as from one module to another.

To improve the validity of the proposed measure, data roles are recognized in it. Data in a C or control role contribute the most complexity. Data in an M or modified role are major contributors. Data in the P or processing role

contribute some complexity. The *T* role data that are only passed through contribute the least to software complexity.

The modules or segments of the software can be visualized as communicating data among themselves.²⁰ Such intermodule data exhibit a simple life history. They start as *M* role data in one module, become *T* role data as they are communicated through other modules, and then terminate as *C* role or *P* role data in a using module.⁵ If only two modules are involved in the communication, then the data skip the *T* role. But the more modules involved in communicating any item of data, the higher is the apparent complexity of the software.

The data communication among modules may be further complicated by the presence of iteration. In modularized programs and systems, the iteration control is observed to be the psychologically most difficult aspect of the "interior" complexity.³⁰ Modularization can reduce most kinds of "interior" complexity, but makes no reduction in iteration complexity when more than one module is involved.^{18,19}

If the duration or extent of iteration is determined by *C* or control role data arising outside of the loop exit module, the complexity of the software increases, but not in a linear manner. This non-linearity at first is of little importance, but becomes critical as the number and arrangement of the *C* role data become larger and more diverse. This is approximated in the computation of the proposed *Q* measure by a conversion formula involving the square of one-third of a weighted count.

In review, the proposed *Q* measure appears, because of its stress on function, to reflect the "exterior" rather than the "interior" complexity. This appearance is deceptive for several reasons. First, as Halstead has pointed out with his " n_2^* ," the externals place a lower limit on the potential interior complexity of a module.¹¹ Second, assuming the presence in the module of an algorithm that avoids work redundant and extraneous to the function within the module, the data (as noted in the input-output table) with their associated domains and ranges, place an upper limit on the interior complexity of a module. And third, the process of identifying modules, apportioning the interior complexity of a system into the programs' component modules, and their interrelationships. Both what is apportioned where, and the interrelationships, are described by the data flow among the modules, and are reflected in the proposed *Q* measure of complexity because of its stress on function.

COMPUTATION OF MEASURE

The high reliability of the proposed measure arises from the simple computational procedure used on the documentation for the program or system. A measure is reliable when different people using the same computational procedure consistently come to the same result.²⁸

The ten steps in the computational procedure for the proposed measure *Q* are:

1. For each module, count in the input-output table⁵ or the equivalent, the number of data items shown in *C*,

P, or *T* roles as input, and in *M* or *T* roles as output. When one data item appears in multiple roles or has multiple sources or destinations, each is to be counted. Data serving as program-wide or system-wide constants or literals are not counted. The reason for distinguishing the roles of data was described earlier.

2. Multiply for each module the total count for each role by the appropriate weighting factor *W*, as follows: 3 for *C*, 2 for *M*, 1 for *P*, and $\frac{1}{2}$ for *T*. The reason for the weights was described earlier.
3. Sum the weighted counts by module.
4. Assign an initial *E* of 0 to all modules. Then examine the documentation to determine which modules are to contain the exit tests for iterations where subordinate modules are part of the iteratively-invoked loop body. The tree structure chart usually shows this most conveniently.^{4,20} The loops or iterations are ignored when they are to be performed entirely within a module with no subordinate modules iteratively invoked. The reason for the concern with iteration control was described earlier.
5. For each iteration-exit module identified in Step #4, examine the *C* items to determine which are to serve in the exit test for the iteration of the subordinate modules that comprise the loop body. Determine where these *C* data come from. If they come from within this module only, or are constants, add 0 to *E* for each such *C* data item. If they come from within the subordinate loop body, add 1 to *E* for each such *C* data item. If they come from outside of the loop body, add 2 to *E* for each such *C* data item. An example (starting with *E* as 0) is an item of data which is initialized to a starting value outside of the loop (add 2 to *E*), and also modified within the loop body (add 1 to *E* to total 3). Note that *E* for any one module cannot exceed three times the count of the number of data items serving in a *C* role for determining the exit from iteration.
6. Convert *E* for each module into a repetition factor *R* by adding 1 to the square of one-third of *E*. For example, if *E* is 6, then one-third of *E* is 2, and 2 squared is 4, 4 plus 1 is 5. Hence *R* is 5. The reason for this formula was described earlier. *R* values for common *E* counts are: *E* of 0 gives *R* of 1.00; *E* of 1 gives *R* of 1.11; *E* of 2 gives *R* of 1.44; *E* of 3 gives *R* of 2.00; *E* of 4 gives *R* of 2.78; *E* of 5 gives *R* of 3.78; *E* of 6 gives *R* of 5.00; and *E* of 7 gives *R* of 6.44.
7. Multiply the sum of the weighted counts from Step 3 by the modules' respective *R* values.
8. Find the square root of the products from Step #7. This is *Q*, the index of module complexity. The computation in this step is easily done on most pocket calculators, and effectively is a computation of the geometric mean of the total weighted counts and the inter-module iteration control.
9. Calculate the *Q* of the program by finding the arithmetic mean (average) of the component modules. This is a simple averaging computation.

10. Calculate the Q of the system by finding the arithmetic mean of the component modules within the component programs, or by weighting the programs' Q from Step #9 by the relative sizes (in terms of modules) of the programs. Note that if in execution a system will iterate the execution of a program, then the E of the module containing the loop exit control is rarely zero.

The practical lower bound on module complexity is 1.0. The exception with a lower bound of zero is a module that has no C , P , T , or M role data—a module that does no function! However, in some delayed-time software, such modules may appear as the root of a system implemented at the root (top) level with JCL (Job Control Language). No upper bound exists for Q , but values beyond 11.0 are uncommon with structured programming and structured design.

EXAMPLE OF COMPUTATION

Figure 1 provides a tree structure chart of a program designed with structured programming techniques. The input-output tables are shown in Figure 2. The computation of module Q and program Q are summarized later. Figure 3 lists in its leftmost column, the identifications of the module.

The numbered paragraphs to follow refer to the step numbers described previously.

1. The counts from the input-output (I-O) tables are shown in the second through fifth column of Figure 3. Thus, for instance, Module S-4 has one C item, one P item, two M items, and two T items of data shown in Figure 2. These counts are the "raw counts" entries for line S-4 in Figure 3.
2. The raw counts are multiplied by the weights. Thus, for S-4, the count of 2 for M is multiplied by the weight W of 2 to give 4.
3. The sum of the weighted counts is shown as W-TOT in Figure 3. Thus, for Module S-4, the total of 9 is the sum of the weighted counts of $3+1+4+1$.
4. A review of Figure 1 shows Modules S-3, S-4 and S-7 to contain iteration exits. The broken ring flags them. If the documentation were less specific, other clues would have to be used to locate repeated functions, such as the reading or writing of records.
5. In S-3, iteration exit items as flagged by a dot in the input-output table are the Record Key East, Record Key West, and Out of Sort. In S-4, it is a Bad End Switch. In S-7, no evidence appears from the input-output table about what the iteration control might

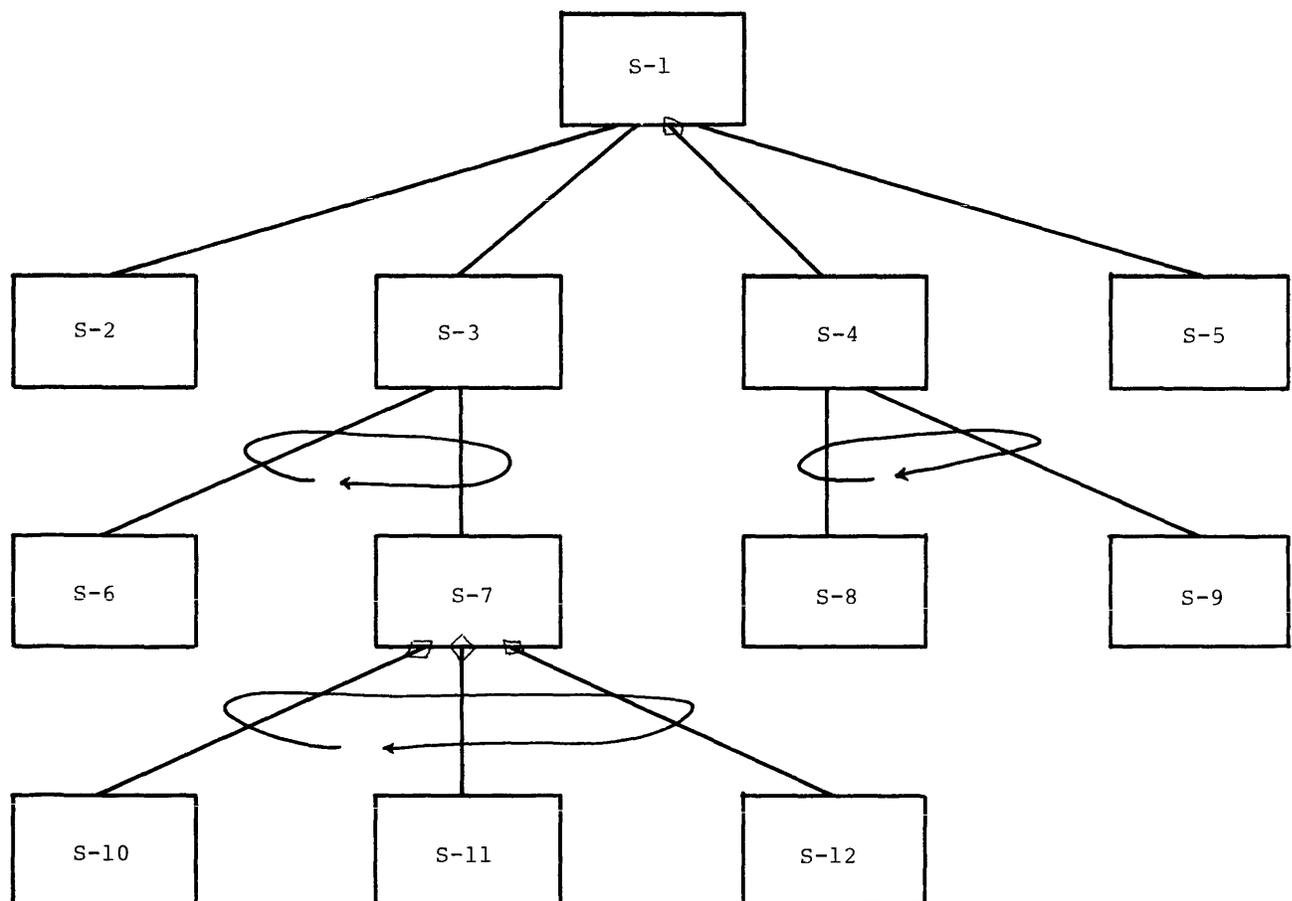


Figure 1—Tree-structure chart for example.

FROM	TO	TYPE	FROM	TO	TYPE
MERGE FILES (S-1)					
OUT OF SORT	C	MERGE GOOD RECORD	OUT OF SORT	M	MERGE GOOD RECORD
IN-POINTER	T	MERGE GOOD RECORD	COUNTERS	T	SUMMARIZE COUNTS
COUNTERS	T	MERGE GOOD RECORD	COUNTERS	T	MERGE GOOD RECORD
COUNTERS	T	SET UP MERGE	OLD KEY EAST	T	MERGE GOOD RECORD
OLD KEY EAST	T	SET UP MERGE	OLD KEY WEST	T	MERGE GOOD RECORD
OLD KEY WEST	T	SET UP MERGE	IN-POINTER	T	COPY BAD
SET UP MERGE (S-7)					
NONE			COUNTERS	M	MERGE FILES
			OLD KEY EAST	M	MERGE FILES
			OLD KEY WEST	M	MERGE FILES
MERGE GOOD RECORDS (S-3)					
OUT OF SORT	CT	GET GOOD RECORD	IN-POINTER	M	GET GOOD RECORD
RECORD KEY EAST	C	GET GOOD RECORD	IN-POINTER	M	WRITE GOOD
RECORD KEY WEST	C	GET GOOD RECORD	IN-POINTER	M	MERGE FILES
RECORD EAST	T	GET GOOD RECORD	RECORD EAST	T	WRITE GOOD
RECORD WEST	T	GET GOOD RECORD	RECORD WEST	T	WRITE GOOD
COUNTERS	T	GET GOOD RECORD	OLD KEY EAST	T	GET GOOD RECORD
COUNTERS	T	MERGE FILES	OLD KEY WEST	T	GET GOOD RECORD
OLD KEY EAST	T	MERGE FILES	OUT OF SORT	T	MERGE FILES
OLD KEY WEST	T	MERGE FILES	COUNTERS	T	MERGE FILES
			COUNTERS	T	GET GOOD RECORD
COPY BAD (S-4)					
BAD END SWITCH	C	READ BAD	IN-POINTER	M	WRITE GOOD
IN-POINTER	P	MERGE FILES	IN-POINTER	M	READ BAD
BAD RECORD	T	READ BAD	BAD RECORD	T	WRITE GOOD
SUMMARIZE COUNTS (S-5)					
COUNTERS	P	MERGE FILES	NONE		
GET GOOD RECORD (S-7)					
IN-POINTER	CP	MERGE GOOD RECORD	OUT OF SORT	M	MERGE GOOD RECORD
RECORD KEY EAST	C	GET GOOD RECORD	COUNTERS	M	MERGE GOOD RECORD
RECORD KEY WEST	C	GET GOOD RECORD	RECORD KEY EAST	M	MERGE GOOD RECORD
OLD KEY EAST	CP	GET GOOD RECORD	RECORD KEY WEST	M	MERGE GOOD RECORD
OLD KEY WEST	CP	MERGE GOOD RECORD	OLD KEY EAST	M	GET GOOD RECORD
OLD KEY WEST	CP	GET GOOD RECORD	OLD KEY WEST	M	GET GOOD RECORD
OLD KEY WEST	CP	MERGE GOOD RECORD	RECORD EAST	T	MERGE GOOD RECORD
OUT OF SORT	P	MERGE GOOD RECORD	RECORD EAST	T	WRITE BAD
COUNTERS	P	MERGE GOOD RECORD	RECORD WEST	T	MERGE GOOD RECORD
RECORD EAST	T	READ EAST	RECORD WEST	T	WRITE BAD
RECORD WEST	T	READ WEST	IN-POINTER	T	WRITE BAD
WRITE GOOD (S-6,9)					
IN-POINTER	P	MERGE GOOD RECORD	NONE		
IN-POINTER	P	COPY BAD			
RECORD EAST	P	MERGE GOOD RECORD			
RECORD WEST	P	MERGE GOOD RECORD			
BAD RECORD	P	COPY BAD			
READ BAD (S-8)					
IN-POINTER	P	COPY BAD	BAD RECORD	M	COPY BAD
			BAD END SWITCH	M	COPY BAD
READ EAST (S-10)					
NONE			RECORD EAST		GET GOOD RECORD
READ WEST (S-11)					
NONE			RECORD WEST		GET GOOD RECORD
WRITE BAD (S-12)					
IN-POINTER	P	GET GOOD RECORD	NONE		
RECORD EAST	P	GET GOOD RECORD			
RECORD WEST	P	GET GOOD RECORD			

Figure 2—Input-output tables for example.

be. It probably is internal within this S-7 module. Care should be taken in such non-appearances to verify them to be reasonable, and not oversights in preparing the input-output tables. In the case of S-7, since the validation appears to be done in S-7 itself, no explicit loop exit data appears reasonable. Hence, *E* is 1 for all modules except S-3 and S-4.

In S-3, each of the three *C* items identified comes from a subordinate module which is within the loop body (hence, *E* is 3). But one *C* item (Out-of-Sort) also comes from outside the loop. Hence, *E* totals to 5 for the module S-3. In S-4, the Bad End Switch comes from the subordinate (loop body) module. Hence, *E* is 1 for S-4.

- The *E* to *R* conversion for all modules is very easy for all but two modules. Since *E* is 0, *R* is 1.0. For module S-3, it is 3.78, and for S-4, it is 1.11, by applying the formula.
- The column PROD is the product of the entries in the W-TOT column times the corresponding entry in the *R* column. Thus, for row S-4, the product of 9 times 1.1 is 9.9.
- The square roots of the entries in the PROD column are entered in the *Q* column. Thus, the square root of 9.9 is 3.1, which is the *Q* entry for module S-4. The *Q* entry is the complexity index for a module.
- The sum of the *Q* entries for the modules is 35.2, which when divided by 12, the number of modules, yields a program complexity index of 2.9.
- Since this example system consists of one program, the system index of complexity *Q* is also 2.9.

The interpretation of the *Q* measure is easy. Low-complexity is indicated by a low *Q*. But also, a relatively even distribution of the complexity is desirable among the modules. Software prepared using the ideas of both structured design and structured programming show low complexity compared to traditionally prepared software. The highest complexity in the structured software is usually along the main branches of the tree-like structure. But even there, the *Q* rarely exceeds 11. Leaf modules rarely exceed a *Q* of 5.

Traditionally prepared software usually shows a higher average complexity. If the software is modular in design, the *Q* can be fairly easily determined. If it is not modular, two alternatives are open. When source code is available, the modules may be taken to be equivalent to the lexical units used to group code, such as paragraph, section, sub-routine, procedure, block, etc. Or when no source code is available, the design documentation may be examined, broken arbitrarily into pieces, and input-output tables prepared. This is rarely an easy process if done in an attempt to highlight and separately recognize functions jumbled together in skimpily-documented traditional designs.

COMPARISON OF COMPLEXITY MEASURES

Five major measures of software complexity have been proposed. McCabe has offered a graph-theoretic measure,¹³ which others have elaborated.^{7,12,22,31} An application of it to the same example presented earlier in this paper is given in the McCabe column in Figure 4. McClure has offered a carefully-thought-out and well tested measure.^{18,19} An application of it is given in the McClure column in Figure 4. Myers has given a basis for the measurement of software quality.^{23,24} While not labeling it a complexity measure, his connectivity matrix measure can be not unreasonably interpreted in that way. An application of it is given in the Myers column in Figure 4. A group of entropy-based measures have been proposed^{1,9,11,26} but are not shown in Figure 4. The Zolnowski measure has been claimed by Zolnowski to be not applicable to an individual program and to individual modules as it has been presented thus far.³² An adaptation

S#	RAW COUNTS				WEIGHTED COUNTS				W-TOT	R	PROD	Q
	C	P	M	T	C	P	M	T				
1	1	0	1	10	3	0	2	5	10	1.0	10.0	3.2
2	0	0	3	0	0	0	6	0	6	1.0	6.0	2.4
3	3	0	3	14	9	0	6	7	22	3.8	83.6	9.1
4	1	1	2	2	3	1	4	1	9	1.1	9.9	3.1
5	0	1	0	0	0	1	0	0	1	1.0	1.0	1.0
6	0	3	0	0	0	3	0	0	3	1.0	3.0	1.7
7	7	7	6	8	21	7	12	4	44	1.0	44.0	6.6
8	0	1	2	0	0	1	4	0	5	1.0	5.0	2.2
9	0	2	0	0	0	2	0	0	2	1.0	2.0	1.4
10	0	0	1	0	0	0	2	0	2	1.0	2.0	1.4
11	0	0	1	0	0	0	2	0	2	1.0	2.0	1.4
12	0	3	0	0	0	3	0	0	3	1.0	3.0	1.7

N = 12 = number of modules

Program complexity index 2.9

Figure 3—Computation of proposed complexity index Q for the example shown in Figures 1 and 2.

of it to modules and programs by McTap²¹ is shown in the $M \times Z$ column in Figure 4. The Q column in Figure 4 is the measure proposed in this paper.

A comparison of these other measures with the measure proposed here reveals both similarities and differences, as well as supports the validity of the proposed measure. All of the measures provide an overall indicator of software complexity suitable for making complexity comparisons between systems, programs and modules. But each measure requires a different interpretation.

The McCabe graph-theoretic has 1.0 as a lower bound, but has no upper bound, and grows faster than the number of modules, because a module never adds less than 1 to the total. The Zolnowski measure shows no consistent relationship to program or system size. The Myers connectivity matrix measure can not exceed 1.0 in value. In practice, for structured programs and systems, the Myers measure typically declines as program or system size increases. The proposed Q measure has a lower bound (zero), no upper bound, and shows a tendency to increase slowly as program or system size increases.

The McClure measure has well defined limits, depending on the number of control variables (C role data items). But as system or program size increases, larger numbers can be expected. This is intuitively reasonable (the growth generally is less than in the McCabe graph-theoretic measure), but a measure that gave "complexity density" would probably be more useful than just "complexity extent." The proposed Q measure does do this.

At the module level, all the measures are suitable for making comparisons between modules from different programs and systems. The McCabe graph-theoretic measure reflects primarily the amount of logic expressible as conditional transfers in the flow of control. In any modular approach, this is a useful type of complexity to control during design, implementation and maintenance.

The McTap-Zolnowski features-measure of complexity depends for its interpretation on the particular features included. A high measure indicates that some of the features are present to an important degree.

The Myers connectivity-matrix measure reflects the relative dependence of the modules. The higher the measure,

MODULE	McCABE	McTAP - ZOLNOWSKI	McCLURE	MYERS	Q	GROUP
S-1	7	6.5	2.20	0.70	3.2	2
S-2	1	0.0	0.55	0.70	2.4	5
S-3	3	6.5	2.75	0.65	9.1	3
S-4	2	2.5	0.55	0.55	3.1	4
S-5	1	4.8	0.55	0.30	1.0	10
S-6	1	1.0	1.10	0.25	1.7	7
S-7	5	7.5	3.74	0.65	6.6	1
S-8	1	1.0	0.00	0.65	2.2	8
8-9	1	1.0	0.00	0.25	1.4	6
S-10	1	1.0	0.72	0.25	1.4	11
S-11	1	1.0	0.72	0.25	1.4	12
S-12	1	1.0	0.92	0.55	1.7	9
PROG	24	5.5	1.25	0.58	2.9	--

Figure 4—Comparison for the example of some measures of complexity.

the more interdependent are the modules. But a high measure may also reflect functional variety and code packaging.

The proposed Q measure reflects what will take the form of actual code complexity in implementation because the more data that are used and produced, the more likely is the processing to be complex. Complexity is harder to achieve with only a few items of data going in and out!

The McClure measure is the most sensitive to the use of data for directing the flow of control. Historically, this has been the source of the toughest bugs in our software. And the McClure measure reflects well the complexity of the patterns of use and value-assignment for the data serving for control.

All of the measures can be used with modular designs characterized by properly nested functions. The Zolnowski measure does not require it. The McClure measure requires it, making provision for only one exception, the equivalent of the job abort. The Myers connectivity matrix and the proposed measures do not require it, but are enhanced by it. More significant for both of them is the size of the module. Both work best for relatively small (less than about 60 imperative instructions expected for the implementing code) and fairly even-sized modules. The McCabe graph-theoretic measure is comparatively independent of the design and

implementation philosophy and practice. In fact, it can be used to limit the size of modules for the aid of the other measures (such as not more than an expected 10 measure and only rarely over an expected 8 measure for the modules).

Some usage difficulties and conveniences distinguish the measures of complexity. For the Zolnowski measure, they depend upon the features selected. The entropy and McCabe graph-theoretic measures are almost always an understatement of the ultimate complexity until the design has been carried fully to debugged code.^{9,11} But by then, it is usually too late to take much corrective action. This can be offset by a tight early discipline in design, but few designers welcome it.

The preparation of the Myers connectivity matrix is a separate additional step—not a normal by-product of design. With experience the matrix preparation can be done fairly rapidly, and it does not require either source code or detailed charts to be available. A tree-structure chart⁴ may be sufficient, but the availability of input-output tables⁵ strengthens the preparation of the connectivity matrix.

To use the McClure measure takes the same detailed review of the design as needed for the Myers connectivity matrix. But the factors looked for are far more objective in

the McClure measure than in the Myers measure. Yet the McClure measure, like the McCabe measure, is a retrospective measure of the complexity already present in the design. By contrast, the proposed Q measure can be used on-the-fly during design, implementation and maintenance. It does not require source code. It rarely misstates the final complexity of the software as coded if the input-output tables were conscientiously prepared.

A more serious limitation for the McClure measure and to a lesser extent the Myers connectivity matrix measure, is a failure to define data usage consistently. Use of the input-output tables helps reduce the problem but lacking good definition, the effect is to understate the complexity for all of these measures. The McClure measure requires a full and precise anticipation of the C and M roles in every module of every item of data used anywhere for control. The proposed Q method requires that any item of data to be accessed or assigned a value in any module be separately identified, such as a record and the key field within that record. But this is only a statement in data terms of the function of the module. And that is knowledge available to the designer.

The complexity measures differ considerably in computational convenience. The most difficult is the Zolnowski measure because it requires extensive data gathering, computation, measurement and further computation. Somewhat easier is the McClure measure because it involves fewer operations to gather the needed data and fewer computational steps. The next easiest is the Myers connectivity matrix. While the arithmetic is easy, the estimates of strength and coupling involve significant human judgments. Some of the entropy measures, such as the Halstead, are as easy to compute, and are free of the need for such extensive judgments. The McCabe graph-theoretic is still easier, and would be the easiest were it not for the difficulty in obtaining firm counts for the lines, the nodes and branches. The proposed Q method is clearly the easiest of all to use if input-output tables⁵ or the equivalent, or code, or Chapin charts,⁶ be available, since simple objective counts and simple arithmetic then yield the proposed Q measure. HIPO¹⁴ can be used but adjustments are needed since a HIPO detail chart is not limited in its view to a single module except at the leaf position in the tree. When input-output tables are available from the design effort, the proposed Q measure can be calculated by someone without even a knowledge of computers or data processing.

DISCUSSION

The validity of any proposed measure of software complexity cannot be assessed with precision. As Zolnowski has well pointed out,³² people view software differently and see its complexity differently. In general, an index is said to be valid if it measures what it purports to measure.²⁸ Thus, changes in what it purports to measure should be reflected faithfully in the measure. The validity of some measures is open to question or left unaddressed.^{10,16,27,31}

Comparing the proposed Q measure against four of the

other measures cited offers a measure of validity, as shown in Figure 4. On that basis the proposed measure seems as good as any of the others in terms of validity. The column in Figure 4 identified as GROUP represents the rank of the average rankings of the modules in the example by 206 programmers and analysts who had access to a full set of documentation. A rank of 1 represents the most complex, and of 12 the least complex. A ranking does not discriminate the extent of the differences in complexity. Thus, modules S-10 and S-11 were ranked virtually identically. The module S-1 includes a four-way CASE structure which some people regarded as complex. Module S-1 nearly tied for second place with module S-3 in the rankings. No ranking was made of the program overall.

It is surprising that the theory of computational complexity, long a part of the mathematical side of computer science, has contributed so little to measuring software complexity.^{3,13} Perhaps in the future, some contribution will be forthcoming to help assess the validity of measures of software complexity. In the meantime, field experience can help evaluate the contribution in the development and maintenance of systems and programs, of the use of measures of complexity.

REFERENCES

1. Belady, Les A., "Software complexity," *Software Phenomenology*, U.S. Army Systems Command, Washington, D.C., Aug. 1977, 13 pp.
2. Belady, Les A., and M. M. Lehman, "A model of large program development," *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 225-252.
3. Borodin, A., "Computational complexity and the existence of complexity gaps," *Journal of the ACM*, Vol. 19, No. 1, Jan. 1972, pp. 158-183.
4. Chapin, Ned, "Aids to producing comprehensible software," *Structured Programming*, Infotech International Ltd., Maidenhead, UK, 1976, pp. 165-181.
5. Chapid, Ned, "Input-output tables in structured design," *Structured Analysis and Design (Volume 2)*, Infotech International Ltd., Maidenhead, UK, 1978, pp. 43-55.
6. Chapin, Ned, "New format for flowcharts," *Software Practice and Experience*, Vol. 4, No. 4, Oct.-Dec. 1974, pp. 341-357.
7. Chen, Edward T., "Program complexity and programmer productivity," *Software Engineering*, Vol. SE-4, No. 3, May 1978, pp. 187-194.
8. Dijkstra, E. W., "Complexity controlled by hierarchical ordering of function and variability," in Buxton, J. M., et al., eds., *Software Engineering Concepts and Techniques*, New York, Van Nostrand Reinhold Co., 1976, pp. 114-116.
9. Elshoff, James L., "Measuring commercial PL/I programs using Halstead's criteria," *SIGPLAN Notices*, Vol. 11, No. 5, May 1976, pp. 38-46.
10. Gideadi, Amos N., and Henry F. Ledgard, "On a proposed measure of program structure," *SIGPLAN Notices*, Vol. 9, No. 5, May 1974, pp. 31-36.
11. Halstead, Maurice H., *Elements of Software Science*, New York, Elsevier North-Holland, Inc., 1977, 127 pp.
12. Hanson, Wilfred J., "Measurement of program complexity by the pair cyclomatic number, operator count," *SIGPLAN Notices*, Vol. 13, No. 3, March 1978, pp. 29-33.
13. Hartmanis, J., and J. E. Hopcroft, "An overview of the theory of computational complexity," *Journal of the ACM*, Vol. 18, No. 3, July 1971, pp. 444-475.
14. IBM Corp. *HIPO*, IBM Corp., White Plains, NY, 1974, 129 pp.
15. Infotech, eds., "Complexity in programming," *Structured Programming*, Infotech International Ltd., Maidenhead, Berkshire, England, 1976, pp. 25-28, 65-96.
16. Martyn, J., and B. C. Vickery, "The complexity of the modeling of

- information systems," *Journal of Documentation*, Vol. 26, No. 3, Sept. 1970, pp. 204-220.
17. McCabe, Thomas J., "A complexity measure," *Software Engineering*, Vol. SE-2, No. 4, Dec. 1976, pp. 308-320.
 18. McClure, Carma L., *Formalization and Application of Structured Programming and Program Complexity*, Ph. D. Thesis Illinois Institute of Technology, Chicago, 1976, 288 pp.
 19. McClure, Carma L., *Reducing COBOL Complexity Through Structured Programming*, Van Nostrand Reinhold, New York, 1978, 192 pp.
 20. McGowan, Clement L., and John R. Kelly, *Top Down Structured Programming Techniques*, Van Nostrand Reinhold, New York, 1975, 288 pp.
 21. McTap, John L., "The complexity of an individual program," publication pending.
 22. Myers, Glenford J., "An extension to the cyclomatic measure of program complexity," *SIGPLAN Notices*, Vol. 12, No. 10, Oct. 1977, pp. 62-64.
 23. Myers, Glenford J., *Composite/Structured Design*, Van Nostrand Reinhold Co., New York, 1978, 174 pp.
 24. Myers, Glenford J., *Reliable Software through Composite Design*, Van Nostrand Reinhold, New York, 1975, 159 pp.
 25. Parnas, David L., "A technique for software module specification with examples," *Communications of the ACM*, Vol. 15, No. 5, May 1972, pp. 330-336.
 26. Shooman, M., and A. Laemmel, "Statistical theory of computer programs in information content and complexity," *Proceedings of the 1977 Fall COMPCON*, IEEE, Long Beach, CA, 1977, pp. 341-347.
 27. Sullivan, J. E., *Measuring the Complexity of Computer Software*, MTR-2648, Mitre Corp., Bedford, MA, 1973, 241 pp.
 28. Thorndike, Robert L., and Elizabeth P. Hagen, *Measurement and Evaluation in Psychology and Education*, John Wiley & Sons, Inc., New York, 1977, 693 pp.
 29. Underhill, L. H., "The growth of complexity of a general-purpose program," *The Computer Journal*, Vol. 6, No. 1, Jan. 1963, pp. 37-38.
 30. Weissman, Larry, "Psychological complexity of computer programs: an experimental methodology," *SIGPLAN Notices*, Vol. 9, No. 6, June 1974, pp. 25-36.
 31. Woodward, Martin R., Michael A. Hennell, and David Hedley, "A measure of control flow complexity in program text," *Software Engineering*, Vol. SE-5, No. 1, Jan. 1979, pp. 45-50.
 32. Zolnowski, Jean C., and Dick B. Simmons, "Measuring program complexity," *Proceedings of the 1977 Fall COMPCON*, IEEE, Long Beach, CA, 1977, pp. 336-340.

Relating computer program maintainability to software measures

by A. R. FEUER

Bell Telephone Laboratories
Piscataway, New Jersey

and

E. B. FOWLKES

Bell Telephone Laboratories
Murray Hill, New Jersey

MOTIVATION

It is no longer a surprise that program maintenance dominates the total cost of a large software system over its lifetime. In response to these costs, the emphasis in program design has largely shifted from the time and space issues of machine efficiency to issues of clear and flexible program structures that can be easily maintained.

The goal of this project is to identify measurable program properties that influence maintainability. More precisely, we examine the effect of various program characteristics on the subsequent frequency and magnitude of program errors.

PROGRAM MAINTAINABILITY

This paper presents a study of 123 PL/I program modules from a business data processing application. The modules fall roughly into three categories: high-level control, numerical and data base management. The maintenance record of each module was followed for approximately one year.

We characterize the maintenance performance of a program module by two values—the total maintenance time spent on the module and the number of maintenance changes made to the module. We considered a module in “maintenance” from the time it left the development programmer and entered system testing. The maintenance performance data came from two sources:

1. Informal (and incomplete) time records, recorded by hand, that include time spent on and cause of maintenance activities.
2. A formal (and complete) maintenance activity data base, recorded automatically when a program is changed, that does not include time spent per activity.

In this paper we present an analysis based upon the informal time records, chiefly:

- The number of errors per program module.
- The total time spent repairing those errors.

PROGRAM MEASURES

Our starting hypothesis was that maintenance performance for a program depends upon:

- The complexity of the algorithm coded.
- The clarity of the coding.

(It is beyond the current scope of this work to consider influences on performance beyond the program itself, such as frequency of execution or instability of user requirements.)

Based upon previous work^{1,4,17} and upon programming folklore we compiled a set of program properties for which the qualifiers complex and clear might have meaning. The list includes:

- Complexity of program control flow.
- Clarity of program control structure.
- Clarity of program data usage.

The next step was to develop measures to quantify these properties. Two primary criteria were adhered to in developing the measures:

1. Each measure should be largely language-independent.
2. Each measure should be noncoercible.

A measure is language-independent if:

- It can be meaningfully and consistently applied to programs written in several programming languages.
- The ordering the measure assigns to a set of algorithms

remains roughly constant when the algorithms are coded across languages of similar power.

One language-dependent measure that has been used to characterize programs is a count of the GOTO statements appearing in a module.^{1,13} Counting GOTO statements is language-dependent because the practical meaning of GOTO changes as control structures change across languages. One language may require a GOTO to implement an innate control structure of another language.

A measure is noncoercible if it in fact measures the underlying program property of interest. Identifying program characteristics that may coincidentally vary with a fundamental property tells us little about the property.

Coercible measures that have been used to characterize programs include calculating the mean length of variable names as a measure of name meaningfulness,¹³ and counting the number and length of comment blocks as an indicator of program documentation.^{13,15} We have called these measures coercible since they can (quite easily) be influenced without any corresponding effect on the property they seek to measure.

Measuring the complexity of program control flow

Program control flow was characterized by measurements taken from the control flow graph for each program. A control flow graph is a directed graph with nodes corresponding to the simple clauses of a program and arcs indicating the sequence of control. The graph is connected, unreachable clauses are ignored. And the graph is single-entrance-single-exit—an entry node points to all entry points and an exit node is pointed to by all exit points (Figure 1).

Two simple graph measures were included for study:

- A count of the nodes in the graph.
- The ratio of binary decisions embedded in the graph to total nodes. (The number of binary decisions at a node is one less than the number of outgoing arcs from the node.)

Two other graph measures, sensitive to the configuration of the graph, were also studied:

- A count of the possible paths through the graph.
- The mean number of decisions per path through the graph.

We define a path through a graph to be a sequence of nodes, from the entrance node to the exit, such that no cycle is repeated. Even with this restriction the number of paths through a typical graph can be very great. For the more complex programs the total path count was estimated by a lower bound, and the mean path length was estimated from a sampling of the paths.

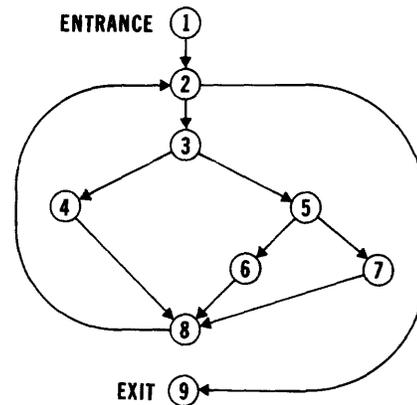
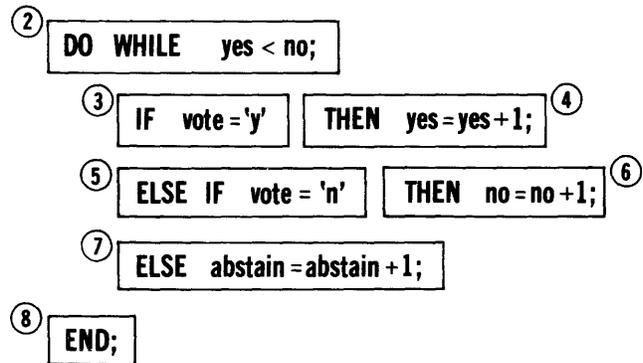


Figure 1—A control flow graph.

Measuring the clarity of program control structure

Structured programming has evolved as a guide to writing more easily understood programs. It is generally agreed that well structured code is composed from blocks, each with a single entrance and a single exit. In the strictest sense, the lowest level block can be taken to be a simple clause in a program, a node in the control flow graph. A simple reduction rule exists to collapse graphs of strictly well structured programs to a single node:

1. Choose a node n that has at most one incoming arc and at most one outgoing arc.
2. Replace n by an arc, preserving direction, that connects n 's neighbors (m is a neighbor of n if there is an arc between m and n).
3. Remove any redundant arcs:
 - Two arcs in the same direction between the same nodes is a redundancy.
 - An arc from a node to itself is a redundancy.

The three steps are repeatedly applied until they are no longer applicable (Figure 2).

After applying the reduction rule to a graph, some subgraph will remain. The subgraph will be a single node for

REDUCTION RULE

STEP A: COLLAPSE ANY NODE THAT HAS AT MOST ONE INCOMING ARC AND AT MOST ONE OUTGOING ARC

STEP B: REMOVE ANY EXTRANEIOUS ARCS

REPEATEDLY APPLY EACH STEP UNTIL NEITHER IS APPLICABLE

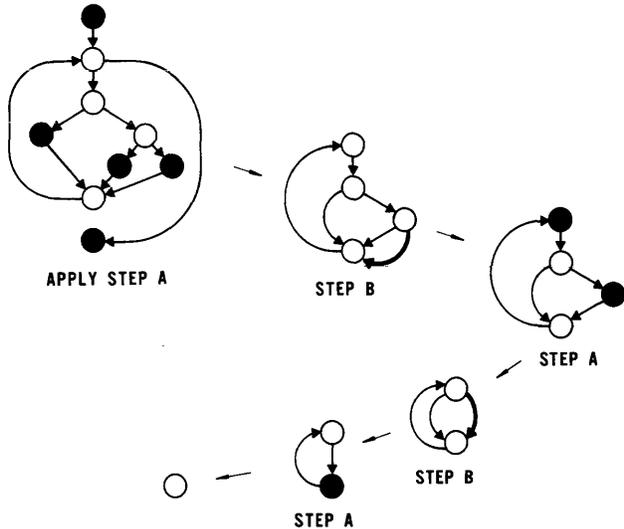


Figure 2—Reducing a Graph.

strictly well structured programs; the subgraph will be more complex for other programs. We characterize the degree to which a graph can be reduced by the ratio of binary decisions embedded in the subgraph to binary decisions embedded in the original graph.

Measuring the clarity of program data usage

One characteristic of data usage is the degree to which variable use has been localized. Measuring the span of a variable is one indicator of localization. A span of a variable is one plus the number of intervening statement clauses between two successive references to the variable. The average span of a variable is the sum of all spans for the variable divided by the number of spans. The variable span for a program is the sum of average variable spans for all the variables within the program (Figure 3).

Software science measures

The field of software science is attempting to establish meaningful relationships among the primitive components of algorithms—operators and operands (Figure 4). Work by others^{10,16} has indicated that some of the software science variables suggested by Halstead⁹ may reliably characterize the overall complexity of a program.

In our analysis we studied six software science variables.

All can be derived from counts of: Unique operators ($n1$), unique operands ($n2$), instances of operators ($N1$), and instances of operands ($N2$).

- *Length*— $N1 + N2$. *Length* is a program size measure of finer granularity than a count of program statements or clauses. A virtue of *length* is that it is largely insensitive to horizontal (deeply-nested expressions) versus vertical (simple expressions) programming style.
- *Expected length*— $(n1 \log_2 n1)(n2 \log_2 n2)$. Empirical studies have shown *expected length* and *length* correlate highly for published, hence presumably well polished, programs.^{2,8} We used the calculation $(length - expected\ length) / length$ to indicate the agreement between *length* and *expected length*.
- *Volume*— $Length \log_2(n1 + n2)$. *Volume* is an estimate of the minimum number of bits needed to represent the executable statements of a program.
- *Level*— $(2/n1)(n2/N2)$. *Level* is a measure of the match between the operations performed by a program and the primitive operators and functions used by the program. In a program at the highest *level* the operation performed is implemented by a single operator.
- *Effort*— $Volume / level$. Intuitively, *effort* is a function of the quantity of information represented by a program and the power of the statements with which the information is encoded. *Effort* rises with increasing information content and with decreasing statement power.

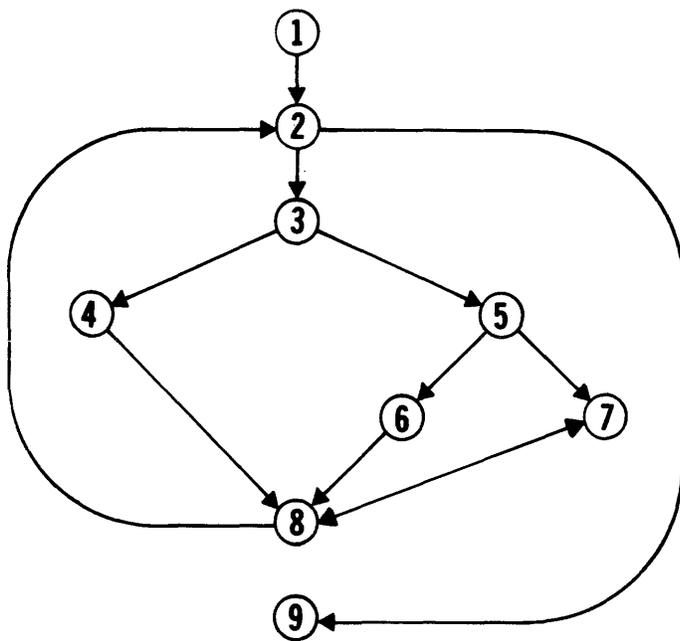
DATA ANALYSIS

An analysis of 12 carriers, or independent variables, and two responses, or dependent variables, is presented here.¹⁴ Figure 5 summarizes the range of values for each of the carriers.

The carriers were first studied singly and in pairs. Normal quantile-quantile plots⁶ for each carrier revealed that most had non-normal, asymmetric distributions. Transformations (usually the natural logarithm) were carried out which rendered the distributions approximately normal and symmetric. We used the carrier *node count* as a simple measure of module size. Scatter plots of the eleven other carriers versus *node count* revealed that seven were highly correlated with module size and four were not (Figure 5).

In order to determine whether there was a significant stratification of the modules according to size, the hierarchical clustering algorithm of Johnson, using Manhattan distance and the complete linkage method,^{5,10} was used for the 123 modules according to the eight standardized size-related carriers. A dendrogram of the result is shown in Figure 6. The figure shows strong clustering. If the tree is cut at a distance of 9.0, there appear to be six clusters.

An attempt was next made to relate the response *time to repair errors* to module size. Program maintenance data showed that there were a total of 124 errors on 45 modules. The 124 errors were classified into six groups according to the assignment of the modules with errors to the six size

**REFERENCES**

② yes, no

③ vote

④ yes

⑤ vote

⑥ no

⑦ abstain

$$\text{MEAN SPAN FOR VARIABLE} = \frac{\text{LAST REFERENCE} - \text{FIRST}}{\text{NUMBER OF REFERENCES} - 1}$$

SPAN FOR A PROGRAM = SUM OF MEAN SPANS FOR ALL VARIABLES

for this graph:

$$\text{yes: } \frac{4-2}{1} = 2$$

$$\text{no: } \frac{6-2}{1} = 4$$

$$\text{vote: } \frac{5-3}{1} = 2$$

$$\text{abstain: } = 0^*$$

$$\text{SPAN} = 8$$

*By convention a variable referenced only once has a span of zero

Figure 3—Variable Span.

```

DO WHILE yes < no ;
  IF vote = 'y' THEN yes = yes + 1 ;
  ELSE IF vote = 'n' THEN no = no + 1 ;
  ELSE abstain = abstain + 1 ;
END ;
    
```

OPERATORS	INSTANCES	OPERANDS	INSTANCES
DO-END	1	yes	3
WHILE	1	no	3
<	1	vote	2
;	5	'y'	1
IF-THEN	2	1	3
= (equality)	2	'n'	1
= (assignment)	3	abstain	2
+	3	<hr/>	<hr/>
ELSE	2	7	15
<hr/>	<hr/>		
9	20		

Figure 4—Software science primitives: operators and operands.

clusters. Average *node count* was calculated for each group, and the *times to repair* (in quarter-hour units) were plotted versus average *node count* (Figure 7). (A random normal deviate from $N(0, \epsilon)$ was added to each of the x and y coordinates of every point so that the density of points in each

group could be seen.) A striking dependency can be seen; the spread of *time to repair* increases with *node count*.

Actually, since at *node counts* of 50, 100 and 220 there appear to be one or two points quite separated from the rest (the observation of 15 hours at *node count* 220 is not a

• Size Related

CARRIER	25% Point	50% Point	75% Point
node count	28.5	50	104
path count	11.5	58	738
mean path length	6	10	21
length	137	252	601
expected length	173.5	340	645.5
volume	725	1523	4140
level	.0231	.0386	.0661
effort	12919	35213	179472

• Non Size Related

CARRIER	25% Point	50% Point	75% Point
decision density	.1263	.1579	.1892
variable span	6.5	29	139
percent reduction	71.82	100	100
(len-exlen)/len	.2311	.3764	.5931

Figure 5—Carrier summary.

mistake), it is not unreasonable to hypothesize two mechanisms in operation here. Figure 7 suggests a model for such a phenomenon superimposed on the data. The model says that there are two types of errors that occur, easy-to-repair and difficult-to-repair. The easy-to-repair errors occur with large probability and the difficult-to-repair with small prob-

ability. Points that might have come from the difficult-to-repair distribution have been circled in Figure 7. An exponential fit was estimated for these points, but this estimate may be very unreliable because of the small number of points. Nevertheless, the model does provide an interesting framework to study the dependency of *time to repair* on module size.

An important question addressed next in the analysis was whether there might be some residual or second-order effect of the seven size-related carriers other than *node count* on the responses. Size-adjusted variables were created for the seven carriers by performing simple linear regressions of the log carrier on log *node count*. Simple linear regression appeared to be adequate for each case. The seven sets of residuals about regression formed new adjusted carriers that were relatively free of *node count*.

A hierarchical clustering of the 123 modules according to the standardized residual carriers was conducted using Manhattan distance and the complete linkage method. A dendrogram is shown in Figure 8. The dendrogram appears to show reasonably strong clustering. If the tree is cut at a distance equal to 8.5, 13 clusters result. As a check, clusterings using different methods and different metrics were tried. It was seen that modules combined in the tree at different heights and in slightly different orders, but the basic grouping remained stable. In the analysis that follows, only the response *error density* will be considered. *Error*

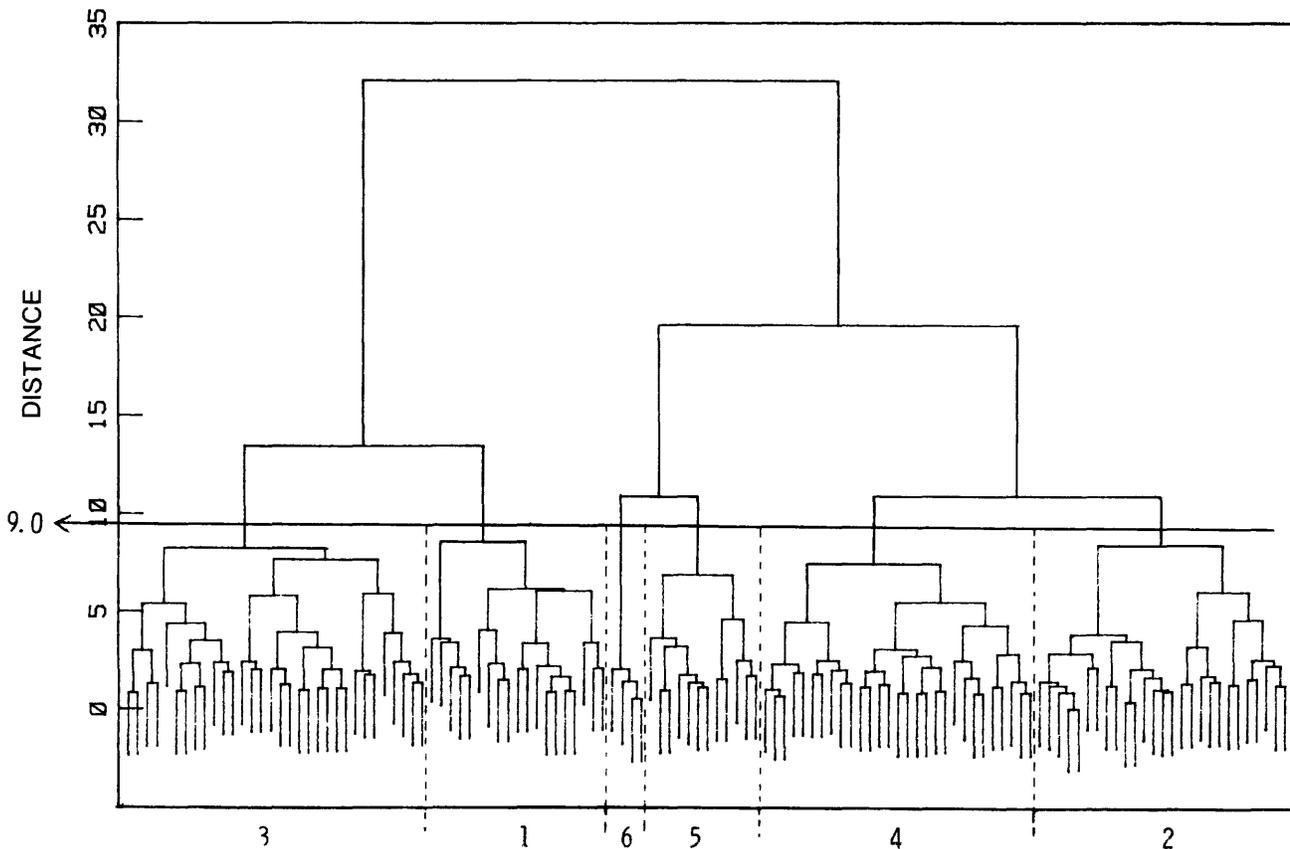


Figure 6—Clustering of eight standardized log carriers using manhattan distance.

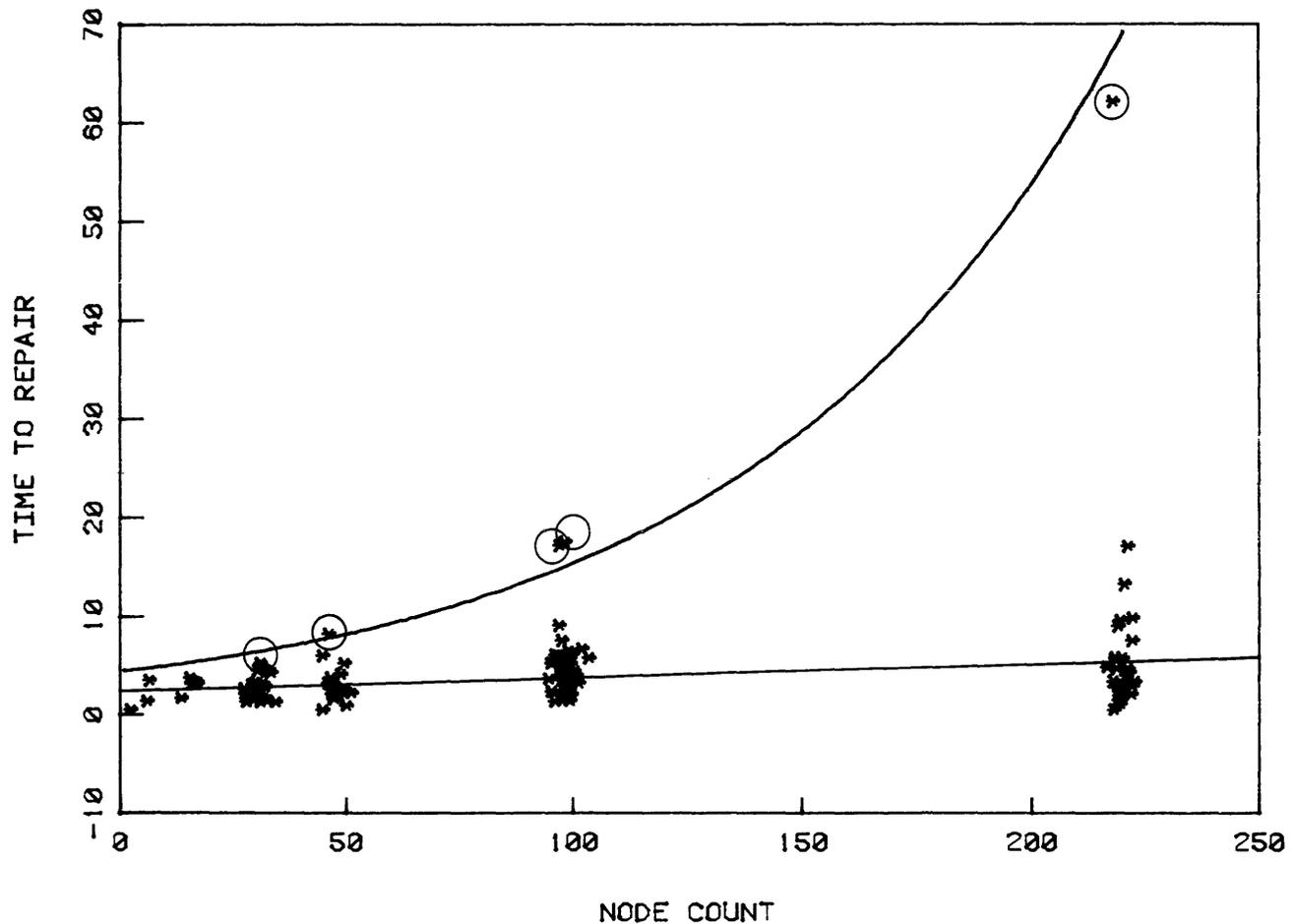


Figure 7—Time to repair versus node count for six size clusters.

density is defined as the total number of errors for a group of modules divided by the number of modules in the group which contain errors.

The seven residual carriers were next investigated for multi-collinearity. A principal components analysis of the correlation matrix⁷ of these carriers revealed that only three of the seven carriers were not collinear since the first three principal axes accounted for 99% of the total variability. Before proceeding further, the multi-collinearity of the carriers was removed. Weighted (by reciprocal variance) averages of residual *length* and *volume* and residual *path count* and *average path length* were taken, thereby reducing the number of carriers to five. Next, the first and fourth carriers of the five-carrier set were eliminated. These carriers had the lowest simple correlation with the response *error density*. This leaves as carriers residual *level*, residual *expected length*, and a weighted average of residual *path count* and residual *average path length*. A principal components analysis for these three variables showed no singularity.

One approach to relating the *error density* to the carriers is to group the modules with errors according to the 13 clusters found previously, calculate average *error density* for each group, and regress these averages on representative

values of the adjusted carriers for each group. The representative value for each group was taken to be the centroid of the final three adjusted carriers. Since each group has a different number of modules with errors, the average responses are based on differing numbers of items and weighted regression is appropriate. In carrying out this grouping-averaging process, regression analysis was used in its classical sense of estimating the mean of the conditional distribution of the response, conditional on values of the carriers. There was a large amount of inherent variability in the original data, in part because the data was not collected as part of a designed experiment. The averaging carried out above reduced this variability and allowed the marginal effects of an adjusted carrier to be seen more clearly.

Next we formed a regression model of *error density* and our three-carrier set. However, it might be possible for the carriers to affect the response interactively. To account for this, we included cross-product terms in the regression model. When this is done, six carriers result (Figure 9).

In order to select usable models containing combinations of the six carriers, the C_p statistic³ was calculated for each possible regression containing the carriers. These statistics were plotted against p , the number of terms in each regres-

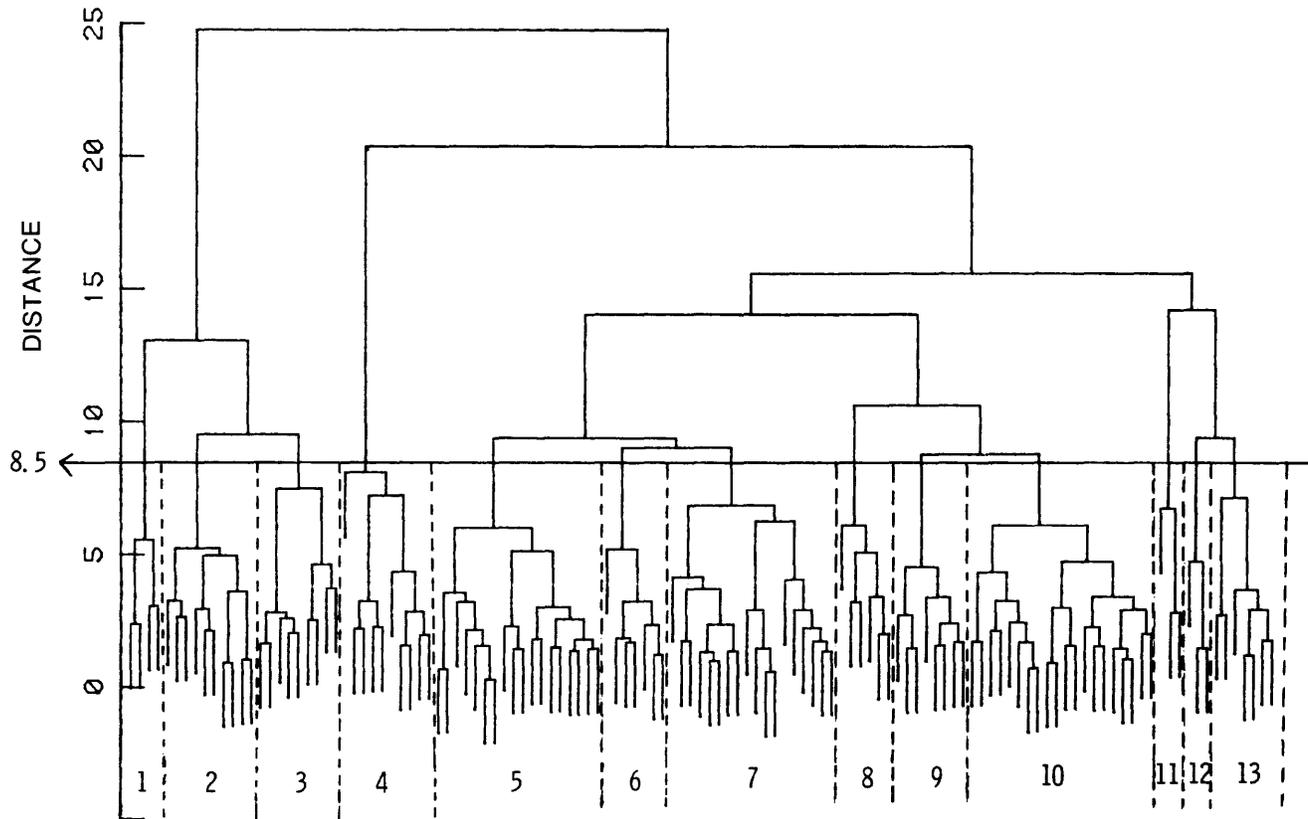


Figure 8—Clustering of standardized residuals from regression on node count using manhattan distance.

sion, and are shown in Figure 10. Here the maximum p equals seven because a constant term was added in each regression. The line $C_p=p$ was added in order to indicate good results. Any combination of variables that lie on this line are good models for the regression. A dashed line was drawn to show reasonable models for the data. We considered a model reasonable if it had a mean square error at least as small as the model with all the terms in it. All the models include variable 1, residual *level*, which by itself almost provides a reasonable model for the data. Notice that some of the candidate models include interaction terms without containing the corresponding main effects; such models were not considered acceptable. One of the best and most plausible models comprises the terms 1, 2, 3, and 4. These are residual *level*, residual *expected length*, their interaction and the residual *path carrier*. The result of this analysis shows that there is an apparent residual effect of three of the seven original carriers.

-
- 1. Residual *level*
 - 2. Residual *expected length*
 - 3. Weighted average of residual *path count* and residual *mean path length*
 - 4. Cross product of carriers 1 and 2
 - 5. Cross product of carriers 1 and 3
 - 6. Cross product of carriers 2 and 3
-

Figure 9—Reduced Carrier Set

The regressions for models that were considered reasonable were all significant at the $\alpha=.01$ level but are not strong enough for accurate prediction (the multiple R^2 is less than .40 for regressions performed on the raw, unaveraged data). However, they do provide ideas for future experiments and food for thought.

SUMMARY OF FINDINGS

Both the data collection and analysis are at an early stage; yet a few results are evident.

1. Most of the variables we studied have a large size component.
2. Module size, by itself, appears to be a good indicator of maintenance performance for the module, though we have not studied the tradeoff of many small modules versus fewer large modules.
3. When adjusted for size, *level* appears to be a fair indicator as to how a group of modules will perform.

While we do not expect to predict the maintenance performance of a single module based upon a static analysis as presented here, we do expect to find program properties characteristic of poor performance. Such properties would

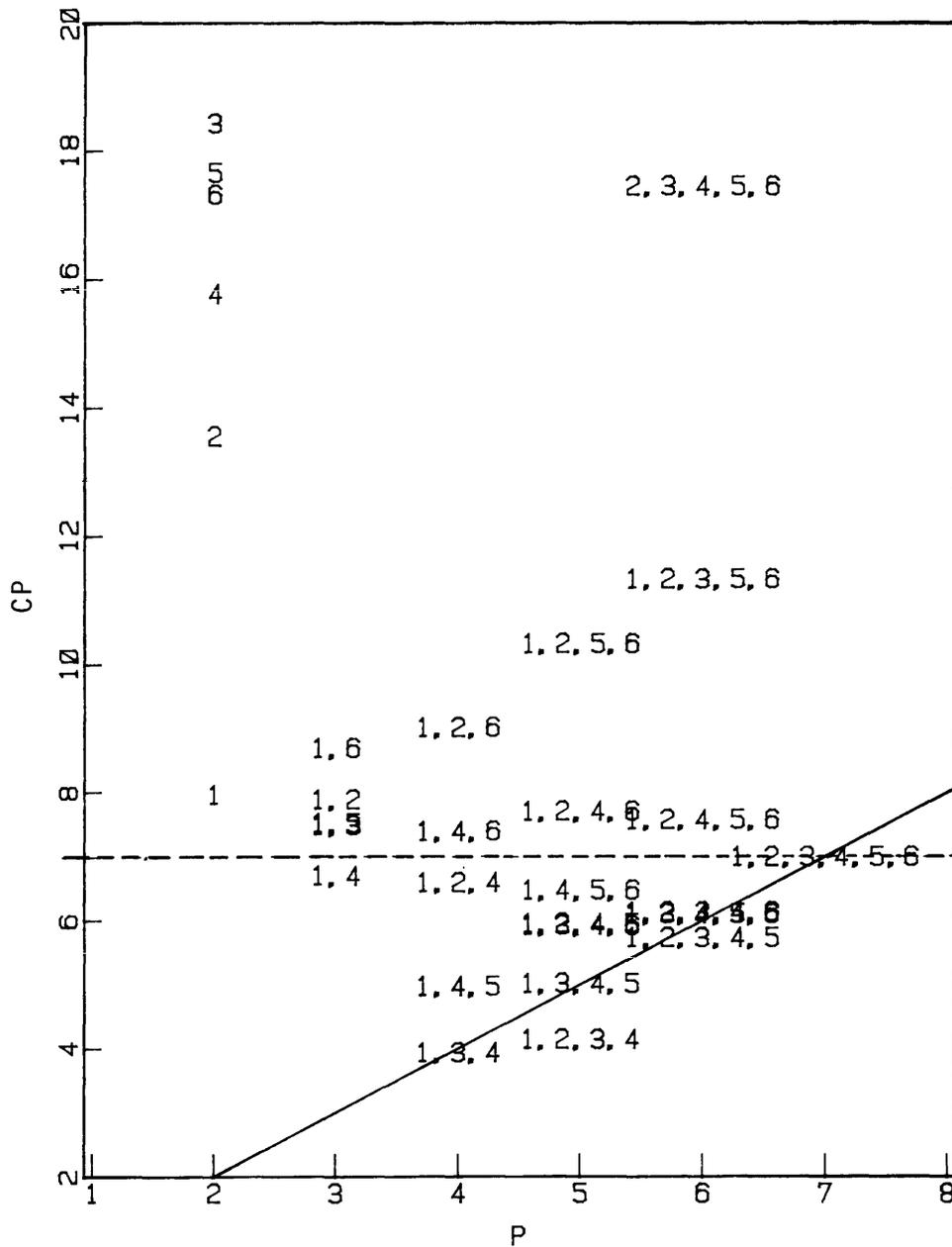


Figure 10—Cp plot for error density.

lead us to firmer ground for establishing guidelines of good programming practice.

ACKNOWLEDGMENTS

We are indebted to J. C. Gear and D. E. Pinkston for their assistance with this work.

REFERENCES

1. Amster, S. J., et al., "An Experiment in Automatic Quality Evaluation of Software," *Bell Laboratories Technical Memorandum*, May 1974.

2. Bohrer, R., "Halstead's Criterion and Statistical Algorithms," *Proceedings of the Eighth Annual Computer Science/Statistics Interface Symposium*, February 1975, pp. 262-266.
 3. Daniel, C. and F. S. Wood, *Fitting Equations to Data*, Wiley, 1971.
 4. Elshoff, J. L., "An Analysis of Some Commercial PL/I Programs," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 2, June 1976.
 5. Everitt, B., *Cluster Analysis*, Halstead, 1974.
 6. Gnanadesikan, R. and M. B. Wilk, "Probability Plotting Methods for the Analysis of Data," *Biometrika*, Vol. 55, 1968, pp. 1-17.
 7. Gnanadesikan, R., *Methods for Statistical Data Analysis of Multivariate Observations*, Wiley, 1977.
 8. Halstead, M. H., "Natural Laws Controlling Algorithm Structure?," *ACM SIGPLAN Notices*, Vol. 7, No. 2, February 1972, pp. 19-26.
 9. Halstead, M. H., *Elements of Software Science*, Elsevier, 1977.

10. Johnson, S. C., "Hierarchical Clustering Schemes," *Psychometrika*, Vol. 32, 1967, pp. 241-254.
11. Love, L. T. and A. B. Bowman, "An Independent Test of the Theory of Software Physics," *ACM SIGPLAN Notices*, Vol. 11, No. 11, November 1976.
12. McCall, J. A., P. K. Richards and G. F. Walters, "Factors in Software Quality," *Rome Air Development Center Technical Report*, July 1977.
13. McGibbon, T. L., et al., "Pattern Recognition Methods for Determining Software Quality," *Rome Air Development Center Technical Report*, October 1977.
14. Mosteller, F. and J. W. Tukey, *Data Analysis and Regression*, Addison-Wesley, 1977.
15. Motley, R. W., and W. D. Brooks, "Statistical Prediction of Programming Errors," *Rome Air Development Center Technical Report*, November 1976.
16. Sheppard, S. B., M. A. Borst and L. T. Love, "Predicting Software Comprehensibility," *General Electric Technical Report TR-77-388100-2*, February 1978.
17. Weinberg, G. M., *The Psychology of Computer Programming*, Van Nostrand Reinhold, 1971.

Program forms and program form analysers for high-level structured design*

by JAYASHREE RAMANATHAN

University of Houston
Houston, Texas

and

MEERA BLATTNER

Rice University
Houston, Texas

INTRODUCTION

Structured programming is the single most important technique currently used in developing software which is both reliable and inexpensive.^{11,27} It has been observed by various researchers that programmer productivity can be vastly improved if the development of software is split into two, equally important phases.

1. The design phase, and
2. The implementation phase.

In this paper we present PSEUDO LANGUAGE (PL)—a program design tool enforcing the functional programming concept discussed in References 4, 10, 13, 17, 21, 22 and 28. A PSEUDO LANGUAGE PROCESSOR (PLP) is also described. The PLP is a translator which examines source programs in PL and provides a listing of these programs together with a variety of messages. These messages can be used by the programmer both during the design and the implementation phase.

A PL program is a *program form* which represents a broad class of possible implementations in any of the standard programming languages. PL program forms resemble pidgin English and are therefore very readable. It is easy to program in PL since the programmer can ignore the messy details necessitated by actual implementation languages (FORTRAN, PASCAL, PL1, COBOL, etc.). An important characteristic of the PL design language is that it requires the programmer to explicitly identify the functional components of the programming task at hand. The implementation details of these functional components may be ignored by the programmer. Instead the programmer can concentrate on the logical interaction between these components. PL is designed to enforce structured programming techniques. A PL program can serve as the documentation for

the implementation version of the program. The advantages of PL mentioned above should also result in better communications between programmers in a team and contribute towards increased programmer productivity.

The PLP described here is constructed realizing that it is cheapest to correct errors during the earlier stages of software development. This is mainly due to the fact that fewer corrective changes have to be made to debug a design program. The messages generated by the PLP indicate errors in the PL program forms and list the functional components in the program form which have already been implemented. In order for the PLP to provide these messages, the PL program forms must have a recognizable structure. The definition of PL given here attempts to strike a balance between allowing program forms with too little syntactic structure and too much syntactic structure (as in implementation language programs).

Existing design tools for structured programming can be generally categorized as a variation of flowcharting (for examples refer to References 6, 8, 9 and 26) or as English sentences with keywords which identify the control flow in the program.⁷ Both types of design tools concentrate on the control logic of the programming task. PL programs also contain fixed control structures and in addition restrict the English sentences to "commands." This restriction on the English sentences forces the programmer to identify the functional components of the programming problem. At the same time, this restriction allows the PLP to employ algorithms for validating PL programs. PLP performs extensive static analysis on PL program forms and uses this analysis to print out messages that can be used by the programmer for validating and debugging the program forms.

The next section introduces the syntax for PL, and the third section discusses how PL enforces structured programming. PLP functions are outlined in the fourth section and the final section presents conclusions and suggestions for future work.

* This research was supported in part by NSF Grant Number MCS77-02470.

DEFINITION OF PSEUDO LANGUAGE (PL)

PL requires that the declarations precede the "executable" statements as in FORTRAN or PASCAL. Using somewhat informal syntax notation we have

- a. <program> → <introduction> <body>
- b. <introduction> → <directives to PLP> <title>
 INPUT PARAMETERS: <noun list>
 OUTPUT PARAMETERS: <noun list>
 READ INTO: <noun list>
 PRINT FROM: <noun list>
 DICTIONARY: <noun description list>
- c. <noun list> → <noun list>, <noun>
 / <noun>
- d. <noun description> → <noun>
 / <noun> ATTRIBUTES: <attribute list>
 / <noun> ATTRIBUTES: <attribute list>
 INITIAL VALUE: <constant list>
- e. <noun description list> → <noun description list>;
 <noun description>
 / <noun description>
 / <empty>
- f. <noun> → <identifier>
 / <subscripted identifier>
 / <structured noun>
 / <empty>
- g. <structured noun> → <structured noun> <constant> <identifier>
 / <constant> <identifier>
- h. <attribute list> → <attribute list>, <attribute>
 / <attribute>

The syntax of the introduction may be modified to allow declarations for general data types.^{20,21} We now give an example of an <introduction> for a Bubble Sort program. Comments on any line are preceded by a double slash (//).

Example

```
//introduction
SORT IN ASCENDING ORDER
INPUT PARAMETERS: SIZE_OF_TABLE, TABLE
OUTPUT PARAMETERS: TABLE
READ INTO:
PRINT FROM:
DICTIONARY:
SIZE_OF_TABLE ATTRIBUTE: INTEGER;
NO_INTERCHANGE ATTRIBUTE: FLAG
INITIAL: 1;
FIRST_ITEM ATTRIBUTE: POINTER; // TO
TABLE
SECOND_ITEM ATTRIBUTE: POINTER; // TO
TABLE
TABLE ATTRIBUTE: ARRAY;
EACH_PAIR
```

As illustrated in the previous example, <directives to PLP> can be empty. ATTRIBUTES: can be followed by any number of words describing the noun. The syntax for <constant list>, <identifier> and <constant> are as in standard programming languages. The syntax for the <body> of a PL program

follows. We shall think of the body as being composed of statement forms which specify the control flow in the implementation program and statement forms which specify the executable, functional components of the implementation program. We shall loosely say that the body of the program form is executable.

- i. <body> → BEGIN <statement list> END
- j. <statement list> → <statement list>; <statement list>
 / <statement list>
- k. <statement list> → <assignment list>
 / <compound statement list>
 / <if statement list>
 / <case statement list>
 / <while statement list>
 / <repeat statement list>
 / <for statement list>
 / <cycle statement list>
 / <with statement list>
 / <cobegin-coend statement list>
 / <I/O statement list>
 / <expression list>
 / <command list>

The fundamental executable components of a program

form are assignments, commands and expressions. The control structures specify the manner in which the fundamental components are to be executed. The non-terminal <statement> goes to the fundamental components and to the control structures containing sequences of fundamental components. Standard (PASCAL-like) syntax details of the control structures are given in the appendix. It should be noted that concurrent execution of statements may be specified by PL. Hence PL allows the design of program forms for a wide variety of applications. In fact, <statement> may be sent to any of the structures supported by the various languages currently in use. Syntax for exit statements, GOTOs and labels may be easily added to the above grammar but this is not done in the grammar presented here.

The fundamental executable component of interest is the command. The following productions give its syntax.

1. <command> <verb>
/ <verb> <comments> <noun>

```

<comments> <noun>
:
<comments> <noun>
/ <verb> <comments> <noun>
<comments> <noun>
:
<comments> <noun>
RETURN
<comments> <noun>
:
<comments> <noun>
    
```

<verb> is to be semantically interpreted as any English language verb. The non-terminal <comments> is to be interpreted as a sequence of English language words (usually prepositions and conjunctions) which add to the readability of the command. Finally, <noun> is to be interpreted as a data structure declared in the introduction of the program form. The body of the SORT program form introduced earlier is given in the following example.

Example PL program form for SORT

```

//<body>
BEGIN
    IF      SIZE_OF_TABLE>1
    THEN
        WHILE NO_INTERCHANGE
        DO     NO_INTERCHANGE=0
              FOR EACH_PAIR=1 TO SIZE_OF_TABLE-1
                //<command>
                DO  GET FIRST_ITEM IN TABLE;
                  //<command>
                  GET SECOND_ITEM IN TABLE;
                  IF FIRST_ITEM IN TABLE>SECOND_
                    ITEM IN TABLE
                THEN
                    BEGIN
                    //<command>
                    INTERCHANGE FIRST_ITEM IN TABLE AND
                    SECOND_ITEM IN TABLE;
                    NO_INTERCHANGE=1
                    END
                OD
            OD
        PRINT TABLE
END
    
```

Examine the first command GET FIRST_ITEM IN TABLE. This command begins with the verb GET and it operates on the nouns FIRST_ITEM and TABLE. The word IN functions as a comment which makes the command easier to read. Similarly, INTERCHANGE is the verb in the last command of the above program. INTERCHANGE operates on the nouns FIRST_ITEM, SECOND_ITEM and TABLE. More specifically,

```

//<verb>      <noun>      <comment>
INTERCHANGE  FIRST_ITEM  IN
              //<noun>    <comment>
              TABLE     AND
              //<noun>    <comment> <noun>
              SECOND_ITEM IN  TABLE
    
```

INTERCHANGE is the abstraction of a routine which has as inputs FIRST_ITEM, TABLE, and SECOND_ITEM.

The actual FORTRAN implementation for even the simple sort program is much harder to read. The FORTRAN subroutine is given below.

Example

```

SUBROUTINE SORT (ISIZE, TABLE)
DIMENSION TABLE (100)
IF (ISIZE.LE.1) GO TO 99
2  INTER=0
   DO 20 I=1, ISIZE-1
     IF (TABLE(I).LE.TABLE(I+1))GO TO 20
     INTER=1
     TEMP=TABLE (I)
     TABLE (I)=TABLE (I+1)
     TABLE (I)=TEMP
20  CONTINUE
   IF (INTER. EQ. 1) GO TO 2
99  WRITE (6, 30) (TABLE (I), I=1, ISIZE)
30  FORMAT (5×2, 10 (F8.3, 2×))
STOP
END

```

Note the PL program form for the above SORT given in the next example suppresses the details of TABLE, INTERCHANGE, GET and PRINT.

Example

```

SORT IN ASCENDING ORDER
INPUT PARAMETERS:  SIZE_OF_TABLE, TABLE
OUTPUT PARAMETERS: TABLE
PRINT FROM:      TABLE
DICTIONARY:      SIZE_OF_TABLE; NO_INTERCHANGE;
                  FIRST_ITEM; SECOND_ITEM; TABLE;
                  EACH_PAIR

BEGIN
  IF          SIZE_OF_TABLE>1
  THEN       WHILE      NO_INTERCHANGE
             DO         NO_INTERCHANGE=0
                   FOR EACH_PAIR=1 TO SIZE_OF_TABLE-1
                   DO  GET FIRST_ITEM IN TABLE;
                       GET SECOND_ITEM IN TABLE;
                       IF FIRST_ITEM IN TABLE>
                           SECOND_ITEM IN TABLE
                       THEN BEGIN
                           INTERCHANGE FIRST_ITEM IN
                           TABLE AND SECOND_ITEM IN TABLE;
                           NO_INTERCHANGE=1
                           END
                   OD
             OD
  PRINT TABLE
END

```

STRUCTURED PROGRAMMING USING PSEUDO LANGUAGE

Note that the PL SORT program form implies the hierarchical structure of Figure 1. The PL program form is not

concerned with the details of TABLE or its elements. In an implementation, the TABLE may be a file and its elements may be employee records. Also, GET and INTERCHANGE may be complicated, machine-dependent subroutines. Here again the PL SORT program form is not concerned about

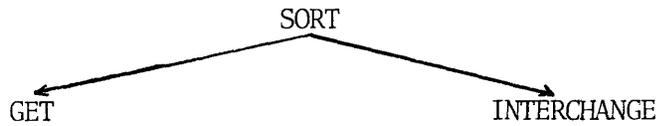


Figure 1

the details of algorithms implementing the functions GET and INTERCHANGE. SORT is written assuming GET and INTERCHANGE work. The logic of the SORT program form is easy to check out. Therefore, if GET and INTERCHANGE work reliably, then SORT works reliably. To reiterate, SORT TABLE might be the main goal. This goal is refined by the PL SORT program form which in turn calls for two functions, GET and INTERCHANGE. The data structures (or nouns) on which the two functions operate are completely specified in the commands in which they appear in the SORT program.

PL forces the programmer to identify the verbs (or functional components) of the program during the design phase. This ensures the program form can be implemented. Design languages which allow flexible English language statements cannot ensure that the design algorithm can in fact be implemented by a process. Another feature that PL has in common with the existing design tools is that control structures must be identified.

The functional components in the PL SORT program form may themselves be implemented by other PL program forms calling on still other functions (verbs). This stepwise refinement can continue until a level is reached where all the verbs are implemented in a desired implementation language. It is easy to see that the resulting implementation program should be well structured.

PSEUDO LANGUAGE PROCESSOR (PLP)

PL syntax is designed so that PL programs are amenable to both structural and symbolic (static) analysis. The PLP

currently being implemented is divided into three logical components—the scanner, the structure analyser and the message generator. PLP is designed to generate a variety of messages. The classes of messages generated are listed below together with a brief discussion of how the messages in each class are generated.

- *Messages detecting violations of standard design practices*—The structural analysis of the PL program, by the parser, detects omissions in the introduction and improper use of control structures in the body.^{1,27}
- *Cross-reference tables*—Tables indicating variable usage are determined during the three PLP phases and printed out appropriately.
- *Global references are listed*—All nouns in the introduction which have a global attribute are listed. This allows uses of critical, global resources to be monitored.
- *Messages indicating anomalies in the use of variables*—PLP uses various current techniques of high-level data flow analysis techniques on the parse tree to detect anomalies in the use of variables.^{2,5,12,14-16,18,19,24,25} The appearance of a variable in the introduction identifies it as a noun. This information is used by the semantic routines to distinguish the nouns in a command from other words which serve as comments. The first word in a command is always a verb. If a noun follows the keyword RETURN it is taken to be defined in the subprocess named by the verb. See Figure 2 for an illustration.
- *A list of possible sub-processes from the PLP library which may be used by the programmer*—An elementary component of the PL program is the command. Each command is a directive to the computer to operate on some nouns (data structures). The verb contained in the command must be implemented by a routine supplied either by the programmer or provided by the PLP library. Consider the PL statement

```

IF      FIRST_ITEM IN TABLE > SECOND_ITEM IN TABLE
THEN    <verb>      <noun>
        INTERCHANGE FIRST_ITEM
        <noun>      <noun>
        IN TABLE AND SECOND_ITEM
        <noun>
        IN TABLE
  
```

Suppose the verb INTERCHANGE is already implemented by a FORTRAN (or PASCAL) or even another PL routine in the PLP library. PLP will list INTERCHANGE as a possible implementation of the verb. The programmer can examine a copy of INTERCHANGE. After an examination, if the programmer so directs, the library implementation of INTERCHANGE can be used as shown below. PLP library implementation of INTERCHANGE may contain the statements

```

TEMP=FIRST_ARGUMENT
FIRST_ARGUMENT=SECOND_ARGUMENT
SECOND_ARGUMENT=TEMP
  
```

Based on the programmer's directive, INTERCHANGE in the PL statement above will be considered as a call to the library routine called INTERCHANGE and the resulting statement synthesized will be

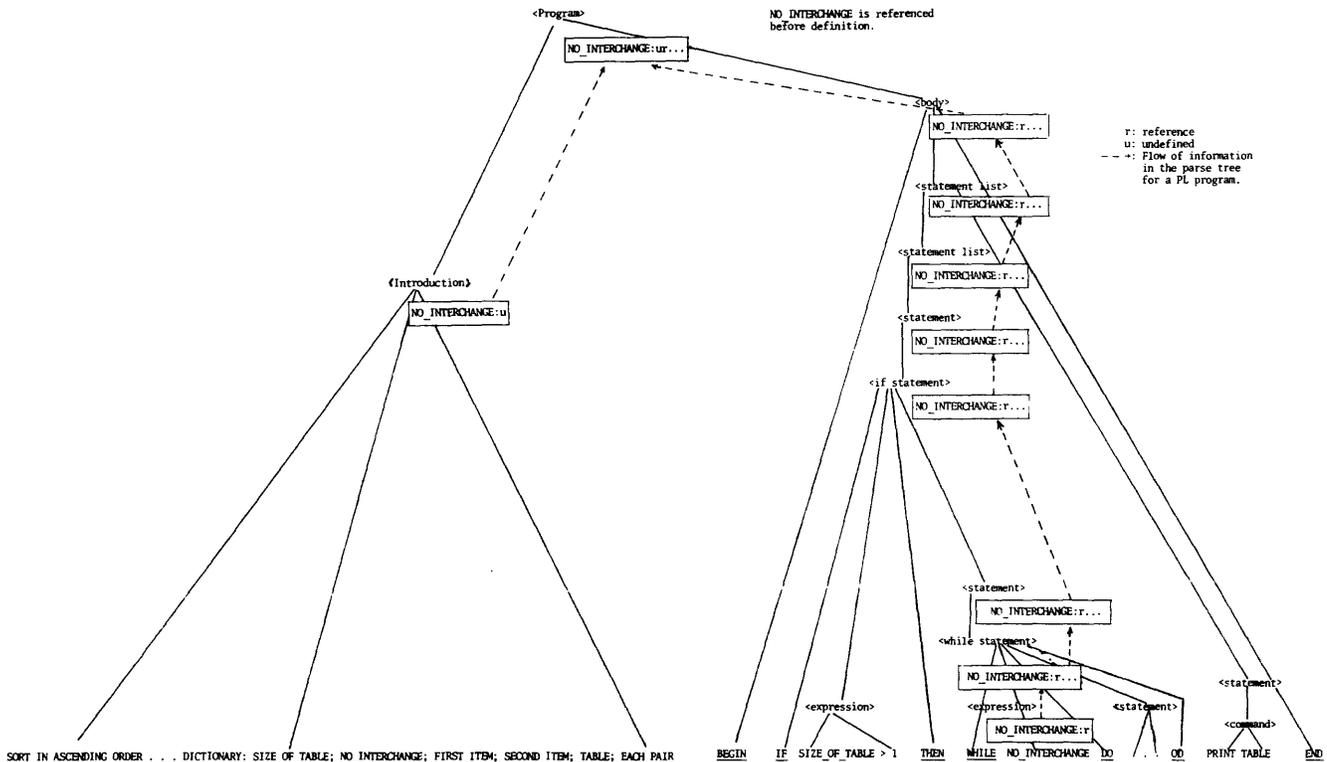


Figure 2

```

IF      FIRST_ITEM IN TABLE>SECOND_ITEM IN TABLE
THEN   CALL INTERCHANGE
        (TABLE(FIRST_ITEM),
         TABLE(SECOND_ITEM))
    
```

The PLP facility just discussed is especially useful when the verbs are implemented by complicated routines. In such cases considerable programming effort is saved and the program itself is more structured. A drawback of this PLP facility is that the programmer may use a verb which is implemented and contained in the PLP library under a different name. To overcome this, a document containing the implemented verbs may be provided to the programmers.

- *Messages on the misuse of a sub-process*—The introduction of each sub-process provides a detailed description of the input parameters and the output parameters of the sub-process. When PLP links together sub-processes or when it recommends (or uses) a particular implementation of a verb, it will check to see whether the actual parameters in the calling sub-process are compatible with the formal parameters in the definition of the sub-process. At the same time PLP will check access rights of each sub-process. These checks will be made semantically by examining the introduction of the two sub-processes involved.³

The last two PLP capabilities listed are currently being designed. The remaining capabilities are being implemented.

Finally, PLP uses the structural analysis of a PL program for creating an output listing with proper indentations.

CONCLUSIONS

A design language called PSEUDO LANGUAGE (PL) has been presented. Programs written in PL are called program forms. Program forms avoid implementation details and are therefore easily readable and understandable. PL also forces the programmer to identify the control structures as well as the functional components of the program system during the design phase.

The cost of finding an error in software increases as the software development comes nearer to completion. Errors found during specification are relatively inexpensive to correct as compared with errors found during total system integration.²³ The PSEUDO LANGUAGE PROCESSOR (PLP), currently being implemented, is an automatic tool for analysing specifications written in PL and printing out messages that indicate

- Violations of good design practices
- Errors

- Incorrect interfacing between programs
- Existing, potentially useful sub-processes that the programmer can use

Current research involves developing the theoretical framework for modeling the translation of PL program forms to implementation programs. Work is also proposed on the PL syntax. There may be many different implementations of a verb in the PLP library. Therefore, the syntax should allow a verb to be further qualified. For example, BUBBLE.SORT implementation would differ from MERGE.SORT. PLP can be designed as an interactive system which aids program form validation and implementation program synthesis. Valuable statistics on the usage of verb implementations can be obtained by the PLP. This would point to verbs that perhaps could be implemented by hardware.

REFERENCES

- Aho, A. V., and J. D. Ullman, *Principles of Compiler Design*, Addison-Wesley Publishing Co., 1977.
- Allen, F. E., and J. Cocke, "A Program Data Flow Analysis Procedure," *Communications of the ACM*, Vol. 19, No. 3, pp. 137-147, 1976.
- Barth, J. M., "An Interprocedural Data Flow Analysis Algorithm," *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages*, pp. 119-131.
- Blazer, R., N. Goldman and D. Wile, "Informality in Program Specifications," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 2, March 1978.
- Bochmann, G. V., "Attribute Grammars and Compilation: Program Evaluation in Several Phases," *Technical Report #54*, Department d'Informatique, Universite de Montreal, Montreal, Canada, 1974.
- Boyd, D. L., and A. Pizzarello, "Introduction to the WELLMADE Design Methodology," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 4, July 1978.
- Caine, S. H., and E. Gordon, "PDL . . . A Tool for Software Design," *Proceedings of the AFIPS 1975 National Computer Conference*, pp. 271-276.
- Chapin, N., "Semi-Code in Design and Maintenance," *Computers and People*, Vol. 27, No. 6, June 1978.
- Chapin, N., "New Format for Flowcharts," *Software Practice and Experience*, Vol. 4, No. 4, October-December 1974.
- Dijkstra, E., "A Constructive Approach to the Problem of Program Correctness," *BIT*, Vol. 8, No. 3, pp. 174-186, 1968.
- Dijkstra, E., *A Discipline of Programming*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.
- Fosdick, L. D., and J. L. Osterweil, "Data Flow Analysis in Software Reliability," *Computing Surveys*, Vol. 8, No. 3, pp. 305-330.
- Gannon, J. D., and J. J. Horning, "Language Design for Programming Reliability," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, June 1975.
- Hecht, M. S., *Flow Analysis of Computer Programs*, Elsevier, North Holland, 1977.
- Kennedy, K., and J. Ramanathan, "A Deterministic Attribute Grammar Evaluator Based on Dynamic Sequencing," *Communications of the ACM* (To Appear).
- Kennedy, K., and L. Zucconi, "Applications of a Graph Grammar for Program Control Flow Analysis," *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages*, Los Angeles, California, January 1977.
- Kernighan, B. W., and P. J. Plauger, *The Elements of Programming Style*, McGraw-Hill Book Co., New York, 1974.
- Knuth, D. E., "Semantics of Context-free Languages," *Mathematical Systems Theory*, Vol. 2, No. 2, pp. 127-145.
- Lancaster, R. L., and V. B. Schneider, "Quick Compiler Construction Using Uniform Code Generators," *Software-Practice and Experience*, Vol. 6, pp. 83-91, 1976.
- Liskov, H. B., "Abstraction Mechanisms in CLU," *Communications of the ACM*, August 1977.
- Liskov, H. B., and S. N. Zilles, "Specification Techniques for Data Abstraction," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 1, March 1975.
- Noonan, R. E., "Structured Programming and Formal Specification," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 4, December 1975.
- Ramamoorthy, C. V., "Testing Large Software with Automated Software Evaluation Systems," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 1, March 1978.
- Reifer, D. J., "Automated Tools for Reliable Software," *Proceedings of the 1975 International Conference on Reliable Software*, IEEE, 1975.
- Rosen, B. K., "Applications of High Level Control Flow," *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages*, Los Angeles, California, January 1977.
- Stay, J. F., "HIPO and Interactive Program Design," *IBM Systems Journal*, 1976.
- Wasserman, A. I., "Case Studies in Software Design," *IEEE Tutorial on Software Design Techniques*, IEEE Catalog No. 76, CH1145-2C, San Francisco, California, October 1976.
- Wirth, N., "Program Development by Stepwise Refinement," *Communications of the ACM*, Vol. 14, No. 4, April 1971.

APPENDIX

- | | |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| k.0 <statement> | → <assignment>
/ <compound statement>
/ <if statement>
/ <case statement>
/ <while statement>
/ <repeat statement>
/ <for statement>
/ <cycle statement>
/ <with statement>
/ <concurrent statement>
/ <command>
/ <I/O statement> |
| k.1 <compound statement> | → BEGIN <statement list>
END |
| k.2 <if statement> | → IF <expression>
THEN <statement>
ELSE <statement> |
| k.3 <case statement> | CASE <expression> OF
<constant> : <statement list> ;
:
END |
| k.4 <while statement> | → WHILE <expression> DO
<statement> OD |
| k.5 <repeat> | → REPEAT <statement list>
UNTIL <expression> |
| k.6 <for statement> | → FOR
<identifier> = <expression>
TO <expression>
DO <statement>
OD |
| k.7 <cycle statement> | → CYCLE <statement list>
END |

k.8 <with statement>	→ <i>WITH</i> <variable>, <variable>, . . . <i>DO</i> <statement>	<directives to PLP>	: list (possibly empty) of directives to the PLP processor
k.9 <command>	as in text	<attribute>	: any relevant attribute Example—real, integer, bit, array, etc.
k.10 <concurrent statement>	→ <i>COBEGIN</i> <statement list> <i>COEND</i>	<identifier>	: as in standard FORTRAN but no length limit
k.11 <I/O statement>	→ <i>PRINT</i> <noun list> <i>READ</i> <noun list>	<subscripted identifier>	: as in standard FORTRAN but no length limit

The terminals of PSEUDO LANGUAGE are the underlined symbols, the special characters (:, :, ,, =) and the digits. The nonterminals, which are not defined in the above syntax, are described below.

<title> : any title

<number> : as in standard FORTRAN
 <constant> : integer
 <expression> : as in FORTRAN
 <verb> : any verb in the English language
 <comment> : any group of English language words which are not verbs or nouns

First-year results from a research program on human factors in software engineering

by SYLVIA B. SHEPPARD, BILL CURTIS, PHIL MILLIMAN, M. A. BORST, and TOM LOVE

*General Electric Company
Arlington, Virginia*

INTRODUCTION

For the past two years the Software Management Research Unit at General Electric has been investigating several areas of human factors in software engineering with support from Engineering Psychology Programs of the Office of Naval Research. There have been two major thrusts in this research. The first thrust investigated the effects of several modern programming practices on programmer efficiency. The second thrust investigated the prediction of programmer performance from software complexity metrics such as those proposed by Halstead and McCabe. This research program consisted of separate experiments on the understanding, modification, debugging, and construction of software, each using professional programmers. Each experiment investigated both the effects of experimentally manipulated programming practices, and the values of complexity metrics computed from the programs employed.

Structured coding techniques, mnemonic variable names and commenting are programming practices which supposedly reduce the complexity of software. Dijkstra⁴ contended that program construction should proceed in a structured, top-down fashion. By limiting the control structures allowed, he assumed that the simplified control flow would make functions performed by the program easier to trace. Mnemonic variable names supposedly simplify the cognitive task of understanding a program by reducing the memory load on a programmer. The inclusion of comments purportedly simplifies modification tasks, although there are different methods of commenting. Global comments preceding a program summarize what objectives are accomplished, while in-line comments delineate how and where the objectives are fulfilled.

In 1972 Halstead first published his theory of software physics (renamed software science) stating that algorithms have measurable characteristics analogous to physical laws. These characteristics provide one assessment of program complexity. According to Halstead,^{13,14,16,18} the amount of mental effort required to generate a program can be calculated from simple counts of distinct operators and operands and the total frequencies of operators and operands. From these four quantities Halstead derives the number of mental comparisons required to generate a program. Correlations

often greater than .90⁸ have been reported between Halstead's metrics and such dependent measures as the number of bugs in a program,^{2,7,9} programming time,^{11,17} and the quality of programs.^{1,6,15}

More recently, McCabe²¹ developed a definition of complexity based on the decision structure of a program. McCabe's complexity metric is the classical graph-theory cyclomatic number which represents the number of regions in a graph, or in the current usage, the number of linearly independent control paths comprising a program. Simply stated, McCabe counts the number of elementary control path segments. When combined these segments generate every possible path through the program.

This paper reports results from the experiments on understanding and modification conducted during the first year of this research program. The first experiment investigated the effect of structured coding and mnemonic variable names on program comprehensibility. The second experiment studied the effects of structured coding and global versus in-line comments on modification tasks.

METHOD

Participants

In each experiment 36 programmers were tested in several General Electric locations. Participants in Experiment 1 had working knowledge of FORTRAN and averaged 6.8 years of professional programming experience ($SD = 5.8$). In Experiment 2 the participants had an average of 5.9 years of professional programming experience ($SD = 4.0$), a working knowledge of FORTRAN, and none had participated in the previous experiment. The majority of participants came from an engineering background.

Procedure

In both experiments a packet of materials was prepared for each participant with written instructions on the experimental tasks. As a preliminary exercise, all participants were presented the same short FORTRAN program and a

brief description of its purpose. In Experiment 1 they studied this program for 10 minutes and were then given 15 minutes to reconstruct a functionally equivalent program from memory. In Experiment 2 all participants were asked to modify the same short FORTRAN program. They were given a brief description of its purpose and were allowed unlimited time to complete a specified modification. This introductory program provided a common basis for comparing the skills of participants and diminished learning effects prior to the experimental tasks. This latter point is important since a pilot study²⁴ indicated that learning may occur during this type of task.

Following the initial exercise, participants were presented in turn with three separate programs comprising their experimental tasks. In Experiment 1 they were allowed 25 minutes to study each program, during which they were permitted to make notes or draw flowcharts. At the end of the study period, the original program and all scrap paper were collected. Each participant was then given 20 minutes to reconstruct a functional equivalent of the program from memory on a blank sheet of paper, but was not required to reproduce the comment section. In Experiment 2 one modification was requested for each of the three programs and was described on a sheet accompanying the program listing. Participants worked at their own pace, taking as much time as needed to implement the modification. A break of 15 minutes occurred before the last program was presented in each experiment.

Independent variables

Program class. Three general classes of programs were used in Experiment 1: engineering, statistical, and non-numerical. Three programs were employed from each class with lengths varying from 36 to 57 statements. These nine programs were selected from among many solicited from programmers at several locations and were considered representative of programs actually encountered by practicing programmers. All experimental programs were compiled and executed using appropriate test data. Experiment 2 used three of the nine programs from Experiment 1.

Complexity of control flow. Three control flow structures were defined for each program in both experiments. Structured control flow was generally consistent with the tenets of structured programming described by Dijkstra.⁴ When the rules for structured programming are applied rigorously, awkward constructions may occur in standard FORTRAN such as DO loops with dummy indices.²⁵ In a second version of each program, these awkward constructions were largely eliminated with a more naturally structured control flow. These conventions included multiple returns, exits from DO loops, and judiciously used backward GO TO's. In the unstructured version of each program, the control flow was not straightforward. Expanded DO loops, arithmetic IF's, and unrestricted use of GO TO's were allowed.

Variable name mnemonicity. In Experiment 1 three levels of mnemonicity for variable names were developed. The

programs were shown to several non-participants who were asked to assign names to the variables. The names chosen most frequently were used in the most mnemonic condition. The medium mnemonic level consisted of less frequently chosen names. In the least mnemonic condition, names consisted of one or two randomly chosen alphanumeric characters.

Comments. Three levels of commenting were tested in Experiment 2: global, in-line, and none. Global comments provided an overview of the function of the program and identified the primary variables. In-line comments were interspersed throughout the program and described the specific functions of small sections of code.

Modifications. Three types of modifications were selected for each program in Experiment 2 as typical changes a programmer might be expected to implement. The level of difficulty for seven of the nine modifications increased as more lines had to be added to the original code, and the hardest modifications for each program required the most additional lines.

Experimental design. In order to control for individual differences in performance, a within-subjects 3⁴ factorial design was employed in each experiment.¹² In Experiment 1 three types of control flow were defined for each of nine programs, and each of these 27 versions was presented in three levels of variable mnemonicity, for a total of 81 programs. In Experiment 2 three levels of control flow were defined for each of the three programs. Each of these nine versions was presented with one of three levels of documentation. Modifications at three levels of difficulty were developed for each program, generating a total of 81 experimental conditions. The first 27 participants in each experiment exhausted the total of 81 programs. The additional nine participants repeated 27 of the previous experimental tasks. Programmers at each location were randomly assigned to experimental conditions in order that they would experience each level of each independent variable. That is, within their three tasks they worked with a program from each class, with each type of control flow, and at each level of documentation (variable mnemonicity or type of commenting). Each of the first 27 participants experienced unique combinations of these levels across their three experimental tasks. The order of presentation of the three programs was assigned randomly to each participant.

Covariates. In order to obtain a measure of programming ability related to the experimental tasks, scores on the preliminary tasks in both experiments were used as a covariate. Participants reported their type of programming experience and the number of years they had been programming professionally. Order of presentation was a situational covariate.

Complexity measures

Halstead's E. Halstead's effort metric (*E*) was computed precisely from a program (based on Reference 22) whose input was the source code listings of the 27 distinct programs in each experiment. Programs differing only in variable mne-

monicity or type of commenting were not considered distinct programs in this analysis. The computation formula was:

$$E = \frac{\eta_1 N_2 (N_1 + N_2) \log_2 (\eta_1 + \eta_2)}{2 \eta_2}$$

where

- η_1 = number of unique operators
- η_2 = number of unique operands
- N_1 = total frequency of operators
- N_2 = total frequency of operands

McCabe's $\nu(G)$. McCabe's metric is the classical graph-theory cyclomatic number defined as:

$$\nu(G) = \# \text{ edges} - \# \text{ nodes} + 2 \text{ (} \# \text{ connected components)}$$

McCabe presents two simpler methods of calculating $\nu(G)$. His metric equals the number of predicate nodes plus 1. Values of $\nu(G)$ can also be computed from a planar graph of the control flow by counting the number of regions.

Length. The length of the program was computed as the total number of FORTRAN statements excluding comments.

Dependent variables

Experiment 1. The criterion for scoring the programs in Experiment 1 was the functional correctness of each separately reconstructed statement. Variable names and statement numbers which differed from those in the original program were counted as correct when used consistently. Control structures could be different from the original program so long as the statements performed the same function. The score on each experimental task was the percent of statements correctly recalled. Three judges scored each program independently. Interjudge correlations of .96, .96, and .94 were obtained across the three sets of scores. The average of the three scores (percents of statements correctly reconstructed) for each program was the dependent variable in the data analysis for Experiment 1.

Experiment 2. The dependent variables for Experiment 2 were the correctness of the modification and the time taken by the participant to perform the task. The individual steps necessary for correct implementation of the requested modifications had been delineated in advance and assigned equal weights. That is, prototypes of each program with each modification correctly implemented were established as the criteria against which participants' work would be compared. A percentage score reflecting the correctness of each modification was computed by comparing participants' changes with the criteria. The time to write a modification was measured to the nearest minute by an electronic timer.

Analysis

Results were analyzed in two phases. The first phase investigated the effects of experimentally manipulated var-

iables, while the second phase evaluated the performance predictions of the software complexity metrics. The experimental effects of programming practices were analyzed in hierarchical regression analyses. In these analyses domains of variables were entered sequentially into a multiple regression equation to determine if each successive domain added significant prediction to that afforded by domains already entered. Effects related to pre-existing differences among participants and programs were entered into analyses prior to evaluating the effects of programming practices. The variables representing the different conditions of experimentally manipulated variables were effect coded.¹⁹

Analyses investigating relationships among Halstead's E , McCabe's $\nu(G)$, number of statements, and performance were conducted with Pearson product-moment correlation coefficients.

RESULTS

Experimental manipulations

Experiment 1. Table I presents the results of the hierarchical regression analyses for Experiment 1. Figures presented in this and succeeding tables indicate the unique percent of variance contributed to the prediction of performance by a variable domain when added into the analysis with preceding domains. Significance levels identified by asterisks indicate the likelihood (expressed as a proportion) that a prediction of this significance could have occurred by chance.

An average of 50 percent of the statements were correctly recalled across all programs and experimental conditions. Pretest scores accounted for 17 percent of the variance among scores on the percent of statements correctly recalled. No relationships were observed for type and length of programming experience or job location.

Differences among the program classes accounted for 8 percent of the variance in performance scores in addition to that accounted for by individual differences among participants. Engineering programs were the most difficult (41 percent of the statements correctly recalled), followed by statistical (52 percent), and non-numeric (57 percent) programs. When the specific program was taken into account, an additional 20 percent of the variance in performance was explained. However, this result was not strictly a function of differences among programs, because variance related to

TABLE I.—Hierarchical Regression for Percent of Statements Correctly Recalled

Variable domain	ΔR^2
Pretest	.17**
Class of program	.08**
Specific program	.20**
Control flow complexity	.07**
Variable mnemonicity	.01
Total R^2	.53***

Note: $n=108$. ** $p \leq .01$. *** $p \leq .001$.

specific programs was confounded with variance related to participants. That is, each participant saw only three of the nine programs. Overall, 45 percent of the variance in performance was accounted for by differences among participants and programs.

The complexity of the control flow affected performance, accounting for 7% of the variance in addition to that accounted for by differences among programs and participants. As expected, unstructured programs were the most difficult to reconstruct. A post hoc analysis²³ showed the means for naturally structured and unstructured programs (56 percent versus 42 percent, respectively) to be significantly different ($p \leq .05$). Performance on structured programs fell between these two. No differences occurred among levels of variable mnemonicity.

Experiment 2. Across all experimental conditions, an average of 62 percent of the steps for each modification were accurately implemented. The 108 accuracy scores ranged from five scores of 0 percent to 24 scores of 100 percent and were negatively skewed. The average time to complete the modifications was 17.9 minutes, ranging from 2 to 59 minutes with a positive skew. Accuracy and time were uncorrelated.

Table II presents hierarchical regression results for the accuracy of the participants' modifications. Only 19 percent of the variance in accuracy scores could be predicted by the variable domains studied. However, there were substantial differences in the degree to which performance on each of the three programs could be predicted. On two of the programs, 35 percent of the variance in accuracy scores was accounted for, while results for the third program were insignificant.

Order of presentation accounted for 5 percent of the variance in accuracy scores. Participants made more complete modifications in less time with each succeeding experimental task. However, the two programs on which performance proved most predictable were more frequently presented second or third in order. Thus, random assignment of presentation orders failed to counter-balance the number of times each condition appeared in each position order.

The difficulty of the modification accounted for 9 percent of the variance in accuracy scores on the two most predictable programs. Performance was poorer on modifications which required more lines of code to be inserted. The complexity of the control flow accounted for 7 percent of the variance in accuracy scores on the two programs for which

TABLE II.—Hierarchical Regression for Accuracy of Modifications

Variable domain	ΔR^2	ΔR^2
	(3 programs)	(2 programs)
Pretest accuracy	.05*	.05
Presentation order	.05*	.13**
Program	.02	.01
Modification difficulty	.02	.09**
Control flow complexity	.04	.07**
Type of commenting	.01	.00
Total R^2	.19	.35***

Note: $n=108$ for 3 programs and $n=72$ for 2 programs. * $p < .05$. ** $p \leq .01$. *** $p \leq .001$.

TABLE III.—Hierarchical Regression for Time to Completion

Variable domain	ΔR^2
Pretest time	.03
Presentation order	.06**
Program	.01
Modification difficulty	.15**
Control flow complexity	.02
Type of commenting	.01
Total R^2	.28***

Note: $n=108$. ** $p \leq .01$. *** $p \leq .001$.

accuracy was most predictable. Modifications made to structured programs were more accurate than those made to unstructured programs. Accuracy scores did not differ among programs, nor among the type of comments included in the program.

Table III presents hierarchical regressions for time to completion. Across all three programs, 28 percent of the variance in the time required to complete the modifications could be accounted for by variables studied here. Time to complete the modifications was more easily predicted than accuracy scores across all three programs.

Results of the hierarchical regression for time were generally similar to the results observed for accuracy. The specific program and type of comments were unrelated to the criterion. Significant variance was accounted for by both the difficulty of the modification and the order of presentation. Again, however, the interpretation of the effect for this latter variable is confounded. Neither the pretest scores nor control flow complexity were significantly related to time, although they had been modestly related to accuracy.

Further inspection verified that the number of additional statements required in the code to accurately complete a modification was related to the time required to insert them. Fitting a curvilinear function to these data using least squares procedures resulted in a curvilinear correlation (second order polynomial) of .80 ($p \leq .05$) and a standard error or estimate of 2.53 minutes. No such relationship was found for accuracy.

Software complexity measures

Since different levels of variable mnemonicity and type of commenting neither affected performance, nor caused any change in the value of the complexity metrics for a particular program, the data reported in this section were aggregated over the three levels of mnemonicity in Experiment 1 and type of commenting in Experiment 2. This procedure resulted in 27 data points for each experiment. Each datum represented the average of at least three performance scores. Table IV presents the correlations among the three complexity measures in both experiments. Correlations in the lower triangle are from Experiment 1; those in the upper triangle are from Experiment 2. Generally these correlations were quite large in both experiments.

Table V presents correlations between the complexity

TABLE IV.—Intercorrelations among Complexity Measures

Complexity Measure	<i>E</i>	$v(G)$	Length
Halstead's <i>E</i>		.88***	.92***
McCabe's $v(G)$.84***		.89***
Length	.47**	.64***	

Note: $n=27$. ** $p \leq .01$. *** $p \leq .001$.

measures and performance criteria in both Experiments 1 and 2. In Experiment 1 the correlations between performance and each of the complexity measures were all negative, indicating that fewer lines were recalled as the level of complexity represented by these three metrics increased. Performance was moderately related to length and McCabe's $v(G)$, but not to Halstead's *E*.

Most of the significant correlations with performance in Experiment 2 were observed for measures computed on correctly modified rather than unmodified programs. Correlations reported in Table V were for measures computed on modified programs. All three measures were moderately correlated with time to complete the modification, while only length and McCabe's $v(G)$ were significantly related to accuracy.

The complexity of the control flow moderated the relationships between performance and the complexity metrics in both experiments. That is, while insignificant correlations were observed when the control flow was structured or naturally structured, this was not the case for unstructured code. Correlations with percent recalled correctly in Experiment 1 of $-.55$ ($p \leq .001$) and $-.45$ ($p \leq .01$) for $v(G)$ and *E* were observed on unstructured programs. In Experiment 2 correlations relating time with Halstead's *E* went from .08 in the structured code, to .28 ($p \leq .05$) in naturally structured code, to .38 ($p \leq .05$) in unstructured code. No such moderating effects were observed for McCabe's $v(G)$, nor for either metric with accuracy scores.

Correlations between the complexity metrics and performance criteria in Experiment 2 were also moderated by the type of commenting. When no comments were included in the program, significant correlations on modified programs for both Halstead's *E* and McCabe's $v(G)$ were observed for both accuracy ($r = -.34$ and $-.35$, $p \leq .05$) and time ($r = .47$ and $.44$, $p \leq .01$). Insignificant correlations were usually observed when either global or in-line comments appeared in the code.

The amount of professional programming experience profoundly moderated the relationships observed between the complexity measures and percent of statements correctly

recalled in Experiment 1 and time to completion in Experiment 2. For programmers with three or less years of professional experience in Experiment 1, correlations of $-.47$ ($p \leq .001$) for McCabe's $v(G)$ and $-.35$ ($p \leq .05$) for Halstead's *E* were observed. Insignificant correlations were observed for programmers with more than three years experience. For time to completion in Experiment 2, correlations of .55 ($p \leq .001$) for Halstead's *E* and .52 ($p \leq .001$) for McCabe's $v(G)$ were observed for programmers with three or less years of professional experience, while no correlations above .20 were observed for programmers with more than three years experience.

DISCUSSION

Experimental manipulations

Several factors were consistently related to programmer performance. Individual differences among participants and the complexity of the control flow were found to influence programmer performance in both experiments. In Experiment 2 the difficulty of the requested modification and the order of presentation influenced both the accuracy and speed of implementing modifications. Each of these factors contributed independently to predicting program comprehension. Contrary to expectations, however, mnemonic variable names and types of commenting did not influence performance.

Control flow complexity was significantly related to both the percent of statements correctly recalled in Experiment 1 and the accuracy of the modifications on two of the programs studied in Experiment 2, but not to the time spent implementing modifications. In Experiment 1 naturally structured code was more easily comprehended than unstructured code. In Experiment 2 more accurate modifications were made to structured rather than unstructured code. It is not clear from the results of these two experiments whether rigidly structured code or code structured with a more natural control flow for FORTRAN can be maintained more efficiently. However, both of these control flows proved superior to unstructured code in at least one of the experiments.

Differences among programs played an important, but difficult-to-explain, role in these experiments. Effects on performance attributed to these differences may have resulted from some familiarity factor specific to the samples of programs and programmers studied. Further, effects due to differences among specific programs were confounded in Experiment 1 with effects related to individual differences among participants.

It is not surprising that the difficulty of a modification in Experiment 2 was related to the time required to implement it. The significant factor in the time spent implementing a modification was the number of new lines to be added rather than the number of in-line changes, such as deletions or substitutions. The difficulty of a modification also affected the accuracy with which it was implemented. Greater cog-

TABLE V.—Correlations between Complexity and Performance Measures

Complexity metric	Percent Recalled (Exp. 1)	Accuracy (Exp. 2)	Time (Exp. 2)
Halstead's <i>E</i>	-.13	-.29	.44**
McCabe's $v(G)$	-.35*	-.36*	.38*
Length	-.53**	-.34*	.46**

Note: $n=27$. * $p \leq .05$. ** $p \leq .01$.

nitive difficulty appeared to be involved in creating new code than in merely deleting or altering it.

The inclusion of mnemonic variable names and either global or in-line comments were expected to improve programmer performance. The surprising lack of effects for documentation aids in both experiments may have occurred for several reasons. First, in Experiment 1 variable mnemonicity was manipulated and global comments were provided with all programs. In Experiment 2 type of commenting was manipulated and mnemonic variable names were provided in all programs. Thus, the existence of one type of documentation may have reduced the additional information available from the documentation aid being experimentally manipulated, reducing its impact on performance.

A second possibility is that documentation aids do not contribute significantly to performance for programs of the modular size (35-55 lines) employed here. In large systems with many modules and thousands of lines of code, documentation may have more impact on performance because of the increased amount of information programmers must remember. Thus, program size may moderate the relationship between documentation and performance.

Finally, although mnemonic variable names did not affect performance in Experiment 1, many participants seemed to prefer them. That is, they used their own, more meaningful names when reconstructing the least mnemonic versions of the programs. For the medium and most mnemonic versions, they tended to use the original names supplied. Thus, the contribution of mnemonic variable names is supported by anecdotal rather than statistical evidence.

Results for the modern programming practices studied here were probably conservative due to the small size of programs studied. The cognitive load placed on programmers attempting to understand or modify approximately 50-line programs did not require the amount of assistance provided cumulatively by structured coding, mnemonic variable names, and comments. While the information provided by these practices was not necessarily redundant, the task could be mastered with less information than presented. In a larger system composed of many modules, however, the cognitive burden of implementing modifications may be so great that each of these programming practices may contribute significantly to efficiency. Thus, future research needs to assess the independent benefits of these practices in substantially larger programs.

Software complexity metrics

The two experiments comprising this study produced empirical evidence that software complexity metrics were related to the difficulty programmers experienced in understanding and modifying programs. Deeper analysis indicated, however, that the Halstead and McCabe metrics predicted programmer performance only on certain programs. Programs on which significant prediction was observed were characterized by the absence of programming practices such as structured coding or commenting which provided assistance in understanding the code. These com-

plexity metrics were more predictive of the performance of less experienced programmers. A more complete presentation and discussion of these results is presented by Curtis, Sheppard, Milliman, Borst, and Love.³

Assessment of the psychological complexity of software appears to require more than a simple count of operators and operands or basic control paths. Many programs have characteristics unassessed by these metrics which may heavily influence psychological complexity. For instance, the use of structured coding techniques or comments may reduce the cognitive load on a programmer in ways unassessed by the complexity metrics. Further, complexity metrics may not be capturing the most important factors for predicting the performance of experienced programmers who may either be conceptualizing programs at a level other than that of operators, operands, and basic control paths, or who can fit the program into a schema similar to one with which they have had previous experience.

Even though moderating effects were observed in these data, stronger relationships with performance may have been masked by the effects of differences between individuals and programs which were enhanced by limitations in the economical multifactor designs employed. Uniformity in the sizes of programs studied may also have limited these results. The range of values assumed by complexity metrics computed on these programs may have been insufficient for correlational tests⁵ to detect the strong relationships reported in other verifications of these theories. Studies reporting higher correlations for Halstead's *E* usually involved a broader range of program sizes.^{8,18}

Further work in the area of software complexity should identify a set of cognitive principles relevant to programming tasks. Metrics could then be developed which would assess the qualities of software which are most closely related to these principles. Such an exercise might not only lead to improved metrics for assessing software complexity, but might also identify programming practices which could lead to more easily maintained software.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the assistance of Ann Fitzsimmons in implementing this research, of Dr. Gerald Hahn in developing the experimental design, and of Beverly Day in manuscript preparation. Careful reviews of this report by Dr. John O'Hare, Dr. Maurice Halstead, and Thomas McCabe have resulted in substantial improvements. The support and encouragement of Gerald Dwyer has also been greatly appreciated.

This study was supported by the Office of Naval Research, Engineering Psychology Programs (Contract #N00014-77-C-0158). The views expressed in this paper, however, are not necessarily those of the Office of Naval Research or the Department of Defense.

Expanded reports of each experiment can be obtained by writing the senior author at: General Electric, Suite 200; 1755 Jefferson Davis Hwy.; Arlington, VA. 22202. Portions of this paper were drawn from articles to be published by

the Human Factors Society and the Institute of Electrical and Electronics Engineers.

REFERENCES

1. Bulut, N., and M. H. Halstead, "Impurities found in algorithm implementations," *SIGPLAN Notices*, No. 3, 1974, pp. 9-10.
2. Cornell, L., and M. H. Halstead, *Predicting the number of bugs expected in a program module* (Tech. Rep. CSD-TR-205), West Lafayette, IN, Purdue University, Computer Science Department, October 1976.
3. Curtis, B., S. B. Sheppard, P. Milliman, M. A. Borst, and T. Love, "Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics," *IEEE Transactions on Software Engineering*, Vol. 5, 1979, in press.
4. Dijkstra, E. W., "Notes on structured programming," in O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare (Eds.), *Structured Programming*, New York, Academic Press, 1972.
5. Edwards, A. L., *An Introduction to Linear Regression and Correlation*, San Francisco, Freeman, 1976.
6. Elshoff, J. L., "Measuring commercial PL/I programs using Halstead's criteria," *SIGPLAN Notices*, 1976, Vol. 11, No. 5, pp. 38-46.
7. Fitzsimmons, A. B., "Relating the presence of software errors to the theory of software science," in A. I. Wasserman and R. H. Sprague, Jr. (Eds.), *Proceedings of the Eleventh Hawaii International Conference on System Science*, Vol. 1, San Francisco, Western Periodicals, 1978.
8. Fitzsimmons, A. B., and T. Love, "A review and evaluation of software science," *ACM Computing Surveys*, Vol. 10, 1978, pp. 3-18.
9. Funami, Y., and M. H. Halstead, "A software physics analysis of Akiyama's debugging data," in *Proceedings of the MRI XXIV International Symposium: Software Engineering*, New York, Polytechnic Press, 1976.
10. Gordon, R. D., *A measure of mental effort related to program clarity*, Unpublished doctoral dissertation, Purdue University, 1977.
11. Gordon, R. D., and M. H. Halstead, "An experiment comparing FORTRAN programming time with the software physics hypothesis," in *Proceedings of the AFIPS 1976 National Computer Conference*, Montvale, NJ, AFIPS Press, 1976.
12. Hahn, G. J., and S. S. Shapiro, *A catalogue and computer program for the design and analysis of orthogonal symmetric and asymmetric fractional factorial experiments* (Tech. Rep. 66-C-165), Schenectady, NY, General Electric, May 1966.
13. Halstead, M. H., "Natural laws controlling algorithm structure," *SIGPLAN Notices*, Vol. 7, No. 2, 1972.
14. Halstead, M. H., *A theoretical relationship between mental work and machine language programming* (Tech. Rep. CSD-TR-67), West Lafayette, IN, Purdue University Computer Science Department, May 1972.
15. Halstead, M. H., "An experimental determination of the purity of a trivial algorithm," *SIGME Performance Evaluation Review*, Vol. 2, 1973, pp. 10-15.
16. Halstead, M. H., *Software physics: Basic principles* (Tech. Rep. RJ-1582), Yorktown Heights, NY, IBM, 1975.
17. Halstead, M. H., *Using the methodology of natural science to understand software* (Tech. Rep. CSD-TR-190), West Lafayette, IN, Purdue University, Computer Science Department, 1976.
18. Halstead, M. H., *Elements of Software Science*, New York, Elsevier North-Holland, 1977.
19. Kerlinger, F. N., and E. J. Pedhazur, *Multiple Regression in Behavioral Research*, New York, Holt, Rinehart, & Winston, 1973.
20. Love, L. T., *Relating individual differences in computer programming performance to human information processing abilities*, Unpublished doctoral dissertation, University of Washington, 1977.
21. McCabe, T. J., "A complexity measure," *IEEE Transactions on Software Engineering*, Vol. 2, 1976, pp. 308-320.
22. Ottenstein, K. J., *A program to count operators and operands for ANSIFORTRAN modules* (Tech. Rep. CSD-TR-196), West Lafayette, IN, Purdue University, Computer Science Department, June 1976.
23. Scheffé, H. A., *The Analysis of Variance*, New York, Wiley, 1959.
24. Sheppard, S. B., and L. T. Love, "A preliminary experiment to test influences on human understanding of software," in *Proceedings of the 21st Meeting of the Human Factors Society*, Santa Monica, CA, Human Factors Society, 1977.
25. Tenny, T., "Structured programming in FORTRAN," *Datamation*, Vol. 20, 1974, pp. 110-115.

The use and abuse of a software engineering system

by D. J. PEARSON

Bell-Northern Research
Ottawa, Ontario, Canada

In 1969, International Computers Limited of England set about the design of its 2900 Series which was to unify the primary thrust of the company and was to provide a hardware and software architecture which was, at least, state-of-the-art.¹ The systems also had to sell. Therefore, they had to satisfy the then market requirement for rich facilities and generally neat features. If ICL was seriously to compete with IBM (in Europe at least) the operating system, subsequently called System VME/B, would have to be comparable with the IBM products. ICL could not afford the staggering investments made in the 360 software systems. On the other hand, its recent track record had been less than outstanding for such software development. Faced with this dilemma, a project was set up to develop a system capable of minimizing the problems of software development by harnessing many of the current software engineering philosophies in order to aid management, reduce error rate and increase productivity. This system was subsequently called CADES (Computer Aided Development and Evaluation Systems). This paper describes briefly the major facets of this system and then goes on to discuss six years of product development experience with a software engineering system which was, by the standards of the day, state-of-the-art.

THE SOFTWARE ENGINEERING SYSTEM

The original CADES system was based on System 4 and was implemented between 1970 and 1972. All the original VME/B software was itself implemented on the System 4 and written in a high-level language based on Algol 68 called S3. It was not until 1973/74 that a real transition to 2900 architecture took place as far as the product development activities was concerned. CADES itself was redesigned and reimplemented on 2900 during the period 1974–76. CADES consists of the following elements:

1. Formal Design Methodology
2. Design Definition Language
3. Product Data Base
4. Formal Data Capture and Control
5. Product Data Base Applications

Formal design methodology

A methodology called Structural Modeling was defined and adopted. It was based on a formalized top-down, levels of abstraction approach with great emphasis being placed on the data-driven emergence of design, and attempting to quantify the iterative nature of design. Instead of the "design until you understand the problem, code until you realize you don't, then iterate" approach to design, the entire process was quantified and documented, with all the possible iteration paths identified and priorities assigned to them.

Using Structural Modeling, the designer was constrained first of all to analyse the problem in terms of *information flow* and to structure this information analysis into a tree-form, each level on the tree defining an *abstract machine*. One of the lower levels of abstract machine would map onto the S3 compiler data structures. This was the *implementation level*. After this information analysis, a function tree was constructed, compatible with the information tree, each level on this function tree representing the functional definition of that abstract machine. This was called the *holon tree* (holon, from Koestler, Reference 3; in this context a holon is defined as a unit of further design).

A lower level of the holon tree would map onto the concept of an S3 module. This was the *implementation level*. When both trees were compatible, that is, complementary from an abstract machine, levels-of-design point of view, the top-down design process was commenced, expressing the functional design of each holon, level-by-level, in terms of the information items at the corresponding level in the data tree and its interactions in terms of its peer-holons. These designs were defined in terms of the design definition language. Techniques were built into the modelling process to preserve the data and functional modularity, and to minimize the functional and data connectivity across the system.

Design definition language

The emerging design was expressed in a formal System Descriptive Language, SDL. The structure of SDL was based on that of the implementation language S3. However,

it was provided with facilities to express design concepts such as an 'event' and a 'virtual machine,' and it contained no facilities for implementation concepts such as data structures and procedure declarations. An SDL definition dealt exclusively with the items defining the abstract machine under definition, and the conditions and assertions governing its function in terms of the recognized design concepts. When the designer, during his top-down design activities, reached the implementation level and started to define that in SDL, an automatic code generator was initiated and S3 code automatically generated from an SDL definition.

The advantages of using SDL were fourfold:

1. It ensured a formal, complete definition of the entire design.
2. Design definitions were machineable and could be captured in a data base.
3. Automatic code generation increased productivity.
4. Automatic code generation dramatically reduced finger-trouble error.

Product data base

One of the primary intentions of the CADES system was the construction of a product data base which would contain the entire set of information defining the VME/B product, from earliest design statement, S3 code and loadable versions, product releases, bug reports and fixes. One major reason for this was in order to be able to relate problems discovered in the field to the appropriate earlier design decision and hence define accurately the scope of such problems, rather than adopting the fire-fighting, piecemeal approach to product maintenance. The greater the contents of

the product data base, the more valuable it was in terms of information inversions. It reached maximum value at product release, and subsequently proved to be invaluable throughout all product maintenance activities. The data base itself is able to contain all *types* of VME/B product definition, as shown in Figure 1.

Updates were made to the data base using the formal data capture mechanism and expressed in SDL. Similarly, retrieval requests were expressed in SDL. The code generator, compiler, construction, loading and maintenance tools interacted directly with the data base. This interaction enabled the mapping between 'types' of operating system representation to be carried out consistently. Note that the mapping between high- and low-level design representations is carried out by the designer. When a change was made to the data base, entries were recorded in queues for action by the appropriate tools. The tools themselves return information such as sizes and indications of success to the data base. This return information itself might then cause further entries to be made on the queues for appropriate processing. In this way a degree of automation was achieved, human interventions, and hence error rate, decreased.

Formal data capture and control

As will be explained in the next section, the size of the project team was large. Data base information integrity in such an environment is very important, and yet very difficult, to achieve. In order to achieve the necessary level of integrity we were forced to implement a semi-batch interface for data capture, with a rigorous control scheme to validate every piece of information to be inserted into the data base. This level of control was achieved by associating an au-

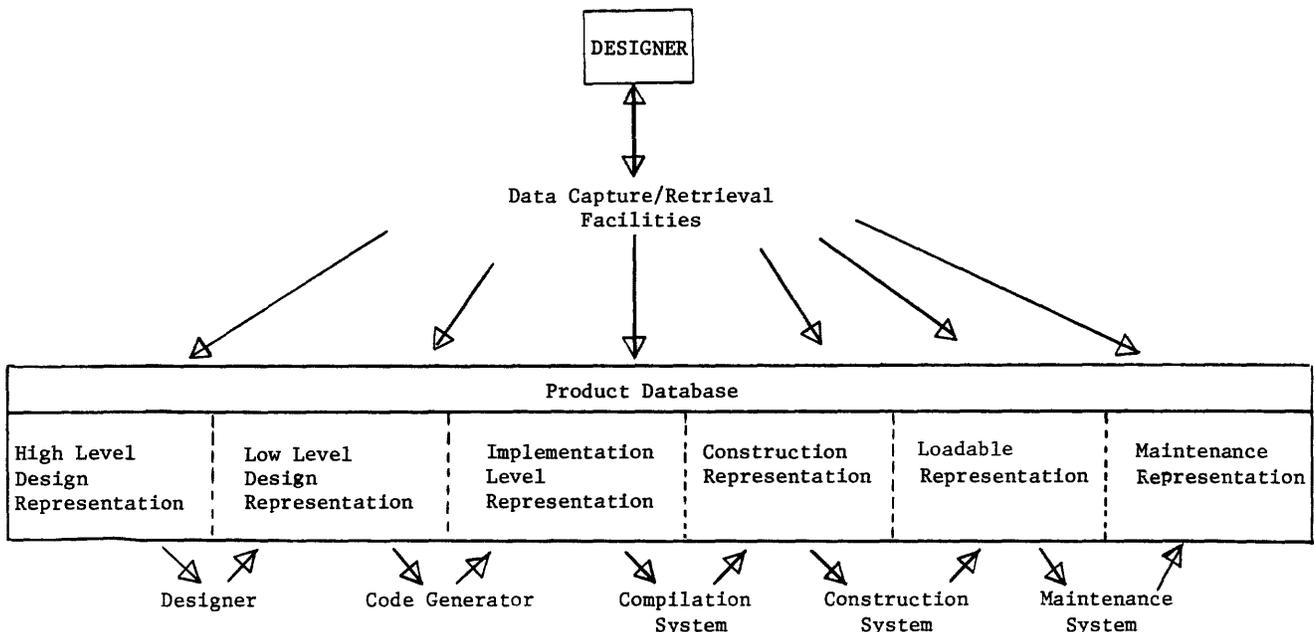


Figure 1

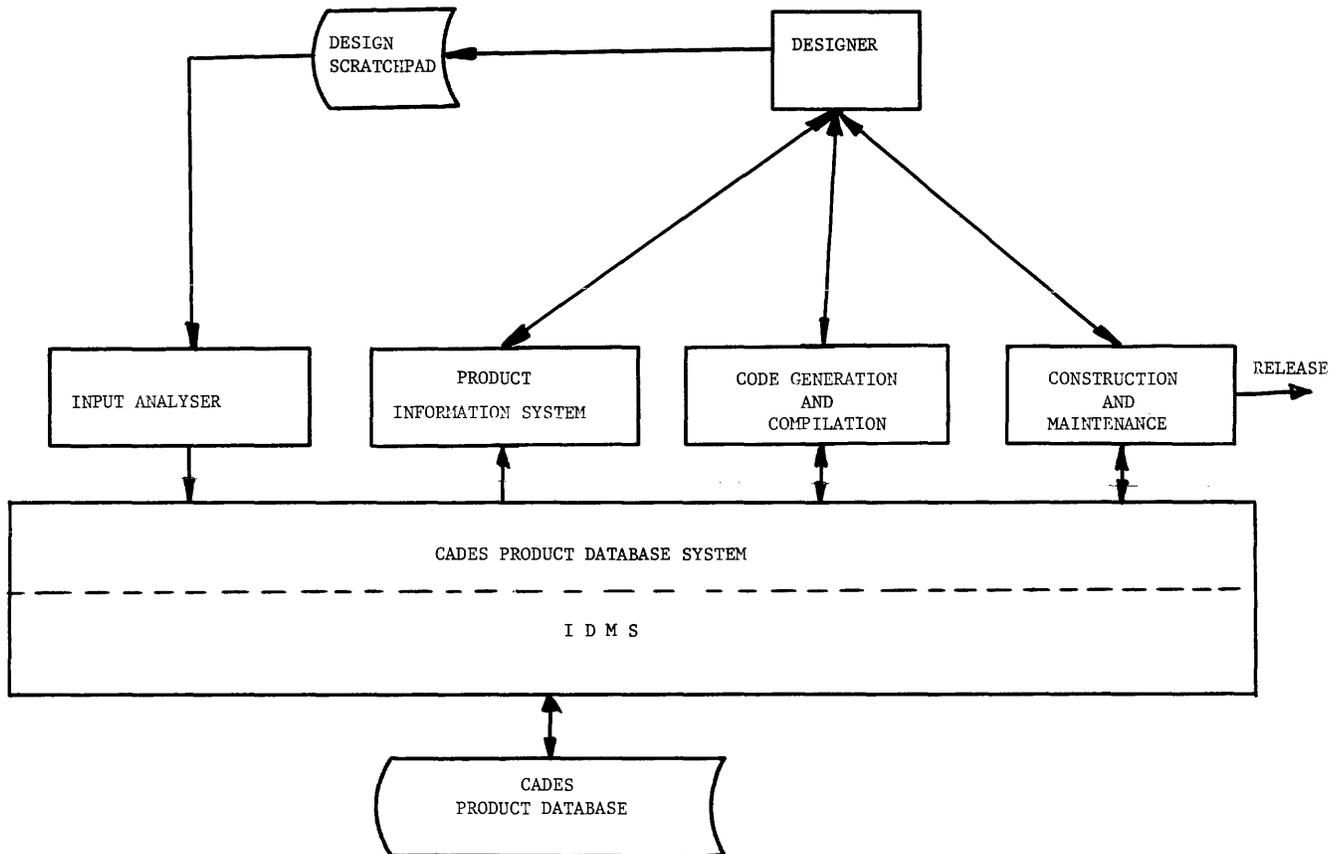


Figure 2

thorization form with every type of input to the data base system. The designer coded the proposed insertion (tree, SDL, management information, etc.), completed the appropriate authorization form and then collected the approval signatures of his team leader, chief designer and project manager before he was able to pass the update to a CADES service project for processing and return of an update acknowledgment. Although this system sounds horribly inflexible and bureaucratic it did achieve two things—the level of data integrity in the product data base after five years' use was very high, and it augmented and reinforced the project management mechanisms to a considerable degree. In fact, these controls placed on the development group for the sake of data integrity more than justify themselves purely in terms of rigorous software management techniques.

Product data base applications

Figure 2 gives an idea of the CADES system as a whole. The four applications shown, Input Analysis, Product Information System, Code Generation and Compilation, and Construction and Maintenance were the most significant in terms of impact, and certainly in terms of success. Less successful applications such as simulation and test program

generation were attempted and then subsequently fell into disuse due to lack of support.

The Input Analyzer was responsible for the syntax and semantic analysis of the design. It ensured that standards had been adhered to, that correct versions had been used, that the connectivity of the proposed design update was acceptable, etc. Less than full use was made of this checking stage. It would have been possible to code very rigorous and comprehensive checks in the Analyser, and for the project manager to choose what levels of validation he was prepared to accept in a trade-off against expediency. This would have been possible and would have led to some interesting results. However, time always seemed to preclude this level of 'luxury.'

The Product Information System presented an easy to use retrieval interface to the designers, in either interactive or bulk-data mode. The service project was the main users of this interface to regularly supply managers and designers with current product details and statistics. The interface also provided answers to 'what if. . .', 'who uses. . .' and 'how did this happen' type of questions.

The Code Generation and Compilation application had a great impact throughout the product development. The application started from design specification in SDL at the implementation level, formed an S3 source code module

from macros and primitives in the product data base, compiled it, and stored the subsequent S3 object code in the data base for subsequent collection. Because the designer did not deal with any sort of declaration and because he was constrained to make maximum use of the primitive macros stored in the data base, code production rates were increased very considerably and error rates dramatically reduced. Other advantages were that the code generator automatically inserted error checking and tracking code, and it, of course, imposed its own standards on the entire product coding.

The Construction and Maintenance application was driven by the master product version/release plan built into the data base. From this the system was able to construct from the object code modules, architectural details, etc., a set of load modules for export to a specific user site. The system kept track of user configurations, trouble reports, etc., and automatically correlated the most recent updates and fixes with the version currently in use on site.

THE ENVIRONMENT

Before discussing the results of using this CADES system on the VME/B product development, it will be useful to look at the target product and project involved.

VME/B is a facility-rich operating system with great emphasis placed on data management and job control language facilities. It supports a virtual machine architecture and is protected by 15 levels of hardware-implemented protection. A current typical release of VME/B represents, in total, between two and three million lines of S3 source code. Detailed design of the product was started in 1971 and the first version was running on 2900 architecture in 1973. The first customer release was made in 1975. The software was initially developed on System 4 machines, and then converted to run on a 2900 hardware simulator (i.e. 2900 architecture, old technology), before finally appearing on a true 2900 machine in 1974.

Until 1974 the project, about 200 people, was split over two sites which were two hundred miles apart. As the size of the project peaked the software engineering disciplines proved insufficient to handle this geographic split and the two teams were moved into one site. At any one time the CADES development and service group, including compiler development, represented about 15 percent of the total project team.

In the view of the above environment, and in view of the late 60s experiences by ICL in terms of operating systems development, the software engineering system was initially developed and controlled in order to promote the following VME/B attributes:

1. Highly structured, that is high modularity and low connectivity. This requirement initially dominated the performance requires.
2. Ease of maintenance and enhancement.
3. High predictability and reliability.

4. Strong management and technical control over a large project team.
5. Subsequent performance improvements.

The next section describes how these attributes were achieved in practice.

THE PRAGMATICS

(*Pragmatic*—'practical as opposed to idealistic' (Webster's New Collegiate Dictionary).)

When we set out to formulate and create the CADES philosophy and system we attempted from the start to be as practical as possible. We certainly adopted a rather basic engineering, rather than scientific, approach. This approach is described in detail in Reference 5. For instance, the form of the design language was based more on the facilities we wanted to remove from the programming languages as far as the designer was concerned than on more esoteric considerations concerning concise, formal definitions of the evolving design. It had to be usable, capable of quick, error-free production and easy to learn. Hence it ended up as being rather inelegant. But we could express emerging design in it, and we could generate S3 from it.

This approach to life pervaded the entire CADES system—methodology, language, data base, applications and service. The goal was to save the company as much product development money and lead times as possible. We set out early in 1970 not knowing a great deal about how we were going to do this and learned a great deal during the next seven years—both in a controlled learning way and also by bitter and expensive mistakes and experience. Some of the lessons are discussed below.

Economics and productivity

Over the seven-year period, the CADES system produced significant increases in programming productivity. At the end of the seven year period we were much better at producing code than at the beginning. Fred Brooks¹ quotes 'typical' IBM code production rates classified by the complexity of the product being generated. These are:

- Very few interactions—10,000 instructions per man-year.
- Some interactions—5,000 instructions per man-year.
- Many interactions—1,500 instructions per man-year.

He also quotes Corbato's (MIT's project MAC) overall production rate on MULTICS of 1200 lines of debugged PL/I per man-year. During the last three years of VME/B production when CADES represented a fully familiar, established and stable product, the production rate for the entire development team was around one module per man-month (four weeks). After generation, the average S3 source module was around 350 lines of code, derived from, perhaps, 100 lines of SDL. This represents a debugged programming rate of around 4500 S3 lines per man-year. This compares

favourably with the MULTICS rate and with the IBM rate for complex software. Hence, in terms of individual productivity CADES would appear to be at least a qualified success.

However, there is a more fundamental point here. In view of the ultimate size of the product and of the project, the experiences of the other manufacturers on products of similar size, and the problems we ourselves had during those seven years, it is *very* unlikely that the VME/B product could have existed at all without the CADES system, or an approach very akin to it. The power of CADES was that it forced software development activities, and in particular quantified design activities, into close proximity with rigorous management mechanisms. They presented sociological problems which will be discussed later. It did, however, provide us with an extremely well controlled and regulated large-scale software development.

Connectivity controls

One of the initial intentions in the CADES philosophy was to minimize and then subsequently control the evolution of the connectivity of the system. Connectivity is a major factor influencing the structure of a software product, the other being its modularity. Lehman⁴ has compared software structure to negative entropy in his Program Growth Dynamics studies. The Law of Increasing Entropy states: "The entropy of a system (which is the cross-product of connectivity and the reciprocal of modularity) increases with time unless specific work is executed to maintain or reduce it." This is another statement of the well known Structural Decay phenomenon in which the difficulty of system modification increases with system release number. I believe that the secret of maintenance and enhancement, and hence economic product longevity, lies in mechanisms to monitor and control connectivity evolution. CADES had such mechanisms built into its methodology and database applications.

A measure of how successful we were can be gained by looking at a system attribute called *Ripple Factor* (RF) and calculating this for VME/B release sequences. The term Ripple Effect was defined by F.M. Haney² and occurs most noticeably in systems with a large number of components. An enhancement or fix in one component causes, as a side effect, further necessary changes in other components, and possibly itself. The RF is the factor by which the work increases relative to that planned. Haney, via various sample systems, quoted RF of around 10 as being representative for a maturing software system of medium-to-high complexity. When a similar study was made of VME/B releases in 1975 the derived RF was found to be 1.4. It represents a remarkable improvement if Haney's figures are taken as representative. Subsequent experience in VME/B maintenance and enhancement would tend to confirm the fact that it is indeed relatively economic to enhance and develop now that it has reached a steady-state maturity. It remains to be seen whether the management controls will remain sufficiently rigorous to present the Law of Increasing Entropy finally

dominating. The current signs are, however, that connectivity is still under control.

Law of management futility

Lehman⁴ states a law which he calls the Law of Statistically Smooth Growth. I prefer to call it the Law of Management Futility, and to state it thus:

Growth trend measures of global system attributes may appear stochastic locally in time and space, but statistically are cyclically self-regulating with well-defined longer-term trends.

I believe that this law is a very fundamental one and explains why so many large-scale software developments go astray because of too much misdirected, rather than too little, management attention. Lehman's law, and my experience, state that management fire-fighting at any stage in the software development process is likely not to work. The fighting and extinguishing of a local bush fire will only raise the temperature of, and the pressure on, another area, probably more critical and expensive, downstream in the development process.

CADES recognized this phenomenon and prevented its abuse in two ways. Firstly, the product data-base was based on a comprehensive, complex schema which defined the *entire* development process, from statement of market requirements through to steady-state product maintenance, and defined in great detail the entire set of interrelationships existing throughout the life-cycle. As a result, little management action could be taken in isolation without its downstream impacts being automatically defined in a suitable degree of detail. I believe this schema, and its evolution in future systems, is fundamentally important and says very significant things about the very nature of the software development process.

Secondly, the CADES system itself, and its control over VME/B production, had a noticeable inertia to violent changes in direction by management dictate. The greater was the proposed change, the more noticeable was the inertia of the system. This may not seem to be a positive attribute of the system. However, Lehman's law says otherwise. Management actions should be smooth, controlled and fully cogniscent of the downstream implications of current action. The CADES system ensured this was true to a marked degree.

Sociological implications

It was obvious early in our formulation of the CADES principles that a large team would ultimately be employed on the VME/B development program. This one fact had a strong influence on the entire CADES system—its methodology, language and software. A decision was made that all aspects of the system would impose a rigid discipline and control over the project team, even at the expense of flexi-

bility and ergonomics. Indeed, one of the constraints on CADES development was that the entire system should "mechanize" as much as possible of the software development process, either with machinery (i.e. software tools) or by formal disciplines and procedures which attempted to take as much arbitrariness out of the process as possible. As a result, initially, some designers felt that their creativity was being stifled and their intellectual freedom curtailed. In some ways they were correct and, as a result, in the first two years of operation, a few designers left the project specifically because of the technology. Perhaps five percent of the workforce left over this two year period, ostensibly for this reason. However, at the end of the two years, the management team was able to agree that, if we had had to shed five percent because of redundancy or whatever, there would have been a close correlation between the two sets of five-percents. As the system proved itself during 73-75, designers and managers joined the project ostensibly for the same reasons.

Multiple versions

In the first version of CADES we were so intent on solving difficult, important problems that we completely underestimated the magnitude of the multiple versions problem for large systems development. As a result, in the new CADES version on 2900 we had to include comprehensive and powerful mechanisms for multiple version handling down to the local procedure and data item level. This consumed a significant amount of effort in the second CADES development. I believe such a mechanism is fundamental to the real usability of any production software engineering system. Not only should multiple, limitless versions of almost everything be allowed, but also automatic "inheritance" mechanisms have to be implemented between these multiple versions in order to accurately reflect the way software systems are developed in practice.

On bridging data bases

The final observation does not refer directly to software engineering or operating systems development at all, but rather to database administration. James Martin once told me that he had no knowledge of a *successful* migration of the entire contents of a large database from one database system to another. At the time I failed to understand the subtlety of the point. I now fully appreciate its meaning. In 1976 we were faced with the conversion of the CADES data base contents from the initial data base system, an in-house transposed file system, to the Cullinane IDMS system. The initial data base size was modest, about 60 MByte. The strategy was very simple; extract the information from Data Base 1, process these files into user input language for Data Base 2, and then input it to Data Base 2. The activity was at least four times as expensive as planned.

The reason was very simple. Data Base 1 had been in daily production use for four years. Although tight controls

had always been placed on the integrity of data submitted, it was inevitable that "semi-corruptions" would creep in from time to time. As the Data Base 1 evolved and was supported by the service group, it "learned" to handle these semi-corruptions (such as missing information) and still give adequate service. However, Data Base 2 and the bridging software knew nothing of this acquired knowledge and as a consequence fared badly when faced with less than 100 percent perfect information. Fortunately, the corruptions were minor and Data Base 2 made a good, but expensive, recovery.

CONCLUSIONS FOR THE FUTURE

I think that I have to conclude that CADES was at least a qualified success in its control of and contribution to the development of VME/B. The very existence today of VME/B would support this. Its use on this project did, I contend, save the company money running into millions of dollars. However, I doubt that the CADES system as it stands today will ever be even approximately repeated.

Mainframe manufacturers have now recognized that adding more and more people to a project does not reduce its risk of failure. Indeed, it can have quite the reverse effect. Although software is still getting more complex, teams are getting smaller. The software engineering systems of the future will reflect this. Such systems will become increasingly important and central to the well-being of even modest projects. However, they will reflect a greater emphasis on ergonomics and less on rigorous management and administrative control. This is the approach being taken at Bell Northern Research Laboratories in Ottawa in our development of a new-generation system called ISES, Integrated Software Engineering System. This system, the prototype of which has just entered trial use, combines many of the lessons learned from the CADES exercise with current, state-of-the-art thinking in software engineering. Some of the features embodied in the ISES system are:

Requirements engineering

In a commercial, industrial or government environment concerned with the development of high-quality, complex software destined for a highly-competitive marketplace, the rigorous definition and acceptance of product requirements is the Achilles' heel of the product development group. In this environment it is not enough to develop high-quality designs and efficient, timely implementations—they also have to be *appropriate*. Without a formal approach to product requirements definition and its total integration into the product-development and life-cycle process there is no quantified manner in which to assess this appropriateness until the customer receives, or refuses, his product. ISES includes an approach to definition of product requirements which is as quantified and integrated into the development process as are other, more accepted, aspects such as unit and integration testing.

Graphical design language

I identified one of the "weaknesses" of CADES as being its gross approach to the control of Chinese-Army-type operations. The industry has moved on. Large teams become less and less the norm. So must software engineering evolve. Textual design languages are weak for describing the back-of-envelopes design activity. They demand completeness of expression. During the early stages of conceptual design completeness tends to be very low on the list of priorities, rightfully so. In this context, we need to be able to support the expressionism associated with the highly-dynamic, creative thought processes and decision making of our most experienced, creative designers. To do this, ISES provides the designer with a powerful abstract symbolic interface supported by an intelligent color raster graphics system. With this interface the designer can sketch out his design and architectural ideas in close to a random fashion whilst the system interprets these into more orthodox design representation, placing this into the global context of the current ISES data base.

Software metrics

The more one formalizes and captures the total software life-cycle process with systems such as ISES and CADES, the more able one is to develop the physics of large software systems, or Software Metrics.⁷ A large proportion of the software engineering community is striving to understand the nature of large-scale, complex software systems. We need to be able to define the parameters of these systems, the relationships governing these parameters, and how best to optimise them. The power of a system such as ISES in this context is that it controls and contains the total flow of information associated with a product's life-cycle. Hence, there is a level of activity-monitoring architecture within ISES—the system monitors its own usage and is able to evaluate and refine in-built metric models as a result of this day-to-day usage. It is a powerful step towards combining the worlds of theoretical and practical software engineering.

Hardware CAD compatibility

In many product development situations the position of interfaces between the levels of hardware and software implementation is arbitrary to a large extent. The factors which dictate the divisions are tactical ones and include flexibility, maintainability and timing criteria. In hardware engineering

this arbitrary nature is well illustrated in the division of a design between custom and silicon chips and printed circuit boards. Indeed, several manufacturers now adopt the technique, with the aid of advanced computer-aided design systems, of prototyping their products in PCB technology and then, once proven, reducing these PCB's each to a single custom chip. This flexibility should also extend to the outer software-implemented levels of a product. In order to aid efficiency in critical areas a manufacturer may want to re-implement a software function, or series of functions, in a chosen hardware technology. Today, this is a highly complex, risk-laden task. The transfer into hardware usually stops at the firmware level. ISES, on the other hand, is being implemented as part of a total-technology CAD system. Eventually this system will be driven exclusively by technology-independent requirement and problem statement languages. The ultimate system will provide complete flexibility to the designer in terms of how his design is implemented, and freedom to move between technologies for a single piece of design. Thus this CAD system, which contains ISES, will support the fundamental concept of the Implementation Technology Independence of the holon design unit.

ACKNOWLEDGMENTS

I should like to express my gratitude to the entire CADES team, both past and present members, for making the last seven years such enjoyable and entertaining ones. I should also like to express my admiration of ICL management for having the courage to follow through with the CADES program during times when the benefits were less obvious than they are today.

Permission to publish extracts from a paper delivered to the Second International Symposium on Operating Systems in Paris in October, 1978, is acknowledged by the author.

REFERENCES

1. Buckle, J. K., *The ICL 2900 Series*, MacMillan, 1978.
2. Haney, F. M., "Module Connection Analysis," *FJCC*, 1972.
3. Koestler, A., *The Ghost in the Machine*, Chapter 3.
4. Lehman, M. M., "Programs, Cities, Students—Limits to growth?," Inaugural lecture at Imperial College, May 1974.
5. Pearson, D. J., "CADES—Computer Aided Design and Evaluation Systems," *Computer Weekly*, July/August 1973.
6. Pratten, G. D., and R. A. Snowden, "CADES, support for the development of complex software," *EUROCOMP*, 1976.
7. Gibb, T., *Software Metrics*, Winthrop, 1977.
8. Brooks, F. P., Jr., *The Mythical Man-Month*.

The integrated control/distributed power software development shop

by JEAN-PAUL RENAULT

CIT-Alcatel
Velizy-Villacoublay, France

INTRODUCTION

In the constantly expanding spectrum of microprocessor applications, there is a large class of systems which used to be implemented with large or minicomputers, in which new software engineering problems have emerged. Typically, these systems involve quite a few hierarchical microprocessors—they perform fairly sophisticated control functions, and must comply with stringent reliability and availability demands. Though the distribution of functions significantly reduces the complexity of some of the technical problems usually encountered in such systems, the volume of software to be developed remains large, and system tests are still a problem.

Furthermore, the development systems offered on the market by microprocessor manufacturers or independent vendors are very well adapted to the development of small applications which involve a unique microprocessor and a limited volume of code, but they are insufficient for the development of large systems. The system builder engaged in distributed system development will, therefore, have problems to solve, until the ideal support software and hardware are made available on the market.

SOFTWARE DEVELOPMENT REQUIREMENTS

General requirements

What a programmer expects from a development system is now well understood for large and mini-computer software development, and since it has to do mainly with the way the user sees the system, or with the way he gains access to it, and not with the way it is built, there is no reason, no matter how different the target processor may be, that the same rules should not apply to microprocessor software development.

The qualities assumed of a development system have to do with ease of use and access, the overriding consideration being, therefore, that the system should be terminal-oriented. This should give the user unrestricted access to all system facilities from his desk. In addition, it should limit the need to handle media; cards should be completely aban-

doned, listings should not be used as the support of run results, but as reference only, when a program has reached a satisfactory level of completion, and magnetic media should be handled for archives, deliveries, etc. . . , by the system itself, and not at all by the programmer. Even cassettes or diskettes, though handy, can create a great amount of confusion.

From his terminal, the user should have access to a certain number of programming aid facilities:

- Program handling aids such as text editors, library managers.
- Documentation support facilities, including document entry, document updating, document editing and printing, either directly on listings, or through phototype-setting for quality printing.
- Programming aids—Compilers, link editors, loaders, debugging facilities, etc.

Another extremely important feature of the development system is the command language it offers. It should be rich (i.e. it should offer many functions), and easy to use, which in fact means that it should be easy to invoke from a terminal, both directly through system commands and indirectly, through catalogued procedures built by the user to help him perform frequently repeated complex operations by means of simple commands.

Specific microprocessor requirements

Programming problems are not significantly different in distributed systems, even large ones, from those encountered in others. Programming proper is not a problem—high-level languages, when available, are perfectly suited for distributed system programming, and the fact that some specific microprocessor features are not accessible through them will never justify going back to assembly languages, except, maybe, in a few sections where performance is critical. This is true, in fact, of any system and of any high-level programming language. On the other hand, in some areas, programming will be even simpler. The kernels which operate the microprocessors are generally message-oriented,

and support only fairly static processes and connections between processes, which makes them easy to understand and develop. Even reconfiguration mechanisms, generally quite sophisticated in other systems, tend to be rather simple in a distributed environment, at least from the programming point of view.

The problems associated with unit testing are of the same order of magnitude as those traditionally found in systems—depending on the detailed internal architecture, it will be more or less practical and economical to create an environment which lends itself to unit tests.

System testing is, however, a completely different problem in distributed systems—simulation works mainly for unit testing. The complexity of interactions, and the number and variety of components at work, generally make simulators uneconomical in such environments. The only economical means is to test software on prototypes, equipped with the appropriate hardware and software probes connected on the microprocessor, the memory and maybe some other critical points. Besides, these tests should be run in interactive mode under a system offering the right kind of debugging facilities, such as traces, breakpoints, snapshots, at the symbolic level.

CURRENT SOFTWARE DEVELOPMENT SYSTEMS

In a system development house, the approach first selected for the development of software for microprocessor systems is to use the existing computer facility. Typically, such a facility is a medium-to-large computer system operated in time-sharing (see Figure 1a). For the benefit of microprocessor software development, existing tools for system and program design, program management and documentation management may be used without problems. However, two questions do have to be answered: What "programming system" should be used? How should testing be done?

In this environment, as far as program translation is concerned, the only possibility is to develop a compiler or adapt an existing one to generate the target code, and to develop a linkage editor and other loader utilities as required by the microprocessor hardware. Further, a microprocessor test simulator has to be developed for unit tests.

When it comes to system testing, programs in load format have to be output on an external medium, which can be used as an input to the prototype system; but, for efficiency, debugging facilities have been implemented on the prototype system to fulfill the requirements stated previously. Though such a development system can work, it has many disadvantages:

1. Development of a cross compiler and of the other programming facilities may be a costly proposition for, after its development, it will have to be maintained and enhanced as required, adapted when new microprocessors have to be used, etc. Even if such a line of translators can be found on the market, adaptation to new microprocessors will generally introduce delays.

2. Simulators have to be developed.
3. Transfer of programs on an external medium is generally not satisfactory because:
 - There is not always a compatible medium on the development system and test system.
 - When an error is found, binary patches will be used on the prototype instead of source corrections, because of the time otherwise involved in updating, compiling, linking and transferring a program. Such a practice is potentially harmful and costly.
4. A debugging facility on the prototype system has to be developed. This may be expensive, if it is to perform symbolic debugging in an interactive mode.
5. When going to other microprocessor compilers, linkage editors, simulators and debugging aids will have to be extensively adapted.

The main advantages are the use of:

1. Existing systems.
2. Existing language-independent software tools, such as design aids, program librarians and documentation aids.
3. If a standard implementation language exists in the user organization, it can still be used on microprocessors at a certain cost, even though it might not give enough visibility to some of the processor functions.

Some of the problems just mentioned, especially in 3, can be alleviated by connecting all consoles to both the development and the test systems. This might be done as shown in Figure 1b, using a concentrator to which both the development system and one or more test systems would be connected via transmission lines. The transfer of programs can then be done automatically, and the user may have access from his console to all resources, whether development or test.

In a development facility equipped with a programmer's workbench,³ the situation is not significantly better. A cross compiler, a linkage editor and a simulator would have to be developed for a "Programming Machine" (see Figure 1c), leaving the program and documentation-handling facilities on the programmer's workbench, which is essentially a file machine, and does not look like the right system in which to place compilers and simulators unless its configuration is expanded. But then it becomes a conventional system, like those described earlier. The programmer's workbench does not help to solve the system test problems any better either, since the required debugging facilities again have to be developed on the prototype system.

Another variant of this type of software development system is the Interactive Session Monitor,² in which microprocessors are connected directly to the existing development system (Figure 1d). Programming tools are run under time-sharing on the development system, and tests can be done either in simulation mode, or by executing the program on the microprocessor. This system again does not solve the problems discussed, since cross-compilers and simulators

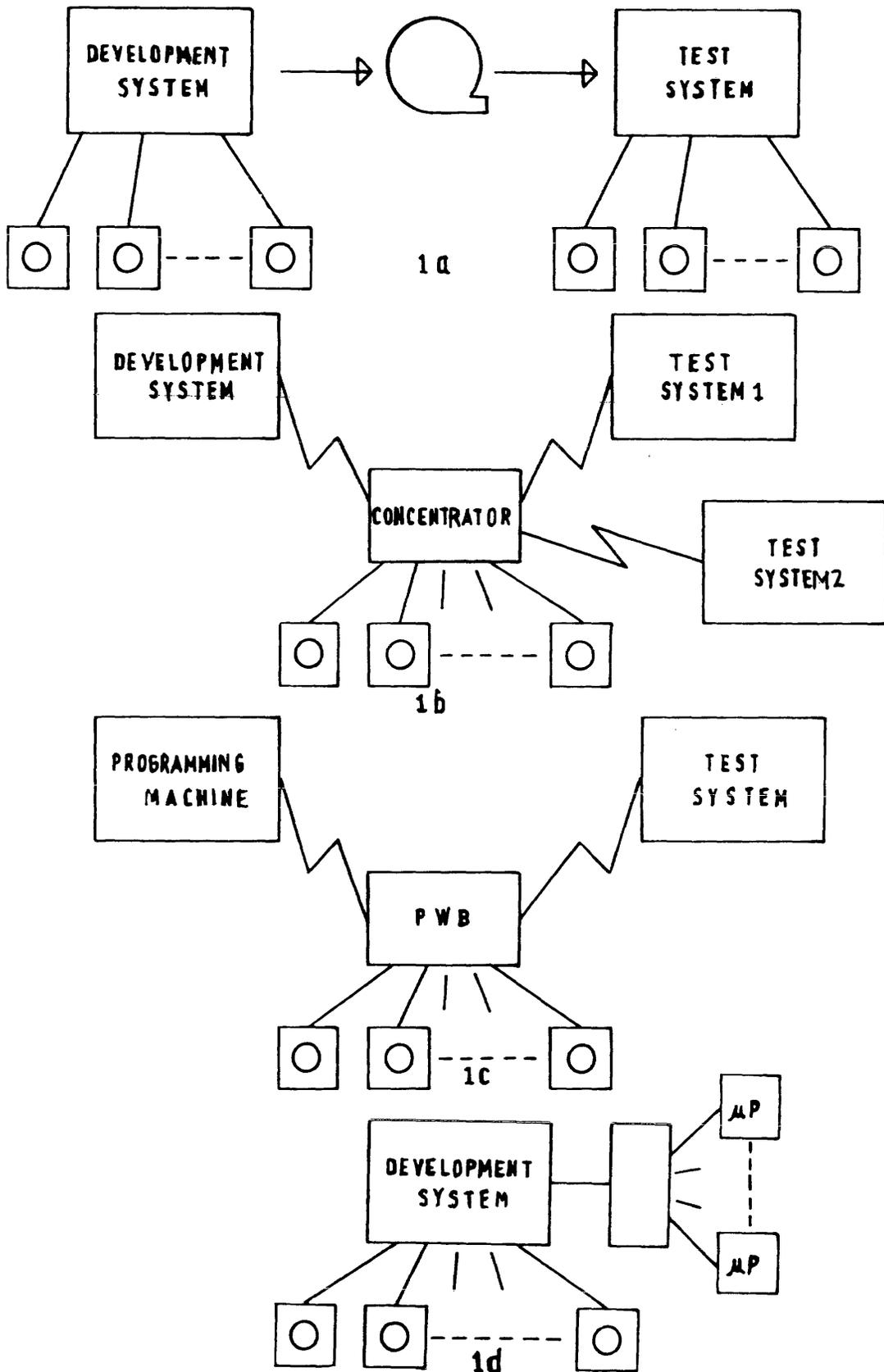


Figure 1—Conventional development systems.

have to be developed, and execution on the microprocessor offers only limited possibilities.

MICROPROCESSOR DEVELOPMENT SYSTEMS

Software development systems offered by microprocessor or independent vendors can be used instead of an existing facility. For example, the software development facilities for the 8080 line of microprocessors found on the INTEL-LEC system are:

- A "Programming system", including
 - A text editor, to enter, update and inspect text.
 - Compilers, for high-level languages.
 - An assembler.
 - Link/loader facilities.
- A simple file system, to store programs in various formats.
- A debugging system, for unit tests on the system.
- An in-circuit emulator, for testing of a microprocessor program within the prototype environment.

The peripheral configuration consists of a single keyboard console, dual floppy disk drives, and optional printers and other peripherals.

The vendors' systems support their microprocessors only, but some independent suppliers offer systems which support several microprocessors. Indeed, microprocessor software development systems offer a very adequate range of functions for program development. However, the tools or utilities found on large or minicomputers are entirely missing; there are generally no program library managers (the file systems offered are insufficient in this respect), no documentation aids, no design aids, no project control utilities. Performances are limited—compilations are slow, printing is slow and secondary storage on floppy disks is limited in throughput and capacity.

The advantages, however, are:

1. The programmer has easy access to the system (a ratio of one system to two programmers is generally used).
2. These systems are relatively cheap (in the vicinity of \$20,000).
3. In-circuit emulation, which allows one or more programmers to debug on the prototype system, is generally very good.

In large projects involving more than a few programmers, where the volume of code to be produced exceeds 50,000 to 100,000 lines, the investment in development systems is no longer negligible, diskettes on which programs are stored proliferate and large compilations for system integration hit the limits of the development system, in terms of speed and secondary storage capacity.⁴

INTEGRATED CONTROL, DISTRIBUTED POWER

The relatively low cost of microprocessor development systems, the programming facilities they offer, the easy ac-

cess to the system resources they provide and their in-circuit emulation functions make them hard not to use. Their shortcomings can be overcome by using a two-level system (Figure 2):

1. A centralized level, or "Central System," performing all functions of concern to the project as a whole, and offering services to the programmer where the microprocessor development system is insufficient.
2. A decentralized level, consisting of microprocessor development systems connected to the central system by transmission lines.

Microprocessor development systems

The following functions are performed at this level:

1. Program preparation—Program entry, update and inspection, and program translations, compilation/assembly, link editing.
2. Unit testing on the development system.
3. System testing on the prototype, using the in-circuit emulation facility.

Large volumes of input, such as the initial entry of large programs in source format, are performed on the central system, as is program library management. Typically, the user has his private work library on as limited a number of diskettes as possible, and access in read-only mode to the central library. To prepare his work he may request, from his console, transfer over the transmission line of any number of programs/files, as needed for his own work.

Individual programmer's programs are included in the central reference library, according to rules depending on the project, the system integration philosophy, and other considerations, but at times that guarantee enough visibility of the state of the work of each individual, throughout the project. A decision to include a new program in the reference library is, however, always an explicit project management decision.

Central system

Each microprocessor development system is seen by the central system as a time-sharing terminal, thus giving the user access to all central system resources.

The central system can be regarded as a *file machine* and *spooling machine*.

As a file machine, it manages program and documentation libraries; as a spooling machine, it performs all volume input/output—card input, listing and documentation editing. In fact, no card reader or printer of any sort need be connected on any microprocessor development system—all input/output can go through the central system. In addition, simple keyboard terminals can be connected for functions such as program or documentation entry/update/inspection, which can be done at the central level or elsewhere. Any other

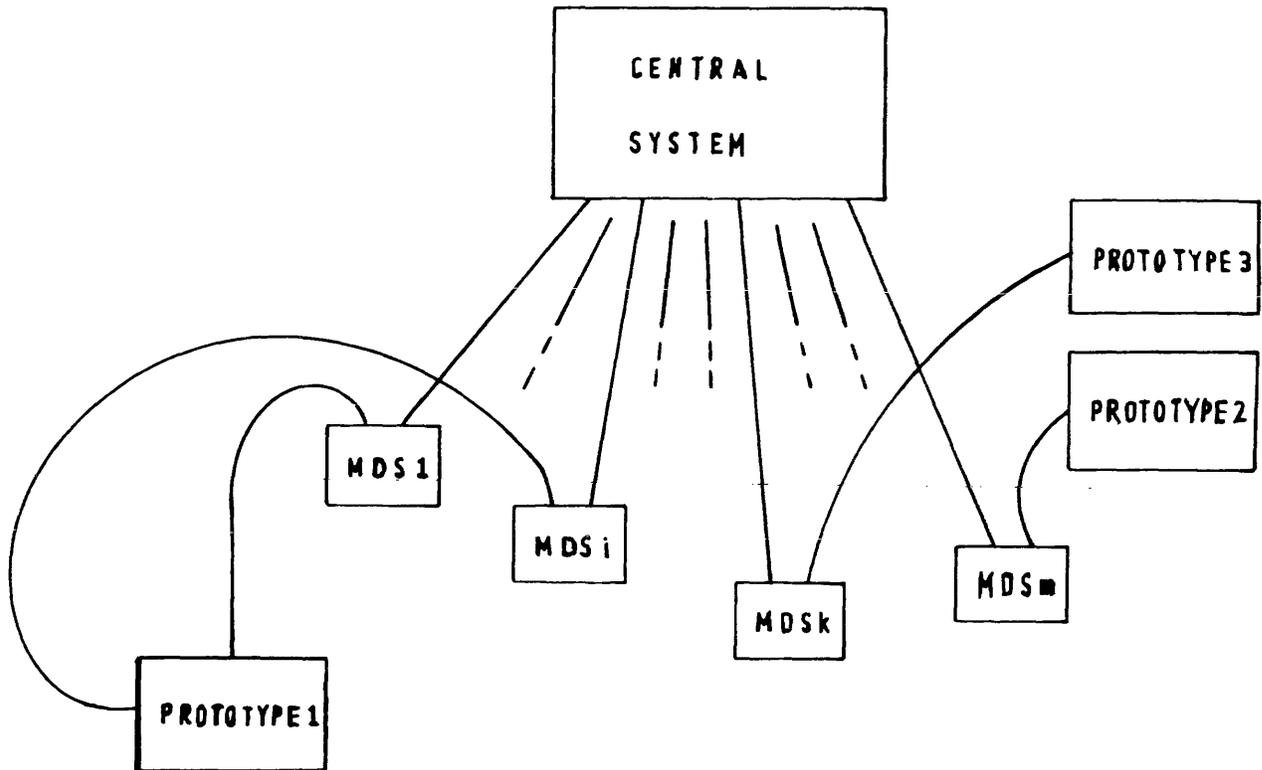


Figure 2—Centralized control distributed power.

utilities needed, e.g. a source code formatter, a macrogenerator, a test data generator, can be implemented singly on the central system, rather than on the several development systems.

The central system plays very much the same part as the programmer's workbench, except that the latter is a "front-end" processor, whereas here it is the microprocessor development systems that are front-end processors.

Advantages and disadvantages

What we have described is not ideal, but it does combine the advantages of centralized systems with those of microprocessor software development systems.

Its residual shortcomings are:

1. The file transfer rate over transmission lines is limited. Most time-sharing terminal throughput is limited to 9600 bauds, which means that the practical maximum transfer rate is around 1000 characters/s. This limitation is quite acceptable for a small program, but if a program of 10,000 lines of source code has to be transmitted over a 4800-baud line, transfer will take between 20 and 30 minutes. In practice, this is not really a problem, because:
 - a. Modules are stored in source and object format.
 - b. Modules are kept fairly small.
 - c. Source is transferred only when it is to be modified.

2. The programming languages and debugging systems are those offered by the vendor with the microprocessor development systems. This precludes the standardization of a single implementation language on all microprocessors within an organization—at least at the present time—and makes the client organization entirely dependent on any decision the vendor might make to enhance, abandon or maintain the corresponding products. Though this is not a satisfactory situation, it is not unusual either.

In compensation, however, it may be noted that the system described offers the advantage of being able to use tools, methods and techniques normally used in software development on computers other than microprocessors, thus yielding the kind of technical proficiency and managerial control that the state of the art in computer software engineering permits. Compared to other types of development systems, the flexibility and costs of this system are attractive features.

Flexibility

It is generally easy to connect a development system to a system running under time-sharing, so that, if various microprocessors are being used, it will be possible to connect the various corresponding development systems offered by the vendors, or some independent vendor. Further, for operations of general interest not requiring compilation, like

program or documentation entry, such as gathering information concerning library status, simple time-sharing terminals may be used. With a single central system it will therefore be possible to develop, manage and test programs for microprocessors of various origins.

This set-up, in fact, offers other interesting possibilities. For instance, concerning programming, although the system is clearly geared to perform compilations on development systems, it does not preclude using cross compilers. As reported in Reference 1, software for microprocessor *A* might be developed using a programming language available on the development system *M (B)*, designed to program microprocessor *B*. Program entry, compilation and unit testing are carried out on *M (B)*; then a cross compiler on the central system is used to generate code for microprocessor *A*. The development system *M (A)*, or an independent system, will be used to perform system tests using *A* on a prototype. Though this process seems fairly involved, it can be made fairly easy in practice, since *M (B)* and *M (A)* can both be connected to the central system.

Costs

The load generated at the central system in such an application is fairly low, in terms of CPU usage, since most operations are file or terminal operations. A medium-to-large minicomputer should therefore be perfectly suitable. Moreover, actual connect time should be fairly low, since it is essentially determined by the frequency and durations of transfers or other operations requested from the central system.

It is assumed here that a minicomputer in the \$300,000 range (hardware only) could provide simultaneous support to 25 to 30 users,³ and therefore that up to 50 microprocessor software development systems could be supported. The actual cost (purchase price) of the total system, per development station, would therefore be about \$6,000, plus the price of the development system itself, which is about \$20,000. If it is assumed that one development system can fulfill the needs of two programmers, that the central system has a lifetime of five years, that a development system will be obsolete after three years and that the cost of operating the central system is the same as the cost of hardware, the cost per programmer, per year should be in the vicinity of \$5,000.

The service thus provided should be of very good quality, combining those of a minicomputer-based time-sharing system with those of microcomputer development systems. The cost of computer time as assessed above should be compared to figures used for many system developments. Typically, on a system such as a 370/158 where an average of two hours of CPU time per programmer, per month is consumed, the cost per year and per programmer is around \$10,000. An overall cost reduction of around 50 percent can therefore be expected with respect to conventional systems.

With respect to stand-alone microcomputer development systems, the concept presented increases the hardware cost by around \$6,000 per development system, plus operating costs. But it is pointed out that:

1. Printers on the development systems are no longer necessary, since printing is done at the central site (typical cost of a printer is \$2,000-4,000).
2. Extra diskettes are no longer necessary, since a large disk capacity is accessible on the central system (an add-on, double density dual diskette costs around \$5,000).
3. Extra services such as library maintenance, documentation aids, etc., and fast I/O devices are accessible on the central system.

CONCLUSION

This approach is being progressively implemented at CIT-Alcatel for the development of distributed microprocessor-based switching systems. Initially, INTELLEC systems were connected to CII-IRIS 80 as Remote Batch stations giving access to all development tools available for other developments, then, to improve access conditions and response times, they were connected to a CS 40 (a computer developed by CIT-Alcatel for switching systems), as time-sharing consoles. Replacement of the CS 40 by a minicomputer, to further reduce the costs of the system, is now being considered.

The system described offers distinct advantages with respect to developments on large systems, or on stand-alone microcomputer development systems, although in its present form the overall services offered are limited by the performance of the individual development stations, and by the transfer rates.

Nonetheless, the services offered and the flexibility gained make the system the real solution to the software development problem for the builders of large systems who want to use commercially available development products.

REFERENCES

1. Aslanian, R., "Production efficace de logiciel portable pour les microprocesseurs d'INTEL et de MOTOROLA," in *Outils de Production de Systèmes Industriels, Journées BIGRE, IRIA 24, 25 Novembre 1977*, pp. 42-44.
2. Gnesotto, G., "Système Interactif de Développement de Microprocesseurs," *Convention Informatique 1978 édition provisoire*, pp. 269-274.
3. Ivie, E. L., "The Programmer's Workbench—A Machine for Software Development," *Communications of the ACM*, Vol. 20, No. 10, October 1977, pp. 746-753.
4. Larnaudie, J. C., R. Aslanian and P. Caizergues, "La Maîtrise des méthodes et outils évolués de programmation fait reculer les limites d'utilisation des microprocesseurs," in *Outils de Production de Systèmes Industriels, Journées BIGRE, IRIA 24, 25 Novembre 1977*, pp. 21-24.

On the fate of software enhancements

by NORMAN K. SONDHEIMER*

Bell Laboratories

and

The Ohio State University
Columbus, Ohio

INTRODUCTION

One of the most serious problems for vendors of software is pressure from users to enhance and extend their products.¹ One important type of enhancement is the change in language form aimed at improving user productivity by making the system easier to learn and use. Ideas for this sort of enhancement appear regularly in such publications as *Sigplan Notices*. As evidenced by the correspondence in that publication, there are many opinions on the quality of these proposals. However, little is known about the actual fate of enhancements that have been added to existing software systems and released to users. When a new feature is released with the best of intentions and the usual fanfare of documentation and instruction, its acceptance by programmers may have little to do with the sincerity of the request or the quality of the feature.

This paper reports on one experimental study of the fate of enhancements to existing software systems. Features whose desirability was established by a survey of the user community were added to an established and heavily used text and program editor. The enhanced editor was then released to replace the old editor for a part of that community, and the reaction was monitored. The remainder of this paper details the experimental method, gives the results, and draws conclusions on the fate of software enhancements.

EXPERIMENTAL METHOD

The experiment used to test the fate of user-requested enhancements to software systems proceeded through several distinct phases. In this section, I explain what was involved, beginning with the choice of software system and user community.

The system used in the experiment was the text editor of the UNIX** time-sharing system.^{2,3} The editor is the main tool for program and document developments on the system. It falls within the QED family of editors tracing back to Deutsch and Lampson's editor.⁴ It is line-oriented and

biased towards the execution of single commands in isolation and identification by a pattern of strings to be modified.

The programming organization involved in the experiment was the Switching Operations Systems Laboratory of Bell Laboratories in Columbus, Ohio. The laboratory produces minicomputer-based real-time systems for the telephone network. The laboratory employs more than 150 people. Over 100 have some involvement with programming. Within the laboratory, large amounts of program and document development are regularly done with the text editor.

The experiment began with a set of proposed enhancements consisting of Features 1 through 5 listed in Table I. Included in the laboratory is a group specifically responsible for operating system support including dealing with user problems with the editor. The majority of ideas for enhancements were drawn from their experience. Each feature was aimed at making the system easier or safer to use by supplying facilities either significantly more complex to access or not available in the existing editor. Feature 1 allows the reduction of a sequence of two or more commands to a single step. Features 2, 3, and 4 allow savings in the size of commands. With Feature 2, surrounding characters need not be included in order to identify a string meant for substitution. With Feature 3, a repeated string of characters can be identified by a shorter string. With Feature 4, repeated replacement strings can be identified by even simpler means. Feature 5 is meant to protect against loss of information by premature exit from the editor. Hence, each feature made a positive contribution to the use of the editor. Features 3, 4, and 5 were suggested by the members of the support group, and Features 1 and 2 were developed with their assistance. All the additions were upward-compatible from the current editor, with the exception that the special character used in Feature 4 would take on a different meaning in a single context.

The entire laboratory then evaluated the proposals through a formal survey which detailed the enhancements. Respondents were asked to identify their usage of the text editor, to rate the proposed features in terms of helpfulness in their own work, to estimate their likely rates of usage, and to rank the features in terms of desire in seeing them implemented.

Based on the survey, Features 2, 3, and 4 were chosen

* Present address: Sperry Univac, Blue Bell, PA.

** UNIX is a trademark of Bell Telephone Laboratories, Inc.

TABLE I.—List and Description of Enhancements Used in Study

No.	Description
1	A command to interchange two groups of lines.
2	The ability to identify for substitution specific instances of matches of patterns within a line. Previously, either the first or all matches had to be substituted for. Now, individual matches or sets of matches could be specified.
3	The ability to define simple macros within an editor session.
4	The ability to use a special character to stand for the second and subsequent references to a string when inserted by consecutive substitute commands.
5	A special warning when exiting from the editor without writing out the work space to a permanent file.
6	The ability to append the work space to the end of an existing file.

for implementation. The special character in Feature 4 was changed to a less-used symbol. In addition, another feature suggested by a member of the support group, No. 6 in Table I, was chosen for implementation. This feature allowed a sequence of editor and UNIX control language commands to be replaced by one editor command. The four features were implemented without noticeably changing the editor's performance characteristics.

The testing phase saw the extended editor replace the original on two selected computers within the laboratory. Two common methods of release were used. The editor was first released on the computer used for systems development by a group of nearly 50 people, herein called Group 1. Each person received extensive documentation on the new features, including examples of the features' usefulness. Group 1 was also alerted to the change in the editor by a message that appeared on its terminals whenever the UNIX system was entered. At a later date, Group 1 received replacement pages for the programming manuals describing the extended editor. Consulting was made available on the features during the first week after release. Announcement of the existence of the features also appeared several weeks later in an internal UNIX users' newsletter.

The extended editor was also installed on the development computer for another project involving a dozen people, herein called Group 2. The small size of this group allowed special attention to be given to the programmers, including a half-hour introductory lecture given to the entire group upon release, and individual follow-up sessions two weeks later. The same introductory memo and message used with Group 1 was used on the day of release to Group 2. The new manual pages were also distributed on that day. At that time, the newsletter had already been issued.

The evaluation of the features involved both automatic monitoring of editor usage and further surveys. All uses of the existing editor were monitored for a time prior to release of the extended editor to obtain a basis for comparison. After release, usage of the new features was monitored for an extended period of time. The monitoring dates are given in Table II. Group 1 had an initial monitoring period lasting over three months. Between 317 and 1151 editor sessions were recorded, with an average of 651 sessions per time

period. Total commands monitored numbered over 375,000. With Group 2, monitoring was done over the three weeks following release. Between 132 and 481 sessions were recorded for each full day; the average was 277 sessions. Total commands numbered over 53,000. All information was obtained while both groups were carrying out normal duties.

Approximately eight months after release of the revised text editor, a follow-up survey was distributed to the users to obtain their impressions of the helpfulness of the new features, especially with respect to other features in the editor. The users were also asked to estimate their familiarity with different features in the editor. A final monitoring period covering 1200 sessions and over 110,000 commands was employed for Group 1 (see Table II). At that time, Feature 3 had been removed from the editor (due to the preliminary interpretation of the results given in the following section).

To summarize the method employed, the user community was first consulted on the choice of enhancements. The usage of the editor was then monitored both before and after the release of the new features. Further polling was done after the user community had the opportunity to use the features. Some further monitoring completed the survey. The results of the experiment are given in the following section.

RESULTS

The pre-implementation survey served as a check on the users' desire for the proposed enhancements. The monitoring and the follow-up survey established the fate of the enhancements. This section presents the quantitative results of these efforts.

The choice of enhancements was clear after the pre-implementation survey. Fifty-six users returned completed surveys; 27 were from Group 1 and 8 from Group 2. Ninety percent of the respondents indicated regular use of the editor. A high level of interest in the survey was reflected in the more than two dozen extra comments and suggestions

TABLE II.—Dates and Working-Days with Associated Time Period Labels on which Editor Usage was Monitored for Each Experimental Group

Time Period	Group 1		Group 2	
	Dates	Working Days	Dates	Working Days
Base	8/31-9/6/77	4	10/17-10/25/77	6.5
1	9/7-9/8/77	2	10/25/77	0.5
2	9/9-9/12/77	2	10/26/77	1
3	9/13-9/14/77	2	10/27/77	1
4	9/15-9/18/77	2	10/28-10/30/77	1
5	9/19-9/20/77	2	10/31/77	1
6	9/21-9/22/77	2	11/1/77	1
7	9/23-9/26/77	2	11/2/77	1
8	10/12, 10/18/77	2	11/7/77	1
9	11/3-11/4/77	2	11/9/77	1
10	11/16/77	1	11/10/77	1
11	12/19-12/20/77	2		
Final	6/2-6/10/78	6		

TABLE III.—Average Estimates of Helpfulness of Each Proposed Feature Obtained from Preimplementation Survey, Indexed by Groups

Group	Features				
	1	2	3	4	5
Entire Lab	3.04	2.64	2.59	2.45	3.15
Group 1	3.00	2.89	2.65	2.44	3.30
Group 2	2.62	2.75	2.62	2.43	3.75

NOTE—Scale for point values show: 1=Very Helpful; 2=Helpful; 3=Somewhat Helpful; 4=No Help; 5=An Impairment.

concerning the text editor that were returned with the survey. Table III shows a near-uniform evaluation of Features 2, 3 and 4 as the most helpful. Table IV shows the three features as being the most preferred for implementation.

Respondents predicted a high usage of these features. They had the choice of predicting use as follows:

1. Every Session
2. Every Few Sessions
3. Occasional Sessions
4. Rare Sessions
5. Never

In Group 1, for example, 54 percent of the 26 respondents predicted use in the highest 3 categories for Feature 2. Seventy-six percent of 25 respondents chose these same categories for Feature 3. Eighty-one percent of 27 respondents did the same for Feature 4. We can conclude that the users see these features as desirable. With the monitoring results, it is easy to determine the accuracy of these predictions.

The surveys were completed by the end of July 1977. On September 7, the expanded text editor and documentation were released to Group 1. The actual usage of the features versus time periods is shown in Figure 1. The scale of the horizontal axis is proportionate for the first three work-weeks. The last four monitoring periods do not maintain the scale. However, results for these periods are approximately the same. The editor was released to Group 2 on October 25, 1977 at the same time the introductory lectures were given. The use of the features by Group 2 is shown in Figure 2. The horizontal axis in this figure is linear. The follow-up visit to Group 2 was on Day 11.

The two figures are very similar. Initially high usage rates decreased quickly. Figure 1 differs by showing a large "spike" for Feature 6. Figure 2 shows apparent recovery

TABLE IV.—Average Rank of Each Proposed Feature on a Preference-For-Implementation Ordering Obtained from Pre-implementation Survey, Indexed by Group

Group	Features				
	i	2	3	4	5
Entire Lab	3.30	2.75	2.50	2.15	3.45
Group 1	3.35	2.92	2.36	2.30	3.80
Group 2	3.14	2.29	1.33	3.00	4.29

NOTE—A rank of 1 indicates highest preference; 5 indicates lowest.

TABLE V.—Comparison of Uses Per Session and Usage Rate Categories of New Features and Eight Selected Old Features, *a-h*, with Values Established by Base and Test Period Monitoring, Indexed by Group

Features	Group 1		Group 2		
	Uses Per Session	Usage Rate Category	Features	Uses Per Session	Usage Rate Category
a	2.807	1	a	1.036	1
g	.841	1-2	g	.458	2
d	.178	2-3	d	.343	2
e	.115	3	e	.111	3
h	.082	3	h	.060	3-4
c	.075	3	b	.017	4
2	.030	4	2	.009	4-5
4	.026	4	c	.006	5
6	.025	4	6	.004	5
b	.019	4	4	.001	5
f	.008	5	f	.000	5
3	.005	5	3	.000	5

NOTE—Values for older features established during base period, except for feature *h* in Group 1 which was from test period. Values for new features established by use in Time Periods 6 to 11 for Group 1, and Time Periods 6 to 10 for Group 2. Features *a-h* represent a typical assortment of commands, e.g., the substitute and change commands, and features of commands, e.g., end-of-line marker and changing all matched strings.

from low values for Features 2 and 6 late in the period monitored, but a decrease on the last day. Also, Figure 1 has generally higher usage rates throughout than Figure 2. The data, therefore, indicates that the special attention given to Group 2 did not improve acceptance.

To get a better insight into the acceptance of the enhancements, their usage rates can be compared to that of older features obtained during the base period. This is done in Table V through a comparison with eight commands and features whose usage are representative of the distribution of rates for all features. In order to obtain fair values for the new features, usage rates are taken from the last half of the time periods monitored—Time Periods 6 through 11 for Group 1, and Time Periods 5 through 10 for Group 2. To aid in comparison with the categories on the pre-implementation survey, a "usage rate category" obtained from a scale shown in Table VI is used to characterize the information. Two numbers show rates between categories. In either evaluation, the usage of the new features is towards the low end of the usage rates.

Individualized rates were also low. Counting individual logon identifiers as distinguishing individual users, there were 45 users in Group 1 throughout the experiment. During Time Periods 6 to 11, 16 percent used Features 2 and 4 more frequently than at the "Rare" level. Eighteen percent used

TABLE VI.—A Numerical Interpretation of Usage Rate Categories Used on Preimplementation Survey

No.	Usage Rate Categories	Sessions Per Use	Uses Per Session
1.	Every Session	1 or less	1 or more
2.	Every Few Sessions	2-4	.5-.25
3.	Occasional	8-16	.125-.062
4.	Rare	32-64	.031-.016
5.	Never	128 or more	.008 or less

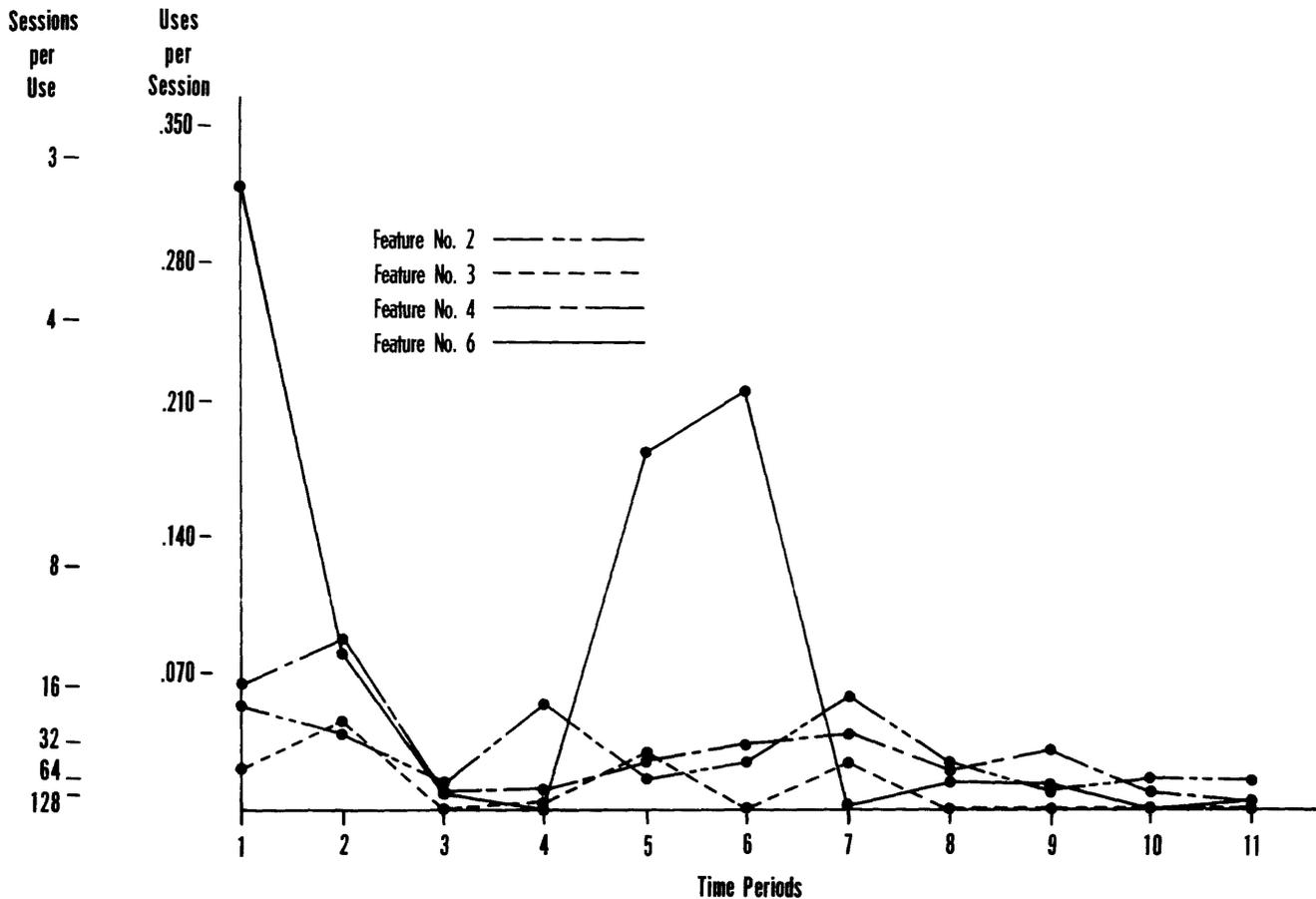


Figure 1—Usage of new features with respect to time by Group 1.

Feature 6 at that level. No one used Feature 3 at that level. This places the usage of Features 2, 3, and 4 well below those predicted on the pre-implementation survey.

Table VII gives another view of individual usage rates through the number of users accounting for percentage of use. It shows how many users in Group 1 accounted for 50 percent, 75 percent, 90 percent and 100 percent of all uses of the four new features during Time Periods 6 to 11. Since the total number of users on the system at the time was 45, no feature was used by a majority of users. Feature 3 is obviously lightly used (it was deleted shortly after this data was obtained). Feature 6 has the fewest users accounting for the most usage. Over the entire test period, two users accounted for 88 percent of its use. The spike seen in Figure 1 is a side effect of this since the heavy users concentrated their use of the feature on several isolated days. All three of the features that were used show a sharp decline in numbers of users as lower percentages of use are considered. Hence, a few users account for most of the usage.

For comparison, individual contribution to the use of four established features and the sum of all commands during the Base Period were analyzed for Group 1. This produced the same type of distribution seen in Table VII. Among other facts, this indicates that there are a few very heavy users of the text editor. Interestingly, these users do not substantially

contribute to use of the enhancements. One is among the 75 percent usage group of Feature 2, and another is among the 75 percent usage group of Feature 6. None fit in any 50 percent group.

Analysis of the individuals who used the new features showed that there is little overlap among users of the new features. Only two users account for usage in more than one feature at the 50 percent or 75 percent levels. In terms of usage per session, there are 12 different users in Group 1 at the Occasional level or higher for at least one feature. The same type of variation is seen among users of the four older features analyzed. Hence, both the style of use of the editor and the changes in style resulting from the enhancements differ widely.

TABLE VII.—Number of Users Accounting for Percentage of Use of New Features During Time Periods 6 to 11 for Group 1

Percentage	Features			
	2	3	4	6
100	13	1	10	11
90	6	1	6	6
75	4	1	4	2
50	2	1	3	1

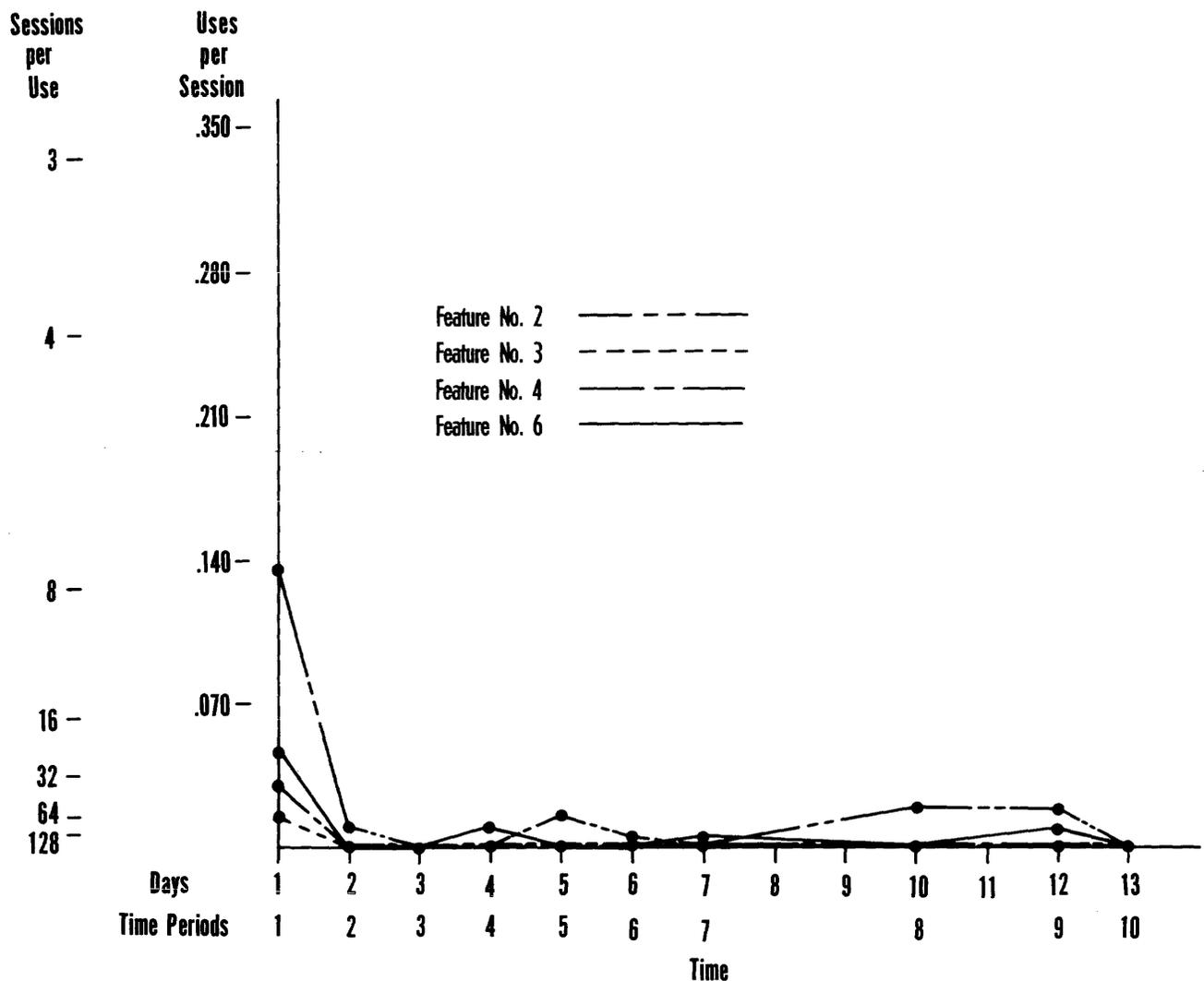


Figure 2—Usage of new features with respect to time by Group 2.

The follow-up survey was distributed a few months after obtaining these values. At that time, I was inclined to think that the features were rejected. However, the response given in Table VIII indicated a different conclusion. Users who were in the two experimental groups at the time of the first survey clearly rated Features 2 and 4 more helpful than they did on the pre-implementation survey. Ignoring the votes of those who stated they were unfamiliar with the features, the popularity of these two enhancements is even higher. Further, users found Features 2 and 4 as helpful as the majority of the older features considered. Users therefore had accepted at least two of the four features.

In order to see if the acceptance of the features translates into new levels of usage, the results of the Final Period of monitoring of Group 1 can be analyzed. In some ways, rates are changed. Usage rates of Features 2 and 4 are .057 uses per session and .089 uses per session, respectively. This is nearly double and more than triple the earlier rates. Feature 2 is now near the top of the Occasional-To-Rare category,

while Feature 4 is well into the Occasional category. At the same time, Feature 6 is now in the Never category with a usage rate of .005 uses per session. As opposed to increases in usage rates, the number of users of the features has not substantially increased. There were 52 on the text editor during the period. Features 2, 4, and 6 were used by 14, 11, and 8 users, respectively. Table IX shows that the identity of the users of Features 2 and 4 during the earlier and later periods is often the same. The level of use is also similar. In particular, few users drop to the Never category. This shows that users have some loyalty to tools they have begun to use, but it also shows that the increases in overall usage rates of Features 2 and 4 are not matched by other increases.

DISCUSSION AND CONCLUSION

To summarize the preceding section, it can be seen that the study produced several distinct results. Users responded positively to a set of features presented to them through a

TABLE VIII.—Average Evaluation of Helpfulness of New Features and Selected Old Features, with Number Responding as Not-At-All Familiar with Each Feature, and Average Evaluation of Only Those Familiar, Indexed by Groups and Obtained from Follow-Up Survey

Group 1 (22 respondents)			
Feature	Average Helpfulness	Number of Respondents Not-At-All Familiar	Average Helpfulness of Those Familiar
a	1.00	0	1.00
e	1.24	0	1.24
h	1.33	0	1.33
d	1.55	0	1.55
4	1.95	2	1.79
g	2.00	2	1.79
c	2.00	1	1.89
2	2.14	5	1.75
f	2.35	5	1.80
6	2.43	2	2.26
3	3.00	5	2.67
b	3.38	5	3.10

Group 2 (7 respondents)			
Feature	Average Helpfulness	Number of Respondents Not-At-All Familiar	Average Helpfulness of Those Familiar
a	1.00	0	1.00
h	1.00	0	1.00
c	1.43	0	1.43
g	1.71	1	1.50
2	1.71	1	1.50
d	1.71	0	1.71
f	2.00	1	1.83
4	2.00	0	2.00
e	2.82	0	2.82
6	2.86	1	2.66
3	3.20	3	2.67
b	3.33	4	2.50

NOTE—All respondents were members of laboratory at the time of the Pre-implementation Survey. Scale for point values show: 1=Very Helpful; 2=Helpful; 3=Somewhat Helpful; 4=No Help; 5=An Impairment.

survey. A high usage rate immediately after release quickly decreases to well below predicted levels and below the levels of use of most other features. The group receiving special instruction shows no higher usage rates. The effect of the enhancements on individual usage styles varies considerably and the style of the heaviest users of the system are less affected. A degree of loyalty to tools is evidenced by users. In addition, users' subjective evaluation shows greater acceptance of the features. These features were clearly desired by users. Assuming that these results are an example of the general reaction to software enhancements, it is important to understand their underlying causes and to consider the consequences the pattern has on computing.

Peter Naur⁵ and many others have commented on the difficulty of new programming languages gaining acceptance. A guess at one reason for this is that people and organizations, through their personal commitment of resources, have developed strong usage habits. I believe that similar habits, those dictating the way in which people use

a particular language, are what produces the pattern that this experiment has uncovered.

There are a number of points that lead me to favor this explanation. Probably the most striking evidence is the peculiar distribution of users over both old and new features, and their loyalty to the features they have adopted. This points to the existence of well developed styles on a level that would interfere with acceptance of new features. The lack of agreement between reality and subjective observation can be explained by the tendency for habitual activities to be performed without conscious recognition. This lack of agreement is found in the users' evaluation of features versus their actual use. It also was evident in discussions with Group 2 where several users indicated they were using new tools more heavily than what concurrent monitoring showed. Finally, habit can be seen in the rejection of the enhancements by the heaviest users. These users are likely to depend most on habit to achieve their high usage rates.

Whether or not habits are the reason for the results arrived at, the consequences of the results must be considered. These consequences must differ with the perspective brought to the problem. As a vendor, the goal is to satisfy user demands. Since subjective evaluation can be high without correspondingly high usage rates, enhancements can be successful. As a user, or better yet, as the party who bears

TABLE IX.—Number of Users in Group 1 of Features 2 and 4 Compared by Usage Rate Categories: Time Periods 6-11 vs. Final Time Period

		Usage Rate Category in Final Time Period									
		5	4-5	4	3-4	3	2-3	2	1-2	1	N.A.
Usage Rate Category in Time Periods 6-11	1										
	1-2							1	1		
	2					1				1	
	2-3										
	3										1
	3-4					1	1				
	4										2
	4-5										2
	5	X		1		1					

a. Users of Feature 2

		Usage Rate Category in Final Time Period									
		5	4-5	4	3-4	3	2-3	2	1-2	1	N.A.
Usage Rate Category in Time Periods 6-11	1										
	1-2										
	2							1	1		
	2-3	1							1		1
	3										1
	3-4					1	1				
	4					1					
	4-5	1									
	5	X			1	2	1		1		

b. Users of Feature 4

NOTE—N.A. (Not Applicable) columns and rows identify users not recorded as using the text editor during given time period. X indicates that no values are given for constant nonusers.

the expense of the enhancements, these same arguments can be presented. However, it is more likely that people will want to see some resulting usage. The experiment has shown that scattered usage is possible. This may be enough to justify the enhancements. There are, however, cases where as general an adoption as possible is desired; for example, when the enhancement is a software tool that will improve productivity. In conclusion, let's consider what to do in these situations.

The problem is to get as many users as possible to use a new feature. New arrivals to a system do not, of course, have established programming habits. Securing their acceptance of the desired feature should be easier. An environment that approaches that of a new arrival to a system appears achievable by revolutionary change in a programming environment. For example, a group at the Columbus laboratory reported success at changing their programming style when they changed from one programming language to another, from FORTRAN to "C."⁶ At the same time, people already using the target language had a more difficult time adjusting.

In addition, there may be ways to reach users in evolutionary situations. The experiment shows that the habits of established users cannot generally be changed by simply presenting information. Even the more elaborate process employed with Group 2 failed. However, from my discussions with that group, one possible tactic emerged. Several people liked the features but did not remember them when they used the editor. Similarly, many people who had been in the two groups at the time of release noted on their surveys that they were Not-At-All familiar with some of them. Several commented that they would use one feature or another now that they knew it was available. For these users, the problem is not learning but remembering; typical aids to memory such as opcode cards, lists of features at work sites, and suggestions for use generated by systems are promising tools.

One final possibility is change via software project management. People respond to guidance. Weinberg and Schulman⁷ have experimental evidence that programmers will dramatically alter the style and structure of the programs they

produce in response to guidance. Hence, in achieving general adoption of a software enhancement, there is the possibility of managing its adoption.

In summary, this paper has reported on a case where desired software enhancements were not generally adopted when introduced into an existing programming environment. I maintain that this indicates the presence of well established habits and suggest the need for more vigorous acceptance methods to ensure that new software tools produce changes in programming style. I conclude that it must be the task of software engineers not only to discover enhancements to software systems but also to discover better methods of ensuring their use.

ACKNOWLEDGMENTS

I wish to thank all my colleagues at Bell Laboratories in Columbus, Ohio. Special thanks are due to Art Kamlet who suggested the experiment; Tom Davison, off whom many of the ideas involved were "bounced;" and, Dale DeJager who helped with the UNIX system.

REFERENCES

1. Lientz, B. P., E. B. Swanson and G. E. Tompkins, "Characteristics of Application Software Maintenance," *Comm. ACM*, Vol. 21, No. 6, June 1978, pp. 466-471.
2. Ritchie, Dennis M., and Ken Thompson, "The UNIX Time-Sharing System," *Comm. ACM*, Vol. 17, No. 7, July 1974, pp. 365-375.
3. *The Bell System Technical Journal*, Vol. 57, No. 6, Part 2, July-August 1978, is devoted to the UNIX Time-Sharing System. See especially "Document Preparation" by B. W. Kernighan, M. E. Lesk, and J. F. Ossanna, Jr., pp. 2115-2135.
4. Deutsch, L. P., and B. S. Lampson, "An Online Editor," *Comm. ACM*, Vol. 10, No. 12, December 1967, pp. 793-799, 803.
5. Naur, Peter, "Programming Languages, Natural Languages, and Mathematics," *Comm. ACM*, Vol. 18, No. 12, December 1975, pp. 676-683.
6. Kernighan, Brian W., and Dennis M. Ritchie, *The C Programming Language*, Englewood Cliffs, Prentice-Hall, Inc., 1978.
7. Weinberg, Gerald M., and Edward L. Schulman, "Goals and Performance in Computer Programming," *Human Factors*, Vol. 16, No. 1, 1974, pp. 70-77.

Experiences in building and using compiler validation systems

by PAUL OLIVER

E.D.S. Federal Corporation
Washington, D.C.

INTRODUCTION

Software Validation

For purposes of this discussion, the term "software validation" refers to the process of testing a completed software product in its operational environment. The scope of this paper is further limited in the following ways: the experiences described are not applicable to applications software; the validation systems discussed must be capable of functioning on a variety of dissimilar hardware and operating systems; the staff performing the validation is not involved in the development or the maintenance of the products being tested; and the result of a validation could impact the eligibility of the product for procurement. This environment imposes unusual and stringent requirements on the portability of the validation systems and the auditability of the validation.

The methodologies available for software validation are few. Design simulation¹ can and is being used, concurrently with software system design, to evaluate the effects of various system requirements or design alternatives once the system has been defined and modeled. Problem statement languages¹ provide a formal syntax and semantics with which to communicate needs of users to analysts, and are applicable during the system definition and design phases of a software development effort. Neither design simulation nor problem statement languages are applicable to the testing of a completed product. The experiences of the Federal COBOL Compiler Testing Service (FCCTS) suggest that functional testing²⁻⁴ is the most thorough technique presently available for testing a completed software product.

Functional Testing

Functional testing is the process of executing a series of generally independent tests designed to exercise the various functional features of a software product. The examples discussed in this paper will address the testing of compilers, but the findings and conclusions are applicable to system software in general. The exclusion of application software is not meant to suggest that such software is not amenable to functional testing, but is, rather, a reflection of the general

inadequacy of specifications for applications software. It is difficult to test for the valid implementation of functions when the functions themselves are ill-defined. This is of course a subjective observation, and it is also true that the specifications of a compiler's architecture (generally, the standard) often leave much to be desired.

Functional testing can be used to test characteristics of software such as performance and integrity, but is most commonly used for specifications testing. Thorough functional testing requires a complete test plan, systematic control of the testing effort, and objective measurements of test coverage. Functional testing is applicable during implementation (verification), during evaluation (validation), and during the maintenance phase of a software product's life-cycle.

This applicability to all phases of test activities is a principal advantage of functional testing. Furthermore, the thoroughness of testing is measurable in terms of number of functions tested, and the revision and evaluation of test specifications is relatively simple. Also, functional testing offers a high degree of visibility to a customer, and is apt to be well understood by that customer.

With these advantages come some disadvantages. The one most frequently cited is that it is generally not possible to assure that all possible features or decision points of a software product are in fact tested. With regard to compilers it is certainly true that it is not practical to test all possible combinations of language components and data types. We have not, however, found this to be a serious shortcoming, since it is certainly possible to test all reasonable combinations. What is "reasonable" is admittedly a subjective judgment, but such subjectivity regarding test limits is hardly unique to software testing. A more serious problem, from the FCCTS experience, is that functional testing can only be as good as the specifications being tested. Thus, it is an unfortunate fact that many important features of a compiler cannot be tested because the pertinent language specifications are ambiguous.

THE FCCTS EXPERIENCE

The Federal COBOL Compiler Testing Service has, during the years 1973-1978, performed official validations of over four dozen COBOL and FORTRAN compilers,⁵ in

support of Federal Government procurement regulations. What follows are descriptions of some of the results of these activities, a summary of the resources expended in performing such validations, a description of the technical problems encountered, and suggestions as to what future work remains to be done in this area. It is the intent of this paper to present data on specific experiences which can hopefully lead to useful generalizations regarding functional testing.

Validation of FORTRAN and COBOL Compilers

The importance and utility of higher-level languages is, in this writer's opinion, no longer at issue in data processing. The steady decline in hardware costs, a growing awareness of the importance of programmer productivity and software reliability and maintainability, and improvements in the quality of code produced by compilers have nearly terminated arguments regarding the relative effectiveness of lower- and higher-level languages.

This trend has made the compiler the most visible component of system software. To many programmers, the compiler *is* the system, since it is the tool which is most frequently used in building a piece of software. It is therefore important that the compiler conform to its specification. For some compilers, most notably COBOL, FORTRAN, and PL/I, these specifications (at the functional or architectural level) are embodied in a standard.

The software development manager is faced with a multitude of problems. The productivity of his average programmer ranges from three to nine lines of code per hour, and has been increasing at a rate of only about 3 percent per year.⁶ Furthermore, the quality of his product is not too good, and he may require as much as 75 percent of his resources just to maintain the product once it is developed.⁷ These are not very encouraging figures. Thus, it is paramount that the manager, and his programmers, at least be able to have some faith in the tools of their trade. In particular, it is not unreasonable to expect that a compiler conform to existing standards in its translation of programs written in standardized languages. There are of course limits to how much a standard can do. A language standard is not like most engineering standards. The state of the art in software technology (or the level of maturity of the data processing industry) is not yet at the point where such precision is possible. Furthermore, rightly or wrongly, most standards still allow quite a bit of latitude to the implementor with regard to the meaning of certain language constructs. Nevertheless, a language standard can and should provide a framework for a workable, well-disciplined approach to software development. This is in fact the fundamental contribution of a language standard. It is important to recognize this because one commonly hears dubious claims made on behalf of standards—the most frequent of which is that the presence of a language standard makes program conversion (within the language, e.g., COBOL-COBOL) costs disappear. This is simply not true. Programming practices, imprecise standards, and environmental factors (e.g., operating system differences) all contribute to keeping the cost of conversion

fairly high. An analysis performed by the FCCTS of some 32 COBOL-COBOL conversions reveals, for example, that the conversion cost of one line of COBOL code can range from \$.50 to \$6.00, and this suggests that conversion costs are still high, even for programs written in "standard" COBOL.

To repeat—the principal purpose of a language standard is to provide a disciplined, predictable, efficient framework for software development. To fulfill this goal, a compiler must perform according to the standard. Testing is required in order to determine conformity.

The FORTRAN and COBOL Compiler Validation Systems

The FCCTS has performed functional testing of FORTRAN and COBOL compilers since its inception in 1973. This testing has been done using the COBOL Compiler Validation System (CCVS)⁸ and the FORTRAN Compiler Validation System (FCVS).⁹ A detailed description of these systems can be found in the references; only a brief summary is presented here. The CCVS74 (COBOL 74) consists of nearly 219,000 lines of COBOL code which collectively exhaust the meaningful constructs in the COBOL language. The FCVS 78 (FORTRAN 77) consists of over 62,000 lines of FORTRAN code, and represents a subset of the full standard. The FCCTS has also produced a HYPO-COBOL Validation System and an FCVS for FORTRAN 66, but the size of these projects was so small (12,000 and 38,000 lines of code, respectively) that data derived from them would be misleading. The interested reader is referred to References 9 and 10.

The Validation Systems consist of syntactically correct programs and an executive routine. The executive routine is used to perform certain text editing (e.g., specification of implementor names), for program and test selection (hardware dependent language elements may not be testable), and for the generation of job control language statements appropriate to the host operating system. Furthermore, an audit log of these actions is produced. These functions are necessitated by the environment in which the FCCTS operates—portability of the validation systems and auditability of the test procedures may not be required for in-house testing (although they are not bad features to have).

The Validation Process

The Compiler Validation Systems used by the FCCTS are necessarily generalized products, almost in the sense that an operating system is generalized when it is first delivered to a customer. A system generation process must take place to produce a product which is tailored to the installation in which it is implemented. In the case of the Compiler Validation Systems, this generation process consists of inserting, through the executive routines, implementor names in the source code, generating operating system control statements, and deleting (actually, not generating) tests which the compiler is unable to process at all, e.g., language fea-

tures which the compiler does not implement. This generation process is performed by the organization whose compiler is being validated, with some assistance from the FCCTS.

Validation is performed by the FCCTS staff, and consists of executing the audit routines on site, reviewing the resulting raw data, and producing a Validation Summary Report.⁵

Results

Experiences with COBOL compiler validations were reported by Baird and Cook.¹⁰ It is encouraging to note that many of the anomalies reported in 1974 have disappeared from compilers today. Errors reflecting sloppiness or poor judgement, such as performing syntax correctness checks on comments, no longer occur. Errors in simple constructs such as ADD, SUBTRACT, and COPY statements in COBOL are no longer prevalent. There is little if any justification for such errors, and their disappearance suggests that the availability of an independently-administered validation service has had an impact on compiler developers.

Table I illustrates the downward trend in both the average and the maximum number of errors discovered during a validation. It should be noted that the failure to implement a particular language feature is regarded as an error by the FCCTS. While this type of error is irritating to a programmer who wishes to use the feature it is not as serious as the presence of a feature which is incorrectly implemented, since this latter type of error is deceiving. We are finding no strong patterns in the errors found—i.e., our data does not reveal any meaningful distribution of errors according to language elements. The distribution according to modules is what one might expect—65 percent of all errors were found in the Nucleus, while 29 percent were found in the I/O modules. It is perhaps surprising that only 1 percent of all errors occurred in the remaining modules. We do note that a large number of "errors" are in fact caused by purposeful implementation decisions on the part of compiler developers, particularly in input-output functions, data represen-

tation, and other language areas which are closely related to operating efficiency of the total data processing system. This seems to suggest that implementors, when faced with the decision of implementing a feature correctly or efficiently, will chose efficiency. Given a community of users which traditionally has favored "efficiency" over correctness this is not surprising; but it is not desirable. The misconception that a rigorous standard inhibits efficient implementation is still a prevalent one.

Finally, it should be noted that the reduction in errors found in COBOL compilers took place during a period of transition from COBOL 68 to COBOL 74. This makes the error trend even more significant since it took place in a development rather than a maintenance context, and occurred in the face of an expanded, more complex version of COBOL.

A less encouraging trend has been the number of language features which the FCCTS has been unable to include in its Validation Systems. The FCCTS COBOL Validation Summary Reports list 24 language elements which are fully or partially implementor-defined. Some, such as "computer-name" in the COBOL SOURCE-COMPUTER paragraph are innocuous and acceptable. Others, such as the representation of a valid sign in certain forms of numeric tests and the number of positions carried for intermediate results of arithmetic statements are, in this writer's opinion, indicative of a poorly-defined standard.

A typical result is that one recent COBOL Validation Summary Report listed 15 results "for information only," i.e., applying to tests whose results are not well-defined by the standard. Six of these referred to input-output functions, and four to computation or comparisons. Thus, a COBOL programmer cannot use such common and useful statements as COMPUTE with any degree of certainty as to what the results are to be from one compiler to the other. This is perhaps to be expected in a language which is defined by a committee, whose national standard is set by a second committee, and whose Federal standard is set by a third committee.

The results of FORTRAN compiler testing are not so dramatic. This is due partly to the fact that testing to date has been with respect to the 1966 FORTRAN standard. The features contained in this standard are but a subset of most FORTRAN dialects, and have remained stable over the past 12 years. Furthermore, FORTRAN is a simpler language than COBOL. It is therefore not surprising that the error rate found in FORTRAN compilers has been much lower than that found in COBOL compilers.

Two generalizations come to mind. One is that methodically and independently applied functional testing of compilers has produced meaningful and beneficial results. The other is that functional testing is limited by the quality of specifications.

Resources Required

Table II summarizes workload and resources expenditures data compiled by the FCCTS during the 1974-1978 period.

TABLE I.—Validation Errors Summary

(1) Average number of errors found for COBOL 68 compilers (1973)	18
(2) Maximum number of errors found for COBOL 68 compilers (1973)	50
(3) Minimum number of errors found for COBOL 68 compilers (1973)	0
(4) Average number of errors found for COBOL 74 compilers (1977)	9
(5) Maximum number of errors found for COBOL 74 compilers (1977)	30
(6) Minimum number of errors found for COBOL 74 compilers (1977)	1
(7) Average number of errors found for FORTRAN 66 compilers (1977)	1
(8) Maximum number of errors found for FORTRAN 66 compilers (1977)	2
(9) Minimum number of errors found for FORTRAN 66 compilers (1977)	0

TABLE II.—Validation Projects Data

	CCVS 74	HCVS	FCVS 66	FCVS 78
(1) Lines of delivered source code	218,560	12,000	37,839	63,188
(2) Calendar months expended	49	13	11	20
(3) Man-hours expended	11,200	781	1,959	4,600
(4) Lines produced per man-hour	19.5	15.4	19.3	13.7
(5) Time distribution (percent) Design/Code/Test	35/42/23	36/48/16	40/37/26	50/30/20

The CCVS74 elapsed time data is somewhat misleading. The 49 months given for the development of CCVS74 were really devoted to development of three distinct versions of the product, each of which was really a distinct, usable product in itself. Thus, periods of relative inactivity are included in this timespan. The reason for this approach is that development of the CCVS had to be coordinated with development of COBOL compilers, and these were developed and released incrementally. The first releases of COBOL 74 compiler implemented subsets of the language, and therefore the audit routines tested only these subsets. It is a peculiarity of compiler testing that development of the test tool must rely on the compiler; it fact many vendors used portions of the CCVS74 in testing their compilers during development. The 49 months cited were therefore caused by a continuing mutual "piggy-backing" of compiler-audit routines development.

Productivity was quite high on all the validation systems development projects. This is due to a variety of reasons—the product specifications were to some extent already in existence (the standard represents specifications for audit routines as much as for compilers), the FCCTS staff was recruited for the specific purpose of software testing from the inception of the FCCTS in 1973, and the experience factor of the staff is quite high. Also, although a validation system may be quite large, as with the CCVS74, it really consists of a large number of relatively small, relatively independent modules. Productivity in developing this type of system will naturally tend to be well above average due to the lowered incidence of interpersonal communication required.

The project time distributions show a greater percentage of the development effort is devoted to coding than one would generally expect. Again, this is attributable to the nature of the product being developed. The major phases of

TABLE III.—Validation Time Requirements

Personnel Type/Phase	Average Hours Required
Professional/Preparation	4.0
Professional/Site Visit	20.0
Professional/Report Production	18.0
Clerical/Report Production	7.5

design, coding, and testing are not as distinct in this type of product as they are generally. The modularity and module independence of the audit routines make it convenient to combine much of the documentation and unit testing activities with the code production phase in a way which makes it impossible to separate the time associated with each.

Table III summarizes the FCCTS resources expended for validations. We believe that a good portion of the professional time presently spent in the report production could be shifted toward clerical time, and are taking steps to do this.

A few observations regarding the makeup of the staff might be of interest. The FCCTS was formed in 1973⁵ and performs some of its functions in support of Federal procurement regulations. Previous experience with compiler validation existed,⁸ but much remained to be learned. Recruitment and staffing therefore leaned toward experienced, highly competent personnel, and the positions established were fairly senior. This was wise at the time, but it is not the optimal staffing pattern once some experience has been gained, nor is it optimal outside the Federal procurement support environment. Development of validation systems requires a high level of skill, ingenuity and maturity during the design stages of the product, and for the development of certain support functions such as the executive routines and code generators; but the coding and testing processes bear such a resemblance to a production line that high-caliber personnel is simply neither required, nor desirable. Rather, such a project should be staffed along the lines of a Chief Programmer Team, with some obvious modification (it perhaps should be called a "chief designer team"). The project manager should also act as the chief designer, with the primary responsibility for developing the test specifications. A software specialist should be responsible for developing support software such as the executive system. The coding task can be relegated to junior programmers. Ideally, a single individual should be responsible for computer-based testing. It has been found very useful to have programmers visually review each other's code—the simple structure of audit routines makes this practice very cost effective. Documentation can be voluminous; the validation system itself should be self-documenting, but such documents as user guides should be produced by a literate technical writer. Finally, a "librarian" is both feasible and useful.

Technical Problems

It was indicated earlier that lack of completeness can be a problem in functional testing. It has not been a problem in compiler validation. The reason is that the potentially combinatorial number of logic paths which should be tested in application software (whose inputs are unpredictable) is not a factor when testing a compiler. Ensuring that all meaningful language constructs are tested may tax the patience of the developer, but not his intellect. The problem is further alleviated by limiting the tests to correct language constructs, i.e., we do not insert erroneous code in the audit

routines. This decision is motivated by the environment in which the FCCTS functions. If we were developing tests for an in-house product we would very likely want to introduce erroneous code in the tests. We would then be faced with the traditional question of what errors to introduce, at what frequency distribution, and by whom should they be introduced.

A problem we *have* encountered is that of properly identifying the building blocks of the audit routines. This is a problem unique to validation of compilers. Since the audit routines must be compiled by the compiler which is being tested it is necessary to identify certain language constructs for which correct compilation is taken as a set of axioms, in the sense that if these features are not correctly implemented there is no point in continuing the validation. An example of this building block approach can be seen from the FORTRAN Compiler Validation System (FORTRAN 77). Some of the assumptions we have made regarding the Subset FORTRAN are:

1. Six (6) character symbolic names and five (5) digit statement labels are permitted.
2. Comment lines do not affect a program in any way.
3. Execution of the unconditional GO TO statement
GO TO s
causes the statement identified by the statement label s to be the next statement executed.
4. Branching to a CONTINUE statement causes the statement following the CONTINUE statement to be the next statement executed.
5. The arithmetic assignment statements
variable = constant
variable = variable
function correctly.
6. The arithmetic IF statement functions correctly:
IF (e) s1, s2, s3
where e is any arithmetic expression of the form
variable + constant
variable - constant
and s1, s2 and s3 are statement labels.
7. The character assignment statements
character variable = character constant
character variable = character variable
function correctly.
8. The logical expression
IF (character relational expression) executable
statement
where the form of the character relational expression is
character variable .EQ. character constant
functions correctly.
9. The following form of the WRITE statement functions correctly:
WRITE (u, f) iolist
where u is a unit specifier, f is a FORMAT identifier, and iolist is an input/output list containing arithmetic or character variables. The format statement contains nH edit descriptors, X edit descriptors, numeric editing descriptors and/or A edit descriptors.

10. In order for the output report to have the correct format, the use of the first character of a formatted record for vertical spacing on a printing device must function correctly.

The two characters which are used in printing the report are:

Character	Vertical Spacing Before Printing
I	To first line of next page
blank	One line

11. The system output device has at least 56 characters per line.
12. An integer datum consists of at least 16 bits of which one bit is a sign bit.
13. A real datum contains at least 16 bits in the mantissa and eight bits in the exponent.
14. A character datum of length 14 is permitted.

These assumptions are necessarily subjective—a different designer might have made different choices. The point is that choices must be made, and that the validity of the choices must be determined prior to full testing.

Because the type of validation systems we are discussing consist of a large number of small independent modules, and because the high-level specifications are a "given" (i.e., the language standard) there is a tendency to bypass the design phase (after having determined the "axioms") and to plunge immediately into the coding phase. The danger with this approach is not that the resulting product will be a faulty one, but rather that it will be an unnecessarily voluminous one. We have found it useful to develop general specifications which define in a broad way a set of tests for a given language module or construct (e.g., arithmetic expressions), and to follow these with a set of detailed specifications which define each test. The general specs allow us to refine the estimates of time and resources required for the project, and to ensure completeness "in-the-large," while the detailed specs provide thorough documentation and enable the actual coding to be done by relatively junior personnel.¹²

Audit routines should be self-checking. The volume of output is far too large to be eyeballed. We have also found it useful to be able to suppress printed outputs during the testing phase of a validation system development.

We have encountered two problems which are unique to an outside testing group which is testing a product on different systems. One is portability of the validation system; the other is auditability of the test process itself. We must be able to execute the audit routines on any compiler-operating system-hardware combination, and we must be able to assure the consistency of our testing procedures. The executive systems allow us to fulfill both the goals. In-house testing would require no more than a good text-editor and a library management function. Additional details of the validation process using the executive routines are to follow.

Although our productivity has been high we believe that it is necessary to improve it. We would like to be able to develop a validation system for a new language, e.g., PL/I, in six months or less, while limiting the project staff to, say,

five people. This is not yet possible. We have taken some steps to increase our productivity. One has been to adopt very stringent and precise development procedures, so that tasks are completely interchangeable among project staff members. These are described in greater detail below. We have also found it useful to use the COBOL Compiler Validation System executive routine in developing the audit routines. This not only ensures consistency but also provides us with a useful and complete log of all significant development tasks. Our major effort at improving productivity has been the development of a pre-processor, or generator, as part of our FORTRAN validation system project.

The generator is used to speed up production of "boiler plate" material and repetitive code which changes only with regard to statement labels. Specifically, the generator is used to:

1. Produce standard comments which appear, unchanged, before each grouping of tests or before each individual test.
2. Produce code to be executed when a test is passed, failed, or deleted.
3. Produce "standard" variables declarations.
4. Generate statement labels which render the common code segments unique.

While the generator has been useful, it should be noted that it is *not* used to generate the test code itself. We have given this possibility serious thought for over three years, and have concluded that while test code generation is feasible, it is not productive. Such an attempt would be useful if either of these conditions were true:

1. We needed to produce multiple but differing versions of a validation system for a given product.
2. The generator could be used to produce validation system for different languages or products.

The former condition does not exist, while the latter is not feasible except for the most simple-minded languages. We are therefore faced with the uneasy feeling that we are reaching a practical limit on productivity with regard to validation systems development.

We do expect to achieve some improvements through a refinement of our programming practices. Our standard development rules presently specify the following:

- Symbolic name conventions
- Assumption or axioms
- Statement label conventions, particularly in distinguishing among test, pass, fail, delete, verify, and I/O code
- Convention for external unit identifiers
- File naming and contents rules
- Composition and organization of routines and tests
- Phases in the development cycle
- Use of the generator

We hope to find additional ways of expanding and using the

generator, but are skeptical as to the chances of making quantum improvements in our productivity.

The Executive Routines

The executive routine is used to control the generation of a specific Validation System, to control the generation of job control language statements, and to perform updates. System generation consists of program selection, identification of options to be used and the validation environment, and the control of report outputs.

Job control statements are generated for a given operating system through a "higher-level language" which is used to indicate the need for accounting statements (e.g., JOB or RUN), for the invocation of processors such as compilers and collectors, and for the initiation of execution.

Finally, a simple set of text-editing statements is available for replacing, adding, or deleting source statements. As was indicated earlier, the need for an executive system is predicated on the degree to which the validation system itself must be portable.

FUTURE WORK AND UNRESOLVED PROBLEMS

Much of our thinking during the past few years has been directed at determining the feasibility of automating the test generation process. Our motivation has been to increase the rate at which we can produce compiler validation systems. Others have been motivated by the perception of additional problems:¹³

1. Lack of a formal construction method
2. No reference to measures of test effectiveness
3. Manual preparation resulting in uneven product quality.
4. High costs

We believe, based on our experiences, that rigorous procedures, augmented by some support tools, adequately address the first three problems. We furthermore do *not* believe that cost is an inhibiting factor. The cost of CCVS74 has been approximately \$400,000 (including maintenance), or somewhat under \$2/line of code. This is an almost insignificant cost compared to development costs in general. Naturally, the fact that we act as a central service facility whose testing product is applied to many compilers reduces the relative cost—\$400,000 spent to test 40 compilers is a less disturbing figure than \$400,000 spent to test one compiler.

Our experience to date suggests to us that attempts at using a formal notation for the specification of programming languages as the means by which a language's syntax and semantics may be described, and by which tests may be generated, are ill-conceived. This approach is certainly feasible. But, as anyone familiar with syntax-directed compilation knows, this approach will not significantly reduce the time required to produce compiler validation systems. The

reason is simple enough—it is no easier to specify a test using a formal notation than it is to specify it in the programming language directly. Furthermore, the differences among languages are great enough to preclude using a single test specification to generate multiple tests, each in a different language. We have not closed the door on this approach, but we have become very skeptical of its merits.

A more serious problem, in our opinion, is that of the poor quality of language specifications. It is in fact somewhat fruitless to concern oneself with the completeness of a test vehicle when so many significant language features are ill-defined. This is a solvable problem, but its solution will require a "consumer movement," whereby users become more critical of what is handed to them by standards committees. As of now, inadequate language specification represent the most significant limitation on function validation. The PL/I Standard is a major step forward in addressing this problem, despite the criticism it has received.

A related problem has to do with the stability of the product being tested. The FCCTS receives numerous complaints that we often insist on revalidating a compiler because of new releases of the compiler or of its operating system. Our rejoinder is that we will drop our requirement for revalidation when offerors of compilers stop putting out a new release every six months. The industry as a whole has standards of software quality control that would embarrass most other professions.

Finally, more attention needs to be given to the development of specifications languages, so that the process of testing the specifications, producing a product from these specifications, and producing a testing system with which to test the product could be a largely automated, coordinated

effort. We are aware of the work that is under way in this regard, but results have been very meager.

REFERENCES

1. Smith, R. L., "Validation and Verification Study," *Structured Programming Series*, Vol XV, IBM Federal Systems Center, Gaithersburg, MD, 20760, May 22, 1975.
2. Scherr, A. L., "Developing and Testing a Large Programming System—OS/360 Time Sharing Option," Computer Program Test Methods Symposium, June 21-23, 1972, *Program Test Methods*, Prentice-Hall Inc., pp. 165-180.
3. Elmendorf, W. R., "Disciplined Software Testing," *Debugging Techniques in Large Systems*, Prentice-Hall Inc., pp. 137-139.
4. Freeman, P., "Functional Program Testing and Machine Aids," *Program Test Methods*, Prentice-Hall Inc., pp. 41-47.
5. Oliver, P., "A Program for Software Quality Control," *Proc. NCC 74*, AFIPS Press, Montvale, NJ, pp. 411-416.
6. Johnson, James R., "A Working Measure of Productivity," *Datamation*, February, 1977, pp. 106-112.
7. Elshoff, J. L., "An Analysis of Some Commercial PL/I Programs," *IEEE Transactions on Software Engineering*, June 1976, pp. 113-120.
8. Baird, G. N., "The DOD COBOL Compiler Validation System," *Proc. FJCC 1972*, AFIPS Press, Montvale, NJ, pp. 819-827.
9. Hoyt, P. M., "The Navy FORTRAN Compiler Validation System," *Proc. NCC 77*, AFIPS Press, Montvale, NJ, pp. 529-537.
10. Cook, M. M. et al., "COBOL for Mini Computer-HYPO COBOL", Presented at the 1976 National Computer Conference, New York, N.Y.
11. Baird, G. N. and Margaret M. Cook, "Experiences in COBOL Compiler Validation," *Proc. NCC 74*, AFIPS Press, Montvale, NJ, pp. 417-421.
12. Cook, M. M., et. al., "Programming Procedures Manual," FCVS78 Version 2.0, Federal COBOL Compiler Testing Service, Department of the Navy, Washington, D.C. 20376, June 29, 1978.
13. Berning, Paul T. et. al., "Automated Compiler Test Case Generation," RADC-TR-78-30, Naval Air Development Center, Griffiss AFB, NY 13441, February 1978.

Automatic program synthesis via synthesis of loop-free segments*

by JOE W. DURAN

The University of Texas at Dallas
Richardson, Texas

INTRODUCTION

Work on theorem-proving-based automatic program synthesis (see Lee, *et al.*,⁷ for example) has been neglected lately. In their 1971 paper, Manna and Waldinger⁸ covered one of the main reasons why—the difficulty of synthesizing program loops within the current state of the art of automatic theorem-proving. However, there is a great deal of continuing work in theorem-proving, and it is important that motivating work in related areas such as program synthesis not be neglected.

Most theorem-proving-based synthesis systems attempt to constructively prove theorems of the form, “for all input values satisfying the desired input predicate I , there exists a corresponding set of values of output variables which satisfy the desired result predicate, R .” The resulting program (if any) is correct with respect to the input and output predicates. In general, inductive proofs are needed to synthesize the loops within a program. This causes difficulty, and in fact previous work has not been notably successful with loops. To quote Manna and Waldinger, “. . . these systems have been fairly limited; for example, either they have been completely unable to produce programs with loops or they have introduced loops by underhanded methods.” Since most interesting programs contain loops in some form, this is a crucial problem for successful synthesis. We will limit our discussion to iterative loops.

Manna and Waldinger outlined an approach to automatic synthesis which attacked the problem of iterative loops, and discussed the use of various forms of induction for reducing synthesis to the proving of loop-free (i.e., induction-free) lemmas. However, they were dissatisfied with the large number of equivalent induction principles required by their approach. In fact, only a single rule is needed. Loop invariants can be used to provide a single, general rule for expressing the synthesis of loops in terms of loop-free lemmas.

We consider only input and output predicate pairs, I and R , which make sense in the context of program synthesis; i.e., they must be decidable and “define” a nontrivial recursive function, in the sense that $\exists F\{I(v)\rightarrow R(v,F(v))\}$ is true. If a recursive function is “defined” by an I,R pair,

then there exists a program for computing the function. We show that such a program can be synthesized via synthesis of only *loop-free* program segments. The necessary formulations are constructed by using loop invariants associated with **while—do** loop forms to state these loop-free lemmas, which can take the standard forms expected by many synthesis systems.

DECOMPOSITION INTO LOOP-FREE SEGMENTS

If a program P , correct with respect to an input predicate I and a result predicate R , can be found, then it computes a function F_P , such that

$$I(v_{in})\rightarrow R(v_{in},F_P(v_{in})) \quad (1)$$

where v_{in} is the vector of values of the program’s input variables. For a large class of recursive functions, there exist natural decompositions of the form $F_n(F_{n-1}(\dots F_1(v)\dots))\equiv F(v)$, $n>1$, where F is the original function and v is its argument. This form of the decomposition still holds for any function for the trivial cases where one of the F_i is F and the rest are identity functions. Let us consider some decomposition of our function F_P , such that

$$F_n(F_{n-1}(\dots (v_{in})\dots))\equiv F_P(v_{in}) \quad (2)$$

Let P_j be a program for computing F_j . A program computing F_P may be constructed by concatenating the programs P_1, \dots, P_n . Though the decomposition (2) may be trivial, for most programs there will exist non-trivial ones.** Now consider the set of all P_j satisfying the decidable input and result predicated defined as follows:

$$I_1\equiv I,$$

$$I_j(v)\stackrel{def}{=} \{\exists v_{in}[I(v_{in})\wedge v=F_{j-1}(\dots F_1(v_{in})\dots)]\},$$

$$\text{for } 2\leq j\leq n,$$

$$\text{and } R_j(v,v')\stackrel{def}{=} \{v'=F_j(v)\}.$$

Any concatenation $P_1P_2\dots P_n$ of P_j s satisfying these

* This work was partially supported by NSF Grant No. GJ-36424.

** It will be recognized that this decomposition is included in the very well known concept of stepwise refinement in program construction (see Mills⁹).

predicates will produce a program which computes F_p , provided that each P_j is constructed to terminate. The program so produced will thus be correct with respect to I and R .

We proceed to show that, under this scheme, there will always be a satisfactory set $\{P_1, P_2, \dots, P_n\}$ such that each P_j contains at most one loop. Since any computable function can be computed by a program with a single loop, each F_j can be computed with at most a single loop. Each such single-loop program has an associated loop invariant which is sufficient for a Floyd-type correctness proof. This follows from the loop invariant existence proofs in References 2 and 4. Therefore there will always be a satisfactory set of P_j s such that each one contains at most one loop, which can always be of the form "*initialization* while B do *loop body*." F_p can thus be computed by a concatenation of single loop program segments which have associated loop invariants. We will use this fact to complete the analysis leading to our characterization of general program synthesis in terms of loop-free lemmas.

Manna and Waldinger⁸ considered synthesis in terms of the proof of first-order theorems of the form

$$\forall v_{in}[I(v_{in}) \rightarrow (\exists v_{out})R(v_{in}, v_{out})], \quad (3)$$

where v_{in} is the vector of input variable values, and v_{out} the vector of output variable values. Let us now consider any arbitrary single loop program with while—do loop form, correct with respect to input and result predicates I and R . Such a program has two segments—the initialization code and the loop body code—and has a loop control predicate, B , and a loop invariant, Q , such that $Q \wedge \sim B \rightarrow R$. We can synthesize the two segments separately and then use the two segments to assemble the program into its while—do structure.

Initialization code must produce output satisfying the loop invariant whenever the input predicate is true. Thus the loop invariant acts as the result predicate of (3) and initialization can be stated in terms of the induction free lemma

$$\forall v_{in}\{I(v_{in}) \rightarrow \exists v_0[Q(v_0, v_0)]\}, \quad (4)$$

where v is the vector of all program variables, with v_0 representing its initial value.

The input conditions for loop body code entry are that the loop control predicate, B , is true and that the loop invariant, Q , is true. The desired output of the loop body code must satisfy the loop invariant. Further, since we wish to synthesize programs which terminate, we require that the input to the loop body code, which satisfies B , must be transformed into output which is closer to not satisfying B . The predicate ensuring this, denoted by $P(v, v')$, is essentially Dijkstra's variant function.³ (Here v is the program variable vector before loop execution and v' is the updated value after execution.) The complete result predicate is thus $(Q \wedge P)$ and the loop body code lemma is stated as

$$\forall v_0 \forall v \{B(v) \wedge Q(v_0, v) \rightarrow \exists v' [Q(v_0, v') \wedge P(v, v')]\}. \quad (5)$$

SYNTHESIS

Existing first order synthesis systems can be applied to the above lemmas after they are generated for a given problem. For instance, Lee, *et al.*,⁷ have presented a resolution based algorithm for producing loop-free programs which is proved to generate correct programs. Their system constructively proves theorems of the form

$$\forall v_{in} \forall v_{out} \{R(v_{in}, v_{out}) \rightarrow ANS(v_{out})\}. \quad (6)$$

The predicate $ANS(v_{out})$ is used to record the unification substitutions which take place for v_{out} during a resolution proof. To use their methods, lemmas 4 and 5 are converted to the form of (6). Our desired result predicate for initialization is Q , and for loop body code is $Q \wedge P$. Although Lee, *et al.*, make no explicit mention of input conditions, clearly these can be included in their form. Thus (6) can be used for initialization by letting $R = \{B \wedge Q(v_0, v) \wedge Q(v_0, v') \wedge P\}$, giving us

$$\forall v_{in} \forall v_0 \{I(v_{in}) \wedge Q(v_0, v_0) \rightarrow ANS(v_0)\} \quad (7)$$

and

$$\forall v_0 \forall v \forall v' \{B(v) \wedge Q(v_0, v) \wedge Q(v_0, v') \wedge P(v, v') \rightarrow ANS(v')\}. \quad (8)$$

If the assumptions (or axioms) that $I(v_{in})$, $B(v)$, and $Q(v_0, v)$ are true are added, (7) and (8) can be shown to be logically equivalent to (4) and (5). Thus we have logical equivalence where I , B , and Q are true, which is the only real case of interest.

The following example illustrates our approach in conjunction with resolution based loop-free synthesis as in Reference 7.

Example 1: Factorial Function

$$I = \{N \geq 0\} \quad R = \{z = f(N)\}$$

f is the factorial function, defined by $f(0) = 1$ and $f(x+1) = (x+1) * f(x)$. We choose $Q(z, k) = \{z = f(k)\}$, $\sim B(k, N) = \{k = N\}$, and $P(k, k') = \{k < k'\}$, so that $Q \wedge \sim B \rightarrow R$. (The non-active elements in v_0 and v are suppressed in the above argument lists.)

- *Initialization code*—From (7), the theorem to be proved is $\forall N \forall z \forall k \{I(N) \wedge Q(z, k) \rightarrow ANS(z, k)\}$, or $\forall N \forall z \forall k \{N \geq 0 \wedge z = f(k) \rightarrow ANS(z, k)\}$.

From $f(0) = 1$, we have $Q(1, 0)$. Expressing the theorem in clause form appropriate for performing resolution, we have

$$(a) \sim I(N) \vee \sim Q(z, k) \vee ANS(z, k).$$

The axioms specifying I and Q , as required for resolution,

are

- (b) $I(N)$ (since we are interested only in cases where I is true) and
- (c) $Q(1,0)$ (by definition of factorial).

Resolving clauses (a) and (b) yields

- (d) $\sim Q(z,k) \vee ANS(z,k)$.

Resolving clauses (c) and (d) yields

- (e) $ANS(1,0)$.

The unification substitutions, carried in ANS , are $1 \rightarrow z$ and $0 \rightarrow k$. From this, initialization code is $z := 1; k := 0$.

- *Loop body code*—From (8) the loop code generation theorem may be stated as the clause

- (a) $\sim B(k,N) \vee \sim Q(z,k) \vee \sim Q(z',k') \vee \sim P(k,k') \vee ANS(z',k')$.

From the definition of factorial, we have the axiom

- (b) $\sim Q(z,k) \vee Q(z*(k+1),k+1)$.

From the hypothesis that control is still in the loop, since that is our condition of interest, we have

- (c) $B(k,N)$, which is the condition, “ B is true”, and
- (d) $Q(z,k)$, which means, “the loop invariant holds.”

The axioms needed to specify the progress requirement predicate, $P \equiv \{k < k'\}$, are expressed by the clauses

- (e) $\sim P(0,L) \vee P(M,M+L)$ and
- (f) $P(0,1)$.

A string of resolutions leading to isolation of $ANS(z',k')$ is as follows:

- (g) $Q(z*(k+1),k+1)$, from (b) and (d)
- (h) $\sim B(k,N) \vee \sim Q(z,k) \vee \sim P(k,k+1) \vee ANS(z*(k+1),k+1)$, from (a) and (g)
- (i) $\sim Q(z,k) \vee \sim P(k,k+1) \vee ANS(z*(k+1),k+1)$, from (c) and (h)
- (j) $\sim P(k,k+1) \vee ANS(z*(k+1),k+1)$, from (d) and (i)
- (k) $\sim P(0,1) \vee ANS(z*(k+1),k+1)$, from (e) and (j)
- (l) $ANS(z*(k+1),k+1)$ from (f) and (k)

Thus the loop body code, derived from the substitutions for ANS arguments leading to (l), is $z := z*(k+1); k := k+1$.

Example 1 illustrates a difficulty with resolution-based methods—the fact that rather oblique axiomatization is necessary to define Q and P .

It is desirable to have an automatic (or semiautomatic) synthesis system which parallels good programming methodology. Thus we might wish to carry out the necessary theorem proving by the use of “natural deduction” rather than resolution. An interactive approach, allowing human intervention, is the best immediate hope for a practical synthesis system. Natural deduction systems, by definition, retain theorems and intermediate steps in a form near to that which humans usually use. The deductions are kept reasonably close to the rules of inference usually used by synthesis. Additionally, such systems have much of the necessary axiomatization built into their deductions, definitions, and rewrite rules. This greatly simplifies setting up a synthesis problem, compared to resolution. Of course, natural deduction systems are usually incomplete, but this is not expected to be a practical handicap. Bledsoe and Bruell¹ have presented a system, called *PROVER*, which combines natural-deduction-like theorem proving with a capability for man-machine interaction. *PROVER* has been modified for use in a practical correctness proving system (Good, London and Bledsoe⁶). Much of this adaptation should be useful also for synthesis proofs.

The lemma schemata (4) and (5) are already in natural form. Example 2 shows the form applied to the integer multiplication problem and illustrates a natural deduction style proof, after the methods of *PROVER*.

Example 2: Integer Multiplication

$$\begin{aligned} I &\equiv \{y \geq 0 \wedge z \geq 0\} & R &\equiv \{x = y * z\} \\ Q &\equiv \{x + c * d = y * z\} & \sim B &\equiv \{c = 0\} \end{aligned}$$

- The initialization theorem is

$$(i) \forall y \forall z \{y \geq 0 \wedge z \geq 0 \rightarrow \exists x \exists c \exists d [x + c * d = y * z]\}$$

The quantifiers are removed by treating universally quantified variables as constants, identified by the subscript o , as in y_o . Existentially quantified variables are left as is. Thus we have

$$(ii) y_o \geq 0 \wedge z_o \geq 0 \rightarrow x + c * d = y_o * z_o.$$

The o subscript indicates that no value substitutions can be made. Since y_o and z_o are general constants, any substitution for x, c, d which satisfies (ii) will satisfy (i). Thus, for our purposes, (i) and (ii) are logically equivalent. In the *PROVER* system, an hypothesis of the form in (ii) is removed and added to the axiom list, leaving the basic theorem to be proved as

$$(iii) x + c * d = y_o * z_o.$$

A key problem for synthesis proofs is existential inference, as necessary to satisfy (iii). A natural deduction system will certainly require heuristic methods for guessing these inferences, which can then be checked for validity within its deductive framework. For instance, (iii) can be solved by term matching to get $x=0, c=y_o$, and $d=z_o$ as

the desired substitutions, which is easily shown to be valid by current formula manipulation and simplification systems.

- The loop body code theorem is

$$\forall y \forall z \forall c \forall d \{c \neq 0 \wedge x + c * d = y * z \rightarrow \exists x' \exists c' \exists d' [x' + c' * d' = y * z \wedge c' < c]\}.$$

Eliminating quantifiers yields

$$(i) [c_o \neq 0 \wedge x_o + c_o * d_o = y_o * z_o] \rightarrow [x' + c' * d' = y_o * z_o \wedge c' < c_o].$$

The splitting heuristics of PROVER yield the subgoals

- (ii) $[c_o \neq 0 \wedge x_o + c_o * d_o = y_o * z_o] \rightarrow x' + c' * d' = y_o * z_o$ and
- (iii) $[c_o \neq 0 \wedge x_o + c_o * d_o = y_o * z_o] \rightarrow c' < c_o$,

which must be proved under the restriction that (ii) and (iii) must be satisfied by the same set of substitutions.

The simplest solution to subgoal (iii) is $c' = c_o - 1$. Trying this in (ii) yields $c_o \neq 0 \wedge x_o + c_o * d_o = y_o * z_o \rightarrow x' + c_o * d' - d' = y_o * z_o$. Matching expressions across " \rightarrow " gives $x_o + c_o * d_o = x' + c_o * d' - d'$ as a necessary and sufficient condition for validity of (ii). A solution to this is $x' = x + d_o$ and $d' = d_o$.

Example 2 indicates that heuristics similar to those presented for second-order synthesis in Reference 5 will be helpful to a PROVER-like system. We are currently constructing a PROVER-based synthesis system to use such heuristics.

CONCLUSION

A program P can be synthesized as a set of programs $\{P_1, \dots, P_n\}$ so that the concatenation P_1, \dots, P_n computes F_P . We have demonstrated that a satisfactory set $\{P_1, \dots, P_n\}$ exists such that each member has at most a single loop of the **while—do** form. Therefore, the synthesis of each P_j requiring a loop can be stated in terms of proving lemmas of the form of (4) and (5) (or (7) and (8)) for each P_j . Any loop-free P_j s can be directly synthesized from the form of (3) without the need for inductive proofs. If nested loops are desired, the synthesis can be performed hierarchically

to expand operations considered primitive during higher-level synthesis.

We have not addressed the problem of mechanically (or otherwise) arriving at a good decomposition and finding loop invariants and variant functions. It has, however, become part of the general lore of programming methodology that it is desirable for a programmer to perform such decompositions before writing programs, and Dijkstra and others have recommended that the loop invariant and variant function be discovered by a programmer before he actually writes a program loop. Thus a successful synthesis system which operates from a specification of the decomposition and associated loop invariants and variant functions is a very desirable goal.

ACKNOWLEDGMENT

The author would like to thank T. W. Pratt and R. T. Yeh for their support and criticism during the early part of this research.

REFERENCES

1. Bledsoe, W. W., and P. Bruell, "A Man-Machine Theorem Proving System," *Artificial Intelligence*, Vol. 5, 1974, pp. 51-72.
2. DeBakker, J. W., and L. G. L. T. Meertens, "On the Completeness of the Inductive Assertion Method," *Journal of Computer and System Sciences*, Vol. 11, December 1975, pp. 323-357.
3. Dijkstra, E. W., "Guarded Commands, Non-Determinacy and a Calculus for the Derivation of Programs," *Proc. International Conf. on Reliable Software*, Los Angeles, 1975.
4. Duran, J. W., "A Study of Loop Invariants and Automatic Program Synthesis," Ph.D. thesis, University of Texas at Austin, August, 1975.
5. Duran, J. W., "Heuristics for Program Synthesis Using Loop Invariants," *Proc. ACM Nat'l. Conf.*, Washington, 1978.
6. Good, D. I., R. L. London and W. W. Bledsoe, "An Interactive Program Verification System," *IEEE Trans. on Software Engineering*, Vol. SE-1, March 1975, pp. 59-67.
7. Lee, R. C. T., C. L. Chang and R. J. Waldinger, "An Improved Program-Synthesizing Algorithm and Its Correctness," *Comm. ACM* Vol. 17, April 1974, pp. 211-217.
8. Manna, Z., and R. J. Waldinger, "Toward Automatic Program Synthesis," *Comm. ACM*, Vol. 14, March 1971, pp. 151-165.
9. Mills, H. D., "The New Math of Computer Programming," *Comm. ACM*, Vol. 18, January 1975, pp. 43-48.

Semantic similarity analysis—A computer-based study of meaning in noun phrases

by LYNN L. PETERSON

*The University of Texas Health Science Center at Dallas
Dallas, Texas*

INTRODUCTION

The purpose of this paper is to present a method by which a computerized data base which contains phrases of natural language information can be created and interrogated in the semantic domain for multiple uses and users. The method takes into account both the individual user's needs and concepts of similarity in meaning of textual material represented by the stored data. The method is applicable to arbitrary bodies of natural language text which occur in the syntactic format of the noun phrase. The method was developed and tested, however, using noun phrases of medical text since its capabilities are directly applicable to needs in that area.

Data base containing medical text

Ways of dealing with a data base which contains natural language information from medical text warrant examination because of the nature of medical records. There are several objectives the medical recording process should fulfill.¹ The first and most important in the present usage is to serve as a self-reminder for the responsible physician. However, in the modern hospital, the medical record has an equally important task—it is the primary channel of communication between many individuals working together as a team. In either case, computer-based files of patient data are generated and maintained first and foremost to improve health care delivery for the benefit of the patient. The same files should also be available for retrieval of that part of the patient information useful and accessible for research and educational purposes.

Integral parts of these files of medical records are non-numeric and best expressed as natural language, e.g. admitting diagnosis, operative procedures, etc. In fact, components of a problem-oriented medical record are normally expressed as noun phrases using large vocabularies.² The non-numerical component of the medical record serves the function of interpretation of the numeric data, as well as the recording of observations and the communication among medical staff.³ As such, this natural language component of the stored data needs to be handled appropriately.

The simple noun phrase is the construct of which the vast

majority of entries in a problem-oriented medical record are composed.⁴ It is therefore appropriate that the simple noun phrase serve as the focus of attention in the consideration of a procedure which would be applicable to a data base containing medical text.

A data base of medical information will characteristically be created and interrogated for multiple uses and users. This requires a procedure for semantic analysis which can deal accurately with the meaning of words and the relationship between words, and can recognize and use the distinction between synonyms and near-synonyms. According to Pratt, in medicine, the distinction between synonyms and near-synonyms is important for the proper interpretation of the medical record, and remains one of the major problems which will have to be resolved to produce effective systems.⁵

Presently we can make things that mean the same thing fall together; we wish to be able to recognize when things mean nearly the same thing, to quantify this semantic nearness in a meaningful and useful way, and to use this knowledge to aid in the retrieval from the data base of semantically-related information.

Individual concept of similarity

A basic premise of this approach is that similarity in meaning between two pieces of medical text is not an absolute notion. On the contrary, this approach recognizes that (a) one user's concept of similarity may differ from that of another user, i.e., things may not mean the same thing to two different people, and (b) an individual user's interests may differ from one retrieval request to another. At one time a user of the data base may wish to retrieve all those records which contain the same operative procedure ignoring all differences in other areas, i.e., similarity sufficient to retrieve requires an exact match on operative procedure and ignores differences in, say, topography. At another time, however, the user may care very much what anatomical area is involved, i.e., similarity of records will require the involvement of an anatomical area close to the one specified. Thus, by applying different weightings, sets of records can be grouped into different "semantic subspaces," stressing or ignoring features based on their perceived importance to the user.

SEMANTIC SIMILARITY ANALYZER

To address the objective of permitting a data base containing noun phrases of text to be created and interrogated for multiple uses and users, a Semantic Similarity Analyzer, SSA, was constructed. Significant features of the system are highlighted.

MEANINGEX-based system

The system called MEANINGEX, developed by Mishel- evich in 1970,⁴ takes the simple noun phrase and performs on it a "semantic analysis." This semantic analysis is composed of (a) a lexical phase in which each input word or term is recognized and then mapped to a standard term and a part of speech, (b) a syntactical phase in which the head term or main noun is identified, the remaining terms are treated as adjectives, and (c) a semantic parse, or "sparse," phase in which a tree is constructed with the head term as root and the structure of the remainder of the tree determined from entries in a "tree directory" for the head term and its modifiers. The sparse phase is generative of semantic information in that entries in the tree directory supply related and relevant terms which are not physically part of the normalized text.

As an example, let us consider the simple noun phrase

MODERATELY SEVERE PNEUMOCOCCAL ARTHRITIS OF THE LEFT KNEE.

The normalized text produced by the lexical and syntactical phases

SEVERE.ACUTE.PNEUMOCOCCUS.LEFT.KNEE ARTHRITIS.

is produced by using a lexicon whose relevant section is shown in Figure 1. The section of a tree directory which shows the relationships among the head term in this phrase and its modifiers is shown in Figure 2, where two types of terms on next node are indicated:

- (a) An inclusive node in which all the terms are used.
- (b) A selector node, indicated by the *, in which only one of the choices is selected.

The tree generated by MEANINGEX from the phrase of normalized text is then shown in Figure 3, where levels of

TERM	STANDARD TERM	PART OF SPEECH
MODERATELY SEVERE	SEVERE	ADJ
SEVERE	SEVERE	ADJ
PNEUMOCOCCUS	PNEUMOCOCCUS	ADJ
PNEUMOCOCCAL	PNEUMOCOCCUS	ADJ
ARTHRITIS	ARTHRITIS	NOUN
ARTH.	ARTHRITIS	NOUN
ACUTE	ACUTE	ADJ

Figure 1—The section of the lexicon dealing with "arthritis" in the Original MEANINGEX.

TREE DIRECTORY TERM	TERMS ON NEXT NODE
ARTHRITIS	JOINT INFLAMMATION
BACTERIAL	*GONOCOCCUS *MENINGOCOCCUS *PNEUMOCOCCUS
DEGREE	*MILD *MODERATE *SEVERE
DURATION	*ACUTE *SUBACUTE *CHRONIC
ETIOLOGY	*INFECTIOUS *OSTEO *RHEUMATOID
INFECTIOUS	*BACTERIAL *VIRAL
INFLAMMATION	DEGREE DURATION ETIOLOGY LOCATION
JOINT	JOINTNAME SIDE
JOINTNAME	*SHOULDER *FINGER *HIP *KNEE *TOE
LOCATION	*POLY *JOINT *CAVITY *ORGAN
SIDE	*LEFT *RIGHT *BOTH

Figure 2—The section of the tree directory dealing with "arthritis" in the Original MEANINGEX. Two types of nodes, depending on the terms on the next node, are indicated: an inclusive node in which all terms are used and a selector node (indicated by *) in which only one of the choices is selected.

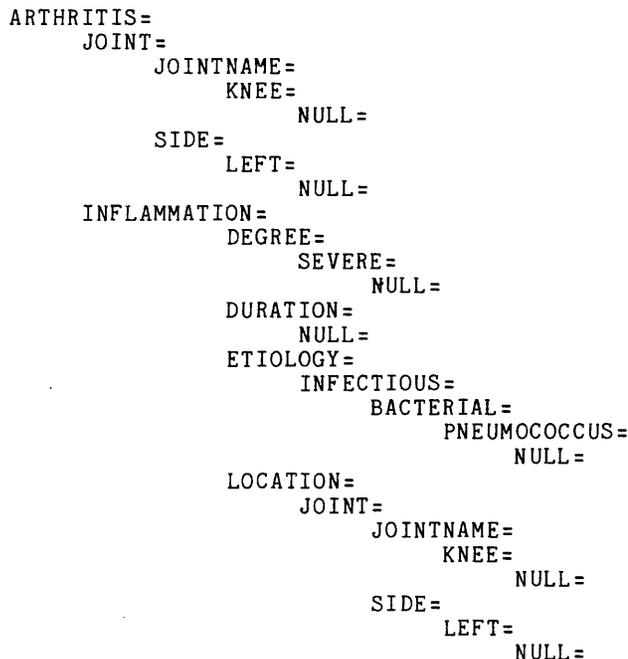


Figure 3—The sparse tree in the Original MEANINGEX for the phrase MODERATELY SEVERE PNEUMOCOCCAL ARTIHRITIS OF THE LEFT KNEE.

indentation indicate levels of hierarchy. The tree constitutes the primary output from the MEANINGEX semantic analyzer. From the tree, a linear string of the nodes of the tree can be constructed by reading the tree bottom up. The string of nodes, which can then function as descriptors or keys for information retrieval, is as follows from the current example:

```
/NULL/LEFT/SIDE/NULL/KNEE/JOINTNAME/
JOINT/LOCATION/NULL/PNEUMOCOCCUS/
BACTERIAL/INFECTIOUS/ETIOLOGY/NULL/
DURATION/NULL/SEVERE/DEGREE/
INFLAMMATION/NULL/LEFT/SIDE/NULL/
KNEE/JOINTNAME/JOINT/ARTHRITIS/
```

Obviously, then, the information which determines the resultant semantic parse of a piece of medical text via MEANINGEX is found primarily in the tree directory. Additionally, however, a synonyms list is made available to MEANINGEX showing terms to be considered to have a meaning equivalent to that of a term already recognized, and an implications list is included to indicate how the presence of one or more recognized terms, Boolean fashion, implies the presence of another term. The generative properties of the semantic parse come about as a result of the semantic information imbedded in the tree directory and the synonyms and implications lists.

The original MEANINGEX system was implemented in 1970 in assembly language for the CDC 3300 computer at The Johns Hopkins University. Published material⁶⁻⁹ indicates that the system was applied to semantic analysis of medical records and to bibliographic indexing, but the approach to semantic analysis was sufficiently general to admit other applications. Retrieval aspects were not addressed by the original MEANINGEX *per se*, but the linear string of descriptors generated by MEANINGEX was in the proper format for processing by the generalized data management system in use at Johns Hopkins.

In 1976 a new MEANINGEX system was implemented on the DECsystem-10 computer by Steve Bush at The University of Texas Health Science Center at Dallas (UTHSCD). The concept and functional description are the same as those of the original MEANINGEX but the new computer system is interactive and has added several features which increase its usefulness in the study of generated meaning. It was implemented primarily in the FORTRAN language, using in addition some well-documented macro-string manipulation routines, adding a measure of transportability to the MEANINGEX system.

The new MEANINGEX, called UTHSCD MEANINGEX, preserves the basic concepts of the original MEANINGEX system. The domain of semantic interest, i.e., the set of terms used in the natural language phrases to be considered together with the relationships among the terms, is specified by the input lexicon, implications list, and synonyms list. However, in the new system, this information can be entered from the keyboard or from a file. Furthermore, additions can be made to the implications and synonyms list from the keyboard, adding a dynamic aspect to the management of this information. Figure 4 shows the list

```
@COMMENT
PORTION OF LEXICON DEALING WITH ARTHRITIS

@LEXICON
ARTHRITIS == JOINT INFLAMMATION
BACTERIAL ==* GONOCOCCUS MENINGOCOCCUS PNEUMOCOCCUS
DEGREE ==* MILD MODERATE SEVERE
DURATION ==* ACUTE SUBACUTE CHRONIC
ETIOLOGY ==* INFECTIOUS OSTEO RHEUMATOID
INFECTIOUS ==* BACTERIAL VIRAL
INFLAMMATION == DEGREE DURATION ETIOLOGY LOCATION
JOINT == JOINTNAME SIDE
JOINTNAME ==* SHOULDER FINGER HIP KNEE TOE POLY
LOCATION ==* JOINT CAVITY ORGAN
SIDE ==* LEFT RIGHT BOTH
```

Figure 4—The section of a lexicon dealing with "arthritis" in the form for input to UTHSCD MEANINGEX. The presence of the * indicates a selector node as in Figure 2.

of statements used to convey to UTHSCD MEANINGEX the same specifications shown in Figure 2 for the "arthritis" example of the original MEANINGEX.

The SSA utilizes the semantic analysis procedures of UTHSCD MEANINGEX and thus also makes use of the input scheme shown in Figure 4. A tree directory can be output by the SSA in a graphic form shown in Figure 5 which is then available for study, elaboration and correction. The tree of knowledge represents a composite view of the terms and hierarchical relationships active in the domain under consideration. The display itself opens for study the details of construction of such a hierarchical directory for a specific domain.

Context weightings/user profiles

Fundamental to the understanding of semantic similarity is the notion that the meaning of a phrase may differ from individual to individual or even within an individual as times or purposes differ. In order to permit differing values of importance or interest to be attached to terms in the vocabulary in a domain of semantic interest, the facility of context weightings is used.

The SSA allows a user of the system to associate with each term in the tree directory a numerical value representing its weight. In some situations, weights are actually assigned by the user to all or only some of the terms in the domain with weightings for the remaining terms derived by inheritance from parent nodes to which weights were assigned. All weights are determined by the user input and the placement of terms relative to others in the tree representing the domain, i.e., the context, so the weights are called "context weights." The entire set of weights then, in a sense,

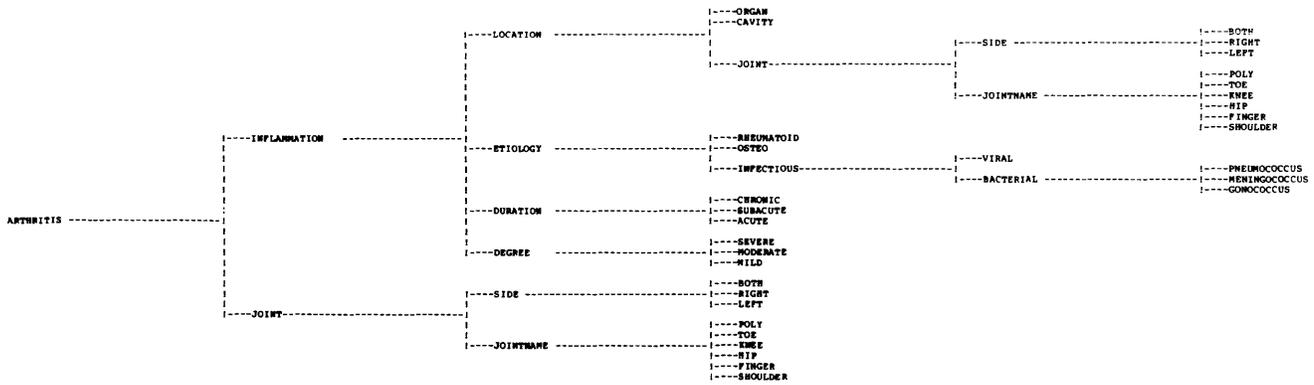


Figure 5—The section of the tree directory dealing with "arthritis" utilizing display techniques of the SSA.

represents the user's notions of importance of or interest in the terms. For this reason, the set of weights together with a short narrative description of the set is called a "user profile."

Numerous methods of specifying weights have been considered during the development of the SSA and two general methods are in use now. The methods fall into the general categories of explicit weighting, or direct generation of the weighting set, and implicit weighting, or generation of the set of weights from general information presented to the system by the user.

Explicit weighting

The most straight-forward method of generating a weightings set is to have the user explicitly provide the numerical weights to be used. In this method of generating a set of context weightings, the user will actually traverse the tree directory for the entire domain of semantic interest by means of an interactive display. Each node is displayed in turn, showing its already established context weight and the names of each of its subnodes. The user is then asked to enter a weight for each of the subnodes. Thereafter the context weight for each node is the product of the context weight of its parent node and the user-entered weight. By this means, context weights are generated for all nodes in the tree directory. As an example, consider a situation in which a user is interested in those childhood illnesses usually accompanied by a rash. Within this context, then, we will wish to carry out the retrieval of those strings similar in meaning to a reference string. To create this context via a weightings set, a dialog such as that shown in Figure 6 would ensue. The result of this dialog is the establishment of a set of context weights and a narrative description of the set, represented thereafter by the user profile "RASH."

However, some drawbacks to explicit weighting do exist. The method does require the user to interact with the SSA via a terminal for a significant period of time before even the simplest retrieval can be tried, if he wishes to use his own user profile. However, it is true that the unit weight option can always be selected, that is, all weights can be

```

*****NOW TO SET UP CONTEXT WEIGHTINGS
TYPE (D) TO DO (CONSTRUCT) A SET OF WEIGHTS
(U) TO USE UNIT WTS ON ALL TERMS
(G) TO GET (USE) AN EXISTING WEIGHTING SET:D
WANT FULL DISPLAY PLUS RECAP?:NO
**NODE:CHILDHOOD ILLNESS           WT= 1
  TYPE
  DEGREE
  SYMPTOMS
ENTER WT FOR EACH TERM
  TYPE           :2
  DEGREE         :1
  SYMPTOMS       :1
**NODE:SYMPTOMS                     WT= 1
  SWOLLEN PAROTID GLANDS
  FEVER
  COUGH
  RASH
  VOMITING
  CONJUNCTIVITIS
  LYMPHADENOPATHY
ENTER WT FOR EACH TERM
  SWOLLEN PAROTID GLANDS :0
  FEVER                  :0
  COUGH                  :0
  RASH                   :10
  VOMITING               :0
  CONJUNCTIVITIS        :0
  LYMPHADENOPATHY       :0
**NODE:DEGREE                       WT= 1
  MILD
  MODERATE
  SEVERE
ENTER WT FOR EACH TERM
  MILD                 :1
  MODERATE             :2
  SEVERE               :3
**NODE:TYPE                          WT= 2
  MEASLES
  MUMPS
  CHICKEN POX
  RUBELLA
  SCARLET FEVER
  WHOOPING COUGH
ENTER WT FOR EACH TERM
  MEASLES              :10
  MUMPS                :0
  CHICKEN POX         :10
  RUBELLA              :10
  SCARLET FEVER       :0
  WHOOPING COUGH     :0
STORE THIS WT SET UNDER USER PROFILES?:YES
TYPE USER PROFILE: RASH
SHOULD THIS REPLACE AN EXISTING PROFILE:NO
  
```

Figure 6—Dialog ensuing as the SSA is used to create a weightings set via an explicit weighting procedure. User responses are underlined.

assigned the value 1. Also, certain user profiles are provided in the SSA system for illustration. Secondly, this method requires the user to "pick numbers out of the air" to assign as weights, to actually enter numeric values, a task which may be unpleasant to some users. So the explicit weighting method has not been felt to be entirely successful in providing a way for a user to generate a user profile.

Implicit weighting via profile-capturing

The difficulties encountered in explicit weighting have led to a second method which involves an implicit determination of the numerical weights and the terms to which the non-unit weights should be assigned. Methods of implicit determination of weights have been utilized in a variety of theoretical and practical settings, and have been referred to by the general title "policy capturing." An admittedly sexist yet easily understood illustration of "policy-capturing" can be seen in the fable of the king who, once upon a time, decided to select a harem larger than King Solomon's. . . .

So the word went out, and soon thousands of young girls were arriving from the various provinces to seek the king's approval.

Early one morning the king began his selection process. As each girl filed by, he looked her over carefully and then expressed his judgment.

"Excellent!" he would say. "This one is very pleasing to my eye." Or perhaps he would hum and haw with indecision. Many times he would show his disapproval in no uncertain terms. "Never!" he would say. "Pass on! Pass on!"

In each instance, the Court Recorder attempted to quantify the king's degree of approval by checking the appropriate level on a nine-point scale which had been devised especially for the occasion by the Chief of the Royal Psychometricians.

By suppertime the king had considered some 300 girls. His eyes and his imagination were beginning to tire.

"Most High First Counselor," he said, "you've been watching me all day, and by now you should know my likes and my dislikes. I've decided to leave the selection of my harem in your hands. But take care! If your choices do not please me, it will be your head!"

After the king retired, the Most High First Counselor summoned the Chief of the Royal Psychometricians. "I'm passing the job on to you," he said. "If you fail to please the king, your head will roll along with mine."

The Chief of the Royal Psychometricians called his staff together and explained the situation.

"We must not fail," he said, "or it will be all of our heads."

"How shall we proceed?" asked one of the young staff members who was fresh out of the Royal Academy.

"Well," responded the Chief, "we know how the king rated the first 300 girls. Right?"

"Right!"

"And we can see everything the king saw when he looked at the girls. Right?"

"Right!"

"Then all we have to do is to uncover the characteristics considered by the king and determine how he weighted them in his judgment. . . ." ¹⁰

The point was to use the knowledge or approximation to the knowledge of how the process was carried out to continue to carry out the process. Since the goal of utilizing a policy-capturing technique in the SSA is the determination of a "user profile," the general technique of determining a profile in this implicit way is called "profile-capturing."

Several profile-capturing methods were tried for use in the SSA. The method currently used makes use of the capabilities of MEANINGEX to process a natural language phrase. Phrases are elicited from a user in response to one or more questions about what the user is or is not interested in. In each of these phrases, the MEANINGEX analysis determines the terms which should be weighted, with the actual weights being assigned by the SSA relative to a user-selected maximum weight. A sample profile-capturing dialog is shown in Figure 7.

It appears that in practice the profile-capturing method of weight determination may not provide enough ability to discriminate between sets of records which are intended to fall in different "semantic subspaces." It may be in practice that some combination of profile-capturing as a first cut and explicit weighting for finer discrimination will become the weighting method of choice.

Similarity measures

The determination of the semantic similarity of a pair of natural language strings in the SSA basically involves the application of a type of function called a "similarity measure" to a pair of vectors representing the semantic characteristics associated with each of the natural language strings. The semantic characteristics relevant in any application of a similarity measure to a pair of strings are the terms or nodes of the smallest tree generated by the MEANINGEX semantic analyzer which still includes the meanings contained in both phrases. Since MEANINGEX is generative of semantic information, the full context surrounding the meaning of both phrases is included in the set of semantic

```

*****NOW TO SET UP CONTEXT WEIGHTINGS
H[HLP],U[UNIT],C[CHNG],D[DO],S[STOR],G[GET],P[PRO],R[RTN]:P
TO DESCRIBE YOURSELF AS A USER OF THE SSA,
  TYPE PHRASE DESCRIBING WHAT TERMS
  YOU ARE MOST INTERESTED IN
*I'M INTERESTED IN THE RESPIRATORY SYSTEM
TERMS SELECTED FOR WEIGHTING ARE:
  RESPIRATORY          OK?(YES,NO,SEE
DETAILS):SEE DETAILS
TO DESCRIBE YOURSELF AS A USER OF THE SSA,
  TYPE PHRASE DESCRIBING WHAT TERMS
  YOU ARE MOST INTERESTED IN
*RESPIRATORY SYSTEM
Tree =
ORGAN SYSTEM
  OS2
    RESPIRATORY
      *UNKNOWN
TERMS SELECTED FOR WEIGHTING ARE:
  RESPIRATORY          OK?(YES,NO,SEE
DETAILS):YES
ENTER MAXIMUM WEIGHT:
5
ANYTHING ELSE TO TAKE INTO ACCOUNT?(YES,NO):YES
  TYPE PHRASE WHICH DESCRIBES WHAT TERMS
  YOU'RE ALSO INTERESTED IN
*INFLAMMATION OR NECROSIS
TERMS SELECTED FOR WEIGHTING ARE:
  NECROSIS
  INFLAMMATION          OK?:YES
ANYTHING ELSE TO TAKE INTO ACCOUNT?(YES,NO):NO
ANYTHING YOU DON'T WANT TO HEAR ABOUT?(YES,NO):YES
TYPE PHRASE DESCRIBING WHAT TERMS
  YOU DON'T WANT TO HEAR ABOUT
*DON'T CARE ABOUT PARASITIC INVOLVEMENT
TERMS SELECTED FOR WEIGHTING ARE:
  PARASITIC          OK?:YES
ANYTHING YOU DON'T WANT TO HEAR ABOUT?(YES,NO):NO
WANT TO DO MORE WITH WEIGHTS?:NO

```

Figure 7—Sample profile-capturing dialog which results in a weightings set in which the term "respiratory" and all terms below it in the tree (all its subnodes and sub-subnodes) are assigned a maximum weight—in this case, 5. Secondly the terms "necrosis" and "inflammation" and all terms below them in the tree are assigned a weight of approximately one-half the maximum value—in this case, 2. Finally, the term "parasitic" and all terms below it are assigned a weight of 0, to decrease in importance any phrase which contains a component of meaning in this area. Weighting of all terms in the tree directory not affected by these phrases remains at 1.

characteristics used to construct the vectors. The elements of the vectors forming a pair are then 0 or 1 depending on the presence or absence of each semantic characteristic as part of the generated tree for the phrase it represents. Certain terms are considered to be "structural" nodes while most are "content" nodes, carrying meaning; an element of the vector will always be 0 if the corresponding term is a structural node. It is on vectors constructed in this way that the similarity measures operate.

The class of similarity measures in use can be characterized as correlation measures, as indicators of the interdependence of the semantic characteristics of the two strings.¹¹ We do not claim or require that these functions measure the similarity of the two strings in an absolute sense but serve as indicators of the association in meaning between the two strings.

The specific similarity measures currently in use in the SSA are derived from some of those most extensively used in bibliographic retrieval according to Salton.^{12,13} In some cases, a weighting scheme has been imposed on a familiar similarity measure to yield a measure more appropriate to the needs of an individual user or a specific type of retrieval request. The form of several of the similarity measures is shown in Figure 8.

For each retrieval request, a determination is made of

similarity between an input string and the set of stored strings using a similarity measure chosen from the set of built-in similarity measures. A stored string is then retrieved if the calculated similarity index is sufficiently high, that is, exceeds a threshold value which is specified by the user based on the selected function and the degree of interdependence required.

Some of these functions have proven more useful than others in similarity determination based on subjective notions of similarity. As utilization of these functions proceeds, it is anticipated that additions and changes will be made to the set of similarity measures built into the SSA, working toward enhancing their utility. However, it is not anticipated that a selection of the "best" function would ever be made. The use of any one function in retrieval or comparison studies obviously depends on what the user wishes to consider as the criteria for similarity determination. Hence it is the philosophy of the SSA to present the features of functions which accomplish similarity determinations based on a variety of criteria and let the user select the one or ones which best meet the current needs.

Structure of the semantic similarity analyzer

The structure of the SSA containing the features just described is shown in the diagram in Figure 9. The SSA System code and Reference Manual are available on request.

Function #1

$$\sum wt(i) * v1(i) * v2(i)$$

Function #2

$$\left(\sum v1(i)*v2(i) + \sum \bar{v1}(i)*\bar{v2}(i) \right) / N$$

where $\bar{v1}(j) = 1 - v1(j)$ is the complement of $v1(j)$ and N is the number of elements in the vector.

Function #3

$$\frac{\sum wt(i)*v1(i)*v2(i)}{\sum wt(i)*v1(i) + \sum wt(i)*v2(i) - \sum wt(i)*v1(i)*v2(i)}$$

Figure 8—Several similarity measures used in the SSA. In each case, V1 and V2 are binary vectors representing the semantic characteristics associated with each of the natural language strings, WT is the vector of weights, and the summation is over all elements of the vector. Function #1 is a weighted vector product representing the sum of the weights of the characteristics occurring in both natural language strings. Function #2 is the unweighted function which counts the number of matching properties divided by the total number of such characteristics. Function #3 is a weighted normalized measure.

which to build a data base. Sample retrievals from this data base demonstrate the differing retrievals resulting from differing specifications of conditions such as user profiles. Given the reference string

ACUTE INFLAMMATION OF THE LUNG

the SSA was asked to find and type out all strings in its data base similar to the reference string, in the sense defined by the conditions set at the time retrieval was initiated. The results showed that the SSA runs efficiently on the DEC-system-10, processing the storage of strings in this domain in from 0.8 to 1.0 CPU seconds and handling a retrieval request from this data base yielding three to seven strings in 2.0 to 2.5 CPU seconds. For this reference string, the set of strings retrieved differed as the choice of retrieval function changed, as was to be expected. In particular, for a function which focused on the occurrence of matching concepts in the pair of phrases, the pattern of retrieval was

all equally good matches

LUNG; FOCAL GRANULOMATOUS
PNEUMONIA
LUNG; MODERATE SUPPURATIVE
PNEUMONIA
LUNG; MILD BRONCHOPNEUMONIA
LUNG; INTERSTITIAL PNEUMONIA,
MODERATE

For functions which penalize the similarity index values for mismatches, the pattern changes to show:

best match

LUNG; INTERSTITIAL PNEUMONIA,
MODERATE

next best matches

LUNG; FOCAL GRANULOMATOUS
PNEUMONIA
LUNG; MILD BRONCHOPNEUMONIA

also good matches

LUNG; MODERATE SUPPURATIVE
BRONCHITIS
LUNG; PULMONARY EDEMA

Secondly, a consideration of a single function as user profiles change shows the way in which the weighting system operates in carrying out weighted retrieval. Utilizing a weighted vector product measure, Function #1 in Figure 8, and a weighting system in which all weights are one results in the retrieval of the strings

best match

LUNG; ACUTE EDEMA AND
HEMORRHAGE WITH BACTERIAL INVASION

next best matches

LUNG; FOCAL GRANULOMATOUS
PNEUMONIA
LUNG; MODERATE SUPPURATIVE
PNEUMONIA
LUNG; MILD BRONCHOPNEUMONIA
LUNG; INTERSTITIAL PNEUMONIA,
MODERATE

However, using a weighting profile which assigns a weight of five to "bacterial etiologic agent," a weight of two to "inflammation," a weight of zero to "non-bacterial etiologic agent" results in the retrieval of the strings just shown with the addition of the following strings as additional good matches:

TRACHEA; MILD SUPPURATIVE TRACHEITIS
TRACHEA; NONSUPPURATIVE
TRACHEITIS, MODERATE

representing inflammations of another part of the respiratory system.

What is retrieved in a request to find all strings similar to a given string is dependent on what is important to a user of the system and how the user wishes the retrieval function to operate. A user-responsive, functionally directed retrieval technique as part of a system which admits natural language input can therefore be applicable to a real situation.

Hence, in the Semantic Similarity Analyzer, it appears we have a system which meets the overall objective of permitting a data base containing phrases of natural language information to be created and interrogated in the semantic domain for multiple uses and users. The system is being applied to the Veterinary Pathology Record Retrieval System currently under development at UTHSCD, and the usefulness of the technique in practice will continue to be evaluated in this setting.

REFERENCES

1. Barnett, G. O., R. A. Greenes and J. H. Grossman, "Computer Processing of Medical Text Information," *Methods Inf. Med.*, Vol. 8, No. 1, 1969, pp. 177-182.
2. Weed, L. L., "Medical Records that Guide and Teach," *N. Eng. J. Med.*, Vol. 278, 1968, pp. 593-600, pp. 652-657.
3. Wong, R. L., and P. Gaynon, "An Automated Parsing Routine for Diagnostic Statements of Surgical Pathology Reports," *Methods Inf. Med.*, Vol. 10, 1971, pp. 169-175.
4. Mishelevich, D. J., "A Computer-Based Semantic Analyzer for Noun Phrase Statements with Examples from Medical Text," Ph.D. Diss., The Johns Hopkins University, Baltimore, MD, 1970.
5. Pratt, A. W., M. G. Pacak, M. Epstein and G. Dunham, "Computers and Natural Language," *J. Clin. Comput.*, Vol. 3, 1973, pp. 85-99.
6. Mishelevich, D. J., "MEANINGEX—A Computer-based Semantic Parse Approach to the Analysis of Meaning," *Proc. AFIPS NCC*, Vol. 39, AFIPS Press, Montvale, NJ, 1972, pp. 271-280.
7. Mishelevich, D. J., "Computer-Based Semantic Analyzers," *Int. J. Comput. Inf. Sci.*, Vol. 1, No. 3, 1972, pp. 267-286.
8. Mishelevich, D. J., "Semantic Analysis of Medical Records," *Bio-Med. Comput.*, Vol. 3, 1972, pp. 163-179.
9. Mishelevich, D. J., "The Application of the MEANINGEX Semantic Parse Analysis to Bibliographic Indexing," in *Adv. Cybernetics Sys*, J. Rose (ed.), Gordon and Breach Sci. Pubs., Inc., New York, 1973, pp. 309-317.
10. Christal, R. E., "Selecting a Harem—and Other Applications of the Policy-Capturing Model," *J. Exp. Educ.*, Vol. 36, No. 4, 1968, pp. 35-41.
11. Kendall, H. G., and A. Stuart, *The Advanced Theory of Statistics*, Vol. II, 3rd edition, Hafner Publishing Co., New York, 1973.
12. Salton, G., *Automatic Information Organization and Retrieval*, McGraw-Hill, New York, 1968.
13. Salton, G., "Automatic Text Analysis," *Science* Vol. 168, 1970, pp. 335-343.

Heuristic control of design-directed program transformations

by CHRISTINA L. JETTE

University of Washington
Seattle, Washington

INTRODUCTION

This paper describes a class of semantic source-to-source program transformations called design-directed program transformations (DDPT) for use in a transformational implementation (TI) approach to programming. A methodology is developed for applying such transformations based on symbolic evaluation and experimental computation of programs. A DDPT is a cognitive model of source-to-source transformations; it knows *what* it is trying to accomplish and contains a strategy of *how* to accomplish it. A DDPT is more than a syntactic pattern replacement rule; it is a semantic program transformation which is intuitively closer to one that a good programmer would invoke in transforming his program.

DDPTs are a natural extension to TI systems.^{1,6,7,8,10,12,13} The importance of a TI approach to programming has been made earlier, most notably by Balzer *et. al.*¹ In a TI environment, the user interactively constructs and modifies a program by applying "correctness-preserving" transformations. Viewed as a programming methodology, this approach strives to relieve the user from worrying about the details of the actual implementation, thereby transforming his role to one of "designer" and "optimizer," delegating the role of "implementer" to the TI system.

To date, TI systems provide only syntactic transformations. A syntactic transformation can usually be represented by a production-like pattern replacement rule of the form "LHS-pattern \Rightarrow RHS-replacement-pattern." (LHS is left-hand side, RHS is right-hand side.) The user *selects* a particular rule. The TI system tries to *match* the LHS-pattern with a portion of the program to be transformed (called the *target program*). If successful, the RHS-replacement-pattern is *substituted* for the matched portion of the program.

A DDPT is not so easily described as a syntactic replacement rule because the LHS and RHS patterns are not "one-to-one" with the statements in a programming language. i.e., they contain user-supplied descriptions of what the program fragment is doing such as "update," "put," etc. Further, the RHS-replacement-pattern cannot always be specified *a priori*. It is instead dynamically generated based on the interaction of the specific target program and the specific DDPT. Consequently, a DDPT is defined by 1) an

input pattern whose language is semantically rich and 2) a *procedure* which derives instances of the RHS-replacement-pattern. This pattern instance encodes the underlying strategy of the transformation. In the next section, we give a detailed example of a DDPT definition called BYPASS-LOOP.

What distinguishes the DDPT method from the more syntactic approach is 1) the goal-directedness of a specific DDPT (e.g., "bypass" a looping computation if possible), 2) the nature of the input pattern (e.g., pattern elements contain descriptive annotations such as "update," "put," etc.) and 3) the synthesis of replacement program fragments which when substituted into the program, achieve the overall goal of the transformation (e.g., bypass a loop for special cases).

Considering only syntactic transformations leads to several operational difficulties.¹ They include 1) Size—An enormous number of transformations exist. How can their numbers be reduced? 2) Selection—How are transformations selected/accessed? 3) Control—How are transformations applied, i.e., in what order should they be tried and how are they invoked? In this paper we show how these problems can be alleviated in some instances using DDPTs.

Summary

In the next section, we present several examples of design-directed program transformations, and compare the design-directed paradigm to a more traditional syntax based paradigm. In the third section we relate this work to others and in the fourth section we give our conclusions.

DDPT EXAMPLES AND COMPARISONS

In this section, we define the steps of the transformation process. Based on these steps, and using detailed examples for illustration, we compare the syntax based pattern-replacement rule paradigm with the design-directed paradigm.

The process of manipulating a program can be factored into four steps. They are *selection*, *matching*, *substitution* and *replacement*. A transformation "rule" is selected (either

automatically or by the user), and matched to a program fragment. If the match is successful, the variable portions of the matched part are substituted for the variable portions of the replacement pattern. Finally, the replacement pattern replaces the matched target program fragment. In the next two examples, we show how the DDPT approach extends each of these steps by making them more "computational" in nature.

BYPASSLOOP DDPT

This example illustrates how the DDPT method extends the matching and substitution processes. We first look at the syntactic approach.

Consider the sequence of syntactic transformation rules necessary to transform the program fragment

(a) **while** $i \leq n$ **do** $(b[i] := b[i] + p * (i-1); i := i+1)$

into the program fragment

(b) **if** $p=0$ **then** $i := n+1$
else while $i \leq n$ **do** $(b[i] := b[i] + p * (i-1); i := i+1)$.

This transformation takes advantage of the special case for which the loop does no "significant" processing, i.e., for " $p=0$." Figure 1 defines six transformation rules, T1 through T6, to transform (a) into (b).

In this approach, T1 is applied to the target program (a) by matching the LHS-pattern of T1 to (a). The result of this match is a list of (argument,value) bindings. e.g.,

$\langle A: (b[i] := b[i] + p * (i-1)) \rangle$

Next, the value for A is substituted in the RHS-replacement-pattern. Note that the predicate " $p=0$ " which is bound to R in the RHS-replacement-pattern must somehow be specified by the user. The RHS-replacement-pattern replaces the matched portion, producing a new program fragment to which T2 will be applied, etc. The program fragment (b) above is the result of applying T1 through T6 to (a).

The problems inherent with this method are:

1. The overall goal to "bypass the loop for the special case that the no computation path is taken," is completely obscured by the details of invoking the correct sequence of applicable transformations.
2. Six transformation rules are invoked in this case, but these six are not necessarily the only sequence of syntactic rules which would transform (a) into (b). How do we select the applicable ones, and how do we know which ones are available for selection?
3. The user is completely responsible for the selection of a correct sequence—the system merely carries out each selection by matching the appropriate code segment and accomplishing a straightforward replacement.

4. The user is also responsible for discovering properties of the program which might be derivable automatically by a more intelligent system, such as the predicate " $p=0$," and the action " $i:=n+1$ " in the example above.

In comparison, the design-directed approach has a single DDPT called BYPASSLOOP, whose strategy is to bypass a looping computation for the special cases in which the loop does no "significant" processing.⁵ We call this strategy the *abstract design* of the DDPT. It is abstract in that it does not specify the "details" of the target program to which it applies. It does, however, specify the necessary constraints of the transformation. The abstract design specifies that the target loop has a "no computation" path, but it does not specify the specific action sequences of that loop. This strategy is encoded as the procedure which generates the replacement program fragment for a DDPT.

A DDPT is defined by giving: 1) An *input pattern*, and 2) A *replacement fragment constructor procedure* which constructs an incompletely specified replacement program fragment based on the given target program and the underlying strategy or goal of the transformation. It contains both *concrete*, or known actions and predicates and *abstract* or unspecified actions and predicates. The concrete parts are derived from the matched input pattern; the abstract parts are exactly those portions of the yet-to-be-transformed program which must be synthesized to achieve the overall goal of the transformation.

For BYPASSLOOP, the input pattern is:

do; (transient-updates | non-transient-updates); **od**

where "do; . . . ; od" is a pattern which denotes a looping computation, "(transient-updates | non-transient-updates)" is an alternation pattern which denotes one of two specializations of "update." (A transient variable is one with a non-repetitive value structure, e.g., a non-array. Hence a transient-update is an assignment to a transient variable, e.g., $i:=i+1$.) Note that the input pattern for BPL is considerably more abstract than the patterns T1-T6 in Figure 1.

The BPL replacement fragment constructor procedure may informally be stated as:

1. Let M be the matched portion of the target program and the input pattern. Let TU be the transient variables, and NTU be the non-transient variables. Create a conditional statement such that:
 - (a) The else branch is M
 - (b) The true branch is $TU := F(TU)$,
 - (c) The predicate condition is r
 \Rightarrow "if r then $TU := F(TU)$ else M "
2. Define r to be exactly that abstract predicate which is true for the bypass path to be taken. That is, $P\{M\}Q \Rightarrow \text{Pand } r\{TU := F(TU)\}Q$.
3. Define $TU := F(TU)$ to be those update actions which may be expressed as non-looping computations. For all x in TU , $F(x)$ is the value x would have had on exit from the loop if the loop had executed.

RULES:

T1: $P\{A\}Q \Rightarrow P\{\text{if } R \text{ then } A \text{ else } A\}Q$

T2: $P\{\text{if } R \text{ then } A \text{ else } B; C\}Q \Rightarrow P\{\text{if } R \text{ then } A;C \text{ else } B;C\}Q$

T3: Partial evaluation, i.e., symbolic evaluation of the result of substituting known values for variables, actual parameters for formal parameters in procedures, etc.

T4: $P\{x:=x\}Q \Rightarrow P\{\text{noop}\}Q$

T5: $P\{\text{while } R \text{ do if } S \text{ then } A \text{ else } B\}Q \Rightarrow$
 $P\{\text{if } S \text{ then } (\text{while } R \text{ do } A) \text{ else } (\text{while } R \text{ do } B)\}Q$
 when $S\{A\}S$ and $S\{B\}S$.

T6: Computing linear relationships and values of variables.

TRANSFORMATIONS:

let target program fragment = while $i \leq n$ do $(b[i]:=b[i]+p*(i-1);i:=i+1)$;

\Rightarrow_{T1} while $i \leq n$ do (if $p=0$ then $b[i]:=b[i] + p*(i-1)$
 else $b[i]:=b[i] + p*(i-1)$;
 $i:=i+1$)

\Rightarrow_{T2} while $i \leq n$ do (if $p=0$ then $(b[i]:=b[i] + p*(i-1);i:=i+1)$
 else $(b[i]:=b[i] + p*(i-1);i:=i+1)$)

\Rightarrow_{T3} while $i \leq n$ do (if $p=0$ then $(b[i]:=b[i]+0;i:=i+1)$
 else $(b[i]:=b[i]+p*(i-1);i:=i+1)$)

\Rightarrow_{T4} while $i \leq n$ do (if $p=0$ then $i:=i+1$
 else $b[i]:=b[i] + p*(i-1);i:=i+1$)

\Rightarrow_{T5} if $p=0$ then while $i \leq n$ do $i:=i+1$
 else while $i \leq n$ do $(b[i]:=b[i] + p*(i-1);i:=i+1)$

\Rightarrow_{T6} if $p=0$ then $i:=n+1$
 else while $i \leq n$ do $(b[i]:=b[i] + p*(i-1);i:=i+1)$

Figure 1—The transformation process using syntactic pattern-replacement rules.

Figure 2 shows the DDPT method applied to BYPASSLOOP. The first step matches the input pattern to the target program fragment, "while $i \leq n$ do. . ." The second step invokes the BPL replacement fragment constructor procedure above. This results in an incompletely-specified program fragment whose abstract elements r and $TU:=F(TU)$ are constrained by the BPL strategy. Finally in Step 3, these abstract portions are synthesized with $p=0$ for r and $i=n+1$ for $TU:=F(TU)$. (The details of the synthesis procedure are beyond the scope of this paper.)

The advantages of this approach are:

1. The overall goal (to bypass the loop) directs the transformation process.
2. Once BYPASSLOOP is selected, any "low-level" analysis is carried out automatically (e.g., partition variable assignments into types).
3. The system derives properties such as the predicate $p=0$, instead of requiring the user to specify them.
4. The input pattern and replacement fragment construc-

input pattern : "do; (transient-updates | non-transient-updates); od"

target program : while i<= n do (b[i]:=b[i]+p*(i-1);i:=i+1)

1. MATCH BPL input pattern to target program. Analyze statements in loop. Partition variable assignments into transient and non-transient updates. (See section 2.1.)

=> <do;...;od : while i<=n do ...>
 <transient-updates : i:=i+1>
 <non-transient-updates : b[j]:=b[j]+p*(j-1), 1<=j<=n>

2. INVOKE BPL replacement fragment constructor procedure.

- (a) Bind M to value of <do;...;od>
- (b) Bind TU to {i}
- (c) Bind NTU to {b}
- (d) CREATE-CONDITIONAL(pred = "r", truepart = "TU:=F(TU)", elsepart = M)

=> "if r then TU:=F(TU) else M"

3. SYNTHESIZE abstract parts "r", and "TU:=F(TU)".
 - (a) Compute F(TU), the final values of x in TU.

=> i:=n+1

(b) Compute predicate "r". Let x_i denote the i th value of x . For each non-transient variable x , for each iteration i , construct a predicate $r_i = AE(x_0 = x_n)$. (AE is abstract evaluation [BIGG77a].) Let $r = r_1$ and r_2 and ... and r_n .

=>> AE($b_0[1]=b_n[1]$) and ... and AE($b_0[n]=b_n[n]$)

=>> AE($b_0[1]=b_0[1]+0$) and ... and AE($b_0[n]=b_0[n]+p*(n-1)$)

=>> T and $p*1=0$ and ... and $p*(n-1)=0$

=>> $p=0$

(c) Return fully instantiated program fragment.

=> "if $p=0$ then $i:=n+1$ else while $i<=n$ do ..."

Figure 2—DDPT method applied to BYPASSLOOP.

tor procedure of the transformation applies to a large class of target program fragments.

Cancel DDPT

A general selection procedure is yet another important aspect of our system which is illustrated in this example. We are experimenting with automating this step of the transformation process by using the program annotations sup-

plied by the programmer for clarity. A database of related annotations is maintained. For example, "loop," a syntactic program control structure element, and "update," a descriptive action verb element, are related by a relation R-BPL. The selection procedure essentially constructs a list of those related elements contained in a target program which are related in the database. Each relation (e.g., R-BPL) is suggestive of one or more DDPTs (e.g., BPL). Hence the selection procedure returns a list of potential transformations to be tried.

The program in Figure 3 processes characters given in an array `text[0:n]`. In and out are two integer pointers such that during execution of the main loop,

- (a) `text[1:out]` is the text processed "so far," and
- (b) `text[1:in]` is the output or "saved" text.

The text processing 1) replaces linefeeds by blanks, 2) removes redundant blanks and 3) removes non-alphabetic characters. (This is an example discussed in Reference 1.)

Abstractly, the main loop of the program is:

```
while moretext do
  (get next character from input;
  put character in output;
  case on character type:
  linefeed: (replace character by blank;
             if redundant blanks then remove character
             from output);
  blank:    if redundant blanks then remove character
             from output;
  alpha:    noop;
  else:     remove character from output);
```

In Figure 3 we "fill in the details" of this abstract program; the abstract operations above are left as annotations.

An applicable DDPT is the CANCEL transformation.² Intuitively the strategy of this transformation is to rearrange the action sequences so as not to have to "undo" or "can-

```
procedure processtext(var text:array[0:n] of char);
var ch:char; in,out:integer;
begin
  in:=out:=0; text[0]:= ' '; /** initialize **/
  while out<n do
  begin
    /** get next character from input **/
    out:=out+1;
    ch:=text[out];
    /** put character in output **/
    in:=in+1;
    text[in]:=ch;
    /** case on character types **/
    case ch in
    linefeed: begin
      /** replace character by blank **/
      ch:=text[in]:=' ';
      /** test for redundant blanks **/
      if text[in-1] = ' '
      then /** remove character from output **/
        in:=in-1;
      end;
    space:   begin
      /** test for redundant blanks **/
      if text[in-1] = ' '
      then /** remove character from output **/
        in:=in-1;
      end;
    alpha:   begin /** noop **/ end;
    else:    /** remove character from output **/
      in:=in-1;
    end; /** case **/
  end; /** while out<n **/
end; /** procedure processtext **/
```

Figure 3—Program to process characters.

cel" any previous action. In this case, CANCEL will eliminate action sequences such as `["put"; . . . ; "remove"]` by replacing the "puts" and "removes" with "noops", i.e., `["put"; . . . ; "remove"] ⇒ ["noop"; . . . ; "noop"]`. Applying CANCEL to the abstract program above yields the transformed abstract program:

```
while moretext do
  (get next character from input;
  noop;
  case on character type:
  linefeed: if not redundant blanks then put blank in out-
            put;
  blank:    if not redundant blanks then put character in
            output;
  alpha:    put character in output;
  else:     noop);
```

Those portions in italics are program parts affected by the transformation. The complete program is shown in Figure 4. In this version, note that the "put" is done only when needed; the "replace" and "remove" actions have been eliminated.

How can one use the program's annotations in Figure 3

```
procedure processtext(var text:array[0:n] of char);
var ch:char; in,out:integer;
begin
  in:=out:=0; text[0]:= ' '; /** initialize **/
  while out<n do
  begin
    /** get next character from input **/
    out:=out+1;
    ch:=text[out];
    /** case on character types **/
    case ch in
    linefeed: begin
      /** replace character by blank **/
      ch:= ' ';
      /** test for redundant blanks **/
      if text[in] ~= ' '
      then begin
        /** put character in output **/
        in:=in+1;
        text[in]:=ch;
      end;
    space:   begin
      /** test for redundant blanks **/
      if text[in] ~= ' '
      then begin
        /** put character in output **/
        in:=in+1;
        text[in]:=ch;
      end;
    alpha:   begin
      /** put character in output **/
      in:=in+1;
      text[in]:=ch;
    end;
    else:    begin /** noop ** / end;
    end; /** case **/
  end; /** while out<n **/
end; /** procedure processtext **/
```

Figure 4—Processtext modified by DDPT CANCEL.

to *select* the CANCEL DDPT? If the program contains a "put" followed by a "remove," then there is a chance that the CANCEL transformation is applicable. A database of relationships is maintained for potentially related annotations. The system searches the program, constructing a list of these related elements. For this example, there are several instances of a relationship named R-CANCEL. In the first abstract program above, there are three paths containing the sequence [put; . . . ; remove]. The pair (put, remove) is contained in the database under the relation R-CANCEL. The system collects three instances:

```
R-CANCEL:
{(PUT001,REMOVE001),(PUT001,REMOVE002),
 (PUT001,REMOVE003)}
```

The presence of these related annotations in the program causes the corresponding DDPT, CANCEL, to be selected. Later, these annotations are used to locate the specific instances of these actions in the match input pattern step of the transformation process. The CANCEL replacement fragment constructor procedure analyzes the target program, reorders and eliminates the offending action sequences, and produces an incompletely specified replacement program fragment (as with BPL). The synthesizer fills in those remaining abstract portions producing the program shown in Figure 4.

In addition to automating more of the transformation process, the advantages to having the transformation system aid in the selection part include

1. Elimination of the problem of "pointing" to the places within the program text where the transformation is to be applied—these places are pointed to by the annotations and remembered by the system.
2. The program is manipulated at the semantic level, e.g., manipulate the verbs "put" and "remove," rather than at the syntactic level, e.g., manipulate "text[in]=ch" and "in:=in-1."

Other DDPTs

Many program manipulations fall naturally into the design-directed paradigm, because of the ability to analyze the "matched" portion of the program in order to synthesize replacement program fragments—the replacement fragment constructor procedure built into each DDPT can be quite general.

In this section, we list a number of DDPTs of interest. The reader is referred to Reference 9 for more details.

FLAG-MONITOR—Replaces an arbitrary boolean test with an equivalent boolean flag variable which "monitors" the original condition.

DO-EITHER—Generates (synthesizes) "special case" program fragments from a specification of the "general case."

GENERALIZE—Synthesizes "general case" program fragments from a sequence of "special cases."

EXTEND—Data structure extension.¹³ (A form of GENERALIZATION.)

RELATION TO OTHER RESEARCH

As noted earlier, most other transformation systems are syntax-based and two "rule-types" have emerged. One is the production-like syntactic pattern replacement rules shown earlier;^{1,8,10,12,13} the other is a system of rules which manipulate recursive programs.^{6,7,11,14} The restrictions imposed by these methods have been pointed out in the second section. Mostly the problems center on control and the degree of complexity allowed in transformations.

The major differences between our rules and their syntax-based counterparts rests in the degree of flexibility in specifying the "patterns" associated with the transformation process. Our matching mechanism allows for more general pattern matching; the substitution and replacement mechanism is more computational in nature, and we provide a mechanism for automatic selection based on the program's annotations. The syntactic rules are both useful and necessary in a TI system; we simply extend the type of rules available to the user.

Design-directed transformations extend the work of Biggerstaff and Johnson,^{3,4} in which automatic program synthesis is viewed as a process directed by known abstract program designs.

CONCLUSIONS

The design-directed approach to program transformations is a natural extension to other, more syntactic-oriented approaches. DDPTs are shown to have more general selection, matching and replacement procedures than their syntax-based counterparts. The advantages of this generality are

- Higher degree of automation.
- Each single transformation has a larger scope (it affects a larger target program fragment).
- Transformations are intuitively closer to ones a good programmer would select in transforming his program.
- Reduction in the number of transformation rules at the user level.
- Makes use of system's ability to analyze and reason about programs (e.g., using symbolic evaluation and program synthesis).
- Each transformation applies to a large class of target programs.

We are currently implementing these transformations and several others in LISP on a DEC 2020, at the University of Washington.

ACKNOWLEDGMENT

The author would like to thank Dr. Ted J. Biggerstaff for his influence, guidance and support in this research. Notable contributions to portions of this work are attributed to his continuing interest. The author is also indebted to Professor David L. Johnson for his continued support and suggestions.

Finally, thanks is due to the referees whose comments greatly improved the content of this paper.

REFERENCES

1. Balzer, R., N. Goldman and D. Wile, "On the transformational approach to programming," *Second International Conference on Software Engineering*, 1976.
2. Barth, J. M., "Shifting garbage collection overhead to compile time," *CACM*, Vol. 20, No. 7, July 1977.
3. Biggerstaff, T. J., "C2—A super-compiler approach to automatic programming," Ph.D. Dissertation, University of Washington, Seattle, Washington, January 1976.
4. Biggerstaff, T. J. and D. L. Johnson, "Design directed program synthesis," *CSCI Tech. Rep. 77-02-01*, University of Washington, February 1977.
5. Biggerstaff, T. J. and C. L. Jette, "Design directed program optimization," *CSCI Tech. Rep. 77-10-01*, University of Washington, (October 1977).
6. Burstall, R. M. and J. Darlington, "Some transformations for developing recursive programs," *Proc. 1975 International Conference on Reliable Software*, 1975.
7. Darlington, J., "Program transformation and synthesis: present capabilities," *Pub. 77/43*, Imperial College, London, September 1977.
8. Gerhart, S. L., "Correctness preserving program transformations," *Second ACM Symposium on POPL*, January 1975.
9. Jette, C. L., "Design directed program transformations," Ph.D. Dissertation, University of Washington, forthcoming.
10. Loveman, D. B., "Program improvement by source to source transformation," *Third ACM Symposium on POPL*, January 1976.
11. Manna, Z. and R. Waldinger, "The logic of computer programming," *STAN-AIM-298*, August 1977.
12. Rutter, P. E., "Improving programs by source-to-source transformation," Ph.D. Dissertation, University of Illinois at Urbana-Champaign, 1977.
13. Standish, T. A., D. C. Harriman, D. F. Kibler and J. M. Neighbors, "The Irvine program transformation catalog," University of California at Irvine, January 1976.
14. Wegbreit, B., "Goal directed program transformation," *IEEE Trans. on Soft. Eng.*, Vol. SE-2, No. 2, June 1976.

A data flow evaluation system based on the concept of recursive locality*

by A. L. DAVIS

University of Utah
Salt Lake City, Utah

INTRODUCTION

In an attempt to increase the performance of computing machines, there appears to be two main approaches—(1) to use faster components in existing architectures and (2) to design new architectures which are capable of exploiting some form of concurrency. The first approach is inherently limited in that the effects of reduced integrated circuit geometry, new process technology, and new logic families can be expected to increase overall system performance by only a couple of orders of magnitude. While this is initially impressive, it does not allow the desired machine performance projected to be necessary to solve large physics problems, or needed for accurate weather prediction.¹⁶ The second approach, while being a considerably more difficult organizational problem, is inherently unlimited in nature. There are numerous levels at which concurrency can be exploited in digital computers, i.e. multiple data paths, more concurrent realization of low-level circuit functions, overlap and pipeline processing within a single processing element, multiple processors, etc. In developing any new "fast as possible" machine, it is important to attempt to implement all of the above suggestions. However the work reported here will mainly be concerned with solving the problem of how to utilize and organize systems containing large numbers of independent processors.

In attempting to escape the fundamental performance bounds imposed by von Neumann architectures, it is insufficient to modify only a few aspects of the von Neumann style system ideas. Alternative proposals to the "clock-driven" von Neumann architectures are numerous. There are at least two areas which have some promise. One is the "demand-driven" approach espoused by Friedman and Wise,¹⁰ Backus³ and Berkling.⁵ Another is the "data-driven"

approach taken by Dennis,⁸ Bahrs,⁴ Davis⁶ and Arvind, Gostelow and Plouffe.² The work described here is of the data-driven variety due to the feeling that the demand-driven approach does not support intra-process pipelining very well. In addition, the propagation of demands takes time, and while demand-driven programs do allow for increased expressive power, the emphasis here is on performance. The data-driven approach naturally describes both pipelined (vertical) concurrency and independent operation (horizontal) concurrency.

The intent of this paper is to present an overview of the "Utah approach" to data-driven computation. The major emphasis will be on the architectural aspects of an existing machine, DDM1 (Data-Driven Machine #1). Details of the method by which data-flow programs are evaluated on DDM1 will also be discussed. The major differences (and motivations for these differences) between the Utah approach and other published data flow groups will be considered. Part of the material presented here has been published elsewhere in a less general and more detailed form.^{6,7} The new topics presented here are principally concerned with automatic resource allocation, and the flow balance during pipelined processing.

It is evident that any machine architecture intended to have a general commercial appeal must be feasible with respect to the changing constraints of integrated circuit technology. For architectures which fit nicely into the VLSI realm, the advantages are numerous. Among these are lower cost, increased reliability, increased speed and decreased power consumption.

The actual machine language of DDM1 is a linearized encoding of a directed graph schema called data-driven nets⁶ or DDNs. DDNs are very similar to the data flow nets of Dennis⁸ and Rodriguez.¹⁵ The asynchronous nature of DDNs makes it easy to decompose a given net into a set of concurrent subnets, which can then be allocated to independent physical resources. The main distinction between the Dennis nets and DDNs is that in DDNs no distinction is made between the net tokens which are used for control purposes, and other net tokens. All DDN tokens are considered to be data items, and no explicit distinction is made to distinguish between classes of tokens. Another difference is that the primitive DDN cell types are slightly more high-level than

* The work reported in this paper was supported by Burroughs Corporation. DDM1 is an operational hard-wired data-driven machine, and was completed in July 1976 at the Burroughs Interactive Research Center in La Jolla, California. Many of the early systems ideas were developed in conjunction with Robert S. Barton. Gary Hodgman, Lawrence Rogers, and Karl Boekelheide were instrumental in the conceptualization and implementation of the actual machine. An improved version of DDM1 now resides at the University of Utah, where the project continues under the support of the Burroughs Corporation.

the Dennis nets. Finally, some primitive activities which are explicitly specified in the Dennis schema are implicit in DDNs. An example is the Dennis "link" which serves as both a transmission and copy site. The functions of such links are implicitly incorporated into the output mechanism of DDN operators. The result is that both the DDN and Dennis schemas share the same properties with respect to ease of program verification, ease of program conceptualization and ease of machine evaluation. Due to slightly higher-level primitives and a less explicit schema, a DDN program graph will typically have less vertices (cells) and arcs (data paths) than a functionally equivalent net in the Dennis schema. This difference is mostly one of style and is not particularly significant, although the differences are reflected in the respective architectures.

The only sequencing constraint in DDNs is that of data dependence, and since no weaker sequencing constraint exists without doing non-productive computation,¹⁴ DDNs are naturally a maximally concurrent representation of a given algorithm. While such concurrency may add to the "naturalness" of the programming experience, it is operationally useless as a speed-up mechanism unless it can be mapped onto a set of physical resources capable of exploiting this concurrency. If this mapping is done at run-time, then the time to map must not overshadow the speed-up attained as a result of the concurrent execution. The attitude about how and when this mapping gets done marks a major difference in the Utah approach and that of other data-driven computation projects.

Lastly, a number of additional goals for the machine structures presented here are felt to be desirable. Namely, it is intended that these machines be general purpose, extensible, reliable, easily programmable, support very high levels of concurrency and also be economical with respect to their performance and existing technology. In particular, this effort is not concerned with one of a kind or special purpose machines. Special purpose machines are perhaps ideal for a given environment, but suffer from inherent limits in their applicability to other problems.

THE IMPACT OF VLSI

The advantages of high density integrated circuit technology are so overwhelming that the constraints of VLSI must be considered as a primary force on future architectures, the global influences of which are summarized here. Due to the tremendous commercial emphasis on MOS VLSI, the following discussion will mainly be concerned with the properties of MOS device integration.

The most highly publicized VLSI benefits are those involving cost. A single custom VLSI chip (64-pin package) currently costs about \$80,000 to \$300,000 to produce. Even then, production typically must be guaranteed for about a quarter of a million parts at an additional cost of \$7 to \$10 per part. This clearly indicates that VLSI cost advantages can be obtained only if any given chip can be used in very large volumes. If a part does not have universal appeal, then

the use of such a part in a new architecture brings about some high-pressure constraints. Either the part must be used a large number of times in a single system, or a single system must have a very high sales volume, or some combination of the two. The number of part types in a given system is also a major concern in that it becomes a multiplicative factor in the system development cost.

Another factor heavily influenced by a VLSI implementation is speed. The dominant speed factor is due to the capacitive effects on a given transmission path. Typical off chip loads are on the order of 100 picofarads, while on chip loads are approximately one picofarad. Since delay times are proportional to the capacitive load (for constant drive current), this implies that signals which can remain on the chip will be driven about two orders of magnitude faster than those which must be driven to destinations off the chip. Additional speed-up can be obtained from the decreased geometries of switching elements and conductor path lengths. This is a very strong argument for architectures which attempt to maximize locality of processing. For architectures in which processing and local storage cannot be done at the same locality, massive delays must be incurred as a result. The only way around the slow off chip drive problem is to drive more current off the chip. This requires a series of relatively large output drivers, which are extremely costly in terms of chip real estate and power consumption. In addition, locality of processing will reduce the amount of contention for a given system transmission path. This contention is important in a highly parallel system in that the resultant sequencing will yield reduced system efficiency.

The number of pins is an important VLSI metric. If chip types are used in sufficient quantities to amortize the initial layout cost, then the physical cost to manufacture a machine becomes approximately linear with pin count. In addition, increasing the number of pins on a particular chip causes decreased yield due to bonding problems. Increased pin count also implies that even more silicon area must be allocated to connection pads and pin drivers.

VLSI implementation also yields the more commonly discussed advantages such as (1) increased system reliability due to reduced part count, (2) decreased system power consumption since voltages on a given chip scale with physical feature size and (3) decreased system maintenance cost as chip replacement policies become more effective in highly integrated systems.

The extent to which these VLSI advantages can be realized is proportional to the logic/pin ratio of the proposed system modules. If the logic/pin ratio is relatively small then the situation is very much that of an SSI machine. If the logic/pin ratio is very high then true VLSI advantages can be obtained. This is a challenge to architects to devise systems which can be modularized into high complexity modules which communicate with their environment infrequently, using relatively few signal paths. Furthermore, as integration technology advances causing feature sizes to shrink even more, these new architectures must remain viable.

ARCHITECTURAL PRINCIPLES

The VLSI constraints indicate that future architectures to support very high levels of concurrency should consist of a set of processing sites capable of performing localized storage and computation of a reasonable complexity. These sites should be essentially the same physical module, which ideally can be constructed as a single part type. An additional goal of the architecture presented here is that of extensibility. More specifically, the architecture should be extensible without bound in the following way:

1. Machine power should be enhanced by the addition of more modules (i.e. allow greater concurrency due to the increased number of processing sites).
2. The addition of new modules should not require any change to the existing operating system in order to manage the resulting larger system.
3. Additional resources should be added simply by "plugging in new modules" without any special tuning of the existing hardware to create consistent system timing and communication for the expanding system.
4. The extension should be available in small quanta.

The first and last points indicate that a user should be able to purchase only the power needed and not much more or much less. The other points indicate that the manufacturer should need to support only a single module, rather than a large number of system configurations.

Systems such as these cannot be implemented in a synchronous, centrally controlled manner. Central control of arbitrarily extensible systems implies that the control must be able to function on an arbitrarily large amount of state information, which either slows the control drastically or requires controller modification to access the new state information. In an arbitrarily extensible synchronous system the problem of unbounded clock skew (maximum difference in the perceived clock time between any two processing sites in the system) will cause failure. The systems described here will therefore be asynchronous, fully distributed systems. Fully distributed systems are defined here to have the following characteristics: (1) no module of a fully distributed system can determine the total system state, and (2) no module of a fully distributed system can enforce simultaneity in other modules. Holt¹³ has shown that the notion of total system state in complex asynchronous systems is counterproductive. Furthermore, the enforcement of simultaneity in physically separate, asynchronous devices is impossible.

There are many ways to organize an extensible set of modules in a distributed control system. The advantages of hierarchical organizations are (1) reduction in the amount of complexity to be dealt with at a given level, (2) verification by inductive methods can be done for uniform hierarchic systems and (3) the superior-inferior relationship can be utilized to resolve problems such as contention and deadlock in multi-resource systems. It will be seen that hierarchy also facilitates a nice resource allocation policy. Recursive hier-

archies are of particular interest in that they imply that the same module can be used at each level.

Recursive systems are nicely extensible. A recursively structured machine is one which has exactly the same structure at every level. Clearly physical recursion must terminate at some point. This point will be seen to be the deepest set of resources in the physical hierarchy. Additional advantages of recursively structured systems have been discussed by Glushkov.¹¹ It will be shown that the width of a level in these recursive hierarchic structures can be used to execute independent operations, while the depth of the hierarchy will be used to facilitate pipelined operations.

THE ARCHITECTURE OF DDM1

The architecture consists of a set of asynchronous modules which communicate by passing messages. The basic computational unit of the architecture is a processor-store element (PSE). A PSE consists of a processor module (P) and its associated local storage module (S). Any PSE can execute any machine language program, providing that it has a sufficient amount of local storage. No module that is not a PSE can perform this function. The architecture is a recursively organized set of these PSEs. The recursive definition of the structure is

$$\begin{aligned} \langle PSE_n \rangle &::= \langle P_n \rangle \langle S_n \rangle \\ \langle S_n \rangle &::= \langle ASU_n \rangle \\ \langle P_n \rangle &::= \langle AP_n \rangle \langle AP_n \rangle \langle PSE.GROUP_{n+1} \rangle \\ \langle PSE.GROUP_{n+1} \rangle &::= \langle PSE_{n+1} \rangle \langle PSE_{n+1} \rangle \langle PSE.GROUP_{n+1} \rangle \end{aligned}$$

Subscripts denote the recursive level at which the module physically resides. $\langle AP \rangle$ is an atomic processor module, which has no further sub-structure (contains no PSEs). Similarly an atomic storage unit $\langle ASU \rangle$ has no PSE substructure. The width of a $\langle PSE.GROUP \rangle$ has a physical bound. For DDM1 this bound is eight. The structure is depicted in Figure 1.

This structure allows for a hierarchical distributed storage organization. Any S or ASU may consist of an arbitrary amount of storage of any desired medium. Higher levels of PSEs are considered logically superior to lower-level PSEs. It is advantageous if higher-level stores (ASUs) are slower and larger than the stores of lower levels. The interface and functional ability of any ASU (regardless of size, speed and level) is the same. The structure also allows for an arbitrary number of processors that can be used concurrently. It is important to note that all APs are identical regardless of level. However, the processors at higher levels will be more powerful, in that they contain more PSE substructure than the processors at lower levels. More substructure implies more internal concurrent processing capability.

When viewed non-recursively this structure is simply a tree structure with a single root and a possibility for up to eight sons at any node. Each node of the tree is a PSE and is capable of executing any machine language program. The leaf nodes have no substructure and therefore consist of an

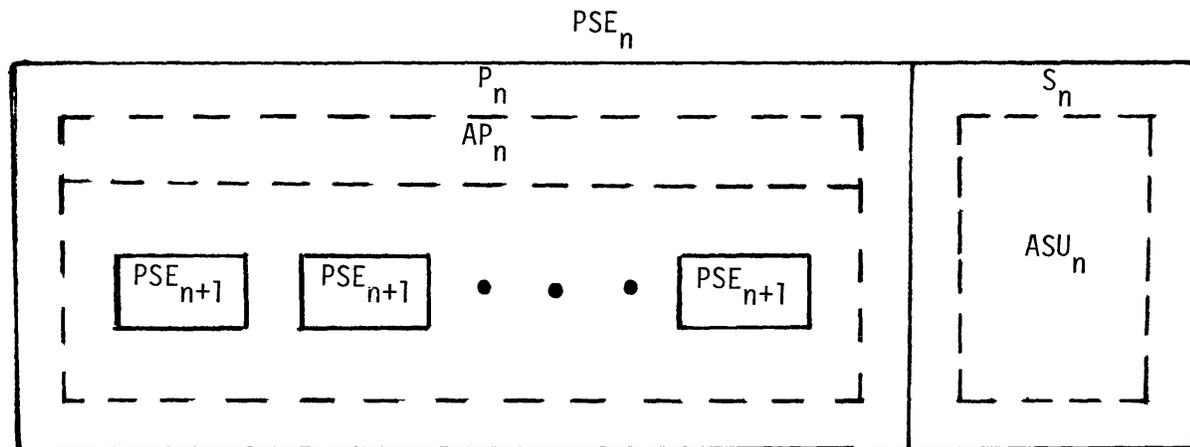


Figure 1—Recursive definition of PSE at level n .

AP and an ASU. At each node the fan-out is fixed but the depth of the tree is arbitrary. In this manner the architecture allows any desired number of PSEs to be configured for a given machine. The desired goal is for machine performance to improve with the addition of more PSEs.

There are a number of ways to enforce this logical tree structure onto a collection of PSEs. All involve some form of a connection network to implement the desired communication paths. A number of general interconnect networks have been considered—busses, crossbars, Banyan nets¹² and permutation networks.¹⁷ For tree-like machines, full connectivity is not required. The expense of crossbar switches vary as the square of the connected elements. Bus conflict could drastically reduce actual parallelism in the machine. Permutation networks present a tremendous problem in that they may need to be totally reconfigured when a single new connection is necessary. This is difficult to do reliably in a multi-path distributed control environment. Banyan networks have some merit, but do not easily allow for the desired hierarchic pipelined communication. Therefore, in DDM1 a simple one-to-eight-switch was chosen as the interface unit between successive levels of PSEs. The result is that the physical and logical recursive structures are the same. The structure is fixed and cannot be dynamically changed.

Information is passed between PSEs as messages which are variable length character strings. Upward traveling messages are passed on by the switch in an arbiter-like manner. Downward going messages contain header fields which indicate their destination. This header is deleted by the switch as the message is passed. Downward and upward messages are dealt with by independent hardware, and therefore are controlled concurrently. This character serial nature of the machine has the following advantages:

1. Hardware modules are made simpler and more applicable for VLSI implementation due to the reduced pin count.
2. Hardware communication paths are more general in that variable length information units can be transmit-

ted as varying numbers of fixed-width base characters. This facilitates a hardware substitution strategy for modules. Each module can interpret the variable length message and perform the indicated function.

These advantages aid in greatly reducing the cost of the hardware modules. Some low-level performance is lost by doing everything serially. The philosophy is to regain that lost performance many times over by providing a systems organization that allows for highly concurrent levels of activity.

Physical queues are placed between levels of PSEs in order to facilitate pipelining and increase physical module independence. Without queues, the sender of a message would need to wait on receiver availability. If a queue becomes full, only then must the sender wait until the receiver has freed up some queue space. If queue sizes are adjusted so that a sender is rarely required to wait for space, then the system would be well tuned for efficient processing. Optimal queue size depends on the average message length. It is therefore impossible to guarantee that no waiting will occur. Strict hierarchical control and a restricted process structure insures that the system does not deadlock. A block diagram of the PSE structure is shown in Figure 2. All DDM1 paths, except for the path between the ASU and the AP, consist of six wires (a two-wire request-acknowledge control link and a character-width data bus; in the DDM1 prototype four-bit characters are used).

The variable length, character serial message structure and DDN representation indicate that the ASU should be a highly flexible storage structure. Further requirements are that the ASU deal with pipelining of data items and their continual destruction upon cell firings. In order to increase efficiency of the PSE, all storage management functions are performed internally by the ASU. The ASU appears as a variable field length file system, which directly executes commands, such as initialize, skip, insert, read, write, delete and index. The free space is managed automatically by the ASU.

This PSE structure allows for a high degree of processing

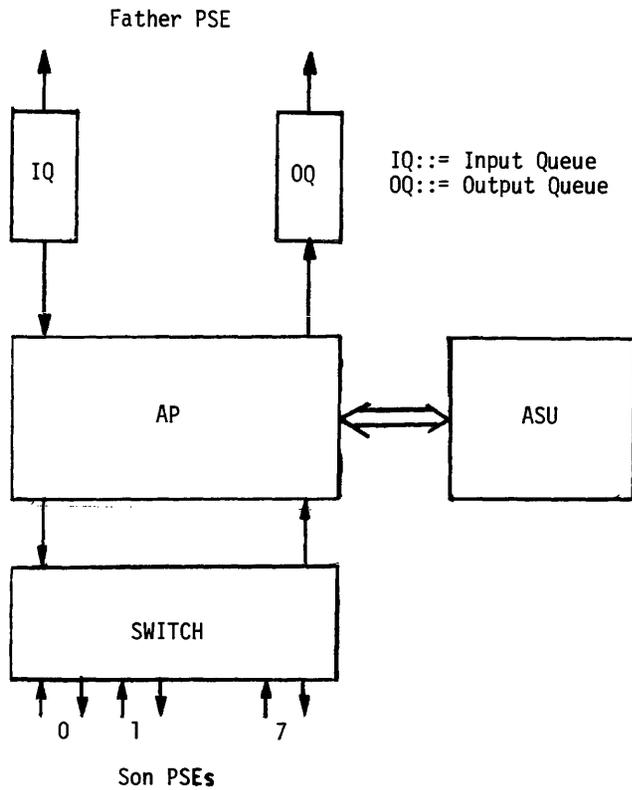


Figure 2—PSE structure.

locality in that any PSE can execute any DDN program (assuming that there is sufficient storage in its local ASU). In addition, the PSE admits nicely to VLSI implementation. The one-to-eight-switch can be implemented using a set of one to two switches of similar function. Using 1:2 switches, DDM1 module complexities (pin and gate count) are shown in Figure 3. The pin counts include pins for power, ground, initialization and extension. The indicated module pin counts are rounded up to coincide with standard package sizes.

These complexities are all within reason for current VLSI designs, and are attractive with respect to the logic/pin ratio. Approximately 30% of the AP and ASU gates are used in statistics and maintenance circuits.

AUTOMATIC RESOURCE ALLOCATION AND EVALUATION

When a message corresponding to a DDN program enters a PSE at any level, the PSE may take one of two actions:

1. *Decomposition and allocation*—If the PSE has substructure and if there exists some set of concurrent subnets in the DDN process, then the PSE may split the DDN and send concurrent subnets to PSEs at the next lower level.
2. *Execution*—If the PSE has no subresources, or if there is no exploitable concurrency in the DDN, then the PSE executes the DDN at that level.

To aid the decomposition process, a structural descriptor may precede the incoming DDN in the message structure. This additional storage can greatly reduce time required for decomposition decisions in the PSE. In addition, each PSE must contain information about the number of available PSEs and the sizes of their respective stores. Problems would result if a DDN were sent to a PSE that was too large to fit in its local store. Only the local store sizes of immediate subresources are known. This ensures the recursive nature of the decomposition process.

The decomposition process takes some time. It is important that the speed-up gained by the extra concurrency resulting from decomposition is not overshadowed by the time to decompose. Experiments have indicated that a "first fit" decomposition is almost always better than a "best fit" decomposition strategy. It also appears not to be generally worthwhile to decompose the DDN structure completely on this architecture. At fine granularities, the slowdown resulting from loss of locality is not regained by the concurrent execution of very small subtasks. The exception to this rule is in the case of pipelining, where subtasks remain allocated for relatively long periods of time and sustain high activity at each site.

If decomposition and resource allocation occur at run-time, it is important that they be simplified as much as possible. It is possible to perform these tasks completely at compile-time. This however is inadvisable since it depends on knowing the run-time availability of PSEs in the system. In a system containing large numbers of PSEs, the probability is high that some PSEs will fail or be busy doing other things. In addition, large portions of a process may only be evaluated conditionally. A compile-time allocation would have to allocate tasks which may never be executed. The strategy is taken here to split the decomposition task into two phases—(1) at compile time do all of the resource and condition independent work and (2) at run-time, dynamically make the actual allocation of executable tasks to available physical resources.

DDNs are quite randomly structured graphs and DDM1 is a very regularly structured set of resources. Direct run-time allocation would be too slow, due to the structural disparity between program and machine. At compile-time, the two-terminal DDN process structure is transformed into a well structured and functionally equivalent series parallel graph (SP-graph). Two-terminal means that the graph contains a single "first" cell and a single "last" cell. This

Module	Gate Count	Pin Count
IQ, OQ (1K Characters)	3,000	16
Ap	20,000	64
ASU (4K Characters)	47,000	64
1:2 Switch	2,000	40
Ap + ASU	67,000	64
Ap + ASU + IQ + OQ	73,000	64
Ap + ASU + Switch	69,000	64
PSE	75,000	64

Figure 3—PSE module complexities.

facilitates the determination of net termination and initiation. SP-graphs are acyclic, two-terminal, directed graph structures which can be formed by successively combining cells and/or SP-graphs in series or in parallel. The SP-graph structures are then allocated as necessary at run-time. Data flow graphs in general admit nicely to arbitrary restructuring due to their asynchronous and local control characteristics.

A detailed description of the compiler algorithms to convert a DDN to a functionally equivalent SP-graph is too lengthy to be presented here. An English description of the process is given, describing the nature of the algorithms. The first steps are:

1. Encapsulate all cycles at the outermost level into single cells. The result is a two-terminal acyclic graph at the outermost level. This can be done since DDNs are required to have well nested cycles.
2. Remove all transitive paths. That is if there exists a path from A to B , B to C , and A to C , remove the A to C path and cause it to pass from A through B to C . This requires the semantics of cell B to be modified slightly. The original inputs are treated the same, while the new inputs are merely passed on unmodified when the cell fires. These new inputs are known as the pass set and may be attached to any cell as a result of this phase of compilation. At this point the graph is a lattice.

For a lattice to be transformed into a functionally equivalent SP-graph, some degree of freedom must be given. There are two meaningful freedoms investigated so far—work and time. If time is free to be changed, then work remains fixed and the resulting graph is called the least-work SP-graph. Conversely, if work is the degree of freedom, then the result is a least-time SP-graph. "Least" is not used in any formal sense but only to indicate that the resultant work or time is equivalent to the original lattice net. Work is defined to be the number of cells which must fire to cause the net to terminate. It will be seen that some additional synchronization cells will need to be executed to enforce the SP topology in the least-work nets. Time is defined to be the critical path length from the first to the last cell at the outermost level (inner levels being those which were encapsulated in Step 1).

For the least-work transformation:

1. Number of paths from the first to the last cell.
2. Make a cut across all paths of equal number.
3. Place a synch cell across all cuts which are not already SP. Non-SP cuts are those which have (1) a set of sender nodes $S=\{s, \dots, sn\}$ and a set of receiver nodes $R=\{r, \dots, rm\}$ such that $n>1$ and $m>1$ and (2) there exists a pair of sender cells si, sj and a pair of receiver nodes ra, rb such that there are at least three connection paths between $\{si, sj\}$ and $\{ra, rb\}$. The result at this point is a least-work SP-graph.

For the least time net, an SP-graph is built from sender-

receiver relationships in the lattice graph as follows:

1. Clear the TODO and SENDER lists.
2. Place the last cell in the TODO list and in the LTSPGRAPH list.
3. Place all cells (which send messages to cells in the TODO list) into the SENDER and LTSPGRAPH lists.
4. For every cell in the SENDER list which is the only sender to the set of cells $\{ril, \dots, rik\}=Ri$ in the TODO list, and where si sends to no cell outside of Ri , build links in the LTSPGRAPH list from si to ril, \dots, rik .
5. Delete $\{ril, \dots, rik\}$ from TODO list and si from SENDER list.
6. For all remaining cells sj in the SENDER list, add $n-1$ copies of sj to the LTSPGRAPH list, where n is the number of output paths of cell sj . Build links in LTSPGRAPH from senders sj to the receivers sj such that each copy of sj has only one outpath. (This step performs the typical node splitting operation which is done in finite state machine reduction. The idea is to duplicate work in order to regularize the structure in a least-time fashion.)
7. Clear the SENDER and TODO lists. Place the T list cells into the TODO list. Clear the T list.
8. Repeat the process starting at Step 2 until all cells in the original lattice graph have been processed. The resulting LTSPGRAPH list is the desired least-time net.

Examples of the lattice to least-time and least-work SP-graphs are shown in Figure 4.

It is interesting to note that the algebraic SP expressions in Figure 4 can be (1) shown to be functionally equivalent and (2) transformed from one form to the other, using techniques similar to algebraic factorization and distribution (of the series and parallel relations). In DDM1 the decision to produce the least-time or least-work SP-graph is made at compile-time by the user. It would be nice to make this decision at run-time based on resource cost and availability. This, however, would increase the run-time overhead to a level which is currently felt to be excessive. The previous procedures are applied recursively to the levels defined by loop encapsulation, and are performed down to the desired granularity.

The allocation of SP-graphs onto tree-structured physical resources is an easy task. If the SP-graph of Figure 5 is folded back onto itself about the middle, the result is a tree-structured SP-graph. The SP-graph, its folding, and the allocation onto a tree of physical resources are all shown in Figure 5.

In this way full upward and downward communication can be carried on concurrently to achieve pipelining. Horizontal parallelism can be achieved by spreading independent subtasks across a given level of the architecture. Resource allocation is performed automatically by the hardware in DDM1 to achieve very high degrees of parallelism. The amount of obtainable concurrency is a function of available hardware resources and the program structure.

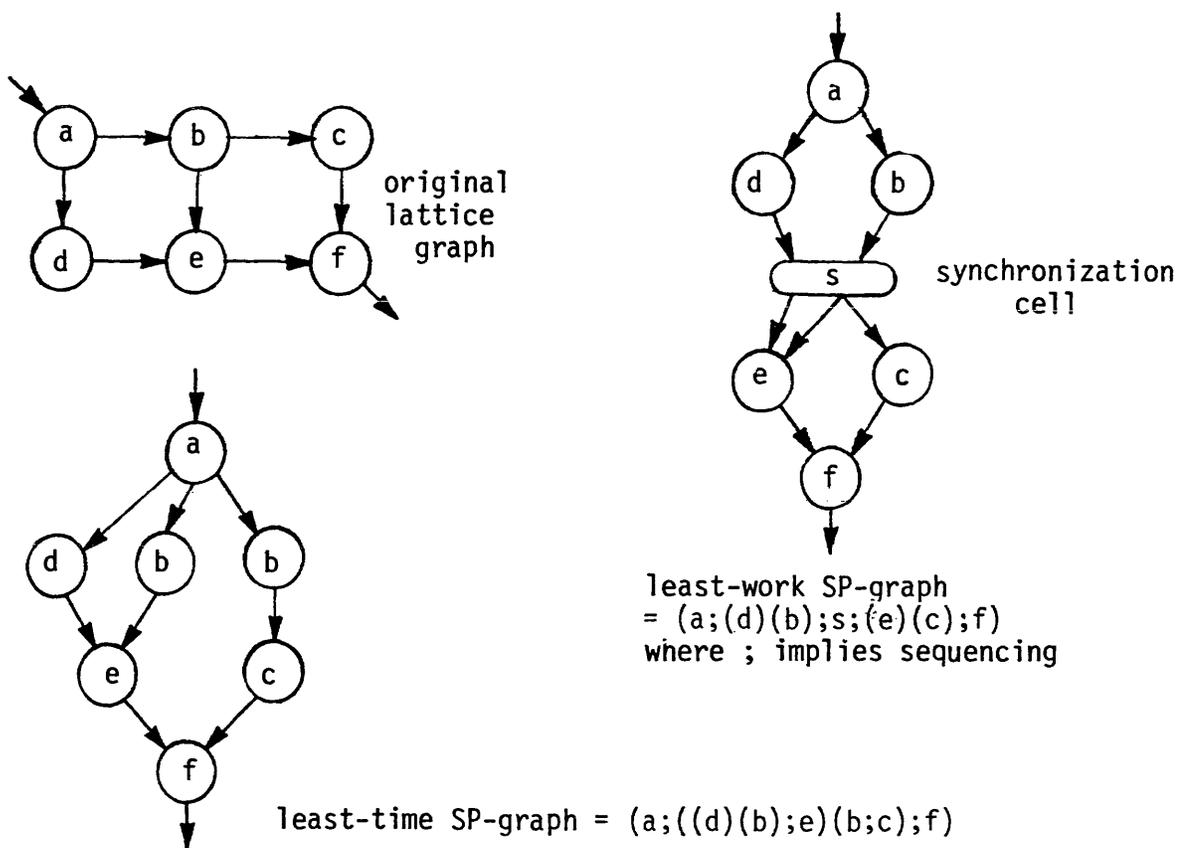


Figure 4—Lattice to least-time and least-work SP-graphs.

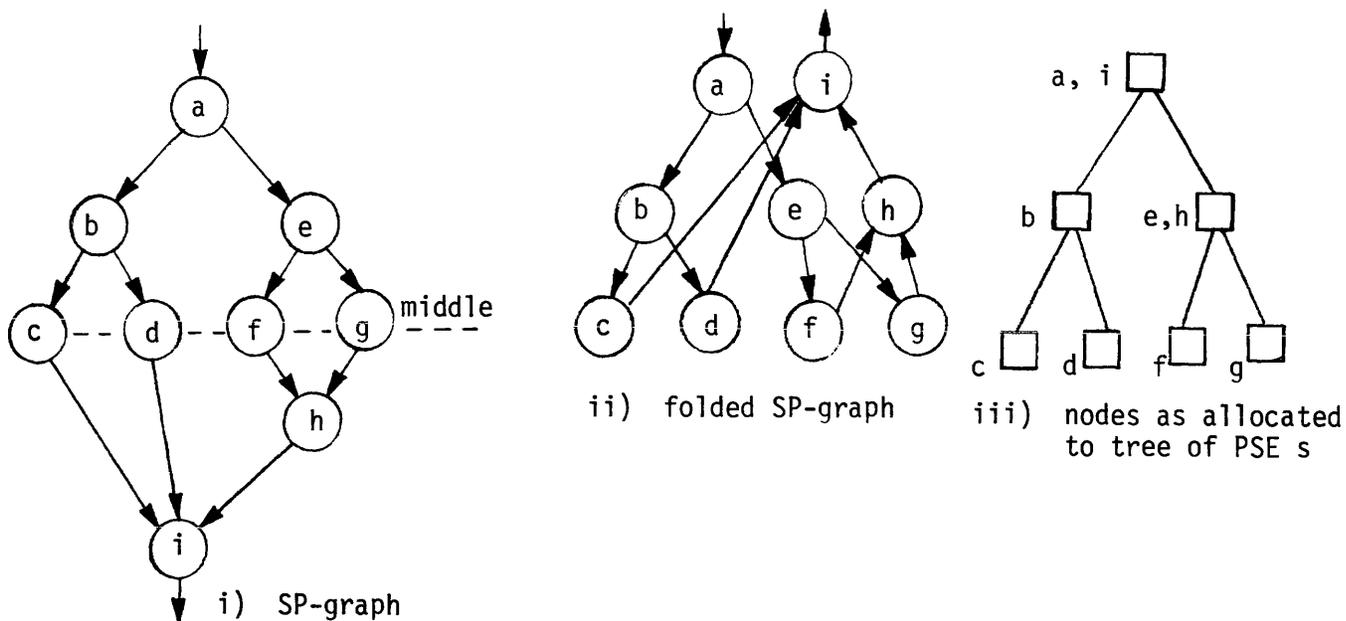


Figure 5—The allocation of SP-graph programs onto a PSE tree.

CONCLUSIONS

An architecture and evaluation scheme for data-flow programs has been presented. The architecture exploits recursive hierarchy to reduce complexity and allows for the arbitrary expansion of system resources. Physical resources are organized such that they can be used to exploit both pipelined and independent tasks. The system exploits the notion of locality that is important for both increased speed and decreased cost aspects of a VLSI implementation. This notion of locality also indicates that this system is not intended to exploit concurrency at the lowest possible level. It is felt that the additional overhead involved to do this would actually reduce overall performance levels.

Current status of the project is that DDM1 is operational and executes DDN programs. DDM1 communicates with a DEC-20/40, which is used to support conventional software tools such as compilers, simulators, and measurement programs. The current programming language is a statement description of a DDN. An interactive graphical programming language is in the works (in both a high-level and a low-level form). A simulator is being written on the DEC-20 which will manage any specified tree of resources (virtual) and use the DDM1 for actual evaluation. A number of large application programs are being written for DDM1. Detailed statistics will be taken during the execution of these programs to aid in formal evaluation of the DDM1 hardware.

The main points of departure of the "Utah" approach and that of Dennis⁹ is the use of a recursive hierarchy of physical resources, the exploitation of physical locality to decrease message frequency and increase the speed of VLSI implementations, dynamic hierarchical resource allocation, the lack of specialized functional modules to reduce the chip type count and a slight difference in the structure of the low-level schema. The architecture of DDM1 differs from that of Arvind and Gostelow¹ in that it does not try to achieve concurrency at all possible levels (because of the locality issue), the interconnection scheme is much simpler and no bus contention is possible, no special address space management needs to be done, allocated tasks may consist of many cells rather than just a single operation, and tasks are allocated only when all of their necessary input operands are present.

The disadvantages of the system described here are:

1. The current ASU design is not nicely extensible to allow more storage capacity to just be "plugged in."
2. The fixed, hard-wire tree structure is not flexible and results in certain PSEs in one subtree remaining idle when another heavily loaded subtree badly needs more resources.

3. There is currently not enough empirical data from test runs on very large programs to accurately quantify the overhead involved in decomposition.
4. Failure of a PSE will cause the entire subtree below the failure to become unusable. In general, the issues of fault tolerance have not been properly attended to.
5. Certain "perverse" SP-graph topologies can not be allocated such that full pipelining can be supported.

BIBLIOGRAPHY

1. Arvind, K. P. Gostelow and W. Plouffe, *The ID Report: An Asynchronous Programming Language and Computing Machine*, University of California at Irvine, Computer Science Department, Technical Report #114, 1978.
2. Arvind and K. P. Gostelow, "A computer capable of exchanging processors for time," *Information Processing '77*, North Holland, New York, 1977, pp. 849-854.
3. Backus, J., "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs," *CACM*, Vol. 21, No. 8, August 1978, pp. 613-614.
4. Bahrs, A., "Programming language semantics and closed applicative languages," *Proceedings of the ACM Symposium on Principles of Programming Languages*, 1972, pp. 71-86.
5. Berkling, K. J., *Reduction Languages for Reduction Machines*, Interner Bericht ISF-76-8, GMD, 1977.
6. Davis, A., *The Architecture of DDM1: A Recursively Structured Data-Driven Machine*, University of Utah, Computer Science Department, Technical Report UUCS-77-113, 1977.
7. Davis, A., *Data-Driven Nets: A Maximally Concurrent, Procedural, Parallel Process Representation for Distributed Control Systems*, University of Utah, Computer Science Department, Technical Report UUCS-78-108, 1978.
8. Dennis, J. B., "Data Flow Schemas," *Lecture Notes in Computer Science*, Vol. 5, Springer-Verlag, 1972, pp. 187-216.
9. Dennis, J. B., and D. P. Misunas, "A computer architecture for highly parallel signal processing," *Proceedings of the ACM National Conference*, 1974, pp. 402-409.
10. Friedman, D. P., and D. S. Wise, "Aspects of Applicative Programming for Parallel Processing," *IEEE TC*, Vol. C-27, No. 4, April 1978, pp. 289-296.
11. Glushkov, V. M., et al., "Recursive Machines and Computing Technology," *IFIPS Proceedings 1974*, North Holland, New York, 1974, pp. 65-70.
12. Goke, L. R., *Banyan Networks for Partitioning Multiprocessor Systems*, Ph.D. Dissertation, University of Florida, 1976.
13. Holt, A., and F. Commoner, "Events and Conditions," *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*, 1970, pp. 3-52.
14. Linderman, J. P., *Productivity in Parallel Computation Schemata*, MIT Project MAC, TR-111, 1973.
15. Rodriguez, J. D., *A Graph Model for Parallel Computations*, MIT Project MAC, TR-64, 1969.
16. Rudy, T. E., "Megaflops from Multiprocessors?" *Proceedings of the 2nd Rocky Mountain Symposium on Microcomputers*, 1978, pp. 99-107.
17. Waksman, A., "A Permutation Network," *JACM*, Vol. 15, No. 1, 1968, pp. 159-163.

Data flow languages

by WILLIAM B. ACKERMAN

Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

There are several computer system architectures which have the goal of exploiting parallelism—multiprocessors, vector machines and array processors. For each of these architectures there have been attempts to design compilers to optimize programs written in conventional languages (e.g. “vectorizing” compilers for the FORTRAN language). There have also been new language designs to facilitate using these systems, such as Concurrent PASCAL for multiprocessors,⁶ and languages that utilize the features of such systems directly, such as GLYPNIR for the Illiac IV array processor¹⁹ and various “vectorizing” dialects of FORTRAN. These languages almost always make the multiprocessor, vector, or array properties of the computer visible to the programmer—that is, they are actually vehicles whereby the programmer helps the compiler uncover parallelism. Many of these languages or dialects are “unnatural” in that they closely reflect the behavior of the system for which they were designed, rather than reflecting the way programmers think about problem solutions.

Data flow computers also have the goal of taking advantage of parallelism. As will be seen below, the parallelism in a data flow computer is both microscopic (much more so than in a multiprocessor) and all-encompassing (much more so than in a vector processor). Like the other forms of parallel computer, data flow computers are best programmed in special languages. In fact, their need for such languages is stronger—most data flow designs would be extremely inefficient if programmed in conventional languages such as FORTRAN or PL/I. However, languages suitable for data flow computers can be very elegant. The language properties that a data flow computer requires are beneficial in their own right, and are very similar to some of the properties that are known to facilitate understandable and maintainable software, such as the absence of undisciplined control structures and module interactions. In fact, languages having many of these properties have been in existence since long before data flow computers were conceived. The principal property of a language suitable for data flow is *freedom from side effects*, which will be described below. The (pure) LISP language²⁰ is the best known example of a language without side effects. The connection between freedom from side effects and efficient parallel computation has been known for over ten years.²⁵

To see why data flow computers require languages free of side effects, we must examine the nature of data flow computation and the nature of side-effects. A detailed description of the mechanism of data flow computers is beyond the scope of this paper. The interested reader is referred to References 2, 12, 15, 21, 22, 23.

There are three “data flow” languages that will be discussed in this paper. VAL¹ and ID³ were developed by the data flow projects at the Massachusetts Institute of Technology and the University of California at Irvine, respectively. LUCID⁴ was developed for program verification, not for programming data flow computers. It nevertheless is a suitable language for data flow computation.

Let us begin by examining a simple sequence of assignment statements written in a conventional programming language such as FORTRAN:

```
1 P=X+Y
2 Q=P/Y
3 R=X*P
4 S=R-Q
5 T=R*P
6 RESULT=S/T
```

A straightforward analysis of this program will show that many of these instructions can be executed concurrently, as long as certain constraints are met. These constraints can be represented by a graph (see Figure 1) in which nodes represent instructions and an arrow from one instruction to another means that the second may not be executed until the first has completed. So the permissible computation sequences include, among others:

```
(1,3,5,2,4,6)
(1,2,3,5,4,6)
(1, [2 and 3 simultaneously], [4 and 5 simultaneously], 6)
```

This type of analysis (commonly called data flow analysis, a term which long predates data flow computers) is frequently performed in two situations—at run-time in the arithmetic processing units of high performance conventional computers such as the IBM 360/91, and at compile time in optimizing compilers. In optimizing compilers, data

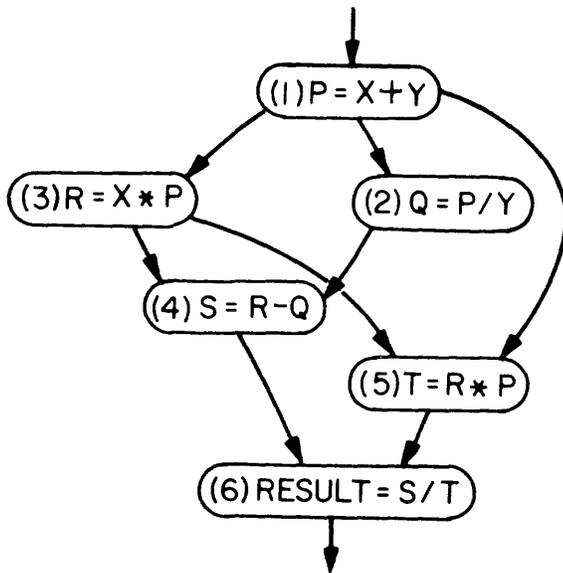


Figure 1

flow analysis yields improved utilization of temporary memory locations. For example, on a computer with high-speed general purpose or floating point registers, this program can be compiled to use the registers instead of core memory for P, Q, R, S and T, if it can be determined that they will not be used again. (This determination is very difficult, principally because of GO TOs, which is one of the reasons why it is very difficult to write optimizing compilers for languages such as FORTRAN.)

In the graph representation, an instruction can be executed as soon as all the instructions with arrows pointing into it have completed. On a multiprocessor system, we could allocate a processor for each instruction, with appropriate instructions (such as semaphore operations¹³) to enforce the sequencing constraints, but execution would be hopelessly inefficient because the parallelism of this example is far too "fine grained" for a multiprocessor. The overhead in the process scheduling and in the *wait* and *signal* instructions would be many times greater than the execution time of the arithmetic operations. A data flow computer, on the other hand, is designed to execute algorithms with such a fine grain of parallelism efficiently. In these machines, parallelism is exploited at the level of individual instructions, as in the previous example, and at all coarser levels as well; in most programs there are typically many parts, often far removed from each other, at which computation may proceed simultaneously.

To exploit parallelism at all levels, the instruction sequencing constraints must be deducible from the program itself. Let us refer again to the previous program to see how this may be done.

The sequencing constraints in Figure 1 are given by arrows. It is not difficult to see that these arrows coincide with data transmission from one instruction to its successor through variables. In fact, the graph could be redrawn with the arrows labeled by the variables that they represent, as in Figure 2.

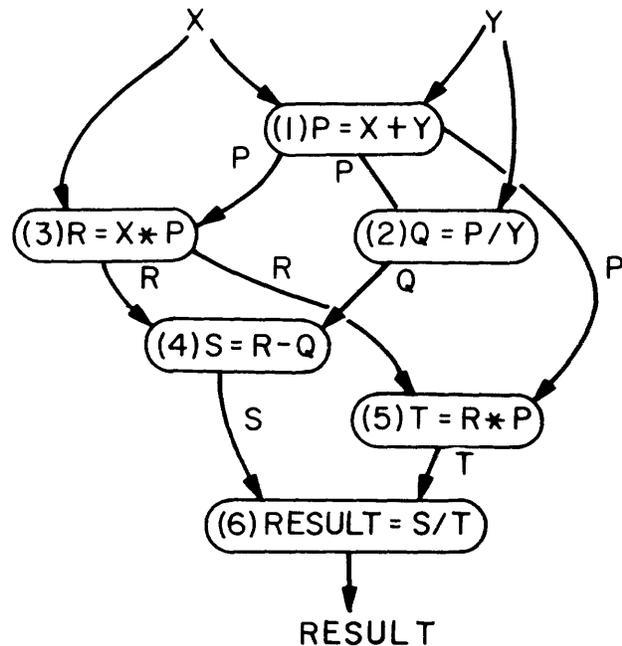


Figure 2

In a data flow computer, the machine level program is represented essentially in this form—a graph with pointers between nodes, the pointers representing both the flow of data and the sequencing constraints. Each instruction is kept in a hardware device (an extremely simple "processor") that is capable of "firing" or executing an instruction when all of the necessary data values have arrived, and sending the result to the processors that hold destination instructions.*

The programming language for a data flow computer must therefore satisfy two criteria:

1. It must be possible to deduce the data dependencies of the program operations.
2. The sequencing constraints must always be exactly the same as the data dependencies, so that the instruction firing rule can be based simply on the availability of data.**

There are two general properties of a language which make it possible to meet these criteria: locality of effect and freedom from side effects.

* Although the language concepts presented in this paper assume that the computer exploits parallelism at a microscopic level, not all "data flow" or "data driven" computers do so. Designs of data flow computers that exploit parallelism only at the subroutine level may be found in References 10 and 24.

** Not all designs for data flow computers accept the second of these criteria or its consequences. The LAU language⁹ is intended for execution on a data flow computer, but it was designed to support data base updating and retrieval, so it has side effects on certain operations. The sequencing of these operations must therefore be constrained by means other than data dependencies, and so it does not satisfy the second criterion. The extra constraints in LAU are specified by path expressions⁷ written into the source program.

LOCALITY OF EFFECT

Locality of effect means that instructions do not have unnecessary far-reaching data dependencies. For example, the FORTRAN program fragment given previously appears to use variables P, Q, R, S and T only as temporaries. A similar program fragment appearing elsewhere in the program might use the same temporaries for some unrelated computation. The logic of the program might be such that the two fragments could be executed concurrently were it not for this overuse of names. (Unfortunately, many conventional languages encourage this style of programming.) Any attempt to execute the program fragments concurrently would be impossible because of the apparent data dependencies arising from overuse of these temporaries, unless the compiler can deduce that the conflict is not real and remove it by using different sets of temporaries.

In languages such as FORTRAN and PL/I, this is not so easy to determine. A reference to a variable in one part of the program does not necessarily imply dependence on the value computed in another part—the variable might be overwritten before it is next read. Careful analysis is required to determine whether a variable is actually transmitting data or is “dead.” This analysis is made much more difficult if unrestricted GO TOs or other undisciplined control structures are allowed.

The problem can be simplified by making every variable have a definite “scope,” or region of the program in which it is active, and carefully restricting the entry to and exit from the blocks that constitute scopes. It is also helpful to deny procedures access to any data items that are not transmitted as arguments, though this is not really necessary if global variables are avoided and procedure definitions are carefully “block structured” as in PASCAL.

SIDE EFFECTS

Freedom from side effects is a necessary property to ensure that the data dependencies are the same as the sequencing constraints. It is much more difficult to achieve than locality of effect. This is because locality only requires superficial restrictions on the language, whereas freedom from side effects requires fundamental changes in the way the language’s “virtual machine” processes data.

Side effects come in many forms—the most well known examples are procedures that modify variables in the calling program, as in the following PASCAL example:

```
procedure GETRS(X, Y:real); (* RS is declared in
begin RS:=X*X+Y*Y          an outer block *)
end;
```

Absence of global or “common” variables and careful control of the scopes of variables make it possible for a compiler to prohibit this sort of thing, but a data flow computer imposes much stricter prohibitions against side effects—a procedure may not even modify its own arguments. In fact, in a sense nothing may ever be modified at all.

To determine what kind of prohibitions against side effects are needed to achieve concurrent computation, we must examine programs that manipulate structured data such as arrays or records, since the problem does not arise when only simple data values are used.

Consider the following procedure which modifies its arguments by a conventional “call by reference” mechanism. SORT2 is a procedure to sort two elements, J and J+1, of array A into ascending order by exchanging them if necessary.

```
procedure SORT2(var A:array[1..10] of real;
J:integer);
var T:real;
begin if A[J]>A[J+1] then begin
T:=A[J];
A[J]:=A[J+1];
A[J+1]:=T;
end
```

end;

- (1) SORT2(AA, J);
- (2) SORT2(AA, K);
- (3) P:=AA[L];

Statements 1 and 2 might interfere with each other and with Statement 3. Since the values of J, K, and L are not known to the compiler, it must assume that the statements will conflict, and execute them in the exact order specified. Any attempt at parallel execution might result in the incorrect results, depending on J, K, L, and unpredictable fluctuations in timing.

A phenomenon known as “aliasing” makes the problem even more difficult. This occurs when different formal parameters to a procedure refer to the same actual parameter, that is, they are “aliases” of each other:

```
procedure SORTREAD(var A, B:array[1..10] of real;
I, J:integer);
begin SORT2(A, I); (* SORT2 is defined above *)
RESULT:=B[J];
end;
```

In this program it would appear that, since A and B are different arrays, the invocation of SORT2 and the reference to B[J] could proceed concurrently. However, if this were part of a larger program and SORTREAD were invoked in the statement

```
SORTREAD(Q,Q,M,N);
```

the arrays A and B would actually be the same. Languages such as FORTRAN and PL/I, in which external procedures are not available to the compiler when the calling program is being compiled, make the problem harder still. Facilities for manipulating data structures by pointers, such as the “pointer” data type in PL/I and the record manipulating operations of PASCAL make it possible for all of these problems to arise without using procedures.

Even if procedures and pointers are not used, the sequencing constraints may be far from clear, as in

- (1) A[J]:=3;
- (2) X:=A[K];

If the convention is made that any statement modifying any element of an array constitutes a "writing" of the array, then Statement 1 clearly passes array A to Statement 2. But then a statement such as the assignment in

```
for J:=1 to 10 do
  A[J+1]:=A[J]+1;
```

depends on itself!

The inescapable conclusion is that, if arrays and records exist as global objects in a memory and are manipulated by statements and passed as procedure parameters, it is virtually impossible to tell, when an array element is modified, what effects that modification may have elsewhere in the program.

One way to solve some of these problems is to use "call by value" instead of the more common "call by reference." This solves the aliasing problem and the problem of procedures modifying their arguments. In a "call by value" scheme, a procedure copies its arguments (even if they are arrays). This way it can never modify the actual argument in the calling program. Call by reference has traditionally been used instead of call by value because it is a more natural way of thinking about computation (and is more efficient) on von Neumann computers.

APPLICATIVE LANGUAGES

For data flow languages, a scheme is used which goes far beyond call by value: all arrays are *values* rather than *objects*, and are treated as such at all times, not just when being passed as procedure arguments. Arrays are not modified by subscripted assignment statements such as

```
A[J]:=S;
```

Instead, they are processed by *operators* which create new array values. The simplest operator to perform the nearest equivalent of modifying an array takes three arguments—an array, an index, and a new data value. The result of the operation is a new array, containing the given data value at

```
function SORT2(A, J)
  if A[J]<A[J+1] then A
    else (A[J:A[J+1]])[J+1:A[J]]
  end
end
% No temporary needed during this
% exchange because A is not modified.
```

Note that the array construction operator may be composed with itself and with other operators in the same way as arithmetic operators.

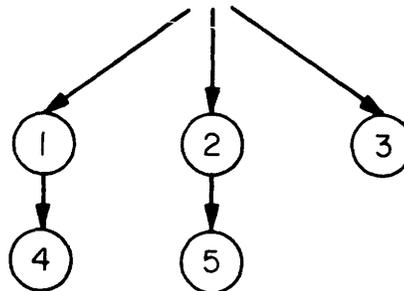


Figure 3

the given index, and the same data as the original array at all other indices.

In the VAL language,¹ the syntax for this elementary operator is

```
A[J: S]
```

In the ID language,³ it is

```
A+[J]S
```

This operation *does not modify* its argument. Hence, in this VAL program:

- (1) B:=A[J:S];
- (2) C:=A[K:T];
- (3) P:=A[L];
- (4) Q:=B[M];
- (5) R:=C[N];

Statements 1 and 2 do not interfere with each other or with array A. Statement 3 may be executed immediately, whether 1 and/or 2 have completed or not, since they would have no effect on Statement 3 anyway. Statement 4 can be executed as soon as Statement 1 completes, whether Statement 2 has completed or not. The sequencing constraints are shown in Figure 3.

Note that this situation is similar to the one in the simple FORTRAN example of the first section. The sequencing constraints are exactly the same as the data dependencies, which is the property we seek for data flow.

An operator-based handling of arrays and records automatically accomplishes call by value. As in a call by value scheme, a routine such as SORT2 would not accomplish its purpose. SORT2 must return the new array as its value. It could be written in VAL (omitting type declarations) as:

In conventional languages, procedures do most, if not all, of their work through side effects—a procedure might be designed to alter dozens of variables in the calling program.

Functions, on the other hand, are typically limited to returning a single value, which typically may not be an array or record. To make functions as powerful and flexible as procedures in conventional languages, applicative languages often allow functions to return several values, or entire records or arrays, or both. If a function returns several values, its call can be used in a "multiple assignment" such as

```
X,Y,Z:=FUNC(P,Q,R) % FUNC returns 3 values
```

Treating arrays and records as values instead of objects is perhaps the most profound difference in the way people must think when writing programs in data flow languages instead of conventional languages. The customary view is of arrays as objects residing in static locations of memory, and being manipulated by statements that are executed in some sequence. As we have seen, this view is incompatible with detection of parallelism among the statements. The correct view for data flow is of arrays and records as values manipulated by operations just as simple values (integers, reals) are. Then the parallelism among the operators can be deduced from the data dependencies just as for simple values.

The value-oriented approach to arrays is confusing to some at first, but it need not be. An integer array should be thought of as a string of integers, just as an integer can be thought of as a string of digits. If J has the value 31416, the statement

```
K:=J-400;
```

leaves K equal to 31016; no programmer would expect the value of J to be affected. If A is an array with elements [3,1,4,1,6], the statement

```
B:=A[3:0];
```

is completely analogous; it leaves B with elements [3,1,0,1,6] and of course does not change A.

Languages which do all processing by means of operators applied to values are called *applicative* languages, and are thus the natural languages for data flow computation. The earliest well known applicative language implemented on a computer is LISP.²⁰ (It is applicative only if RPLACA, RPLACD, and all other functions with side effects are avoided; this subset of the language is often called "pure" LISP.) The connection between applicative languages and the detection of parallelism has been reported by Tesler and Enea²⁵ and by Friedman and Wise.¹⁴ The development of LISP and other applicative languages, and the Tesler/Enea paper, all predate the data flow computer concept by several years. The concept of computation by applicative evaluation of expressions goes back to the invention of the lambda calculus in 1941.⁸

EFFICIENCY CONSIDERATIONS IN APPLICATIVE LANGUAGES

Applicative languages have recently given rise to controversy concerning their time efficiency in practical situations.

It is claimed that many algorithms cannot be executed as efficiently in applicative languages as in conventional statement-oriented languages. The issue is only one of exploiting parallelism, not any fundamental limitation in the computing power of applicative systems. This is because any program written for a conventional von Neumann computer can be rewritten in an applicative language by treating the entire memory space as one array. Statements that manipulate the memory then become operations on that array. The array must be passed from each operation to the next, so execution in such an applicative system must be strictly sequential—no parallelism can be exploited. However, in the original program written in a conventional language, a knowledgeable programmer might be able to explicitly specify parallelism, making the program run more efficiently than in the applicative system.

Consider the conventional program

- (1) A[J]:=S
- (2) A[K]:=T;
- (3) P:=A[M];
- (4) Q:=A[N];

If the programmer knows that the set of indices for the array A can be divided into two disjoint sets, with J and M in one set and K and N in the other, then Statements 1 and 2 could be executed simultaneously, 3 would only need to follow 1, and 4 would only need to follow 2. If the programmer has the ability to control parallelism explicitly (say, by semaphores, monitors, or path expressions), he could specify exactly those constraints. This is not possible in a data flow language without some additional mechanism. However, the problem can often be avoided by dividing the array into separate parts, manipulating them separately, and combining them only when necessary. For example, suppose array A1 contains only those elements that J and M are known to index, and A2 contains those that K and N index. Then

```
B1:=A1[J:S];
B2:=A2[K:T];
P:=B1[M];
Q:=B2[N];
```

exploits the parallelism exactly. There are other ways of overcoming the tendency for arrays to limit parallelism, which will be discussed later. The question of how to exploit parallelism in applicative languages for a wide variety of programming problems is an active area of research.

DEFINITIONAL LANGUAGES AND THE SINGLE ASSIGNMENT RULE

Having accepted an applicative programming style and a value-oriented rather than object-oriented execution model, we next examine the implications of this style upon the meaning of assignment statements.

Except in iterations, (which will be discussed later), an assignment statement has no effect except to provide a value

and bind that value to the name appearing on its left hand side. The result of the assignment is accessible only in subsequent expressions in which that name appears. If the language uses blocks in which all variables are local to the block for which they are declared, the places where a variable is used can be determined by inspection. If the expression on the right hand side of an assignment is substituted for the variable on the left hand side in all places where that variable appears within its scope, the resultant program will be completely equivalent.

$$\begin{array}{ll} S:=X+Y; & \\ D:=3*S; & \text{is equivalent to } D:=3*(X+Y); \\ E:=S/2+F(S); & E:=(X+Y)/2+F(X+Y); \end{array}$$

(The program on the left is clearly more efficient, requiring only two additions instead of four. We are not proposing that the substitution be made in practice.)

Now this situation is the same as a system of mathematical equations. If a system of equations contains " $S=X+Y$ ", it is clear that " $3*S$ " and " $3*(X+Y)$ " are equivalent. Hence the statement

$$S:=X+Y;$$

means the same thing in the program that the equation " $S=X+Y$ " means in a system of equations, namely that, in the scope of these variables, S is the sum of X and Y . The correspondence can be thought of as holding for all time; it is not necessary to think of the statements as being executed at particular instants. (In fact, perhaps the word "variable" is an inappropriate term.) Of course, the addition of X and Y to form S must take place before any of the operations that use S can be performed, but the programmer doesn't need to be directly concerned with this.

Hence the statement $S:=X+Y$ should be thought of as a *definition*, not an assignment. Languages which use this interpretation of assignments are called *definitional languages* as opposed to conventional *imperative languages*. Such languages are well suited to program verification because the assertions one makes in proving correctness are exactly the same as the definitions appearing in the program itself. In conventional languages one must follow the flow of control to determine *where* in the program text assertions such as " $S=X+Y$ " are true, because the variables S , X , and Y can be changed many times. Assertions must therefore be associated with points in the program.

In a definitional language the situation is extremely simple: If a program block contains the statement

$$S:=X+Y;$$

then the assertion $S=X+Y$ is true. Of course, care must be taken to prevent circular definitions such as

$$\begin{array}{l} X:=Y; \\ Y:=X+3; \end{array}$$

We can either have the compiler allow definitions to ap-

pear in any order as long as there is some consistent order, or we can require them to appear in a consistent order. A consistent order is one in which no name is referred to (on the right hand side of a definition) before it is defined. This condition is easily checked by a compiler. So the actual proof rule is: If a program block contains the statement

$$S:=X+Y;$$

and the program compiles correctly, then $S=X+Y$ is true. Strictly speaking, it is only true in the statements after the one defining S , but, since S does not appear in any earlier statements, the assertion can be treated as being true throughout the block.

The power of definitional languages for program verification is well known outside of the data flow field. LUCID⁴ is an example of an applicative definitional language designed expressly for ease of program verification.

There is a problem that could ruin the elegance of definitional languages—multiple definition of the same name. Definitional languages almost invariably obey the *single assignment rule*: A name may appear on the left hand side of an assignment (definition) only once within its scope. The single assignment rule prevents program constructs which imply mathematical abominations, such as

$$J:=J+1;$$

Since the appearance of J on the right hand side precedes the definition of J , it implies an inconsistent statement sequencing that the compiler would diagnose. The prevention of such abominations is of course necessary if the definitions in the program are to be carried directly into assertions used in proving correctness, because the assertion " $J=J+1$ " is absurd.

It is not actually necessary that a data flow language conform to the single assignment rule. A data flow language with multiple assignments could be designed in which the scope of a variable extends only from its definition to the next definition of the same variable. The next definition in effect introduces a new variable that simply happens to have the same name. A program written in such a way can be easily transformed into one obeying the single assignment rule: simply choose a new name for any redefined variable, and change all subsequent references to the new name. However, the advantages of single assignment languages, namely, clarity and ease of verification, generally outweigh the "convenience" of re-using the same name.

ITERATIONS

There remains one area in which statements such as

$$I:=I-1;$$

or

$$A:=A[J:X+Y];$$

seem to be necessary, explicitly or implicitly, in conven-

tional languages, and that is in iterations. The technique of renaming variables to make a program conform to the single assignment rule only works for straight-line programs: If a statement appears in a loop, renaming its variables will not preserve the programmer's intentions. For example:

```
for I:=1 to 10 do      cannot be      for I:=1 to 10 do
  J:=J+1;              transformed to    J1:=J+1;
```

If the language allows general GO TOs, with the resulting possibility of complex and unstructured loops, the problem is indeed difficult. However, data flow languages generally have no GO TO statement, and require loops to be created only by specific program structures such as the "while. . .do. . ." and similar statements found in PL/I and PASCAL. This makes the problem easy to solve, and makes it possible to give iterations a very simple and straightforward meaning.

To develop the data flow equivalent of a "while. . .do. . ." type of iteration, we must consider what the "do" part of such a structure contains. Since there are no side effects, the only state information in an iteration is the bindings of the loop variables, and the only activity that can take place is the redefinition of those variables through functional operators. An iteration therefore consists of

1. Definitions of the initial values of the loop variables.
2. A predicate to determine whether, for any given values of the loop variables, the loop is to terminate or to cycle again.
3. If it is to terminate, some expression giving the value(s) to be returned. These values typically depend on the current values of the loop variables.
4. If it is to cycle again, some expressions giving the new values to be assigned to the loop variables. These also typically depend on the current values of the loop variables.

An iteration to compute the factorial of N could be written (omitting type declarations) in VAL as follows:

```
for I, J:=N, 1;          % Give loop variables I and
                        % J initial
                        % values N and 1
                        % respectively. I will
                        % count downward. J will
                        % keep
                        % accumulated product.
do if I=0                % Decide whether to
                        % terminate.
  then J                 % Yes, final result is current
                        % J.
  else iter I, J:=I-1, J*I;
                        % No, compute new values
                        % of I and J,
                        % and cycle again.
end
```

It could be written in ID as follows:

```
initial I←N; J←1
while I≠0 do
  new I←I-1;
  new J←J*I;
return J
```

Its representation in LUCID is similar to that in ID.

Although the values of the loop variables do change, they change only between one iteration cycle and the next. The single assignment rule, with its prohibition against things like "I=I-1" is still in force within any one cycle. All redefinitions take place precisely at the boundary between iteration cycles (though they need not actually occur simultaneously). This is enforced in VAL by allowing redefinitions only after the word *iter*, which is the command to begin a new iteration cycle. In ID and LUCID, the "new" values become the "current" values at the boundary between cycles.

Since the single assignment rule is obeyed, and names have single values, within any one iteration cycle the mathematical simplicity of assertions about values still exists. The assertions typically take the form

"In any cycle, $S=X+Y$ "

Assertions used in proving correctness of an iteration are usually proved inductively. Because assertions take such a simple form, such proofs are usually simpler than in conventional languages. For example, the assertion

$I \geq 0$ and $J*(I!) = N!$

is used to prove correctness of the previous program. The basis of the induction is that it is true for the initial values $I=N$ and $J=1$. (We assume $N \geq 0$.) The induction step is that, if another cycle is started with the values $I-1$ and $J*I$, they will obey the assertion, that is,

$I-1 \geq 0$ and $(J*I)*((I-1)!) = N!$

which is clearly true if we observe that a new cycle will only be started if $I > 0$ and hence $I-1 \geq 0$.

ERRORS AND EXCEPTIONAL CONDITIONS

Locality of effect requires that errors such as arithmetic overflow be handled by *error values* rather than by program interruptions or manipulation of global status flags. If an error occurs in an operation, that fact must be transmitted to the destinations of that operation and nowhere else. This can be easily accomplished by enlarging the set of values to include error values such as overflow, underflow, or zero-divide.

If the intention is to abort the computation when an error occurs, this can be achieved by making the error values *propagate*—if an argument to an arithmetic operation is an

error, the result is an error. When an error propagates to the end of an iteration body, that iteration always terminates rather than cycling again. In this way the entire computation will come to a stop quickly, yielding an error value as the result. If the computer keeps a record of every error generation and propagation, that record will provide a detailed trace of when and where the error occurred, and what iterations and procedures were active.

If the intention is to correct an error when it occurs (perhaps keeping a list of such errors in some array), that can be accomplished through operations that test for errors. For example, a program to set Z to the quotient of X and Y, or to zero if an error occurs, could be written in VAL as follows:

```
ZZ:=X/Y;
Z:=if is__error(ZZ) then 0 else ZZ end;
```

METHODS OF OBTAINING MAXIMUM PARALLELISM

To achieve the greatest parallelism, it is necessary that computations not be performed sequentially unless necessary. The iteration constructs described previously imply a sequential execution of the various cycles. If the values of the iteration variables in one cycle depend on those of the previous cycles (as they do in the factorial example given previously), nothing can be done about it, although a data flow computer can often execute *part* of a cycle before the previous one has completed. If the values in one cycle do *not* depend on those of the previous cycles, the cycles can be performed in parallel. In VAL this is accomplished with a *forall* program construct which does not allow one cycle to depend on the others, and directs the computer to perform all cycles simultaneously. In ID the same effect is achieved automatically by tagging the values of the iteration variables with their cycle number, and allowing them to be processed out of sequence, or simultaneously, whenever they do not depend on each other.

Another potential "bottleneck" in data flow computation is the requirement that all elements of an array be computed before any element of that array may be accessed. If function "F" creates an array value by filling the array one element at a time, and then passes the array to "G," which reads the elements one at a time, G cannot begin until F completes. In many instances this delay is unnecessary, and various techniques have been proposed for eliminating it without departing from the principle that the sequencing constraints are exactly the data dependencies. One method, mentioned in the fifth section, is to explicitly divide the array into pieces, or use separate data items instead of an array. This method is quite general, but it requires specific calculation by the programmer of which parts of the array are needed at which time.

Streams

A method of overcoming the array bottleneck is the use of "streams."^{2-11,26} A stream may be thought of as an array

that is fragmented in time and is processed one element at a time.

In the previous example of F creating an array one element at a time and passing it to G, a stream would be the natural way to do this. G would receive each element as soon as F created it, so G would be processing the Nth element while F computes the N+1st, resulting in parallel "pipelined" computation of F and G.

The constraint that stream elements be created and consumed in strict sequence may be enforced by placing some restrictions on the source program to prevent "random access." A program to manipulate streams may be written in a recursive style,^{11,26} in which a stream is treated like a list in LISP, or in an iterative style³ in which the rebinding of an iteration variable denoting a stream causes that stream to advance to the next element. Either method enforces the sequencing constraint if certain rules are followed regarding the permissible recursions or iterations.

When streams are viewed not as temporally separated arrays but as sequences of data items, functions that process streams have a few interesting and useful properties that pure mathematical functions do not have: they can emit more (or fewer) outputs than their inputs, and they can exhibit "memory" from one element to another. This makes stream functions suitable for "on-line" applications such as updating a data base. Stream functions are also useful for operations normally performed by coroutines, such as a function to remove all (newline) characters from its input, or insert a (newline) after every 80 characters. A stream function is equivalent to a coroutine that communicates by transmission and reception of data values. A data flow program using streams is a network of parallel communicating coroutines, a computational model that has been of some theoretical interest in the last few years.¹⁶⁻¹⁸

Streams and input/output operations

Streams form a natural mechanism for handling input and output. By treating the sequence of characters read in from an external medium as a stream, it is possible for a program to operate on the data as it is read in. The program can generate an output stream, which is printed as it is produced.

CONCLUSION

There have recently been calls for the abandonment of the traditional "von Neumann" computer architecture as a good way of realizing the enormous potential of VLSI technology.⁵ There has also been widespread recognition that proper language design is essential if the high cost of software is to be brought under control, and that most existing languages are seriously deficient in this area.

Fortunately, the implications of these trends for language design are similar—languages must avoid an execution model (the "von Neumann" model) that involves a global memory whose state is manipulated by sequential execution of commands. Such a global memory makes realization of

the potential of VLSI technology difficult because it creates a "bottleneck" between the computer's control unit and the memory. Languages that use a global memory in their execution model also exacerbate the software problem by allowing program modules to interact with each other in ways that are difficult to understand, rather than through simple transmission of argument and result values. Future language designs based on concepts of applicative programming should be able to help control the high cost of software and to meet the needs of future computer designs.

REFERENCES

1. Ackerman, W. B., and J. B. Dennis, "VAL—A Value-Oriented Algorithmic Language: Preliminary Reference Manual," *Computation Structures Group*, Laboratory for Computer Science, MIT, Cambridge, Massachusetts.
2. Arvind, and K. P. Gostelow, *Dataflow Computer Architecture: Research and Goals*, Department of Information and Computer Science (TR 113), University of California-Irvine, Irvine, California, February 1978.
3. Arvind, K. P. Gostelow and W. Plouffe, *The (Preliminary) Id Report*, Department of Information and Computer Science (TR 114), University of California-Irvine, Irvine, California, May 1978.
4. Ashcroft, E. A., and W. W. Wadge, "Lucid, a Nonprocedural Language with Iteration," *Communications of the ACM*, Vol. 20, No. 7, July 1977, pp. 519-526.
5. Backus, J., "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs," *Communications of the ACM* Vol. 21, No. 8, August 1978, pp. 613-641.
6. Brinch-Hansen, P., "The Programming Language Concurrent Pascal," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, June 1975, pp. 199-207.
7. Campbell, R. H., "Path Expressions: A Technique for Specifying Process Synchronization," Department of Computer Science (Report UIUCDCS-R-77-863), University of Illinois at Urbana-Champaign, Urbana, Ill., 1977.
8. Church, A., "The Calculi of Lambda-Conversion," *Annals of Mathematical Studies*, Vol. 6, Princeton University Press, 1951.
9. Comte, D., G. Durrieu, O. Gelly, A. Plas and J. C. Syre, "Parallelism, Control and Synchronization Expressions in a Single Assignment Language," *SIGPLAN Notices*, Vol. 13, No. 1, January 1978, pp. 25-33.
10. Davis, A. L., "The Architecture and System Method of DDM1: A Recursively Structured Data Driven Machine," *Proceedings of the Fifth Annual Symposium on Computer Architecture, Computer Architecture News*, Vol. 6, No. 7, April 1978, pp. 210-215.
11. Dennis, J. B., "A Language Design for Structured Concurrency," *Design and Implementation of Programming Languages: Proceedings of a DoD-Sponsored Workshop* (J. H. Williams and D. A. Fisher, Eds.), *Lecture Notes in Computer Science*, Vol. 54, October 1976. Also, *Computation Structures Group*, Note 28-1, February 1977. Laboratory for Computer Science, MIT, Cambridge, Massachusetts.
12. Dennis, J. B., D. P. Misunas and C. K. C. Leung, "A Highly Parallel Processor Using a Data Flow Machine Language," *Computation Structures Group*, Memo 134, Laboratory for Computer Science, MIT, Cambridge, Massachusetts, January 1977. To appear in *IEEE Transactions on Computers*.
13. Dijkstra, E. W., "Cooperating Sequential Processes," *Programming Languages* (F. Genuys, Ed.), Academic Press, New York, New York, 1968.
14. Friedman, D. P., and D. S. Wise, "The Impact of Applicative Programming on Multiprocessing," *Proceedings of the 1976 International Conference on Parallel Processing* (P. H. Enslow, Ed.), August 1976, pp. 263-272.
15. Gurd, J., I. Watson and J. Glauert, "A Multilayered Data Flow Computer Architecture," Department of Computer Science, University of Manchester, Manchester, England, July 1978.
16. Hoare, C. A. R., "Communicating Sequential Processes," *Communications of the ACM*, Vol. 21, No. 8, August 1978, pp. 666-677.
17. Kahn, G., "The Semantics of a Simple Language for Parallel Programming," *Information Processing 74: Proceedings of the IFIP Congress 74* (J. L. Rosenfeld, Ed.), 1974, pp. 471-475.
18. Kahn, G., and D. MacQueen, "Coroutines and Networks of Parallel Processes," *Information Processing 77: Proceedings of IFIP Congress 77* (B. Gilchrist, Ed.), August 1977, pp. 993-998.
19. Lawrie, D. H., "GLYPNIR Programming Manual," ILLIAC IV Document no. 232, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Ill., August 1970.
20. McCarthy, J., et. al., "LISP 1.5 Programmer's Manual," MIT Press, 1966.
21. Miranker, G. S., "Implementation of Procedures on a Class of Data Flow Procedures," *Proceedings of the 1977 International Conference on Parallel Processing* (J. L. Baer, Ed.), August 1977, pp. 77-86.
22. Patil, S. S., R. M. Keller and G. Lindstrom, "An Architecture for a Loosely-Coupled Parallel Processor (Draft)," Department of Computer Science (UUCS-78-105), University of Utah, Salt Lake City, Utah, July 1978.
23. Plas, A., D. Comte, O. Gelly, and J. C. Syre, "LAU System Architecture: A Parallel Data Driven Processor Based on Single Assignment," *Proceedings of the 1976 International Conference on Parallel Processing* (P. H. Enslow, Ed.), August 1976, pp. 293-302.
24. Rumbaugh, J. E., "A Data Flow Multiprocessor," *IEEE Transactions on Computers*, Vol. C-26, No. 2, February 1977, pp. 138-146.
25. Tesler, L. G., and H. J. Enea, "A Language Design for Concurrent Processes," *Proceedings of the AFIPS Conference*, Vol. 32, 1968, pp. 403-408.
26. Weng, K.-S., *Stream-Oriented Computation in Recursive Data Flow Schemas*, Laboratory for Computer Science (TM-68), MIT, Cambridge, Massachusetts, October 1975.

1979 NATIONAL COMPUTER CONFERENCE COMMITTEES

PROGRAM COMMITTEE

Chairman

Richard E. Merwin
George Washington University
Washington, DC

Noah Prywes
Moore School of Electrical
Engineering
Philadelphia, PA

Data Base Management

P. Bruce Berra
Syracuse University
Syracuse, NY

Publications Liaison

Patricia Murray
Library of Congress
Washington, DC

Robert L. White
U.S. Census Bureau/DUSD
Washington, DC

Software Technology

Richard E. Nance
Virginia Polytechnic Institute & State
University
Blacksburg, VA

Advisor

Stanley Winkler
IBM Corporation
Armonk, NY

Social Implications
Robert P. Campbell
Department of the Army
Washington, DC

Special Topics

Karen A. Duncan
The MITRE Corporation
McLean, VA

Applications

Paul F. Bohn
Johns Hopkins University
Laurel, MD

Anita Cochran
Bell Laboratories
Murray Hill, NJ

Donald Hollis
Chase Manhattan Bank, N.A.
New York, NY

David C. Hartmann
Library of Congress
Washington, DC

Architecture
Stephen F. Lundstrom
Burroughs Corporation
Paoli, PA

William Wewer
Sutherland, Asbill & Brennan
Washington, DC

Stephen R. Kimbleton
National Bureau of Standards
Washington, DC

Chris Tomlinson
Burroughs Corporation
Paoli, PA

Program Committee Staff

Louisa Hull
Kristine Jensch
William D. Passero
Susan Seiderman
Jacqueline T. Zanca

CONFERENCE STEERING COMMITTEE

Conference Chairman

Merlin G. Smith
IBM T. J. Watson Research Center
Yorktown Heights, NY

Vice Chairman

Robert C. Spieker
AT&T
Piscataway, NJ

Advisors

Stanley Winkler
IBM Corp.
Armonk, NY

Program Chairman

Richard E. Merwin
George Washington University
Washington, DC

Assistant Chairwoman

Anita Cochran
Bell Laboratories
Murray Hill, NJ

Carl Hammer

Sperry Univac
Washington, DC

AFIPS Headquarters Liaison

Gerard F. Chiffriller
AFIPS
Montvale, NJ

Operations

Lori Capodanno
Bell Laboratories
Holmdel, NJ

Society Liaison

Robert L. White
U.S. Census Bureau/DUSD
Washington, DC

Communications

Arnold P. Smith
IBM Corporation
White Plains, NY

Personal Computing

Richard A. Kuzmack
Mathematica, Inc. (MATHTECH)
Arlington, VA

Special Media

Roger M. Firestone
Sperry Univac
Blue Bell, PA

Fiscal

James F. Welch
F. W. Woolworth Co.
New York, NY

Professional Development

William A. Baker
Aetna Life & Casualty
Hartford, CT

NCCC Liaison

Irwin J. Sitkin
Aetna Life & Casualty
Hartford, CT

COMMUNICATIONS COMMITTEE

Chairman

Arnold P. Smith
IBM Corporation
White Plains, NY

Charles Floto
Phillips Publishing, Inc.
Washington, DC

Herb Seidensticker
Combustion Engineering
Stamford, CT

B. Garland Cupp
American Express Card Division
New York, NY

Betty A. Henderson
IBM Corporation
White Plains, NY

A. E. Smith
IBM Corporation
North Tarrytown, NY

Philip H. Dorn
New York, NY

James D. Tupac
PRC
McLean, VA

Robert L. White
U.S. Census Bureau
Washington, DC

George Field
Honeywell Information Systems, Inc.
Waltham, MA

Tom Pritchard
IBM Corporation
White Plains, NY

Thomas C. White
AFIPS
Montvale, NJ

FISCAL COMMITTEE

Chairman

James F. Welch
F. W. Woolworth Co.
New York, NY

Rita M. Lane
Chemical Bank
New York, NY

OPERATIONS COMMITTEE

Chairwoman

Lori Capodanno
Bell Laboratories
Holmdel, NJ

Vice Chairman

Doug Rochester
Manufacturers Hanover Trust Co.
New York, NY

Thomas A. D'Auria
City of New York
New York, NY

George Brucia
New York Telephone
New York, NY

Ed DuCasse
Pace University
New York, NY

Harriet Shair
The Computer Corner
White Plains, NY

Claire Chase
City of New York
New York, NY

Mike Goldberg
Pace University
New York, NY

Robert Soudant
New York Telephone
New York, NY

PERSONAL COMPUTING COMMITTEE

Chairman

Richard A. Kuzmack
Mathematica, Inc. (MATHTECH)
Arlington, VA

Charles Floto
Phillips Publishing, Inc.
Washington, DC

Joe Kasser
Comsat Laboratories
Silver Spring, MD

Vice Chairwoman

Burchenal Green
Creative Computing Magazine
Morristown, NJ

Edward J. Fox
Department of Defense
Fort George G. Meade, MD

Jay P. Lucas
U.S. Patent and Trademark Office
Washington, DC

Russell Adams
U.S. Patent and Trademark Office
Washington, DC

Portia Isaacson
Electronic Data Systems Corp.
Dallas, TX

Harriet Shair
The Computer Corner
White Plains, NY

PROFESSIONAL DEVELOPMENT SEMINARS COMMITTEE

Chairman

William A. Baker
Aetna Life & Casualty
Hartford, CT

Richard K. Edwards
Aetna Life & Casualty
Hartford, CT

G. Robert Grover
Lever Brothers Company
New York, NY

Ed Marks
Software Design Assoc.
New York, NY

SPECIAL ACTIVITIES

Philately

Ira W. Cotton
National Bureau of Standards
Washington, DC

Science Film Theater

Adrian J. Basili
AT&T
Piscataway, NJ

Pioneer Day

Henry P. Stevenson
AT&T
Basking Ridge, NJ

Special Functions

J. William Woythaler
Summit, NJ

Amazing Micro Mouse Finals

Susan L. Rosenbaum
Honeywell Information Systems
Westfield, NJ

“TODAY AT THE NCC” COMMITTEE

Chairman

Roger M. Firestone
Sperry Univac
Blue Bell, PA

George Field
Honeywell Information Systems
Waltham, MA

Stanley Winkler
IBM Corporation
Armonk, NY

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

OFFICERS AND BOARD OF DIRECTORS

President

Albert S. Hoagland
IBM Corporation
San Jose, CA

Vice President

J. Ralph Leatherman
Hughes Tool Co.
Houston, TX

Secretary

Sylvia Charp
The School District of Philadelphia
Philadelphia, PA

Treasurer

Walter Johnson
Consolidated Paper Co.
Wisconsin Rapids, WI

Executive Director

Robert W. Rector
AFIPS
Montvale, NJ

Association for Educational Data Systems

E. Ronald Carruth
Minneapolis School District
St. Paul, MN

American Society for Information Science

Harold Borko
University of California
Los Angeles, CA

American Institute of Aeronautics and Astronautics

H. Lewis Parker
Comsat Labs.
Clarksburg, MD

American Statistical Association

George Minich
World Bank
Washington, DC

Association for Computational Linguistics

A. Hood Roberts
Roberts Information Service, Inc.
Fairfax, VA

Merlin G. Smith
IBM T. J. Watson Research Center
Yorktown Heights, NY

Association for Computing Machinery

Aaron Finerman
University of Michigan
Ann Arbor, MI

Steven S. Yau
Northwestern University
Evanston, IL

Daniel McCracken
Ossining, NY

Institute of Internal Auditors

William E. Perry
IIA
Altamonte Springs, FL

Stuart Lynn
University of California
Berkeley, CA

Instrument Society of America

Arthur C. Lumb
Proctor & Gamble Co.
Cincinnati, OH

Data Processing Management Association

Delbert Atwood
Utah State Board of Education
Salt Lake City, UT

Society for Industrial and Applied Mathematics

Donald L. Thomsen, Jr.
SIAM Institute for Mathematics
New Canaan, CT

Barry D. Lynch
Mercantile National Bank
Dallas, TX

Society for Information Display

Carlo Crocetti
Rome Air Development Center
Griffiss Air Force Base, NY

Robert Marrigan
Mail Communications, Inc.
Everett, MA

Society for Computer Simulation

Per Holst
Foxboro Co.
Foxboro, MA

IEEE-Computer Society

Tse-Yun Feng
W. Bloomfield, MI

AFIPS Immediate Past President

Theodore J. Williams
Purdue University
West Lafayette, IN

AFIPS EXECUTIVE COMMITTEE

Albert S. Hoagland
IBM Corp.
San Jose, CA

Walter A. Johnson
Consolidated Papers, Inc.
Wisconsin Rapids, WI

J. Ralph Leatherman
Hughes Tool Co.
Houston, TX

Sylvia Charp
School District of Philadelphia
Philadelphia, PA

Aaron Finerman
University of Michigan
Ann Arbor, MI

E. Ronald Carruth
Minnesota School Districts Data Processing Joint Board
(TIES)
St. Paul, MN

Robert Marrigan
Mail Communications, Inc.
Everett, MA

Merlin G. Smith
IBM T. J. Watson Research Center
Yorktown Heights, NY

Per Holst
Foxboro Co.
Foxboro, MA

NATIONAL COMPUTER CONFERENCE BOARD

Chairman and AFIPS Representative
J. Ralph Leatherman
Hughes Tool Company
Houston, TX

Treasurer and AFIPS Representative
Walter A. Johnson
Consolidated Papers, Inc.
Wisconsin Rapids, WI

Secretary and DPMA Representative
Robert Marrigan
Mail Communications, Inc.
Everett, MA

AFIPS Representative
Albert S. Hoagland
IBM Corporation
San Jose, CA

AFIPS Representative
Harold Borko
University of California
Los Angeles, CA

ACM Representative
W. Smith Dorsey
North American Rockwell
Orange, CA

SCS Representative
Carl Malstrom
Clemson University
Clemson, SC

IEEE-CS Representative
Dick Simmons
Texas A & M
College Station, TX

NATIONAL COMPUTER CONFERENCE COMMITTEE

Chairman
Jerry Koory
On-Line Business Systems
San Francisco, CA

Morton M. Astrahan
IBM Research Laboratory
San Jose, CA

Secretary
Irwin J. Sitkin
Aetna Life and Casualty
Hartford, CT

Harvey L. Garner
University of Pennsylvania
Philadelphia, PA

Jeffery D. Stein
On-Line Business Systems, Inc.
San Francisco, CA

Portia Isaacson
Electronic Data Systems Corp.
Dallas, TX

Al Hawkes
Sargent & Lundy Engineers
Chicago, IL

Merlin G. Smith
IBM T. J. Watson Research Center
Yorktown Heights, NY

Russell K. Brown
Moore Paper Companies
Houston, TX

Herbert Safford
GTE Data Services, Inc.
Marina del Rey, CA

INDUSTRY ADVISORY PANEL

Chairman
Frederick M. Hoar
Fairchild Camera and Instrument Co.
Mountain View, CA

S. A. (Sandy) Lanzarotta
Xerox Corporation
El Segundo, CA

Russell Berg
Hewlett-Packard
Palo Alto, CA

William Lonergan
Xerox Development Corp.
Beverly Hills, CA

Gordon Daggy
Fairchild Systems
Santa Fe, CA

Richard Mau
Sperry Rand Corp.
New York, NY

Arnold Lerner
IBM Corp.
Armonk, NY

AFIPS HEADQUARTERS AND CONFERENCE SUPPORT STAFF

Robert W. Rector
Executive Director

Deborah Reinfreid
Receptionist

Diane Lancaster
Secretary

Ann Lonergan
Actg. Manager Finance

Jane Smith
Actg. Manager Administration & Volunteer Support Services

Sue Welch
Bookkeeper

Olive M. Shilland
Secretary

Marie Stewart
Manager, Exhibit Sales

Thomas C. White
Director of Communications

Rita Henderson
Coordinator, Office Automation Conference

Marjorie Greimel
Coordinator of Communications

Deborah Colombo
Secretary

Lorraine J. Stanley
Administrative Assistant

Katherine D. Morrison
Secretary

Rustine Tilton
Secretary

Sandra Rider
Secretary

Gerard F. Chiffrieller
*Director of Conference Operations & Conference
Coordinator*

Alexander D. Roth
Director, AFIPS Washington Office

Pender M. McCarter
Research Associate

Carol Sturgeon
Manager, Conference Operations

Linda Martin
Administrative Assistant

NCC '79 SESSION LEADERS AND ORGANIZERS

Abraham, J. A.
Adams, D.
Ahuja, V.
Albus, J. S.
Ampert, F. R., Jr.
Arvind
Austin, J. M.

Batson, A.
Behringer, J. W.
Berg, J. L.
Berra, P. B.
Bonn, T. H.

Campbell, R. P.
Cancian, M. S.
Capraro, G. T.
Carroll, A. B.
Cartun, J.
Chapin, N.
Coleman, T. G.
Colton, R.
Cooper, D. W.
Cotton, I. W.
Couger, J. D.
Crouch, C. J.

Davida, G. I.
Deane, W. J.
Dennis, J. B.
Doll, D.
Downey, A. T.
Duvall, L. M.

Eilers, S.
Eldridge, I. A.
Emery, J. C.
Engel, G. L.

Fernandez, E. B.
Feustel, E.
Findley, G. W.
Firestone, R. M.
Fox, R. G.
Frailey, D. J.
Fromholz, H.
Fu, K. S.

Gabrieli, E. R.
Gardner, L. B.
Glaseman, S.
Gligor, V. D.
Goel, A. L.
Golland, M.
Gordon, S.
Gregory, F. M.
Grinker, W. S.
Gwin, R. D.

Hall, D. G.
Harmon, G. O.
Hartson, H. R.
Henry, H. E.
Herzfeld, R. W.
Herzog, B.
Hillis, D.
Hiltz, R.

Ichikawa, T.

Jackson, A.
Joslin, E. O.

Kartashev, S. I.
Kartashev, S. P.
Kerschberg, L.
Kimbleton, S. R.
Klein, S.
Kossack, N. E.

Lackmann, J.
Landa, S.
Landis, C. P.
Landstein, J. E. K.
LaPlant, W. P., Jr.
Lefkovits, H. C.
Leventhal, L.
Lewin, W. B.
Lykos, P.
Lyons, L.
Lytle, N.

Madden, P. B.
Mamrak, S. A.
Markus, M. L.
Mason, I. W.
McConnell, V. C.
McGregor, P. V.
McGregor, P.
McKenzie, R. G.
McLeod, D.
McLeod, J.
Medley, D. B.
Michael, G. A.
Mills, D. L.
Mills, R. L.
Mischelevich, D. J.
Morgan, H. L.

Navathe, S.
Nielsen, N. R.
Niimi, B.
Nycum, S. H.

O'Neil, L. D.
Oren, T.

Peake, J. W.
Pengov, R. E.
Peuto, B.
Pitchell, L. A.
Plagman, B. K.
Popek, G. J.

Raben, J.
Ramamoorthy, C. V.
Raskin, J.
Robbins, R. R.
Robertson, L.
Roth, A. D.

Sabeck, R. V.
Safford, H. B.
Sargent, R. G.
Schutzer, D.
Sherin, R. M.
Silver, A. N.
Shneiderman, B.
Spievack, E. B.
Stern, N.
Stevenson, H.
Stotz, R.
Su, S. Y. H.
Swearingen, J. K.

Talavage, J.
Thomas, R. H.
Towsen, J. F.
Tropp, H.
Tunkelang, B.
Turn, R.

Unger, B.

Vairavan, K.
Van Belleghem, D.

Wagner, G. R.
Walker, P. B.
Walker, R.
Walker, S. T.
Wallace, V. L.
Wellar, B. S.
Wesselkamper, T. C.
Westervelt, F.
Wexelblat, R. L.
Winkler, S.
Wohl, A. D.
Wood, H. M.
Worth, R.

Yao, S. B.

Zimmerman, E. K.

NCC '79 SPEAKERS AND PANELISTS

Abramowitz, M.
Aggleton, D.
Ahern, V.
Alexander, M.
Anderson, C.
Anderson, R. E.
Armer, P.
Armstrong, J. E.
Arvind
Atalla, J.
Atkinson, J. B.

Barefoot, D. B.
Barracough, W. G.
Bates, M.
Bennett, A.
Bezilla, R.
Biles, W.
Bishop, R.
Blackshear, B.
Boer, G. L.
Borden, R.
Braun, M.
Brewer, G.
Brieden, H. C.
Brill, K.
Bromberg, H.
Brown, D.
Brown, J. S.
Brown, R. L.
Brown, V. R.
Browne, P. S.
Bryant, R.
Buchta, C. J.
Buffalano, A. C.
Burke, E.
Burns, J.
Burris, H.
Bushkin, A.

Carasso, M. G.
Carlisle, J. H.
Carpenter, R. J.
Carr, F. J.
Caywood, C. S.
Cerf, V. G.
Chaleff, S.
Chapdelaine, R. F.
Chartrand, R. L.
Cheng, E.
Chern, B.
Chestnut, H.
Christy, P.
Churchill, N.
Clemons, E. K.
Clow, F. C.
Clubb, J. M.
Coldren, J. D.

Cole, J. A.
Collette, W.
Cook, R. P.
Corger, P. A.
Couger, J. D.
Cranfill, L. D.
Curran, P.
Curtice, R. M.
Curtis, B.

Davida, G. I.
Davidson, C.
Daya, M.
DeBarge, R. E.
DeBenedictis, L.
DeGross, W.
Deland, E.
DeLutis, T. J.
Denning, P. J.
Dennis, J. B.
Deramo, S.
Dewitt, D. J.
Deutsch, D. R.
Dietrich, W.
Diffie, W.
DiGiuseppe, J. L.
Discount, N.
Donaghue, H.
Duket, S.
Duncan, J. W.
Duncan, K. A.

Edward, R.
Eggert, G. R.
Eibner, J. A.
Elin, L.
Emery, J. C.

Farber, D. J.
Fahy, E.
Feiertag, R. J.
Fenwick, P. R.
Fischer, R. A.
Foster, C. C.
Frahm, J.
Frank, H.
Frank, R. L.
Fromholz, H. J.

Gannon, J.
Gannon, W.
Garrison, K. T.
Gass, S. I.
Gaubatz, D. A.
Gerritsen, R.
Gilb, T.
Gilbert, B. K.
Godwin, G. K.

Goldberg, N.
Goldberg, V.
Golden, J. J.
Goldman, R.
Goodwin, N.
Gray, J. N.
Greenlee, M. B.
Gregory, N.
Greguras, F. M.
Griffis, J. G.
Grimes, J.
Grzebinski, R. D.
Gudes, E.

Halbrecht, H. Z.
Hammer, C.
Hammer, M.
Haner, J. E.
Hanft, R.
Hanratty, M. E.
Hansen, G.
Harden, R.
Hartson, H. R.
Harvey, C.
Hecht, H.
Hellerman, H.
Helm, H. A.
Helmers, C. T., Jr.
Hemmings, D. F.
Henderson, J.
Hendrickson, P. H.
Herbert, W. J.
Herzfeld, R. W.
Hespos, R.
Hewett, J. M.
Higgins, D. A.
Hoberman, R.
Hoekstra, W.
Hoffer, J.
Hoffman, L. J.
Hogan, E. J.
Hokom, R. A.
Holberton, F. E.
Holland, L. D.
Holloway, P.
Hopper, G. M.
Hough, R. R.
Hoversten, E. V.
Howard, J.
Hsiao, D. K.
Hudson, B. G.
Hummer, L.
Hutin, C. T.
Hynes, J. P.

Ichikawa, T.
Iley, H. W.
Ingargiola, G. P.

Issacson, P.
 Itkin, S.

 Jackson, A.
 Jacobson, R. V.
 Jensen, E. D.
 Jerumanis, A.
 Johnson, M.
 Johnson, M. E.
 Johnston, W. F.
 Jones, J. L.
 Jones, R. R.
 Joslin, E. O.

 Kaiser, G.
 Kalow, S. J.
 Kambayashi, Y.
 Karas, J. A.
 Karplus, W. J.
 Keen, P. G. W.
 Kent, S.
 Kerr, E. B.
 Kerschberg, L.
 Kile, F.
 King, O.
 Kinney, E. H.
 Kirschenbaum, F.
 Kiviat, P. J.
 Klassen, D.
 Kleuters, W. J.
 Kolba, M. A.
 Korins, L.
 Krauss, L. I.
 Kreager, P.
 Kruszewski, D.

 Labreque, T.
 Lamont, V. C.
 Lashof, J.
 Lennon, R. E.
 Leon, M.
 Leonard, J.
 Levey, J. G.
 Lewis, J.
 Lieberman, M.
 Lien, D.
 Lipovski, G. J.
 Liskov, B.
 Littell, H.
 Littlewood, B.
 Loren, A. Z.
 Love, T.
 Lowe, A. M.
 Lowenthal, E.
 Luckstone, J.
 Lynch, T.
 Lynn, M. S.
 Lyons, J. F.
 Lyons, J. M.

 Machover, C.
 Madden, P. B.
 Manola, F.
 Marks, G. A.
 Marks, H.
 Martin, D. N.
 McArdle, T.
 McCracken, D.
 McGimsey, K.
 McGovern, J. P.
 McGregor, P.
 McKissick, J., Jr.
 Medley, D. B.
 Meetze, H. W.
 Meserve, W.
 Milgrim, R. L.
 Miller, D. J.
 Miller, S. W.
 Mills, R.
 Mishevich, D. J.
 Mitchell, M. P.
 Mize, E. I.
 Moody, B.
 Moore, W. H.
 Morrison, A.
 Moulton, R. K.
 Muller, M.
 Musselman, F. H.
 Myers, J.
 Myers, J.

 Nance, R. E.
 Neumann, P. G.
 Nevins, J.
 Newborn, M.
 Newman, R. A.
 Nichols, P. J.
 Nihan, C. W.
 Niimi, B.
 Nijssen, G. M.
 Nitti, J.
 Norcio, A.
 Novotny, E. J.
 Nunn, P.

 Oettinger, A. G.
 Olson, M.
 Organick, E. I.
 Osborne, A.
 O'Shields, J. T.
 Ostlund, N.

 Parfet, R.
 Park, L. A.
 Parker, D. B.
 Pehrson, G. O.
 Perito, P.
 Perskey, H.
 Pew, R.
 Phillips, C.

 Phillips, G.
 Pick, R.
 Pieper, O. R.
 Plessner, R. L.
 Pohlman, B.
 Pool, J. C. T.
 Popek, G. J.
 Popper, R. D.
 Pordy, L.
 Potter, J. L.
 Potter, T. W.
 Pottle, C.
 Price, R. A.
 Priven, L. D.
 Proske, A.
 Pugh, A. L.

 Quan, J. M. J.

 Ramamoorthy, C. V.
 Randall, J. E.
 Ravenel, B.
 Raysman, R.
 Read, C.
 Reed, S. K.
 Reynolds, J.
 Reznick, L.
 Richard, M.
 Ricketts, D.
 Rivest, R.
 Roark, M. L.
 Robertson, E.
 Robertson, G.
 Robertson, J.
 Robinson, A. L.
 Romberg, B. W.
 Rome, J.
 Rook, P.
 Rose, S.
 Ross, R.
 Ross, W.
 Roth, P. F.
 Rothnie, J. B.
 Rudolph, J.
 Rummel, J.

 Sadowsky, G.
 Saltman, R. G.
 Sammet, J. E.
 Schaeffer, K.
 Schaeffer, L.
 Scheffield, E.
 Scherzer, A.
 Scheuer, J. H.
 Schibuk, N.
 Schreiber, R.
 Schuman, R.
 Schuster, S.
 Schwartz, M.
 Senne, K.

Shah, A.
Shar, L.
Shaw, E.
Sherin, R. M.
Sherman, R. H.
Shneiderman, B.
Siegel, A. J.
Simpson, D.
Sitkin, I.
Skall, G.
Smith, C.
Smith, D.
Smith, S.
Soha, S.
Soloman, J.
Spracklen, K.
Springer, E. C.
Stay, J. F.
Stephenson, J. L.
Strassmann, P. A.
Stritter, S.
Stults, K.
Sukert, A. N.
Sunderland, S. E.
Swearingen, J. K.

Taggart, K.

Tartar, J.
Taylor, L.
Taylor, R.
Taylor, R.
Teichroew, D.
Tenkoff, P. A.
Teorey, T. J.
Thomas, J. C.
Thompson, B.
Thompson, G. B.
Thompson, K.
Thrash, W. D.
Thurber, K.
Tinker, R. F.
Trubow, G.
Turoff, M.

Vajda, S. A.
Vanassche, F.
Van Slyke, D.
Veza, A.
Viananza, P.
Vichnevetsky, R.
Vick, C. R.

Watkins, G. A.
Waxman, B.

Webb, R. D.
Weiner, M.
Weissman, C.
Weldon, J.
Wellar, B. S.
Wess, B. P., Jr.
Westermeier, J. T., Jr.
Wei, R. A.
White, J. W.
Will, P.
Willis, J. R.
Wilson, S.
Winkler, J.
Winston, J.
Wrucke, L. T.
Wyatt, J. B.
Wyrick, T. F.

Yormark, B.
Youstra, R.

Zatkyo, D.
Zemsky, B.
Zevnik, N.
Zimmerman, B.

NCC '79 REFEREES

On behalf of the NCC '79 Program Committee, I wish to thank the following referees for their efforts, without which this Proceedings could not maintain its high quality. Although an attempt was made to list as referees all individuals who participated in this procedure, there will undoubtedly be omissions.

RICHARD E. MERWIN
Editor

Abbey, Duane C.	Bruell, Steven C.	Dixon, Louis F.
Abd Alla, A. M.	Buelow, F. K.	Druseikis, F. C.
Abraham, Jacob	Burton, William D., Jr.	Dunaway, Donna K.
Adams, Larry		Duncan, Karen A.
Adams, Russell E.	Campbell, Robert P.	Dunning, Carolyn M.
Agrawal, D.	Campbell, Roy	Dutton, Ronald
Agrawala, Ashok K.	Cannon, George R., Jr.	
Ahuja, Vijay	Capraro, Gerard T.	Eastman, Caroline M.
Aiken, Robert M.	Carano, Joseph P.	Eccles, William J.
Althoff, James C., Jr.	Carey, Bernard J.	Eger, John M.
Amenta, Joyce	Carpenter, Keith R.	Elliott, Glenn
Ames, Stanley R., Jr.	Carroll, William D.	Emery, James C.
Archibald, J. A., Jr.	Carter, W. C.	Enfield, Ronald L.
Arden, Bruce	Chandrasekaran, B.	Escobar, Carlos
Athey, Thomas H.	Chapin, Ned	Estrin, Gerald
Atkins, Dan	Charney, Reginald B. J.	Estrin, Thelma
Aylor, James H.	Che, Her-daw	
	Chen, Di	Fay, Timothy J.
Babaoglu, Ozalp	Cheydleur, B. F.	Feng, T. Y.
Baer, J. L.	Chu, Yaohan	Field, George
Bail, William G.	Cieslowski, Richard J.	Fife, D.
Baird, George N.	Clema, Joe K.	Firestone, Roger
Baker, F. T.	Clemons, Eric K.	Flynn, Robert J.
Balakrishnan, A. V.	Cobb, Gary	Foley, Jim
Balbo, Gianfranco	Cochran, Anita	Fong, Elizabeth
Banerjee, J.	Coffman, James E.	Foster, Garth H.
Barbacci, Mario R.	Coke, Esther	Fowler, B. R.
Barnes, Bruce H.	Condon, J. H.	Fox, Phillip W.
Batcher, Kenneth	Connors, R. R.	Fraccola, Louis S.
Bateman, Barry L.	Cook, Betty	Franta, W. R.
Bates, Peter	Coopridge, Lee W.	Fraser, A. G.
Bauer, M.	Cornell, John A.	Freeman, Herbert
Belford, Geneva	Couperus, J.	Friedman, Arthur
Belz, Frank C.	Cowan, George	Friedman, Lee A.
Bennett, A. Wayne	Crenshaw, Edsel G.	Fu, K. S.
Bering, Doug		
Berk, Toby	Daniels, Walter E.	Gajski, Daniel
Berra, P. Bruce	Danner, Lee	Galkowski, J. T.
Beverage, R. A.	Davida, George I.	Gallier, Jean H.
Bise, Robert G.	Davies, D. J. M.	Gannon, Thomas F.
Bishop, Judy M.	Davis, Alan M.	Garcia, Oscar
Blomgren, George H.	Davis, Carl G.	Gehani, Narain
Bohn, Paul F.	Day, William H. E.	Geraghty, John J.
Bork, Alfred	DeJong, Kenneth	Ghosh, Sakti
Bowen, David M.	DeKock, Arlan R.	Gillett, Billy E.
Bracken, Peter A.	de Maine, Paul	Goel, Amrit L.
Branstad, Dennis	Denning, Dorothy E.	Goldhirsh, Isadore L.
Brocato, Louis J.	Dewdney, A. K.	Goldman, Neil M.
Brown, Nander	Dittrich, Klaus R.	Gonzalez, Mario J., Jr.

Gottlieb, Allan
Green, Murray
Green, Theresa
Grishman, Ralph
Grogono, Peter
Grosch, Audrey N.
Gross, Arthur G.
Gupta, Ravi K.

Hafer, Lou
Hakozaki, Katsuya
Hamblen, John W.
Hanna, William E., Jr.
Hart, Peter E.
Hartmann, David C.
Hartson, H. Rex
Hatta, Hiroshi
Hattori, Mitsuhiro
Hecht, Herbert
Hellman, Martin E.
Herman, K. M.
Hisgen, Andrew L.
Hoffman, Lance
Hollaar, Lee A.
Holmes, Harvard
Hopewell, Lynn
Hopper, Grace M.
Hord, R. Michael
Horgan, Joseph R.
Horsted, Burt
Howden, William E.
Hoyt, Patrick M.

Ichikawa, Tadao
Idesawa, Masanori
Isaacson, Portia

Jacobs, Barry E.
Jacobs, Morton C.
Jensen, E. Douglas
Jette, Christina
Johnson, L. Arnold
Johnson, Mark Scott
Jones, Alexander M.
Jordan, Harry F.

Kahn, Kevin C.
Kampen, G.
Kaufman, Arie
Kesselman, Murray
Kimbleton, Steven R.
Kooiman, Don
Kornfield, N. R.
Kovar, Don
Kubitz, W. J.

LaBelle, Charles D.
Lamprey, Roger
Landis, Carolyn P.

Langdon, Glen
LaPadula, Leonard J.
Leeman, George
Leive, Gary W.
Levin, Roy
Lien, David A.
Lipovski, G. J.
Little, Elizabeth R.
Little, Joyce
Liu, C. L.
Liu, Jane W. S.
Liu, Wen-Tai
Liuzzi, Raymond A.
Lockett, JoAnn
Long, Harvey S.
Longacre, Andrew, Jr.
Loomis, Herschel H., Jr.
Lucido, A. P.
Lukas, George
Lundstrom, Stephen F.

Machover, Carl
Madrigan, Orlando S.
Madron, Thomas W.
Madron, Beverly B.
Maekawa, Mamoru
Magel, Kenneth
Magnuson, W. G., Jr.
Maher, Austin J.
Maltz, Irwin
Managaki, Masao
Mander, K. C.
Maniotes, John
Mariani, Mike
Maskewitz, Betty F.
Mathews, Walter M.
Matsumoto, Hiroshi
Matsushita, Yutaka
Maynard, Denis R.
Mayne, Glenn
McCrea, Donald R.
McDonald, Nancy H.
McHenry, Stephen
McNulty, Frederick L.
Medley, Don B.
Meltzer, Arnold
Merwin, Richard E.
Metzner, John R.
Michalski, Ryszard S.
Mickunas, M. Dennis
Mikami, Toru
Miller, Dale W.
Mills, David L.
Mills, Wayne
Minsky, Naftaly
Mittal, Sanjay
Modesitt, Kenneth L.
Mollenauer, J. F., Jr.
Moore, Ken

Mudge, Trevor
Muraoka, Yoichi
Murphy, Robert E.

Naemura, Kenji
Nagle, H. Troy, Jr.
Nash, Michael S.
Navathe, Sham
Navlakha, Jai
Nelson, Victor P.
Nestman, Chadwick H.
Neumann, Peter G.
Nielsen, Norman R.
Nutt, Gary

Ogden, Neal R.
O'Kane, Kevin
O'Neal, Beverly
O'Rourke, Donald J.
Orr, Patrick K.

Parrish, Edward
Peralta, L. A.
Perry, James M.
Pfaltz, John L.
Pfleeger, Charles
Pickholtz, Raymond
Pinkert, James
Pitts, Gerald
Pizer, Stephen M.
Porti, Edwin H.
Potter, J.
Pottinger, Hardy J.
Powell, John E.
Prywes, Noah
Purcell, John J.
Purcell, Thomas D.
Purkayastha, S.

Quann, John J.

Rader, J. A.
Raskin, Jef
Rauscher, Tomlinson G.
Reddy, Sudhaker M.
Reho, Andrew M.
Reilly, Dorothy
Rice, Rex
Richardson, Debra J.
Richter, Michael D.
Riddle, William E.
Roberts, Eric S.
Roggio, Robert F.
Rosenfeld, A.
Rosin, R. F.
Roth, R. Waldo
Roth, Scott
Rowe, Lawrence A.
Rudwick, Bernard
Ruschitzka, Manfred

Saal, Harry J.
Sanders, R.
Sangal, Rajeev
Sayward, Frederick G.
Schafer, David J.
Scheuermann, Peter
Schlanger, G. Gary
Schlegel, C. T.
Schmidt, David
Schneider, Michael
Schneidewind, N. F.
Schulze, R. G.
Segal, Ronald
Sethi, Ravi
Shaffron, Nancy
Shapiro, David S.
Shapiro, Michael D.
Shelly, Gary B.
Shetler, Toni
Shimamura, Kuzuya
Shooman, M.
Shore, John E.
Short, Gerald
Siewiorek, Dan
Skeel, Robert D.
Slotnock, D. L.
Smid, Miles
Smith, David N.
Smith, Eugene B.
Smith, Robert Ellis
Smith, Robert J., II
Smoot, Oliver R.
Snyder, James
Snyder, Larry
Soma, Takashi

Sondheimer, Norman K.
Spaniol, Roland
Srodawa, Ronald J.
Stavely, Allan M.
Steele, Stuart A.
Stemple, David
Stevens, D. F.
Stevens, W. Richard
Stranart, J. C.
Stritter, Edward
Stuck, B. W.
Sugano, Takuo
Sugimoto, Masakatsu
Svigals, Jerome
Szolovits, Peter

Tarabar, David
Taulbee, Orrin E.
Tausner, Miriam R.
Taylor, Robert W.
Teichroew, Daniel
Teorey, Toby
Tessar, Paul A.
Thalmann, D.
Thalmann-Magnenat, Nadia
Thurber, Kenneth J.
Tinaztepe, Cihan
Toda, Iwao
Tomaru, Keisuke
Tomlinson, Chris
Towsen, James F.
Troutt, Marvin D.
Tucker, Edwin K.
Turn, Rein

Unger, Brian W.

van Cleemput, W. M.
Van der Gaag, M. R.
Vandergraft, J.
Volz, Richard A.

Ward, Ronnie G.
Warren, Jim
Wecker, Stuart
Weinberger, Peter J.
Weiss, Stephen F.
Weldon, Jay-Louise
Welsch, Lawrence A.
Wesselkamper, T.
Westbrook, Charles K.
Wewer, William
Wexelblat, Richard
Wharton, R. Michael
White, Robert L.
Whiting-O'Keefe, Patricia
Wiederhold, Gio
Wileden, Jack
Willman, Herb
Williams, John D.
Winkler, Stanley
Wu, Chuan-lin

Yamada, Hiroshi
Yamamoto, Masahiro
Yasnoff, William A.
Young, J. W.
Yovits, Marshall C.

Zislis, Paul M.

AUTHOR INDEX

- Abu-Sufah, W., 969
Ackerman, William B., 1087
Agrawala, Ashok K., 51
Aharonian, Donald J., 117
Aird, Charles L., 425
Aiso, Hideo, 499
Alagar, V. S., 775
Amer, P. D., 781
Anderson, Karen K., 385
Arden, Bruce W., 95
Artis, H. Pat, 193
Austing, Richard H., 407
- Baker, Paul, 73
Barksdale, G. J., Jr., 365
Barth, C. Wrandle, 135
Battarel, G. L., 649
Ben-Dor, Avner, 73
Bernstein, Philip A., 813
Berson, T. A., 365
Biba, K. J., 373
Blakley, G. R., 313
Blattner, Meera, 1013
Borst, M. A., 1021
Bravin, Philip W., 385
Briggs, Fayé A., 255
- Campbell, Robert P., 293
Cannon, Edward Ray, 657
Cave, William C., 67
Chang, S. K., 147
Chapin, Ned, 995
Chevance, R. J., 649
Chin, Y. H., 175
Chlamtac, Imrich, 57
Chu, Yaohan, 657
Chung, Richard J., 905
Clark, N. Bruce, 7
Clemons, Eric K., 689
Colton, Kent W., 443
Comer, Douglas, 209
Crow, Franklin C., 157
Curtis, Bill, 1021
- Davis, A. L., 1079
Dowdy, Lawrence W., 51
Drongowski, P. J., 345
Duran, Joe W., 1059
- Elmquist, Kells A., 587
Engel, Gerald L., 407
Enslow, Philip H., Jr., 29
Estrin, G., 975
- Feiertag, Richard J., 329
Fenchel, R., 975
Ferrari, Domenico, 789
Feuer, A. R., 1003
Finkel, Raphael A., 455
Finnin, G. R., 749
Fowlkes, E. B., 1003
Franta, W. R., 57
Fu, King-Sun, 255
- Gardarin, Georges, 681
Goel, Amrit L., 769
Gold, B. D., 355
Goodman, Nathan, 813
Gostelow, Kim P., 629
Gouda, Mohamed Gawdat, 469
Gowan, Curtis R., 493
Gurd, John, 623
- Hamlet, R. G., 267
Hamstra, J. R., 595
Hayashi, Takaki, 217
Henry, J. Shirley, 513
Hinojosa, Sergio M., 917
Horak, Wolfgang, 125
Hord, R. Michael, 243
Horowitz, Michael L., 455
Hurson, Alireza, 727
Hwang, Kai, 255
- Iizuka, Hajime, 499
Irons, Edgar T., 273
- Jette, Christina L., 1071
- Kambayashi, Yahiko, 217
Kampe, Mark, 355
Kartashev, Steven I., 543
Kartashev, Svetlana P., 543
Katz, Randy H., 741
Keller, Robert M., 613
Kimbleton, S. R., 821
Kitamura, Takuo, 557
Klein, Daniel, 199
Kline, Charles S., 355, 831
Kling, Rob, 107
Konishi, Osamu, 217
Krumland, Rand B., 89
Kuck, D., 969
Kutsch, James A., Jr., 383
Kutsch, Kimberly B., 383
- Lafue, Gilles M. E., 713
Lai, Larry Kwok-Woon, 565
Lala, Jaynarayan H., 481
Landa, Suzanne, 1
Landgrebe, David, 233
Lawrie, D., 969
Lee, Hikyu, 95
Lemke, Johann, 697
Lewis, George R., 513
Lim, Raymond S., 205
Lin, B. S., 147
Linde, R. R., 335
Lindstrom, Gary, 613
Lipinski, Hubert, 411
Liu, Jane W. S., 917
Love, Tom, 1021
Lozinskii, E. L., 717
Lynn, Charles, Jr., 139
- MacDonald, John E., Jr., 605
MacDougall, M. H., 39
Malaiya, Yashwant K., 577
Mamrak, S. A., 781
Markus, M. Lynne, 397
McBride, E. J., 749
McCauley, E. J., 345
McCune, Brian P., 513
McMillen, Robert J., 529
Michelman, Eric H., 305
Milliman, Phil, 1021
Minoli, Daniel, 875
Modesitt, Kenneth L., 403
Monroe, Robert M., 461
Morokuma, Tatsushi, 499
Morris, Robert J. T., 887
Mueller, Philip T., Jr., 529
Mukhopadhyay, Amar, 727
Mundy, J. L., 227
Musa, John D., 941
- Navathe, Shamkant B., 697
Neely, R. B., 373
Neumann, Peter G., 329
Nielsen, Norman R., 805
- Oestreicher, Donald, 839, 847
Okumoto, Kazu, 769
Oliver, Paul, 1051
O'Neal, Beverly, 797
Osman, I. M., 733
- Padlipsky, M. A., 373
Paisner, William L., 165

Patel, Janak H., 255
Patil, Suhas, 613
Paulish, Daniel J., 935
Pearson, D. J., 1029
Peeler, R. J., 335
Peterson, Lynn L., 415, 1063
Plummer, Robert, 411
Popek, Gerald J., 355, 831
Pramanik, Sakti, 273
Price, Ronald J., 957
Pujolle, G., 893

Ramamoorthy, C. V., 543, 667
Ramanathan, Jayashree, 1013
Raveling, Paul, 855
Razouk, R., 975
Renault, Jean-Paul, 1037
Risley, Jean, 139
Robey, Daniel, 391
Roman, Gruia-Catalin, 947
Rosenfeld, A., 267
Rosenkranz, Evelyn, 67
Rothenberg, Jeff, 855, 863
Ruggiero, W., 975
Rupert, Sally J., 103
Rybczynski, A. M., 905

Safirstein, Peter, 279
Sakamura, Ken, 499
Sands, Gerald A., 293

Santhanam, Viswanathan, 637
Sarna, David E. Y., 185
Scacchi, Walt, 107
Schaefer, M., 335
Scheid, J. F., 335
Schneidewind, N. F., 989
Schutzer, Daniel, 927
Schwabe, D., 975
Schwetman, Herb, 45
Segal, Ronald, 797
Sekino, Akira, 557
Selden, Jon, 73
Sheppard, Sylvia B., 1021
Siegel, Howard J., 529
Silver, Aaron N., 83
Smith, Charles J., 481
Solomon, Marvin, 455
Sondheimer, Norman K., 1043
Soochan, C., 775
Spangler, Kathleen, 411
Srodawa, Ronald J., 461
Stepp, Robert, 917
Stotz, Robert, 839, 855
Stoughton, Allen, 355
Su, Stephen Y. H., 577
Sutphen, Steven F., 13
Suwa, Izumi, 917

Thomas, Robert E., 629
Todd, Barbara H., 425
Tolopka, Steve, 45

Tonik, A. B., 749
Troutman, Michael A., 7
Tugender, Ronald, 839, 847, 869
Turn, Rein, 283

Urban, Michael, 355
Utsuqi, Tsutoma, 917

Vernon, M., 975

Wah, Benjamin W., 667
Walser, R., 147
Walsh, Thomas J., 761
Walton, Evelyn J., 355
Ward, P. D., 335
Watson, Ian, 623
Weldon, Jay-Louise, 709
Wells, Robert, 139
Westervelt, Franklin H., 461
Wilczynski, David, 839, 847
Wilson, E. A., 19
Wirth, James F., 721
Woborschil, Walter, 125
Wood, Helen M., 419, 821
Woodward, John P. L., 319

Yajima, Shuzo, 217
Yu, S. H., 175