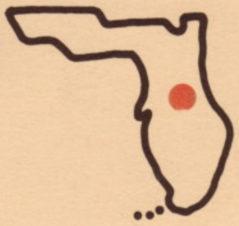
VOLUME I

HEWLETT·PACKARD GENERAL SYSTEMS USERS GROUP

1981 International Meeting Orlando, Florida

April 27-May 1



PROCEEDINGS

HP 3000 INTERNATIONAL USERS GROUP
ORLANDO 1981

PROCEEDINGS OF THE HP GENERAL SYSTEMS USERS GROUP 1981 INTERNATIONAL MEETING

APRIL 27 - MAY 1, 1981
ORLANDO, FLORIDA

TABLE OF CONTENTS

FOREWARD 1

ACKNOWLEDGMENTS 2

INTRODUCTION TO THE PROCEEDINGS 3

PROCEEDINGS 4

INDEX TO PAPERS BY TOPIC AND SESSION CODE 5

INDEX TO PAPERS BY CLASSIFICATION 12

INDEX TO PAPERS BY AUTHOR 19

FOREWARD

An expression of gratitude is offered to all participants in the 1981 HPGSUG Orlando International Meeting. Your enthusiastic efforts have made this year's conference a valuable and enjoyable experience.

The HP3000 International Users Group is a growing organization of individuals dedicated to maximizing their usage of the HP3000 Computer Resource. Though such a group may experience growing pains from time to time, it survives by participation of its membership. This volume represents such an achievement by presenting much of the "state-of-the-art" work being done today.

The papers were written by professionals and current leaders of our industry, and contain significant ideas for 1981 and the years ahead. It represents what I believe to be the most important product of a Users Group;

"An Exchange of Current Information"

To all of you who contributed to the success of the 1981 HPGSUG Orlando International Meeting, my warmest thanks.

JDDavis

J.D. Davis
Conference Chairman
HPGSUG 1981 International Meeting
Orlando, Florida

ACKNOWLEDGMENTS

Special thanks to:

--The Florida Conference Committee

Jerry Davis
Larry Gerringer
Jake Bruckhart
Greg Stewart
Mark Conroy
Jim Oliverio

--Conference Proceedings Committee and
HPGSUG Publications Committee

Dr. John R. Ray, Chairperson
Dr. Lloyd Davis
Dr. John True
Mr. Joe Schneider
Mr. Gary Johnson

INTRODUCTION TO THE PROCEEDINGS

The Hewlett-Packard General Systems Users Group 1981 Orlando International Meeting was especially designed for you -- the HP 3000 System User. This Meeting provided practical, up-to-date information relating to more effective management and utilization of your HP 3000 System. The sessions were planned to introduce, define, and explore the 1981 Theme - "Distributed Processing."

Although the Program was "Distributed Processing," technical sessions on other subjects were included. Examples were system performance, operation, programming techniques, hardware selection, etc. Hewlett-Packard technical specialists presented sessions allowing the user insights into the Hewlett-Packard product line.

Material for use at the Conference was reviewed by the HPGSUG Publications Committee and appropriate members of the Conference Committee. Final papers and/or abstracts were supplied, in camera ready form, by the author(s). Information concerning any paper or abstract should be directed to the author(s).

Proceedings

A major goal of the Conference Committee was to provide attendees with the printed proceedings at the time of registration. The HPGSUG Publications Committee and the Florida Conference Committee have worked to this end.

The volume contains papers and selected abstracts (where papers were not appropriate and/or available) organized by presentation day and session. Additionally, a list of presentations organized by Classification is included. Finally, a list of presentations by author is given.

Within each day, the papers are numbered sequentially with pagination following this plan:

```
Day      A  -  1  -  01

 *       *     *     *
 *       *     *     *
 *       *     *     * * * Page number within this paper
 *       *     *
 *       *     * * * * Session Location - Please see Conference
 *       *             Program
 *       *
 *       * * * * Session Time --(A-1:15 pm Monday; B-2:45 pm Monday;
 *               C-8:30 am Tuesday; D-10:00 am Tuesday; E-1:15 pm
 *               Tuesday; F-2:45 pm Tuesday; G-4:00 pm Tuesday,
 *               H-8:30 am Wednesday; I-10:00 am Wednesday; J-1:30 pm
 *               Wednesday; K-3:00 pm Wednesday; L-8:30 am Thursday;
 *               M-10:00 am Thursday; P-4:00 pm Thursday)
 *
 * * * * Session Day -- Monday, Tuesday, Wednesday or Thursday
```

INDEX TO PAPERS BY TOPIC AND SESSION

Author	Title	Session	Classification
Monday 1:15 pm			
Curtis, Terence	A Systems Development Methodology Based Upon An Active Data Dictionary/Directory	A-1	500
Kramer, Jim	The Technology of the Quad Editor	A-2	300
O'Brien, Matthew	Distributed Processing - A Hewlett-Packard Solution	A-3	050
Garvey, Robert B./ Womack, Robert L. IV	Online Database - Start to Finish	A-4	200
Davis, Dr. Lloyd D.	Faculty Perceptions of Computing Facilities (Based on a study of UTC Faculty in 1981)	A-5	760
Harjula, Esa A.	JOBLIB/3000	A-6	300
O'Neill, Daniel R.	Distributed 6250 BPI Tape	A-7	630
Foote, Richard L.	Software Maintenance and Support In The Distributed Environment	A-8	100
Larson, Orland	QUERY-Directions for the 1980's	A-9	200
Monday 2:45 pm			
Black, Tom/ Roselie Tobes	For First Time Users	B-1	500
A. Steven Wolf	Evolutionary Systems Development in a Distributed Environment	B-2	100
Kurtz, Barry D.	Application System Operation and Control	B-3	050
Belcham, Mick	Manufacturing Control, Planning and Feedback In A Distributed Processing Environment	B-4	720
Heidner, Dennis	Transaction Logging and Its Uses	B-5	200
Damm, Jack	Designing Friendly Interactive Systems	B-6	000

Author	Title	Session	Classification
Leonard, William Jr.	Software Contracts; Preventative Maintenance to Ensure Subsequent Quality Services	B-7	050
Beckett, John	Distributed Control Using One CPU	B-8	050
Holt, Wayne E./ Payne, Delores R.	Programming Standards: A Tool for Increased Programmer Productivity	B-9	050
McCauley, George	Archive Retrieval System	B-10	500
Souder, Duane	A Database Test and Repair Facility	B-11	200
Rego, Alfredo F.	Database Therapy: A Practitioner's Experiences	B-12	200
Gloss, Greg	ANSI Cobol 198X: A Sneak Preview	B-14	420

Tuesday 8:30 am

Eikenbary, Beth	Success with Manufacturing Applications: Implementation of Materials Management/3000	C-1	720
Carlson, Boyd/ Hennen, Ralph	A Subsystem That Improves Response Time for Applications With Large Numbers of Terminals	C-3	120
Davis, Dr. Lloyd D.	Computer Aided Learning at UTC	C-4	760
Garvey, Robert B./ Womack, Robert L. IV	A Generalized Name Indexing Method for Image Databases	C-5	200
Brand, Jim	Using Serial and Demountable Disk Volumes	C-6	110
Storla, Chuck	Measuring Transaction Processing Response Times	C-7	100
VanBempt, Gurdo/ VanDamme, Jaak	Data Base Normalization	C-8	200
Langley, Jim	Laser Printer Paper	C-9	600
Kernke, Jutta	Business Computer Graphics/ Decision Making	C-11	300

Tuesday 10:00 am

Primmer, Paul	Pseudo-Devices	D-1	950

Author	Title	Session	Classification
Green, Robert M.	HP3000/Optimizing Batch Jobs	D-2	100
Kiefer, Karl H.	Data Base Design: Polishing Your Image	D-3	200
Volokh, Eugene/ Volokh, Vladimir	MPEX/3000: Effective Use of MPE Fileset Concept	D-4	100
Berkun, Kenneth/ Evers, Penny/ Juhasz, Thomas	Technical Aspects of Large Geographic Databases on the HP/3000	D-5	700
Farquharson, Ian	The Obsolessence of Programming Genasys/3000	D-6	000
Tuesday 1:15 pm			
Kernke, Jutta	Data Capture on the HP3000	E-1	500
Greer, David J.	Experiences With Pascal	E-3	470
Perkin, Brian M.	The Field Software Coordination Process	E-4	050
Fraser, Thomas L.	Distributed Processing in High Volume Transaction Processing Systems	E-5	050
VanBempt, Gurdo/ VanDamme, Jaak	Increasing Program Productivity/ Full Screen Editor	E-6	000
Tuesday 2:45 pm			
Loffler, Ivan	Using Hardware Monitor To Solve Problems on HP3000-III	F-1	120
Federman, Nancy/ Steiner, Robert	The Development of a Large Application System for the HP3000 Computer	F-2	700
Ventresca, Benjamin, Jr.	Moving Toward Information Management In An Integrated Environment	F-4	050
Mason, J. Michael	Micro-Distributed-Processing	F-7	950
Busch, John R.	The MPE IV Kernel: History, Structure and Strategies	F-9	120
Busch, John R.	The MPE IV Kernel: A High Performance, Integrated Foundation for MPE - The Design Process	F-10	100

Author	Title	Session	Classification
Langley, Jim	The Role of Printers In a DS Environment--An Engineering Feedback Session	F-11	600
Black, Tom/ Turner, Ed./ Beetem, Jim	Data Communications--A Technical Roundtable	F-12	950
Tuesday 4:00 pm			
Geffken, Joachim	User Friendly Applications In Commercial Realtime Data Processing	G-1	050
Peckover, Doug	English 3000-A Natural Language On A Mini-Computer	G-2	700
Foster, M. B.	Systems Life-Cycle-A Framework for Success	G-4	050
McQuillen, Jack	Successful Conversion From Two IBM System/3 to A HP3000	G-6	100
Byers, Peter	VTEST/3000 On-Line Test Harness-User View	G-9	100
Mead, Robert L./ Rakusin, Robin P.	Introducing The HP On-Line Performance Tool (OPT/3000)	G-9	100
Kramer, Jim	Saving the Precious Resource--Disc Accesses	G-11	100
Kurtz, Barry	HP's Manufacturing Software: Engineering	G-10	720
Federman, Nancy/ Steiner, Bob Beetem, Jim	Terminal I/O Interface--An Engineering Feedback Session	G-12	600
Wednesday 8:30 am			
Colmer, Earl E., Jr.	VIP/3000 and VIP-TNC/3000	H-11	200
Wednesday 10:00 am			
Kneppelt, Lee, R / Sneed, Jerry L.	Real-Time, On-Line, Distributed In the Manufacturing Systems Environment	I-1	720
Colmer, Earl E., Jr.	Alter/3000 Quick Modification Program	I-2	300
Kalman, David	E-ZV The Easy Way to Use V/3000	I-3	

Author	Title	Session	Classification
Vigmalou, Dr. Joseph	Dialoguer/3000: Forms Management for Every Screen	I-4	300
Sera, Arthur	Time and Resource Accounting System	I-5	300
Damm, Jack	Budgeting and Profit Planning on the HP3000	I-6	050
Warwaruk, Marshall	The Great Companion (Quiz)	I-9	300
Wednesday 1:30 pm			
Kneppelt,Lee R/ Sneed, Jerry L.	Implementing Manufacturing Systems/ Difficulty of Task	J-1	720
Colmer, Earl	LOGOFF/3000	J-2	300
Carnes, Lance	REX/3000 As a Programmer Productivity Tool	J-3	000
VanSickle, Larry	Applications Program Transformations	J-4	000
Hoff, Roger	The All Purpose Computer	J-5	100
Colmer, Earl	QCALC/3000	J-6	300
Warwaruk, Marshall	The Great Companion (Quick)	J-9	300
Thursday 8:30 am			
Peterson, Don Wright, Norm	Centralized Support of A Distributed Systems Environment	L-1	100
Holinstat, David	Building A Faster Machine: Architecture of the HP3000 Series 44	L-2	100
Dooley, Pat Rowe, Denis	COCAM - A Data Management System	L-4	500
Brown, Barry	Conducting a Long-Range Study	L-6	050
Wimer, Ted	HP7976A: High Performance, 6250 Streaming Tape Storage Subsystem	L-7	600
Fraser, Tom/ Dale, Allan	New Vistas for the HP3000 in On-Line Distributed Networks, A Non-Myopic View	L-8	100
Folkins, Dale	KSAM - It's Alive and Well!	L-9	300

Author	Title	Session	Classification
Black, David	Implementation of A Large Scale Application On A Hewlett Packard 3000 Series III	L-10	100
Larson, Orly/ Kernke, Jutta/ Souder, Duane/ Rego, Alfredo	A Technical Roundtable	L-11	200
Thursday 10:00 am			
Dowling, Jim	Log DB-A System Logfiles Data-Base and Reporting System	M-1	200
Wheatley, Jack E.	Handling Distributed Applications With A Front End Processor	M-2	100
Beasley, Dave Stamps, Bob	Introducing MPE IV	M-3	100
Turner, A. Edward	Data Communications Support Manager, HP	M-4	950
Dailey, Roy N.	Management of Distributed Systems	M-5	050
Trasko, Mark S.	Keyed Sequential Access Capability In Data Base Systems-The IMSAM Enhancement to Hewlett Packard Image	M-6	200
Crow, William M.	Life At The End Of A Phone Line: An Investigation of Asynchronous Terminal Data Communications In An HP/3000 Environment	M-7	950
Kell, Jess	TERMNET: Terminal Network Controller	M-8	950
Thursday 4:00 pm			
Carnes, Lance	Distributed Text Editing	P-1	300
Smith, Terry	HP3000 User Standards for Structured Cobol, Image, and V/3000	P-2	200
Ehrhart, Rick	IML/3000: An Overview	P-3	900
VanLeeuwen, J.F.	EDP Productivity and The HP/3000	P-4	050
Lessey, Ken	On Line System Design and Development A Case Study: Accounts Payable	P-5	050
Langlois, David	Using a Packet Switched Telecommunications Network	P-7	950

Author	Topic	Session	Classification
Bergquist, Rick	Centralized Support of Distributed Systems	P-8	050
Souder, Duane	IMAGE Engineering Feedback	P-9	200
Valby, Nancy	Alternative Hardware Growth Paths for the Series II/III	P-10	050
Goodman, Robert	Industry Software Evolution	P-11	000
Birkwood, Ilene/ Workman, Ted/ Roland, Vince/ Dudley, Sally	Product Assurance Management Roundtable	P-12	050

PRESENTATIONS BY CLASSIFICATION

000 Programming	Session
Designing Friendly Interactive Systems Jack Damm	B-6
The Obsolessence of Programming Genasys/3000 Ian Farquharson	D-6
Increasing Program Productivity/Full Screen Editor VanBempt, Gurdo/VanDamme, Jaak	E-6
REX/3000 As A Programmer Productivity Tool Lance Carnes	J-3
Applications Program Transformations Larry VanSickle	J-4
Industry Software Evolution Robert Goodman	P-11
050 DP Management	
Distributed Processing - A Hewlett Packard Solution Matthew O'Brien	A-3
Application System Operation and Control Barry D. Kurtz	B-3
Software Contracts: Preventative Maintenance to Ensure Subsequent Quality Services William Leonard, Jr.	B-7
Distributed Control Using One CPU John Beckett	B-8
Programming Standards: A Tool for Increased Programmer Productivity Wayne E. Holt/Delores R. Payne	B-9
The Field Software Coordination Process Brian M. Perkin	E-4
Distributed Processing in High Volume Transaction Processing Systems Thomas L. Fraser	E-5
Moving Toward Information Management In An Integrated Environment Benjamin Ventresca, Jr.	F-4

User Friendly Applications in Commercial Realtime Data Processing G-1
Joachim Geffken

Systems Life-Cycle-A Framework for Success G-4
M. B. Foster

Budgeting and Profit Planning on the HP3000 I-6
Jack Damm

Conducting a Long-Range Study L-6
Barry Brown

Management of Distributed Systems M-5
Roy N. Dailey

EDP Productivity and The HP/3000 P-4
J. F. VanLeeuwen

On Line System Design and Development A Case Study: Accounts Payable P-5
Ken Lessey

Centralized Support of Distributed Systems P-8
Rick Bergquist

Alternative Hardware Growth Paths for the Series II/III P-10
Nancy Valby

Product Assurance Management Roundtable P-12
Ilene Birkwood/Ted Workman/Vince Roland/Sally Dudley

<u>100 Systems Management</u>

Software Maintenance and Support In The Distributed Environment A-8
Richard L. Foote

Evolutionary Systems Development in a Distributed Environment B-2
A. Steven Wolf

A Subsystem That Improves Response Time for Applications With Large Numbers of Terminals C-3
Boyd Carlson/Ralph Hennen

Using Serial and Demountable Disk Volumes C-6
Jim Brand

Measuring Transaction Processing Response Times C-7
Chuck Storla

HP3000/Optimizing Batch Jobs D-2
Robert M. Green

MPEX/3000: Effective Use of MPE Fileset Concept D-4
Eugene Volokh/ Vladimir Volokh

Using Hardware Monitor To Solve Problems on HP3000-III F-1
Ivan Loffler

The MPE IV Kernel: History, Structure and Strategies F-9
John R. Busch

The MPE IV Kernel: A High Performance, Integrated Foundation for MPE - The Design Process F-10
John R. Busch

Successful Conversion From Two IBM System3 to A HP3000 G-6
Jack McQuillen

VTEST/3000 On-Line Test Harness-User View G-7
Peter Byers

Introducing The HP On-Line Performance Tool (OPT/3000) G-9
Robert L. Mead/Robin P. Rakusin

Saving the Precious Resource--Disc Accesses G-11
Jim Kramer

The All Purpose Computer J-5
Roger Hoff

Centralized Support of A Distributed Systems Environment L-1
Don Peterson/Norm Wright

Building A Faster Machine: Architecture of the HP3000 Series 44 L-2
David Holinstat

New Vistas for the HP3000 in On-Line Distributed Networks, A Non-Myopic View L-8
Tom Fraser/Allan Dale

Implementation of A Large Scale Application On A Hewlett Packard 3000 Series III L-10
David Black

Handling Distributed Applications With A Front End Processor M-2
Jack E. Wheatley

Introducing MPE IV M-3
Dave Beasley/Bob Stamps

200 Data Base

Online Database - Start to Finish Robert B. Garvey/Robert L. Womack IV	A-4
QUERY-Directions for the 1980's Orland Larson	A-9
Transaction Logging and Its Uses Dennis Heidner	B-5
A Database Test and Repair Facility Duane Souder	B-11
Database Therapy: A Practitioner's Experiences Alfredo F. Rego	B-12
A Generalized Name Indexing Method for Image Databases Robert B. Garvey/Robert L. Womack IV	C-5
Data Base Normalization Gurdo VanBempt/Jaak VanDamme	C-8
Data Base Design: Polishing Your Image Karl H. Kiefer	D-3
VIP/3000 and VIP-TNC/3000 Earl E. Colmer, Jr.	H-11
E-ZV The Easy Way to Use V/3000 David Kalman	I-3
A Technical Roundtable Orly Larson/Jutta Kernke/Duane Souder/Alfredo Rego	L-11
Log DB-A System Logfiles Data base and Reporting System Jim Dowling	M-1
Keyed Sequential Access Capability In Data Base Systems-The IMSAM Enhancement to Hewlett Packard Image Mark S. Trasko	M-6
HP3000 User Standards for Structured Cobol/ Image, and V/3000 Terry Smith	P-2
IMAGE Engineering Feedback Duane Souder	P-9

300 Utilities

The Technology of the Quad Editor Jim Kramer	A-2

JOBLIB/3000 A-6
Esa A. Harjula

Business Computer Graphics/Decision Making C-11
Jutta Kernke

Alter/3000 Quick Modification Program I-2
Earl E. Colmer, Jr.

Dialoguer/3000: Forms Management for Every Screen I-4
Dr. Joseph Vigmalou

Time and Resource Accounting System I-5
Arthur Sera

LOGOFF/3000 J-2
Earl Colmer

QCALC/3000 J-6
Earl Colmer

The Great Companion (Quick) J-9
Marshall Warwaruk

KSAM - It's Alive and Well! L-9
Dale Folkins

Distributed Text Editing P-1
Lance Carnes

400 Language Support

ANSI Cobol 198X: A Sneak Preview B-14
Greg Gloss

Experiences With Pascal E-3
David J. Greer

500 Data and Test Processors

A Systems Development Methodology Based Upon An Active Data Dictionary/Directory A-1
Terence Curtis

For First Time Users B-1
Tom Black/Roselie Tobes

Archive Retrieval System B-10
George McCauley

Data Capture on the HP3000 E-1
Jutta Kernke

COCAM - A Data Management System L-4
Pat Doöley/Denis Rowe

600 Peripheral Software

Distributed 6250 BPI Tape — A-7
Daniel R. O'Neill

Laser Printer Paper — C-9
Jim Langley

The Role of Printers In a DS Environment--An Engineering Feedback Session — F-11
Jim Langley

Terminal I/O Interface--An Engineering Feedback Session — G-12
Jim Beetem

HP7976A: High Performance, 6250 Streaming Tape Storage Subsystem — L-7
Ted Wimer

700 Business

Faculty Perceptions of Computing Facilities (Based on A Study of UTC Faculty in 1981) — A-5
Dr. Lloyd D. Davis

Manufacturing Control, Planning and Feedback in A Distributed Processing Environment — B-4
Mick Belcham

Success with Manufacturing Applications: Implementation of Materials Management/3000 — C-1
Beth Eikenbary

Computer Aided Learning at UTC — C-4
Dr. Lloyd D. Davis

Technical Aspects of Large Geographic Databases on the HP/3000 — D-5
Kenneth Berkun/Penny Evers/Thomas Juhasz

The Development of a Large Application System for the HP3000 Computer — F-2
Nancy Federman/Robert Steiner

English 3000-A Natural Language On A Mini-Computer — G-2
Doug Peckover

HP's Manufacturing Software: Engineering — G-10
Barry Kurtz/Nancy Federman/Bob Steiner

Real-Time, On-Line, Distributed In the Manufacturing Systems Environment — I-1
Lee R. Kneppelt/Jerry L. Sneed

Implementing Manufacturing Systems/Difficulty of Task — J-1
Lee R. Kneppelt/Jerry L. Sneed

800 Science and Engineering

900 Demonstrations and Games

IML/3000: An Overview — P-3
Rick Enrhart

950 Telecommunications

Pseudo-Devices — D-1
Paul Primmer

Micro-Distributed-Processing — F-7
Michael J. Mason

Data Communications--A Technical Roundtable — F-12
Tom Black/Ed Turner/Jim Beetem

Data Communications Support Manager, HP — M-4
Edward A. Turner

Life At The End Of A Phone Line: An Investigation of Asynchronous Terminal Data Communications In An HP/3000 Environment — M-7
William M. Crow

TERMNET: Terminal Network Controller — M-8
Jess Kell

Using A Packet Switched Telecommunications Network — P-7
David Langlois

PRESENTATIONS BY AUTHOR

Beasley, Dave	M-3
Beckett, John	B-8
Beetem, Jim	F-12, G-12
Belcham, Mick	B-4
Bergquist, Rick	P-8
Berkun, Kenneth	D-5
Birkwood, Ilene	P-12
Black, David	L-10
Black, Tom	B-1, F-12
Brand, Jim	C-6
Brown, Barry	L-6
Busch, John R.	F-9, F-10
Byers, Peter	G-9
Carlson, Boyd	C-3
Carnes, Lance	J-3, P-1
Colmer, Earl E., Jr.	H-11, I-2, J-2, J-6
Crow, William M.	M-7
Curtis, Terence	A-1
Dailey, Roy N.	M-5
Dale, Allan	L-8
Damm, Jack	B-6, I-6
Davis, Dr. Lloyd D.	A-5, C-4
Dooley, Pat	L-4
Dowling, Jim	M-1
Dudley, Sally	P-12
Ehrhart, Rick	P-3
Eikenbary, Beth	C-1
Evers, Penny	D-5
Farquharson, Ian	D-6
Federman, Nancy	F-2
Folkins, Dale	L-9

Foote, Richard L.	A-8
Foster, M. B.	G-4
Fraser, Thomas L.	E-5, L-8
Garvey, Robert B.	A-4, C-5
Geffken, Joachim	G-1
Gloss, Greg	B-14
Goodman, Robert	P-11
Green, Robert M.	D-2
Greer, David J.	E-3
Harjula, Esa A.	A-6
Heidner, Dennis	B-5
Hennen, Ralph	C-3
Hoff, Roger	J-5
Holinstat, David	L-2
Holt, Wayne E.	B-9
Juhasz, Thomas	D-5
Kalman, David	I-3
Kell, Jess	M-8
Kernke, Jutta	L-11, E-1, C-11
Kiefer, Karl H.	D-3
Kneppelt, Lee R.	I-1, J-1
Kramer, Jim	A-2, G-11
Kurtz, Barry D.	B-3
Langley, Jim	C-9, F-11
Langlois, David	P-7
Larson, Orland	A-9, L-11
Leonard, William Jr.	B-7
Lessey, Ken	P-5
Loffler, Ivan	F-1
Mason, J. Michael	F-7
McCauley, George	B-10
McQuillen, Jack	G-6
Mead, Robert L.	G-9
O'Brien, Matthew	A-3
O'Neill, Daniel R.	A-7

Payne, Delores R.	B-9
Peckover, Doug	G-2
Perkin, Brian M.	E-4
Peterson, Don	L-1
Primmer, Paul	D-1
Rakusin, Robin P.	G-9
Rego, Alfredo F.	L-11, B-12
Roland, Vince	P-12
Rowe, Denis	L-4
Sera, Arthur	I-5
Smith, Terry	P-2
Sneed, Jerry L.	I-1, J-1
Souder, Duane	B-11, L-11, P-9
Stamps, Bob	M-3
Steiner, Robert	F-2
Storla, Chuck	C-7
Tobes, Roselie	B-1
Trasko, Mark S.	M-6
Turner, Edward A.	M-4, F-12
Valby, Nancy	P-10
VanBempt, Gurdo	C-8, E-6
VanDamme, Jaak	C-8, E-6
VanLeeuwen, J. F.	P-4
VanSickle, Larry	J-4
Ventresca, Benjamin Jr.	F-4
Vigmalou, Dr. Joseph	I-4
Volokh, Eugene	D-4
Volokh, Vladimir	D-4
Warwaruk, Marshall	I-9, J-9
Wheatley, Jack E.	M-2
Wimer, Ted	L-7
Wolf, A. Steven	B-2
Womack, Robert L. IV	A-4, C-5
Workman, Ted	P-12
Wright, Norm	L-1

A SYSTEMS DEVELOPMENT METHODOLOGY BASED UPON AN ACTIVE DATA DICTIONARY/DIRECTORY

T. M. Curtis
Quasar Systems Ltd.
March 1981

1. Introduction
2. Historical Perspective
3. Proposed Approach
4. Method and Technique
5. Obstacles to Implementation
6. Conclusions

Introduction

Computers are not very tolerant of humankind communications. Phrases such as

"you know"

"things"

"what do you call it"

"etc."

are incomprehensible to the average COBOL compiler. Similarly people are not tolerant of the machine's "pickiness" and need for detail. As a result of this communications gap, the EDP professional has leapt into the breach to become a translator between the two uncompromising camps, translating the needs of the user into language and terms that may be manipulated by the computer as well as explaining the strengths and limitations of the machines to unsophisticated users.

The problem with this approach is that the translator has become the key element in the cycle. All communications dealing with the development of computer systems must pass through the EDP professional in both directions.

In recent years there have been dramatic increases in the demand for automated systems and the power of machines to provide services.

However, the supply of EDP professionals (translators) has not kept pace with either the demand for services nor the hardware capacity to deliver the required services. As a result, the limitation on fully utilizing the new hardware power to address the burgeoning demand is _us_, the EDP professionals.

The traditional approach taken to solve this problem has been to increase the productivity of the EDP professional. A procession of analytical, design and programming techniques has been combined with more powerful languages, data management systems and utility software to address the EDP productivity problem. Although these facilities are worthwhile in their own right, they are merely treating the symptoms rather than the problem.

Our challenge is to develop more sophisticated tools for computers and to raise the level of technical literacy of their users so that they may directly interact with the computer for "routine" development processes. This is a natural continuation of the process that has relieved EDP organizations of the burden of data preparation and entry by using hardware to switch from card input (controlled by EDP organizations) to direct data entry using DDP and on-site terminals. That is, we have turned over operational control of systems to the user. The next evolutionary step is to return routine development tasks to the user.

The problem of increasing the level of technical literacy within our society must be left to our educational systems.

This paper will address the opportunity presented by the need to support direct user/computer communications to effect the development of automated data processing systems.

Historical Perspective

Although each of us may follow slightly different analytical, design and development methodologies, the underlying principle is the same:

--determine requirements

--design a computer system that will satisfy the requirements

--develop the system

The requirements are usually, or should be, phrased in non computer oriented language, comprehensible to the user. These requirements are then transformed into a computer design, and the functions and data are translated into process descriptions.

We have traditionally organized our approach to preparing detailed procedures (programs) into data and processing specifications. The file structures, record layouts and field descriptions are prepared. The programmer must then combine these data descriptions with the procedural process descriptions in a program.

We know from experience that the result has been large, unwieldy, incomprehensible, and unmaintainable programs and systems. To overcome this problem we have adopted structured techniques that stringently define the domain of a process and the size of the resulting module. This is essentially an attempt to limit the number of variables and levels of data and processes the programmer must concurrently deal with.

By limiting the size and complexity of modules we have been able to keep the entire complex of data and processes within the intellectual grasp of the programmer. The result of this structured technique has been to achieve greater productivity and dependability through simplifying and standardizing the fundamental system building block, the module.

However, these techniques do not produce the increases in productivity necessary to respond to current and projected demands.

Proposed Approach

General

The dramatic increase in the power of computing hardware coupled with the relative decrease in cost provides us with the basis of a solution: if we can divert some of the power of the machines from "getting the work done" to easing the man machine interface we can reduce the comprehension gap between users and machines. This approach has previously not been feasible because of the cost of machine power necessary to support this level of interface as well as the requirement to get the job done, ie., application systems required all available cycles.

Theoretical Framework

All systems are composed of two elemental items:

a) an entity

b) a relationship

It is possible to comprehensively describe a system in terms of the component entities and relationships that make up that system.

Definitions:

Entity:	"Thing's existence as opposed to its qualities or relations."

We have problems manipulating concepts on a machine, therefore for our purposes an entity may be considered to be;

: That characteristic of something that identifies, describes or quantifies it.

Relation:	"What one person or thing has to do with another."

for our purposes a relation may be considered to be;

: A characteristic or series of characteristics that establishes a link between entities based upon some common identity, description or value.

There are three types of entities that can be used to describe the <u>nodes</u> of a system:

a) data

b) processes

c) users

Data, or course, refers to the information maintained, manipulated or produced by the system.

Processes refers to the rules, precedences, time sequences and operations to be performed on the data handled by the system.

Users refers to the owners, users, controllers and authorities responsible for and involved with the system.

Each of these entity types may be further subclassified as follows.

<u>ENTITIES maintained by DD/D</u>

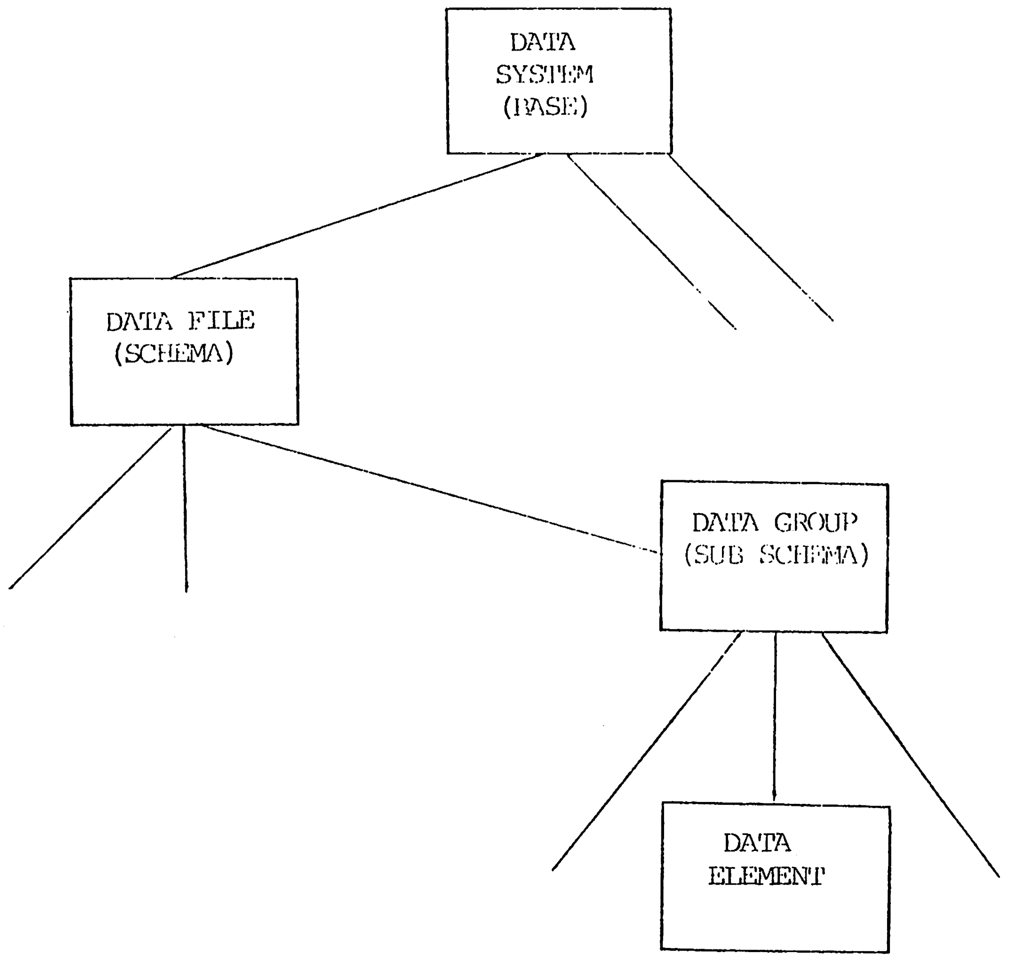
<u>DATA</u>

- Data item	- a primitive	- data definition
- Data group	- sub schema	- (record) 1st order assoc.
- Data file	- schema	- 2nd order association
- Data system	- (base)	- 3rd order association

<u>PROCESSES</u>

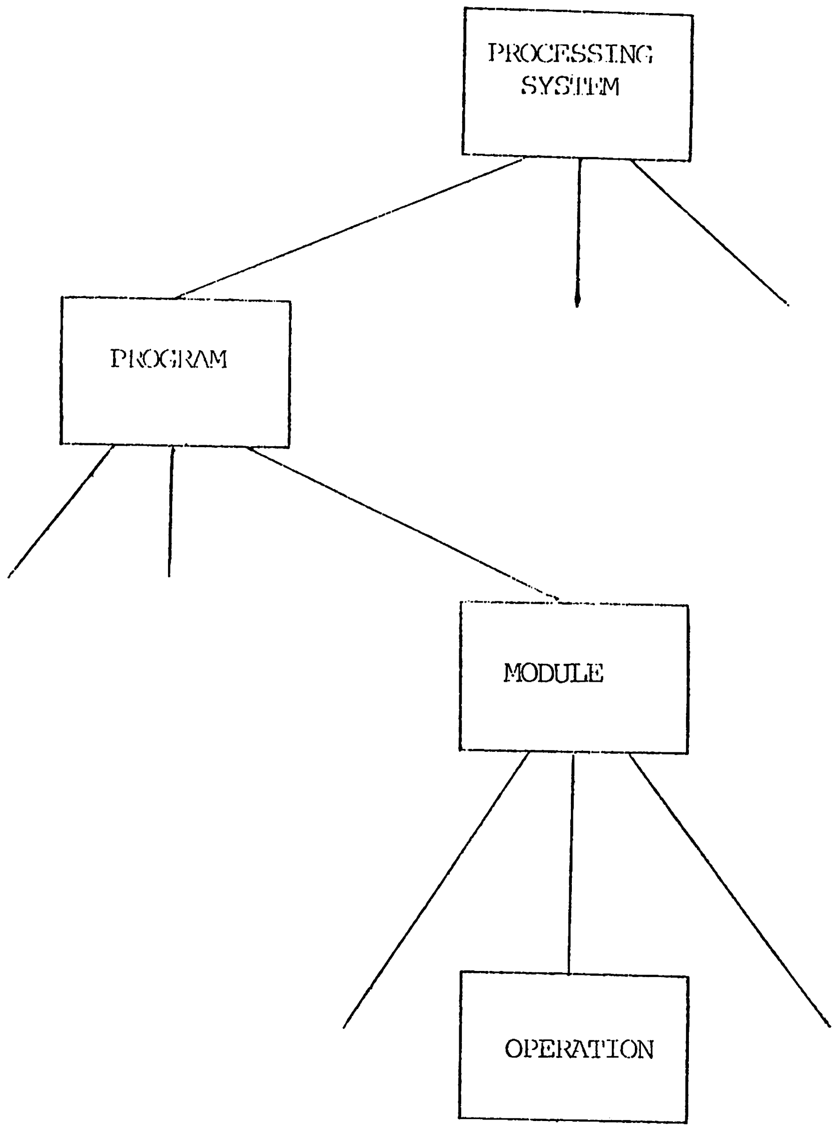
- Operation	- a primitive	- "+" "-" etc. QUIZ & QUICK commands
- module	- an association of functions	
- program	- an association of modules	
- system	- an association of programs	

A physical hierarchical view of the entity types is as follows:



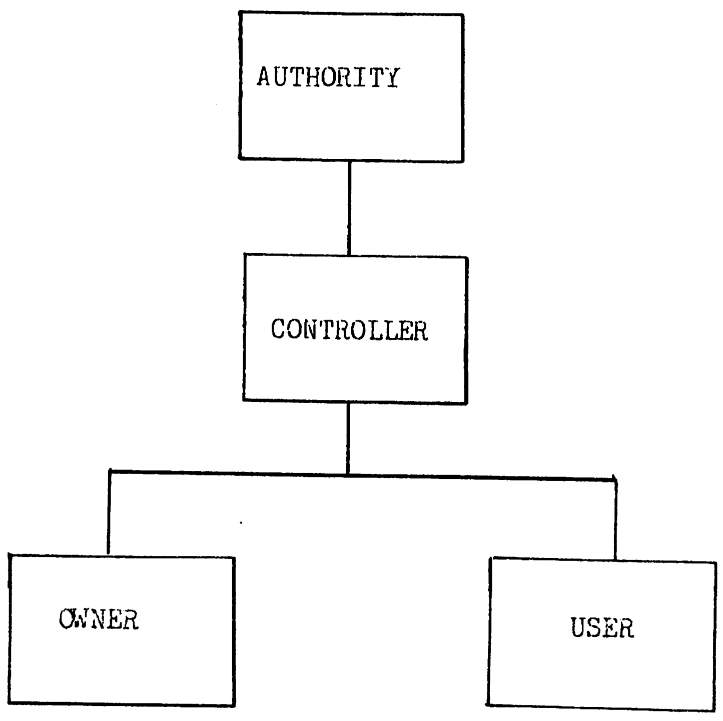
DATA HIERARCHY

Fig. 1



PROCESS HIERARCHY

Fig. 2



USER HIERARCHY

Fig. 3

<u>USER</u>

- Owner	- person or process responsible for accuracy and timeliness of value
- User	- person or process that "uses" the data
- Controller	- person responsible for controlling access to the data or process
- Authority	- person responsible for the definition of the entity

Relationships between entities may be of two categories and take one of three forms.

Relational categories are:

a) absolute: the relation between the associated entities exists at all times and under all conditions.

b) conditional: the relationship between associated entities does or does not exist based upon the value of another entity or the result of another relationship.

The three forms of a relationship are:

a) Relative

b) Associative

c) Algorithmic

a) Relative: "What one person or thing has to do with another."
"Kind of connection, correspondence, contrast or feeling that prevails between two persons or things."

for our purposes we will consider a Relative Relationships to be;

: A grouping of entities that collectively identify, describe or quantify a higher level entity.

e.g., all information maintained on an employee is related and provides an identification, description and "value" for that employee.

b) Associative: combine for common purpose

: connection between related ideas

: thing connected with another

for our purposes we will consider an associative relationship to be;

: A grouping of entities based upon a common or related value of individual or grouped elements.

e.g., all personnel working in the products office are "associated" entities.

c) Algorithmic: process or rules for calculation

for our purposes we will consider an algorithmic relationship to be;

: A procedural relationship established between entities with the purpose of identifying, describing, quantifying or deriving another entity.

e.g., the entity "net pay" may be derived algorithmically as Gross Pay minus Total Deductions.

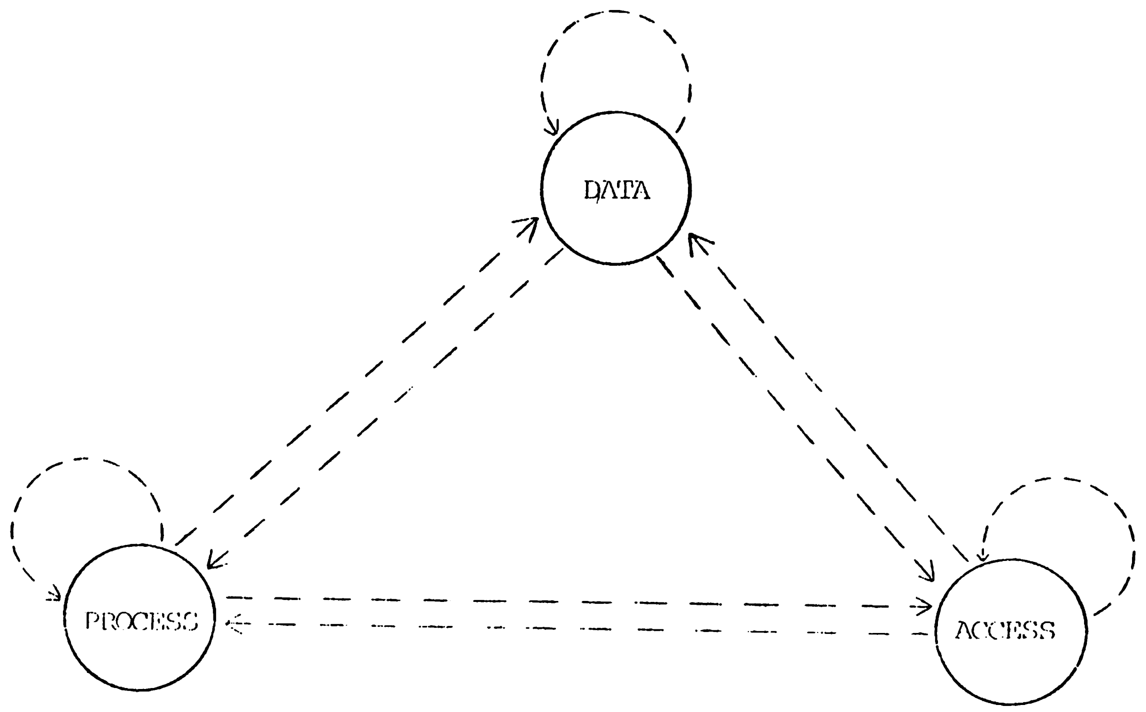
Relationships may be established (may exist):

a) between like entities

b) between dissimilar entities

c) both like and dissimilar entities concurrently

d) recursive (a part may itself be composed of parts, etc.)



RELATIONSHIPS BETWEEN ENTITIES

Fig. 3a

Method and Techniques

The active Data Dictionary/Directory appears to be a viable tool to implement an entity/relationship description of a computerized system. We are all familiar with a number of passive data dictionaries used basically for documentation and data structure source language generation. Packages exhibiting these characteristics have been on the market for years. More recently some dictionaries have become more active, actually resolving references to stored data entities.

The passive Data Dictionary/Directory has the typical structure shown in figure 4. Of course, most current data dictionaries do not maintain process descriptions below the compile unit level, typically a program or subroutine. This structure is not conducive to efficient or effective handling of entity/relationship descriptions. A proposed structure for an active Data Dictionary/Directory is provided in figure 5.

TYPICAL DD/D STRUCTURE

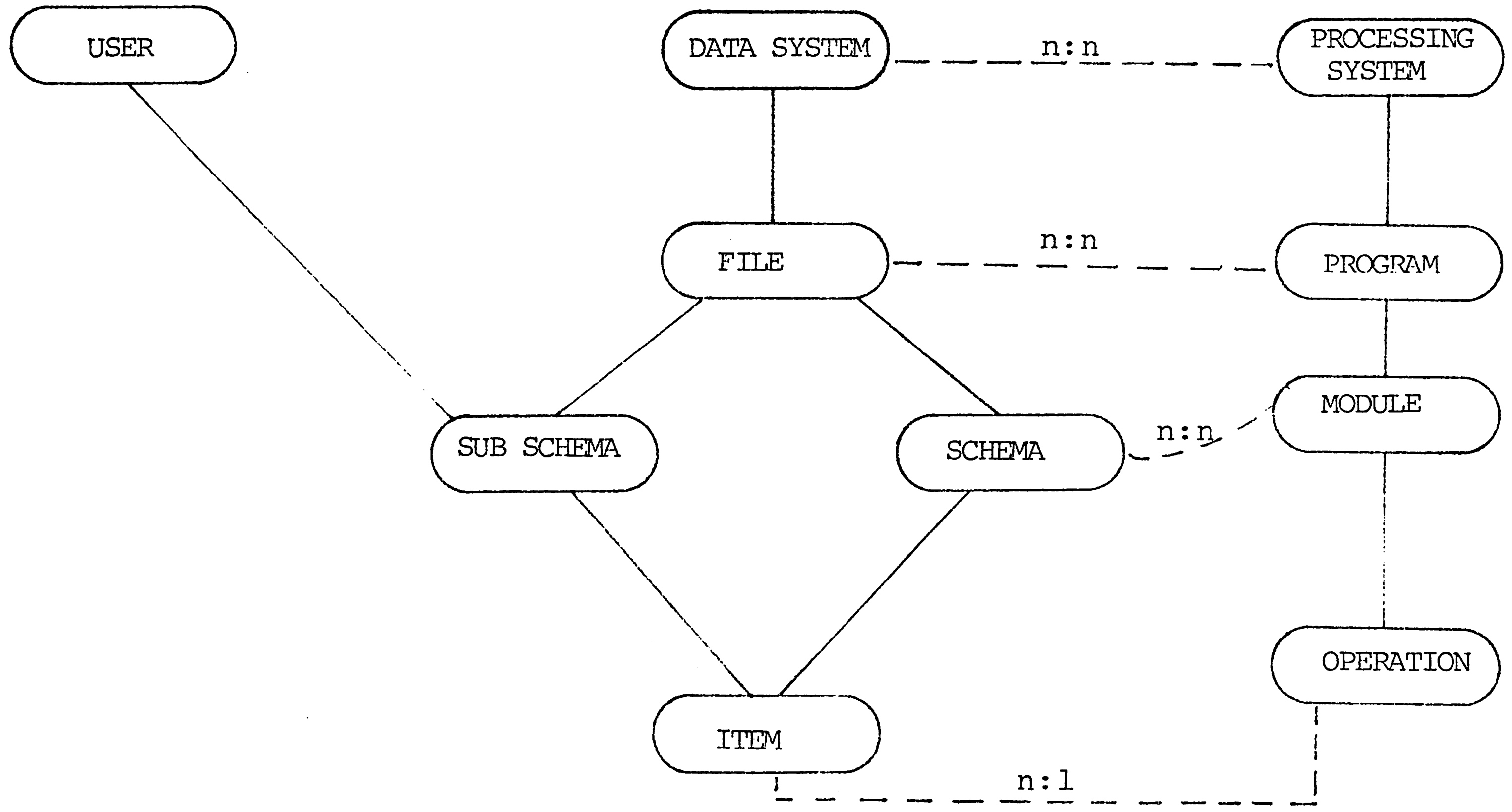
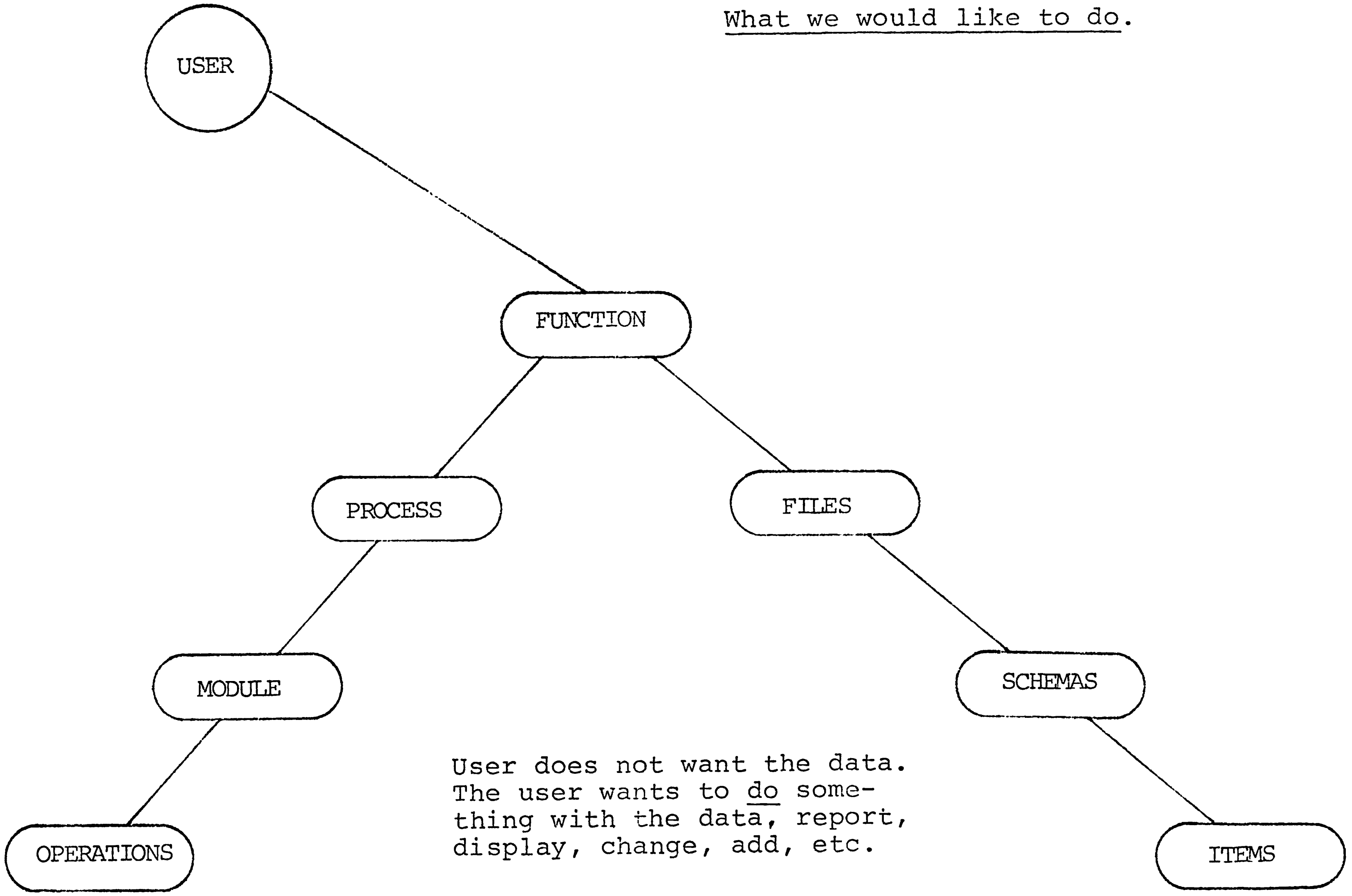


Fig. 4

What we would like to do.

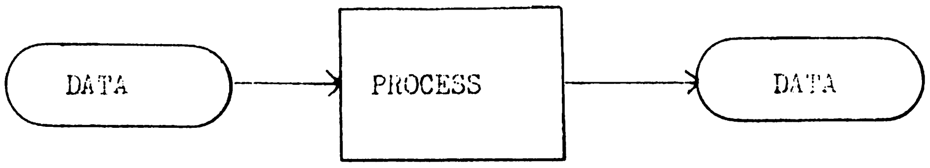


User does not want the data. The user wants to do something with the data, report, display, change, add, etc.

Fig. 5

The structure may be interpreted such that a series of hierarchical processes and data entities form a set joined by relationships. This set of entity/relationships has been organized and termed a function that is "owned" or "used" by a user entity. Therefore, to use structured terminology, if we can establish and define data/process relationships at each level of the hierarchy we wil be able to describe a system. If this description can then be maintained and manipulated by an active Data Dictionary/Directory, we will have established a Function Processor.

A very simple example of this concept is shown by:



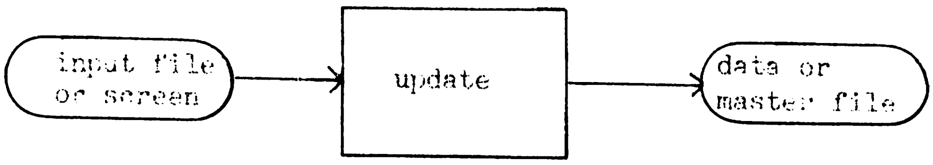
Data can, of course, be a structure

- input file (screen, etc.)
- output report (file, etc.).

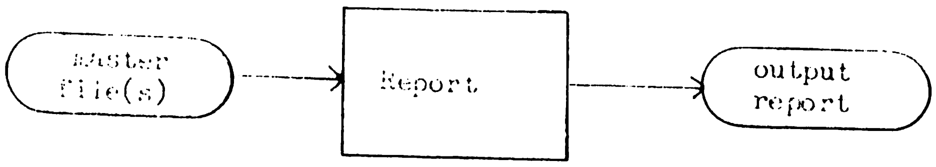
Process can be

- move
- add
- display, etc.

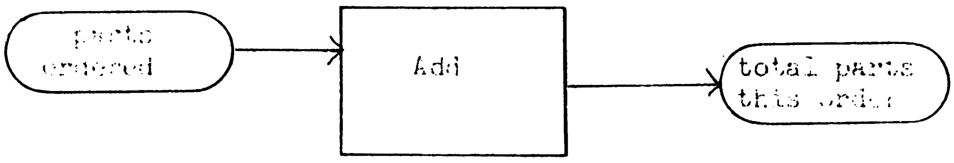
It is possible to describe processes as a series (time sequence) of such entities. At higher levels the description takes the form:

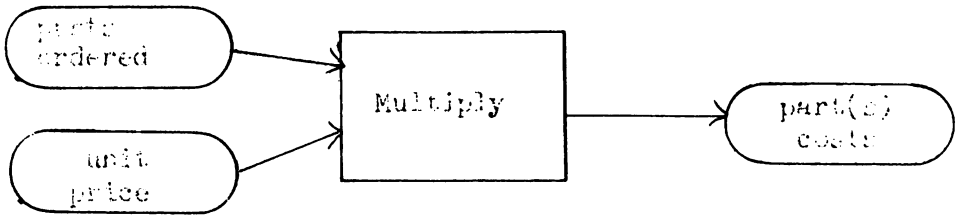


or



At a lower level, the process/data relationship may be described as:





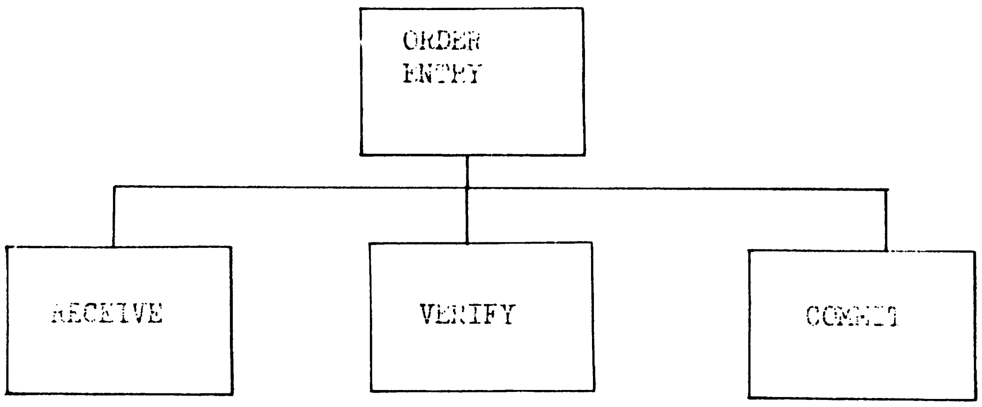
The best way of describing how this approach may work is through the use of an example. An order entry application has been selected for demonstration purposes. The processes to be performed are:

a)	Receive order	(Receive)
b)	Perform verification and credit checks	(Verify)
c)	Commit stock from inventory	(Commit)
d)	Back order "shorts"	(Back Order)

e)	Print picking slips	(Pick slips)
f)	Generate invoice	(Invoice)

for purposes of this discussion we will consider Back Orders, Picking Slips and Invoices as products of Commitment.

We may therefore describe the process hierarchy as:



The corresponding data entities are:

a) Orders

b) Client file

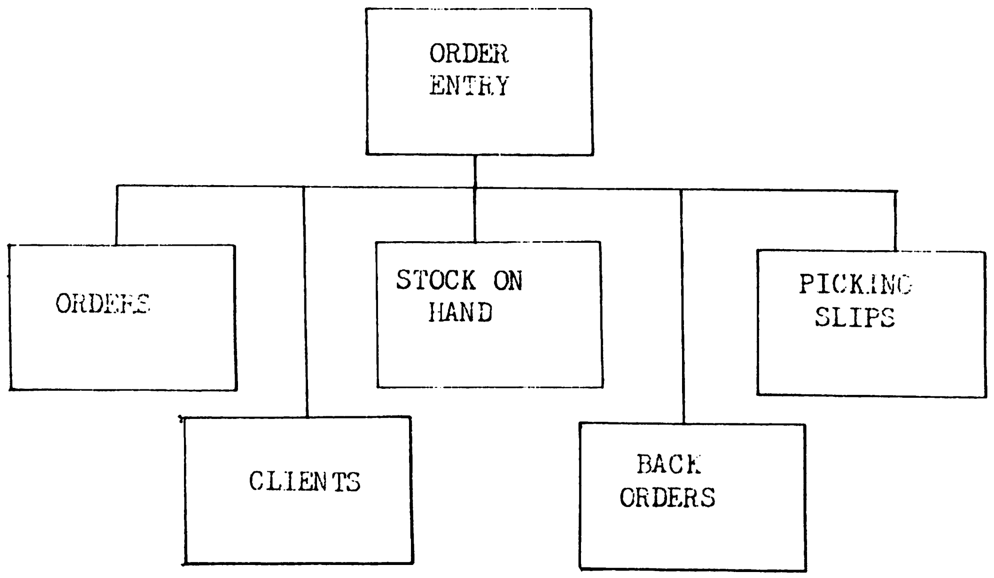
c) Stock on hand file (INVENTORY)

d) Back Orders

e) Picking Slips

f) Invoices

The data hierarchy may therefore be represented as:



It should be pointed out that the association of a data entity to a process entity does not imply ownership. Rather, the relationship may be classified as:

a) use or input

b) update

c) create

d) delete

e) derive

We may define this function as

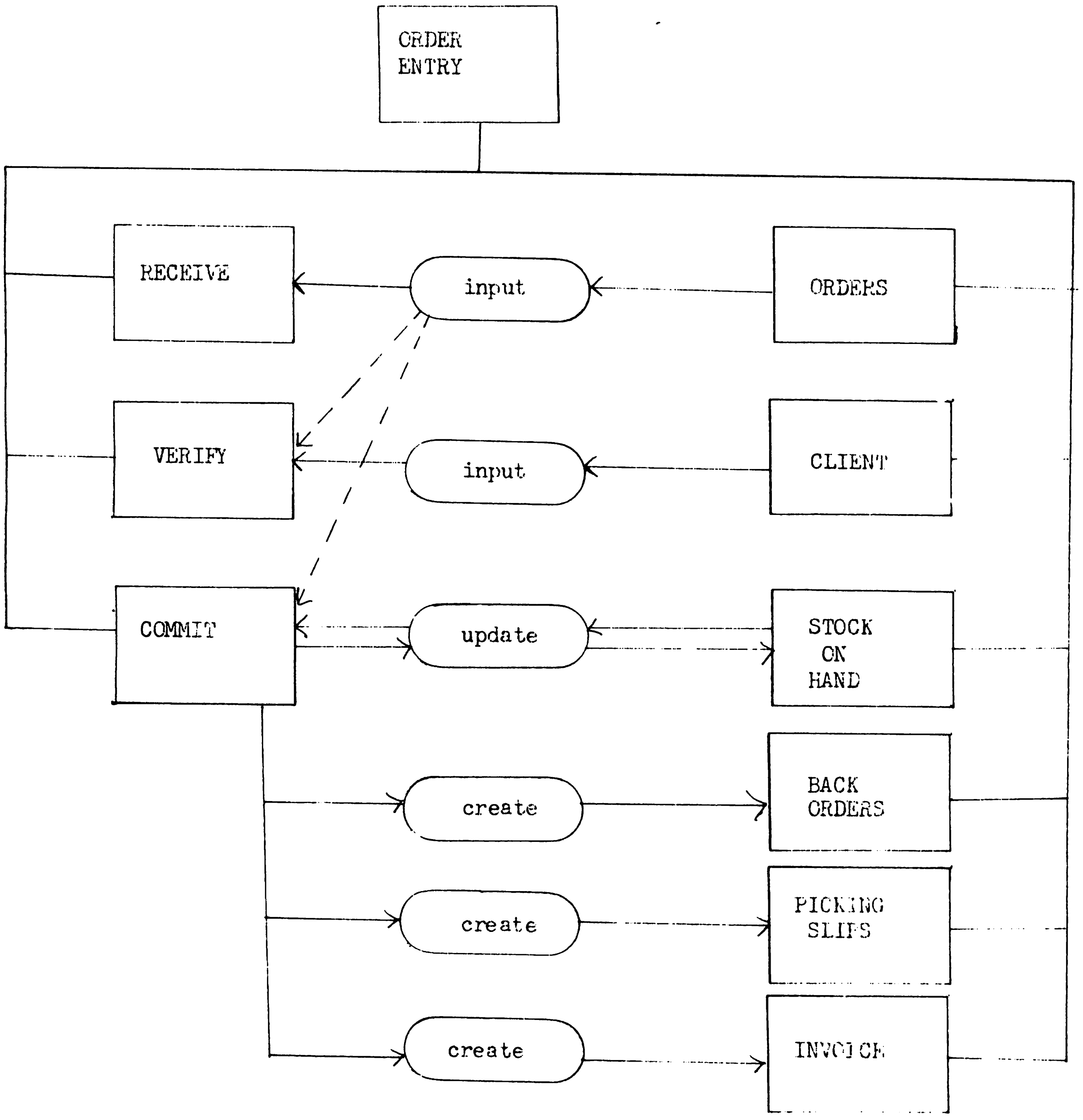
Function: ORDER ENTRY

PROCESS	DATA ENTITY USED	USE	AVG. VOL.	DISTRB	PEAK
RECEIVE	ORDERS	INPUT	Information to be used to support the structure design.		
VERIFY	ORDERS	INPUT			
	CLIENTS	INPUT			
COMMIT	ORDERS	INPUT			
	STOCK	UPDATE			
	BACK ORDERS	CREATE			
	PICK SLIP	CREATE			
	INVOICE	CREATE			

The RECEIVE process is straightforward and does not have to be elaborated upon here.

The VERIFY process is interesting. In essence VERIFY is meant to apply the validation rules that are part of the data definition for each item in the source structure. A sample source structure, in this case an order, is presented below.

```
ORDER NUMBER
CLIENT IDENTIFIER
CLIENT ADDRESS
CLIENT CONTACT NAME
CLIENT CONTACT PHONE
SALESPERSONS IDENTIFIER
DATE
ORDER LINE
            PART NUMBER
            PART DESCRIPTION
            PART UNIT PRICE
            LINE EXTENSION
SALES TAX
TOTAL
```



ORDER ENTRY ENTITY RELATIONSHIPS

Fig. 6

Each data item of the structure is defined as a data element within the dictionary.

Typical definitions will contain:

- unique name
- synonyms
- description and purpose
- type of data
- edit/validation rule(s), severity and messages
- source or derivation
- principal site
- responsibility
- authority
- security restrictions
- links to processes
- links to other data items.

Once the relationship between the process (VERIFY) and the data entity (ORDER) has been established the functional processor may then sweep the data structure applying the edit/validation rules. These rules are not limited to range and type checks but can reference other data items described in the dictionary. As an example, the verification of the CLIENT IDENTIFIER may involve the application of a number of rules.

a) Type is numeric

b) Range 000 - 999

c) Registered on client file CLIENT FILE: PRESENT

d) CREDIT equal OK

The first two rules are simple checks. The third and fourth rules require the resolution of data entities contained in other (but related) data structures. The projected operations of the Function Processor in resolving these references would be:

a) retrieve definition for CLIENT IDENTIFIER

b) resolve "immediate" rules

d) determine other entities required: CLIENT FILE: CREDIT

e) resolve physical location

f) obtain physical representation (record)

g) apply rule(s)

h) set status, produce message, etc.

As this example has illustrated, it is possible to perform the VERIFY function by invoking a primitive operation (VERIFY) and relying upon the data and process specifications maintained by the Data Dictionary/Directory.

Similarly the COMMIT process may be simplified to four subprocesses:

- update the stock on hand
- create back orders
- create picking slips
- create invoices

Again the data and process structures used to perform these operations may be defined in the active data dictionary/directory. The relationships between origin data structure, process and target data structure are given by the structure and linkages of the dictionary.

Utilizing a top down approach to system specification we are able to comprehensively describe the application in a hierarchical fashion.

Once we have the hierarchy of functional blocks we may further decompose the problem until the leaves of our hierarchical tree represent primitives in terms of data and process entities. Anyone familiar with the Jackson methodology will be acquainted with the technique and representation of processes and data.

- Data structures are represented hierarchically.
- Relationships are expressed as correspondences.
- Processes are "operations lists" and are merged with the consolidated data structures, ie., hierarchically structured.

This organization can be maintained by a data dictionary. Indeed many data dictionaries already maintain most of the information necessary to support this type of process/data description.

Obstacles to feasible and practical implementations.

There are two major obstacles to a feasible implementation of this model. The principal difficulty is the development of a non procedural grammar that is concurrently

a) natural language like - to allow the end user to specify data and processes in familiar terms

b) structured enough to provide comprehensive and unambiguous data and process descriptions to the system.

The second obstacle deals with the operating efficiency of such a system. The hierarchical trees of entities and their relationships can quickly become extremely large and complex for even medium sized applications. The organization, maintenance and reference of such a structure will require considerable sophistication to provide the responsiveness, performance and reliability necessary to produce a workable tool.

Conclusion

The techniques and approach outlined in this paper depend upon tools and technology currently available. What is necessary is the impetus required to revise our thinking about how we specify, design and develop computerized systems. The challenge of closing the gap between the unsophisticated user and the uncompromising computer can be addressed from either end. This approach attempts to use the power of the computer to accept non procedural, non technical descriptions of functions, data and processes, and generate automated systems.

The Technology of the Quad Editor

by
Jim Kramer
Hewlett-Packard Co.
St. Louis, Missouri

I. Introduction

The Quad editor is a simple text editor that is being contributed to the Users Group library at these meetings. It has several features which make it notable and useful, the most important of which are that it texts files instantaneously and that it can undo any or all editing changes. The purpose of this paper is to explain the technology behind these and other features.

II. A Brief History and Description of Quad

Quad was created to avoid the high overhead of Edit/3000's text and keep commands, which are essentially file copy operations.

Early versions of Quad simply opened the texted file and made changes directly to it. This precluded the possibility of adding or deleting lines from the file. It also made editing a somewhat risky business, since changes were being made directly to the file rather than to a work file copy ("QUAD" originally meant QUick And Dirty). Nonetheless Quad was very fast for looking at files and making simple changes.

The current version of Quad retains most of the advantages of the early versions and eliminates the main deficiencies. A file is still texted just by opening it, so that the overhead of a file copy is eliminated. However changes are kept in a work file, so that a user is not committed to them until he does a keep of the file.

Having separate text and work files requires Quad to logically merge the two to give the user the illusion of working on a single file. However it also makes it possible to cancel any and all changes just by removing them from the work file; Quad's undo command returns all lines within a specified range to their original state.

In general Quad must, when keeping the edited file, create a new file which merges the text and work files. However if the keep is back to the texted file and no lines have been added or deleted, then the keep is done just by updating existing records in the text file. Thus whenever possible the file copy on keeping is eliminated also.

It is important that Quad be able to find lines in the texted file quickly. Quad starts out with no knowledge of the location of lines in the file, and must find requested lines using binary search. However Quad keeps a record of all blocks read during the search process and uses this record to shorten subsequent searches. The method is described in a paper titled "A New Tool for Keyed File Access (Sometimes)" in the proceedings of the Users Group's 1980 North American meeting.

III. The Work File

Quad creates a work file only when the user first makes a change to the texted file. This can be any type of change -- adding or deleting a line or modifying an existing line.

The work file is a keyed file which allows both keys and data to be variable length. It can contain two types of entries -- deletes and changes.

Quad allows deletion of a range of records -- for example "D 2/9" is a command which deletes line numbers 2 through 9. To do the deletion, Quad makes a single entry in its work file as follows:

```
D0000200000009000
```

This is just a 17 character key. The first character is a "D" (for delete), the next 8 characters are the lower line number in the range, and the last 8 are the upper line number

Since deletion is achieved with a single work file entry, it is very fast, and the speed is independent of the number of lines being deleted.

Now suppose that the command was given to delete records 8 through 14. This command would normally result in an entry of D8/14 into the work file. However this range overlaps an already existing delete range of D2/9, so that Quad would combine the two into a single entry of D2/14.

If the user now undid changes over the interval from 5 to 8, it would be necessary to "undelete" over this range. Therefore the entry D2/14 would have to be split into two -- D2/4.999 and D8.001/14.

The other type of entry in the work file is a change entry. There is a change entry for every line added during editing and every line modified.

A change entry consists of both a key and data. The key is just the letter "C" followed by the 8 character line number, and the data is the line of text corresponding to that line number.

IV. The Structure of the Work File

The work file used by Quad was originally designed for another purpose -- a different editor. The desire was for a file access method which gave random access to variable length records, and re-used space from deleted records.

The solution is a file access method which I call ticket files.

With most file access methods, the user who wants data stored specifies where it is to be stored -- a record number. With ticket files the user does not specify; instead he just supplies the data to the access method and receives back a "ticket" telling him where the data has been stored. In order to retrieve the data, he just supplies the ticket.

It is important to recognise that this technique gives enormous flexibility to the file access manager. The data can be put in the most convenient spot, for example a block that is already in a buffer in main memory. Within the block the record can be placed wherever there is space. With ticket files a record need not even be placed contiguously within the block -- it can be broken into pieces.

Although ticket files might at first seem unnatural or even clumsy, they turn out to be perfectly suited to those applications in which data is found through pointers: tickets are really just pointers.

Image detail data sets are an example. If we neglect serial access, a detail data entry is found through pointers which are stored in other detail entries or in a master entry. Ticket files could in fact be used to implement Image details and would relieve Image of the burden of keeping track of free space. Further they would add a capability that Image details currently do not have -- variable length records.

KSAM files are another example -- all data in a KSAM file can be found through pointers if only the location of the root block in the key file is known. Ticket files will, by the way, remember one ticket for the user.

In order to make ticket files satisfactory as work files, it was necessary to implement a keyed sequential access method based on ticket files. The implementation is significantly different from KSAM and actually more powerful: both keys and data can be variable length, space is re-used, and keyed sequential access can be either forward or backward.

The keys are stored in a tree-structure called a trie. In a trie a key value is actually distributed through various levels of the tree structure; branching occurs when two keys which are identical for some number of beginning characters first differ.

For example the keys "ANDY, "ARMOR", and "ANDROID" would result in the following structure:

```
          A
         / \
       ND   RMOR
      / \
     Y   ROID
```

Generally this structure will have more levels than KSAM's B tree. On the other hand its structure is a function of the data itself, not of the order in which the data is loaded. Therefore tree re-organization is never necessary during loading, whereas with KSAM it usually is.

When a key is stored, a ticket is stored with it. The ticket points to data. Thus storing data by key is a two-step process:

1. Store the data and receive a ticket.

2. Store the key and the ticket.

Retrieving data by key reverses the two steps:

1. Supply the key and receive the associated ticket.

2. Use the ticket to retrieve the associated data.

V. The Help Facility

From the start I wanted Quad to be a single program file not requiring any auxiliary message or documentation files. The reason is that I wanted the acquisition and use of Quad to be as simple and foolproof as possible.

This meant that Quad had to have a very good help facility -- a built-in manual. Quad does in fact now have quite an extensive menu-driven help facility. All of its text can be printed offline to make an adequate user manual.

A help facility presents some technical problems. In the first place it should be very easy for the programmer to write the help text. Moreover the text must be prevented from making the program enormous: text takes up program space very quickly. Therefore blank compression is very desirable.

The solution adopted in early versions of Quad was simple but not fully satisfactory. A line would be written to the screen by doing an SPL move of a literal to a buffer followed by a call to a procedure to write the buffer to the screen. By using SPL defines it was possible to make the code to write a line look like the line itself bordered on both sides by a bit of auxiliary text. Thus the code

```
      BQO "This is a line to go to the screen" EQO
```

was shorthand for the code:

```
      MOVE  MSG:=("This  is  a  line  to  go  to  the screen",0);
      WRITE'TO'SCREEN(MSG);
```

This made help text very easy to write. Unfortunately it also wasted code space. Each line to be written required its own move and call, and there was no blank compression (although trailing blanks could be suppressed).

An efficient solution to save code space is to have each screen of help text stored in a PB-relative array in a compressed form. (A PB-relative array is an SPL array which is part of a code segment). The procedure containing the array would fetch the text line by line in a loop and call another procedure to write each line to the screen. In this way there is only one set of move and call code for the entire screen rather than one set per line.

The problem is that it is very difficult to write code that initializes arrays with blank-compressed text. It would be far better to just write:

```
               (*    Help'proc : Help'seg       *)

     This is a block of help text, which we would like to be
     converted to a space efficient procedure named Help'proc
     (for the segment Help'seg)

                               (**)
```

The solution of course is to have a program which converts such blocks to the desired procedures. Quad's help facility was written using such a program which served as a pre-processor to the SPL compiler.

VI. The Command Interpreter

There were two main objectives in the implementation of Quad's command interpreter:

1. To catch as many errors as possible during command analysis, rather than command execution.

2. To emit the most meaningful possible error messages.

As an example, editors must have commands which include file names. The editor could be designed do no checking on the file name; any errors will be detected later on by the file system. Quad, however, assures that the name is syntactically valid first. For example it checks that there are no more than three parts to the name (file, group and account), that each part has between 1 and 8 alphanumeric characters with the first being alpha, that there is no more than one lockword, that the lockword follows the file part, etc. Any error found results in a message which describes that particular error.

One aspect of emitting meaningful error messages is to point out where in the command the error occurred. To this end Quad points to the error, just as MPE does for command errors.

DISTRIBUTED PROCESSING
A HEWLETT PACKARD SOLUTION

Matthew O'Brien
Section Manager
Hewlett Packard General Systems Division
19410 Homestead Road
Cupertino, California 95014

The purpose of this paper is to present a new concept in the way in which data processing is done within any organization which presently utilizes a central mainframe computer with terminal access distributed between many users.

The term distributed processing has had various meanings through the development history of different computers. One meaning that might be attached to the term is that which also might be called array processing. This involves an array of processors distributing the power of the CPU and performing tasks in parallel to accomplish the computation in a shorter period of time. This is definitely not the meaning that I wish to attach to the term distributed processing.

For the purpose of this discussion, the following phrases characterize 'distributed processing':

- localization of some computational power and program memory

- maintenance of a central node for computation and data base
- minimization of datacommunication traffic
- utilization of the relative strengths of distributed CPUs
- maintenance of privacy by means of local data bases
- utility of shared central mass storage and peripherals
- concept of synergy of "one man - one machine"

This definition warrants an easily understood clarification, as the concepts are more easily grasped with the presentation of a concrete example. The distributed processing referred to is that which is achieved by clustering together a group of what has been termed 'personal computers' around a central node consisting of a mainframe CPU. Unlike the simple terminal interface to a central CPU which has been prevalent, this configuration leads to clear advances in price, utility, performance security, etc. Before proceeding, the terms personal computer and mainframe CPU need clarification.

The mainframe computer was the first result of constructing electronic devices to perform large amounts of computation or calculation. Prior to the late 1930's and the early 1940's, rudimentary machines had been constructed to handle either calculation with numbers or some other sorting or controlling function. In order to handle problems which involved extreme efforts of mental and hand calculation, investigations were begun into constructing an electronic machine which would automate the calculation process. Perhaps one of the most famous examples were the calculations to produce a book containing tables of artillery projectile paths under varying conditions of shell mass size, charge mass and volatility, wind conditions, atmospheric density and of course barrel elevation and azimuth. As so many variables were involved and such great accuracy was desired, it was necessary to perform many hundreds of thousands of calculations to produce a satisfactory result.

This example serves well in showing the emergence of the mainframe computer for two reasons:

- the machine was constructed largely for a single purpose, to perform large numbers of similar calculations
- it was technilogically impossible to produce a computer capable enough, portable enough, and in great enough numbers to couple them directly with the artillery units to produce real-time computation

The artillery projectile computer project was successful and interest grew rapidly in performing diverse computational tasks. However, fundamental limitations still existed, the primary for this discussion being the great expense of producing the central processing unit and the amount of maintenance to keep it performing correctly.

As the years went by great improvements were made in refining the CPU, however it's expense, bulk and necessary level of maintenance continued to justify it's name -

central processing unit.

The purpose of this immediate topic is to stress that the computational structure of the mainframe developed not due to its inherent suitability for the job, but due to technological limitations in producing inexpensive, portable and reliable computational machines of enough capability to allow each user his own processor. Granted this limitation, the only practical solution required a central processor with multiusers timesharing the CPU through terminal ports. This multiuser aspect allowed sufficient utility to amortize the comparatively expensive CPU, and continues to be reflected today in the continuing drive to allow greater numbers of users to share the same machine, driving down the per-user cost of computational power.

Turning now to defining the meaning of personal computer, it must be stressed that the term can produce varied opinions. The preferred definition here is a microprocessor-based processing unit with additional local program and data memory and some form of mass storage and I/O capability. More abstractly, a machine with sufficient power and utility to be used in a stand-alone mode with the capability of being programmatically altered to perform a very wide range of tasks. The last point is important as it is wished that programmable calculators be excluded, their use being too limited to manipulation of numbers and device control.

The element that has made possible the personal computer is the large scale integration of many semiconductor devices onto monolithic chips. This has led to the realization of an effective processing unit which is inexpensive, very portable and highly reliable.
Personal computers cost a fraction of the price of their computing counterparts of ten years ago, and fill the requirements of cost, reliability and portibility necessary for personal use.

Subsequent to the emergence of the first microprocessor and the continued density improvements of RAMs and ROMs in the late 1960s, there emerged the use of these components as a replacement for large amounts of combinational circuitry that had previously been needed to perform certain electronic control functions. These first uses of microprocessors did not justify the name computer, as no means of user programmability was available.

By the mid-1970s the personal computer began to emerge, tentatively and lacking in capability, amount of memory, sufficient I/O and most importantly, software. Given these realities, the machines generally found usage solely as means of technological amusement and as a means of playing simple games. By the late 1970s a fundamental change had occurred and personal computers began to be used in serious applications in science and business.

Today, the personal computer is recognized as a cost-effective means of automating many previously manual operations. Computationally the processor is able to manage many demanding tasks and performs quite well in many applications. Increasing emphasis on increasing the performance of the processor and lowering the cost of the necessary I/O functions and peripherals continues and can be expected to yield new generations of increasingly cost-effective personal computers.

Having discussed these two classes of computers and having brought their development to the present, the next issue that needs to be examined is where do these computers go from here? Will increasingly more advanced technology allow personal computers of ever increasing performance and ever lowering price to become so capable and affordable as to displace forever the mainframe?

My perception of this question is that the answer is no, that the mainframe will continue to serve an important portion of the data processing system requirements of most organizations for the foreseeable future. It is important to note the restriction is made to be most organizations, and the validity of this restriction is easily shown as many small organizations today do rely only on a personal or microcomputer as their data processing needs are sufficiently limited in scope as to be adequately met by the microcomputers and small peripherals.

However, the characteristics of computer usage in a large organization are usually different. To corroborate the contentio that the day of the mainframes demise is not immediate, a few specific examples of the differences can be made and broken into two categories, immediate and future:

Immediate

* vastly higher performance of mainframe is needed to perform tasks of high numerical accuracy or time consuming tasks
* very involved and large applications require large core or program memory to successfully execute
* cost effectiveness of sharing expensive mass storage and peripherals

These points as to the need for the mainframe might possibly begin to change or weaken as the evolution of technology continues. However, another larger list can be made which will not as easily be displaced by technological change as they are not technology-dependent but rather are a fundamentally desirable feature:

Future

* the mainframe concentrates and universalizes data bases which are accessed by many individuals
* allows control of the processing functions of the organization to be visible and controlled by management

* allows managerial control of the security of data bases
* makes the backup and physical security of important data more predictable and controllable
* removes from the hands of unskilled operators the necessity for determining the validity of the data base and the funtionality of the computer
* ensures all data processing of critical nature uses the same revision application
* inherently allows communication between users as it implies a common network
* allows access to higher levels of networking as mainframe serves as efficient port
* additionally, it is most probable that while technology will bring cheaper peripherals and memory to the personal computer, it will probably always do so to the mainframe
* finally, it appears that perhaps a new generation of supercomputer might appear using Josephson junction technology, but the cooling requirements will obviate the small size and portability of microcomputers

Enough said regarding the essentiality of the mainframe and the inevitability of the microcomputer. Let us now consider a pair of specific computers; the HP 3000 mainframe and the HP 125 personal computer. Explaining the HP 125 and its interaction with the HP 3000 shows where Hewlett Packard believes the computational system for the medium-to-large organization is headed.

The HP 125 has been designed to be the foremost personal computer available today. As is the case with all Hewlett Packard products, we like to think that the HP 125 offers the customer not a piece of equipment, but also what we believe is more fundamentally important - it is a solution. It brings what we believe are the typical strengths of Hewlett Packard to what is now a somewhat chaotic and young product area. Hewlett Packard has been recognized for some fundamental precepts by which it does business; that the satisfaction of the customer is most important. This is not only the correct attitude, it also has proven to be a good business practice as it has over the years built a clientele of loyal customers. As such, the HP 125 stresses good price/performance, reliability, serviceability, and presents a total solution composed of not just the product but also the system interaction and software to make the hardware investment meaningful.

The HP 125 is structurally based upon the HP 262X terminal family, sharing some common assemblies. The terminal and CPU portion appear outwardly much like a HP 262X terminal, with the mass storage and peripheral devices being connected to an extended I/O panel on the rear.

The HP 125 combines three functional abilities within one package:

* it serves as an autonomous microcomputer
* it serves as solely a data terminal
* it creates a synergy of use by combining the function of the microcomputer with the data terminal

As a microcomputer, the HP 125 operates using the CP/M operating system. This operating system has become a defacto industry standard for use with the 8080 or Z-80 microprocessor. To support the operating system, a Z-80 with 64K bytes of system RAM is used. This constitutes the bulk of the CPU, the only other significant electronics being a boot ROM to load the operating system from the disc connected to the IEEE 488 interface connector and the byte-parallel interface to the terminal portion of the system. With this relatively simple CPU, the CP/M operating system standardizes within the memory space the necessary functions like input/output, file system, etc. which allow applications software to be hardware independent. Manufacturers of hardware who desire to utilize the standard operating system merely customize those portions which are necessary to allow the hardware to correctly perform the hardware dependent I/O functions.

The benefit of supporting the CP/M operating system is that the HP 125 then is able to directly run many hundreds of applications that run under CP/M. Applications include accounting packages, mailing list programs, word processing, languages, etc. with more applications being added to the list daily.

One drawback of the standardized CP/M operating system is that the author of a generalized application package has had to depend upon the least common denominator of hardware I/O capability. This becomes most readily apparent with the terminal interface. Most CP/M systems have been constructed by building a box to contain the CPU. The user then selected a terminal which he connects to CPU box. This of course means that the application written for the CP/M operating system has been forced to assume the least capable set of terminal features as more advanced features are not supported on many terminals.

Acknowledging this shortcoming, the HP 125 will be released with a great deal of specialized software, some of which has been customized for the superior capabilities of the machine by authors of existing software applications and some of which has been written by Hewlett Packard. With these two sources of software in addition to all generalized CP/M software, the HP 125 will bring an unprecedented amount of microcomputer software to the purchaser.

As mentioned, the terminal portion of the HP 125 is a fairly advanced data terminal, utilizing softkey structure to access such features as the mode of logging data from video memory to either the integral thermal printer or the serial printer connected to the I/O port. Softkey tree selection of functions now only serves to lessen the amounts of keystrokes necessary to select functions, but also serves to guide the user.

The softkeys within the HP 125 not only have the inherent functions embedded within them to implement the terminal features, but are also user programmable to contain up to 80 bytes which can be used for everything from string substitution to escape sequences which actuate execution of subfunctions contained in applications. Each user programmable softkey can be accessed from either a keypad stroke or an application program for user selection. An application or user programmed pneumonic label can be placed within the bottom two rows to correspond to each of the eight programmable keys.

With these advanced terminal features, the HP 125 offers advanced features for a CP/M stand-alone computer system.

The HP 125 maintains a separate terminal functionality within its operating capabilities. When power is applied to the system it normally defaults to the terminal mode of operation, with the selection of loading the operating system to become a microcomputer being selectable by the depression of a single softkey. As a data terminal, the HP 125 has capabilities similar to those of the HP 2621, with some enhancements common to more advanced members of the HP 262X terminal family. Additionally, it presents some features not previously available.

First a brief description of the terminal capabilities of the HP 125 before a discussion of those terminal features unique to it.

As a terminal, the HP 125 presents the user with 24 lines containing 80 characters of text. Also on the screen are a 25th and 26th row containing the labels for either the embedded softkey tree structure, or when selected, the user programmable softkey pneumonics. The terminal allows selection of half-bright, underline, inverse video, or blinking enhancements on a line-to-line basis.

The keyboard is the full extended keyboard which contains dedicated cursor control, scrolling, softkey, numeric pad, and screen-oriented editing keys.

Input/output is provided by an IEEE 488 port and two serial ports. One serial port is nominally dedicated to a serial printer, the other to datacommunications.

Datacomm runs at 9600 baud and supports various handshakes necessary for use with different CPUs and modems. The datacomm port also supports the 13265A direct-connect modem. The printer port is configurable for variable amounts of nulls, parity, and the sense of the rate-pacing handshake. This allows the HP 125 to directly use a large amount of serial printers without the necessity of any special logic or cables.

As an option, the HP 125 supports a thermal printer which is integrated into the top of the terminal package. Either this printer or a serial printer (if configured) are supported within terminal firmware by a softkey tree which allows the direct printing of the entire contents of video memory, the visible screen or a selected line. Additionally, logging modes can be set so that all data coming to the video memory or only that data overflowing video memory is printed.

All configuration information is stored in a CMOS RAM which has battery backup, allowing the user-selected configuration to be maintained when the system is powered down.

The terminal supports remote operation and configuration by use of escape sequences. As an example, the keyboard has a 'home cursor' key which positions the cursor at the first character in video memory. An application program can also home the cursor by transmitting the correct escape sequence to the terminal. By this means, applications running in either the CP/M CPU within the system or an application running on a mainframe can efficiently manipulate the terminal features to provide a friendly applications interface to the user.

The afore described features make the terminal portion of the HP 125 a high performance terminal for use with both the CP/M CPU and when used with a remote mainframe. These features are fairly comparable to those which are supported within the HP 262X family.

Additional to these, the terminal implements several unique features which are fundamental for its use as a CP/M terminal interface and which also generally provide better performance.

Within the terminal, an I/O map is maintained which allows the mapping of any source devices to any destination devices. (For the purpose of this discussion, note the terminal considers the output of the CP/M processor to be an input!) An example may better illustrate this:

In order to diagnose a difficulty in running a CP/M-based application, the HP 125 user can map the output (console out) of the CP/M CPU to be not only the CRT screen, but also datacomm port 1. To this port he has connected a modem which ties over the phone lines to another HP 125 (or terminal) on which a knowledgeable user of the application is viewing. By this means, the output of the application and keystrokes entered by the user (CP/M operates in a full duplex mode) can be viewed for debugging. Further, were the user to map datacomm port 1 as the input for the CP/M CPU (console in), the remote viewer can also run the program and allow the direct operator to watch in order to learn the correct manner in which to run the application.

As another example of the value of this feature, consider a CP/M application written to perform an accounting function. Within the application, various output is routed to either the screen or to the printer for hardcopy. Often it is desired that this fixed output routing be altered, perhaps to obtain hardcopy of items normally sent to the screen. With the HP 125 I/O map, this is easily accomplished.

Another distinctive feature of the HP 125 is that all the ROM-based routines which give the terminal portion of the product its capabilities are vectored through locations in RAM upon powering the system on. By this means, an application which doesn't prefer to use the terminal capabilities as dictated by the ROM routines can intercept the routine call and substitute in RAM its own specialized routine. An example of this ability is also illustrative:

In the normal mode of operation, the cursor control and editing keys as supported by terminal firmware allow the user to manipulate the text on the screen directly. However, this 'feature' may not be desirable while in the midst of running an application. The application can consequently be written to intercept the keystroke processing routine and can then trap keystrokes which are extraneous to the application previous to returning control to the terminal ROM code for keystroke execution. Or by this means, the functionality of keys can be altered.

By this method of embedding a high degree of functional capability in ROM but yet allowing customization of routines critical to certain applications, the HP 125 goes well beyond the capabilities of most microcomputers. Very sophisticated terminal features are ROM resident, and specialized features are application programmable.

Understanding the HP 125 from the physical and features standpoint allows us now to address the unique capability that Hewlett Packard brings to the field of making distributed processing an asset for organizations with large and diverse computing needs.

In a previous section, the permanent and essential nature of the mainframe was discussed. As present users of the HP 3000 computer can probably attest, a major usage of the system involves the creation, maintenance and access to data bases which allow the smooth functioning of large organizations. This automation of data base with instant and accurate access has been the principle benefit of the computer to the business world.

Granted that the personal computer and the mainframe have been discussed and the individual merits of both are appreciated, an examination of the interaction of the two for doing distributed processing is appropriate.

Personal computers have begun to appear within the ranks of large organizations for use either by individuals or for the needs of a small department. While the personal computer has obviously fulfilled a purpose, the utilization factor could be greatly larger. The HP 125 performs well the tasks being addressed by the personal computer, but brings much greater utilization without a greatly appreciable higher price.

The function that is easily recognized for a personal computer within a large organization is what may be called data display and analysis. This term is meant to describe the typical interaction of a manager with those performance criteria of his organization represented by a collection of data.

For the display and analysis of data, the personal computer of today tends to fail to efficiently perform its function. The data base for most organizations is large, communal in nature, subject to frequent correction or update, and most necessarily must be current and correct throughout the organization. Using a stand-alone personal computer, much time consuming and detailed analysis has been done only to find the raw data was incorrect due to an error in transcription or a recent update.

Additionally, most information within organizations comes from a multitude of sources. Using a typical division within Hewlett Packard as an example, data bases are maintained that updated or accessed by accounting, personnel, purchasing, scheduling, manufacturing, quality assurance, research & development, administration, etc. This is the data that is the subject of display and analysis.

With todays typical personal computer, the transfer of data between the micro and the mainframe is at best tedious if not impossible or prone to error. The HP 125 strives to make this process the most expedient, error-free and simple process possible. With a wealth of data base management capability available on the HP 3000 computer, the HP 125 leverages great power into the hands of the person who analyzes or updates the data base.

As an example, the HP 125 supports a screen-oriented calculator which allows management personnel to easily create, display and manipulate data. It allows the manager to quickly explore "what if" questions regarding the vital numerical data which represents his success or failure.

Additionally the HP 125 supports a graphical display package which allows significant data to be displayed by means of bar charts, pie charts, etc. With the HP 125, the data for display, analysis and charting can interactively flow over the terminal data comm port to and from the personal computer and the mainframe. All data is from the common base of the mainframe and represents the organizations most recent and accurate figures. All results of analysis can immediately be re-entered into the common data base. Standardized reports from functional areas can access the database from other areas in which they don't necessarily have involvement as to the generation of data, but from which their respective areas can be directly affected.

All functional areas can present reports that are standardized across the organization as to format. Data flows efficiently between organizations, as data entered by one area becomes immediately accessible for all users. The security of the data base is cared for by the information services group, guaranteeing against the hazards of losing critical data. The access of individuals to data is controlled by management; the HP 125 can be programmed to allow only visual display of the data without user copying to printer or disc while the initial access can be protected by the HP 3000 using passwords.

The strength of the HP 125 is its interactive ability to dynamically perform as a port to the mainframe, a stand-alone personal computer, or a synthesis of the two functions. Stressing the dual nature of mainframe access for data interchange with local analysis, the HP 125 features utility programs which greatly simplify the user interface and lessen the need for sophistication in performing complex or powerful analysis of mainframe data.

As an example, take the purchasing department in a large organization. One of the areas with the greatest potential for cost minimization is the timely and careful control of inventory. Suppose that this organization does basic manufacturing of a wide line of products with many subcomponents and consequently has fifty buyers interacting with a thousand vendors regarding tens of thousands of purchased parts.

Due to the common and large data base needed to track the tens of thousands of parts, the HP 3000 presents a good choice for a central mainframe, probably also functioning for other purposes within the organization. By utilizing the HP 125 as a personal tool for each of the fifty buyers, an extremely powerful controlling application can be quickly written for use by each of the buyers.

Organizing the overall data base using the HP 3000 and IMAGE, the HP 125 can be used serve as the user interface into the larger database for each user. Data is taken from the mainframe into each of the fifty buyers personal computers.

The data resides locally and is manipulated by each buyer for programmed action items such as overdue shipments, low inventory items, high inventory items, changes in scheduling affecting inventory needs, etc. Purchasing management can control and standardize the means of analysis of each buyers proficiency through a common local program. Each buyer using his own data base can generate reports with a common format with all buyers reports. Using the HP 125 graphical package to generate bar or pie charts, the performance indicators can be directly analyzed and evaluated.

In this example, the HP 125 served as the individuals port to the HP 3000 data base, it performed local analysis of data, reduced datacomm overhead and expense, and allowed local generation of reports and graphical analysis.

To summarize, it is believed that the manner in which computers are used by organizations to enter, display and analyze data is evolving towards a new distributed network of processing units. The change on the scene is due to the technological ability to produce processing units that are inexpensive, reliable and capable. The ability to place a personal computer in the hands of an individual has shown to be not only cost effective, but by being personal has involved individuals not previously utilizing computing power directly. While personal computers have these benefits, they have not fully utilized the greater advantage of being part of the entire organizational data processing network within most organizations.

The HP 125 used with the HP 3000 shows the first step in the evolution of data processing. This evolution will bring computer usage into the hands of increasingly greater amounts of individuals within organizations. Data processing will become more convenient and cost effective.

O N L I N E

D A T A B A S E

START TO FINISH

Robert B. Garvey

Robert L. Womack IV

Witan Inc.
Kansas City, Missouri

Presentation Outline

i Introduction, wnat will be covered.

A The Foundations:

- 1 Goals; A System Language and Methodology
- 2 System Principles
  - A Elements
    - 1 Components
    - 2 Relationships
  - B Use in System Phases
    - 1 Analysis 2 File Design 3 General Design
- 3 Information System Architecture
  - A General System Architecture
    - 1 Detailing 2 Development 3 Implementation
  - B Use of Image and View
- 4 Interactivity and Control
  - A Menu Programs
  - B Control Tables
  - C Data Area Control
  - D Quiet Callability

B Dynamically Callable Programs:

- 1 SL's USL's

1. Introduction

Foundation

Bob Garvey will first lay the a foundation for the understanding of an approach used to understand, design, and implement interactive systems.

A system language, Goals, will be introduced to render systems and components.

A general set of principles will be presented incorporating the components and structures inherent in a structured system. The use of these components in the system life cycle and as a documentation system will evolve.

A general system architecture will be presented and an approach to interactivity will be discussed.

Callables

Bob Womack will present the detailed use of callable programs in the 3000 environment.

G O A L S

A System Language

Goals was designed to meet the following criteria:

* Provide good documentation
* Ease maintainence
* Expedite development
* Provide users early understanding of System functions and restraints
* Improve project management and reporting
* Reduce resources required for documentation
* Optimize System performance

Many of the above criteria can be achieved through reasonable structuring of the system . But many of the structuring techniques that are now popular are simply more trouble than they are worth. Yourdon, Jackson and certainly IBM's HIPO involve more work in their maintainence than rewards merit. Warnier comes closest to being worthwhile but cannot be reasonably maintained in machine sensible form.

Goals will be described as a methodology only because it seems to accomplish all the criteria of the popular "Methodologies", and much more. We do not feel that any of the methodologies should be considered ends in themselves and more sacred than the system at hand. Once the principles are learned and applied the implications should be obvious and the apparent need for a methodology forgotten.

Documentation

1 General Statement
2 Goals, Structural Notation

1 General Statement

The purpose of documentation is to assist in the maintainence and operation of a system. To those ends software documentation must be flexible, easily modifiable, current and easy to read. Witan has developed a system of documentation called Goals which uses simple text files associated through control numbers to meet the criteria listed above. The following sections (2 and 3) describe the general features of the structural notation used in Goals and the General system structure used in system projects.

Goals is used throughout the life of a project. It is used to:

1 To state requirements
2 Render flow and components in the analysis phase
3 To develop, test and render a general design
4 As a pseudo code or structured english for detail design
5 As a high level programming language
6 As a project network descriptor.

2 Goals, Structural Notation

Formal structuring permit three primitive operations:
Sequence, Repetition and Alternation. Structural Notation was developed to meet the criteria of formal systems in a generalized way and was guided by the assumption that systems must be rendered in a machine sensible form. Goals relies upon text sequences and key words as it's basis. Structural Notation is the basis of the syntax of Goals.

Following are the representations of the primitive structures using flowcharts and Goals. The word process is used to represent a step, a process or an item depending on the use of the notation at the time.

Goals Primitive Structures

S E Q U E N C E

```
FLOW              ---------
                <   BEGIN   >
                  ---------
                      !
                !-------------!
                ! PROCESS 1   !
                !-------------!
                      !
                ---------------
                ! PROCESS 2   !
                ---------------
                      !
                ---------------
                ! PPROCESS 3  !
                ---------------
                      !
                  ----------
                <    END     >
                  ----------


GOALS         1  PROCESS 1
              2  PROCESS 2
              3  PROCESS 3
```

A L T E R N A T I O N

FLOW

```
                ----------
              <   BEGIN    >
                ----------
                     !
                     *
                   *   *
                 *       *                 !-------------!
               *  IF X     * --- true----->!  PROCESS 1  !---!
                 *       *                 !-------------!   !
                   *   *                                     !
                     *                                       !
                     !                                       !
                   false                                     !
                     !                                       !
                     *                                       !
                   *   *                                     !
                 *       *                 !-------------!   !
               *  IF Y     * ---true------>!  PROCESS 2  !---!
                 *       *                 !-------------!   !
                   *   *                                     !
                     *                                       !
                     !                                       !
                   false                                     !
                     !                                       !
                     *                                       !
                   *   *                                     !
                 *       *                 !-------------!   !
               *  IF Z     *---true------->!  PROCESS 3  !---!
                 *       *                 !-------------!   !
                   *   *                                     !
                     *                                       !
                     !                                       !
                   false                                     !
                     !                                       !
                     !<-------------------------------------<-
                     !
                 ---------
               <   end    >
                 ---------
```

```
GOALS       IF X IS TRUE
               PROCESS 1
            IF Y IS TRUE
               PROCESS 2
            IF Z IS TRUE
               PROCESS 3
```

R E P E T I T I O N

```
        FLOW
                      ----------
                     <   BEGIN    >
                      ----------
                           !
                           *
                         *   *
                       *       *
          <-false---<*   IF Y    *----true----
          !            *       *             !
          !              *   *               !
          !                *                 !
         ------            !           !---------------!
        < END  >           !<----------!   PROCESS 1   !
         ------                        !---------------!
```

```
        GOALS       REPEAT UNTIL Y IS FALSE
                       PROCESS 1
                          PROCESS 1A
                          PROCESS 1B
                          PROCESS 1C
                    !
```

The exclamation point is used to signify control in the REPEAT loop. If the condition is met the control passes to the statement following the (!) on the same level. If the condition is met the control passes to the first statement following the condition.

Processes 1A through 1C were added to show a simple subsequence.

DATA STRUCTURING

Goals is also used to represent data structure. As with control structure there are three general structures which can be represented.

Data items listed line after line represent sequence:

```
1  item-1
2  item-2
3  item-3
```

Subsequences are represented as sequences on a level below the item of which they are are a part.

```
1  item-1
1A    item-1A
1B    item-1B
1C    item-1C
2  item-2
3  item-3
```

LEVELS: are represented graphically with the use of indentation. The first character in a line is considered to begin an "A" level subsequent levels are indented an additional three spaces each.

Successively lower levels (higher value characters and more deeply indented) represent subordinate processes. As will be seen in the general system structure the highest most levels are controled by increments of time; years, quarters, months, days, etc. while lower levels are controlled by events or conditions.

CONTROL NUMBERS: The control numbers used in Goals are developed by alternating the use of numbers and letters to represent sucessively lower levels within the system. The system is similiar to English outlining except that only capital letters and numeric characters are used. For a given statement there is nothing to indicate its position in the hierarchy unless the entire control number is dipicted or the starting control number on the page is given. When Goals statements are machine stored the entire control number is either stored or is assumed.

Repetition in data structuring can be represented by "(S)" at the end of the item name which is repeated, this can take the form of an expression (i.e. (0>s<15)).

```
1 item-1(S)
1A   item-1
```

Example: a file of accounts

```
        Account File
1 Account(s)
1A   Account
1B   Account number
1C   Name
1D   Address(s)
1D1     Address type (h=home,w=work)
1D2     Street number
1D3     Direction
1D4     Street name
1D5     Affix
1E   Amount due
1F   Order(s)
1F1     Order number
1F2     Item(s)
1F3        Item
```

ALTERNATION:
Alternation is represented with the IF control word or with the notation (1,0).

```
2A   IF segment descriptive code = 1
2A1     material
2B   IF segment descriptive code = 2
2B1     supply
```

This convention is seldom used because the REPEAT handles most situations for the case of data structuring.

The other type of alternation is within a string of data items where the item can either exist or not exist. Another way of rep- resentin a non-required item.

```
1 item-1
2 item-2
3 item-3(1,0)
```

This says that items 1 and 2 must exits or are required and item 3 is optional.

Discussion:

The highest level of repetition within a data structure is assumed to be the key to the file or at least the major sort sequence. If additional keys are required they can be represented with the word KEY (i.e. item-3 (KEY)) or an additional data structure can be presented to represent the structure represented when the KEY is used.

Goals can be used to represent logical structures as well as the physical implementations. It is important that the required logical views of data be derrived and documented before any phy- sical structures be planned. A recommended goal in system design is to have a one to one relationship between the physical and the logical structures of the system. The coding complexity is reduced appreciably as well as the maintainence activity. An additional byproduct is the ability to use Query or other general inquiry languages in a more straight forward fashion.

2 Principles

An Information system is distinguished from operating systems, command interperters, compilers and the like. An Information System is that set of communications, operations, files and outputs associated with a single conceptual "file".

I am not talking about a single program. Historically I am talking more about an application area.

A. Elements

A1 Components

First an analogy: All purely mecanical devices are made up of elemental components; the incline plane, the wheel and axle, the lever and the chamber. The physics of these basic components and the materials from which they are constructed define the limits of their application. You may be saying, that list does not sound correct or "what about the screw". In listing elemental components certain definitions are inherent. I define the screw as a "rolled incline plane".

For information systems I assert that the list is: Communications, files, operations and outputs. The limits for such systems are defined by the ordering of the elements using the primitive structures (sequence, alternation and repetition).

As a note; to date the list of elemental components may have been input, process and output without reguard to to structure. This is more elemental considering all computer processes but is unbounded. This makes a general system design technique very difficult. Adding hierarchy to the above does not enhance these primitives to any great extent.

A2 Relationships

With these boundries and definitions in hand, lets look at the relationships that develop.

There is generally a one to one relationship between file structure and operations structure, between communications structure and operations structure, between output structure and operations structure. In other words the operations or control structure mimics the other components of the system and each componet is related to the other in structure. The structure begin with the file structure.

Example; if you have a file of accounts and you want to report them; the report program will have to be structured exactly the same as the file or database to report all the data in the file. Most often there is a one to one relationship between files and outputs. In the report example the report structure could be expected to look exactly like the file. If the report is to look different than the file there would be in intervening operation usually a sort or selection to convert an intermediate output to the final output.

The same is true of communications which on the data processing level are the transmissions to the uses, the screens and the messages. The structure of a communication is generally the same as the operation structure which is the same as the data structure and thus the communication structure is the same as the data structure. This substantiates the theory that systems can be completly described knowing only the data structure. True but limited. Knowing the structure of any part should in theory give you the whole.

If everything describable about a system can be described in simple structures (and thus in Goals) and the components of a system include only communications, files, operations, and outputs and Goals can be used in all system phases then we have a framework for a general system covering conception through maintainence.

Lets look at any application. Traditionally you would begin with a requirements statement and do an analysis of the existing system. Forget flowcharts, classic narratives, and other charting techniques. Think of progressively decomposing the system using simple english outlining starting with the functions. Functions fit into the operations structure discussed. You will note that as you get down a level or two you will encounter repetitive tasks dependant on conditions, add REPEAT and IF to your outlines and keep describing. Remember that users can understand outlines and repetition and alternation are not difficult to understand.

Operations will include existing machine processes, manual proceedures, paper flows, sorting processes ect. As you are going through the operations keep a list of the files that are mentioned and note the file keys (and sorts) and any advantages or requests for multiple keys.

List any outputs or reports prepared by the organization or required in the future.

Communications will be minimal at this stage but note any memos that may go from one section to another of a "file" of notes used as crossreference or duplicate of any more perminent file.

Your documentation is now shaping up; your notebook and I assume that the whole world has change to 8 1/2 by 11, should be divided into communications, file, operations and outputs.

The starting point for design is the detailing of the files in your file list. You will want to reduce the files as much as possible to a single file. By way of naming conventions the "file" should have the same name as the system at hand.

You will notice that many of the manual files are really communications in that they are "views" of the file that are required in a particular subfunction.

The design of the conceptual file must be validated against the required operations. I am going to leave this hanging for a moment to discuss a General System Structure.

3 General System Structure

A General System Structure is presented on the following page in Goals.

This structure is not applicable in all systems but is used as a pattern for system discription, design and understanding.

The key elements of design of this structure are:

1. File unity; a system with this structure has only one conceptual file. It may have any number of datasets of or physical files but they must be formalized into one.
2. Journalizing or logging; all changes made to data items can be (and normally should be) logged.
3. Last action dating; incorporated as part of logging, permits an offline log.

One detailed implication of this is need to have a date stamp in each detail set and a master date stamp in the master file.

General System Architecture

```
Begin system
REPEAT until EDSystem
  REPEAT until EOYear
     REPEAT until EOQuarter
        REPEAT until EOMonth
           REPEAT until EODay
              REPEAT for each user
                 Begin online
                 identify operator and security
                 Open system file
                 Open current files
                 REPEAT for each Communication
                    IF control transfer
                       transfer control
                    IF batch request
                       initiate request
                    IF update , add or delete
                       Begin
                       Memo to LOG
                       LOCK
                       Update ,add or delete
                       UNLOCK
                       End
                    IF inquiry
                       perform communication operation
                    .
                 !
              !
           !
           End Online
           Begin daily batch
           Perform daily batch processing
           Run LOG analysis
           If end of week
              Perform Monthly Processing
           .
           ROLL FILES
        !
     !
     perform Monthly processing
     Perform Quarterly processing
  !
  Perform end of year processing
!
Close system
End
```

A GENERAL DESIGN

With this Architecture and database design complete we have the basis for the development and implementation of any application.

Step 1 is inquiry into our file; if there is only one search criteria then we calculate into to file and return the master data or a summary. Once positioned in a master we can chain through our detail sets or follow appropriate programatic paths.

The master screen (a communication) should provide inquiry, update, and addition ability.

Each detail set should have a screen providing the same update add and inquiry ability. Our screens will be one for one with the detail sets. Think of a detail set as having a buffer that will correspond to communication (VIEW) buffer. Moving date within one program is facilitated with this concept.

The list of detail sets becomes a list of programs which must be written to handle the retrieval, update, addition, deletion and editing of data for the detail set.

When this is complete you will have a functioning system; it will not function well. I have intentionally oversimplified. The office proceedures which may be in place or will evolve will dictate what combination of sets will appear on a screen but no effort was be lost in developing the barebones system according to this method. Each set (detail set) should have its own program to handle retrieval and update. When requirements demand inclusion the programs can usually be used with few changes. You can take this one step further to include a general scheme to handle multiple data sets on one screen.

The question then becomes; "How do I tie this all together?".

4 Interactivity and Control

Lets say that we have written a system composed of a series of programs that correspond to our data sets. The way in which we permit interactivity is through a control program called MENU.

4A Menus

A master data set will exist at the top of the conceptual file and the primary search path will be the file key. Other search paths will be provided through subsystems such as "Name Family" or through automatic masters. For all detail sets associated to the master there will be a program to handle that data set. Your analysis will dictate all the processes that the operator may wish to perform. As other requirements develop associating more than one data set the code can be combined and new screens developed.

The menu control program provides transfer of control. It can do this either "quietly" of "loud". Loud is the obvious implementation; the operator choses a data set from a menu screen, the control is transfered via a "call" to a dynamic subprogram the data set is accessed updated, ect. and control returns to the controlling menu. But let us give the operator the ability to "tell" the system where he wants to go next. If he does a common area flag can be set to say don't display the menu simply transfer control to some other subprogram. We call are common area for data SYSBLK and out flag(s) Q1, Q2, ect. (you are not limited to one level of menu).

A menu structure may look like this:

```
        MAIN MENU
     REPEAT UNTIL PARENT OR END OF SYSTEM
       IF LOUD
          GET MAIN MENU SCREEN
          SEND (SHOW) SCREEN
          REPEAT UNTIL EDITS PASS
            EDIT FIELD
            IF EDITS FAIL
               SEND SCREEN
            .
          SET MODE TO QUIET
       IF QUIET
          IF NEXTPROCEEDURE=A
             CALL A
          IF NEXTPROCEEDURE=B
             CALL B
              ...
          IF NEXTPROCEEDURE =N
             CALL N
          ELSE
             CALL CONTROL'NUMBER'TABLE
          .
       !
```

Through this technique those programs which are not being used are not using memory resources. The CONTROL NUMBER TABLE refers to implementations which have levels of menus. If the control reference is not handled at that menu level control is appropriatly passed to the proper level where a control program can handle it.

The quiet "CALL" technique can be used for any of the data set programs discussed by putting the quiet call structure "around" the program and requiring the passing of appropriate data into or from the communication buffer. Bob Womack will describe this technique in the "NAME FAMILY" discussion.

SL's and USL's

S L ' s

o Modules, Entry points, Programs referenced in require CST entries if they are not allready referenced in a running program.

o Code is sharable by all programs. The PUB.SYS SL is avalable to all programs. Account and group SL's are available to programs being run out of that Account.

o You need exclusive access to the SL to make an entry in it.

o When SL entries are made you do not need to prepare the SL. It is available after you have exited the segmenter.

U S L ' s

o Programs compiled into a USL must be prepared before they are runnable.

o Many programs may be compiled into the same USL. When a program is run the system will look to the USL for resolution of called programs, it then looks to the PUB.SYS SL unless a library is specified in the RUN. (RUN prog;LIB=G)

o All USL resolved entries create XCST entries except the outer block.

CST's and XCST's

o There are 192 CST entries available to user processes

o There are 1028 XCST entries available to user processes.

COMPILE INTO A USL

```
!JOB JOBNAME,username/ serpass.accountname/accountpass;OUTCLASS=,1
!COBOL progname,$NEWPAS ,$NULL
!SEGMENTER
USL $OLDPASS
NEWSEG progname,progname'
PURGERBM SEGMENT,progna e'
USL yourusl
PURGERBM SEGMENT,progna e
AUXUSL $OLDPASS
COPY SEGMENT,progname
EXIT
!TELL user.acct; yourprog ---> yourusl
!EOJ
```

P R E P   O F   U S L

```
!JOB DyourUSL,user/userpass.account/accountpass;PRI=ES;OUTCLASS=,1
!PURGE yourrun
!CONTINUE
!BUILD yourrun;DISC=2500,1,1;CODE=PROG
!SEGMENTER
USL yourusl
PREPARE yourrun;MAXDATA=16000;CAP=MR,DS
EXIT
!TELL user.acct;  yourrun ---> yourrun
!EOJ
```

C A L L A B L E S   I N T O   S L ' s

```
!JOB D!SL,user/userpass.account/accountpass;OUTCLASS=,1
!COBOL yourprog,$OLDPASS,$NULL
!SEGMENTER
AUXUSL $OLDPASS
SL SL
ADDSL yourprog
EXIT
!TELL user.acct;  yourrun ---> yourrun
!EOJ
```

M E N U

```
REPEAT until parent or  nd of system
   IF load
      get menu screen
      show screen
      REPEAT until edits pass
         edit fields
         IF edit fail
             send screen
      !  .
      set mode to quiet
   .
   IF quiet
      IF nextprocedure = "O"
          CALL "O" USING ., ., .
      IF nextprocedure = "I"
          CALL "I" USING ., ., .
           .
           .
           .
      IF nextprocedure = "n"
          CALL "n" using ., ., .
      ELSE
          CALL "CONTROLNUMBERTABLE" using nextprocedure
      .
!  .
```

FACULTY PERCEPTIONS OF COMPUTING FACILITIES
(Based on a study of UTC Faculty in 1981)

Dr. Lloyd D. Davis

Purpose of Study

The University of Tennessee at Chattanooga is very typical of many of the public institutions operating today. It had a history of minimal computing until 1974 when major computer acquisitions were made. Further significant upgradings have been made since then with another major acquisition being consummated in 1978. UTC, therefore, is a model of interest to many concerned with the effects of these computing resources on faculty attitudes towards the curriculum, general educational issues, relevance of current computer facilities, and the entire computing mileau. Hence, other institutions may utilize this model and perhaps the data and analyses presented as a barometer for assessing the impact of instructional/research computing needs, resources, and values at their own institutions.

Background

UTC is a Masters granting institution located in urban Chattanooga, Tennessee, and is a primary campus of the University of Tennessee. With approximately 8,000 students and 250 plus full-time faculty, UTC is primarily an undergraduate institution. Its role within the State of Tennessee is to provide quality education at the baccalaureate level to a largely commuting student body. Students have degree options in the many areas of the Arts & Sciences and the professional areas of Engineering, Computer Science, Nursing, Business Administration, Human Services and Education. Somewhat unusual for a state institution is its private foundation with a $10,000,000 plus endowment which is used to enrich academic areas at UTC.

Included in this enrichment was the acquisition of an HP2000 in 1975 strictly for the use of faculty and students. This timesharing system replaced an IBM 360/30 with RJE capability to a large 360/65 system a hundred miles away. In 1978, UTC purchased a second academic computer system, the HP3000 Series II with funds from a capital development campaign. Currently, the HP2000 and HP3000 are connected to over 80 terminals and provide 63 ports of instruction and research to students and faculty; UTC continues to provide RJE activity to two large IBM 3031's at a remote location.

Methodology

In the spring of 1981, all faculty at the University of Tennessee at Chattanooga were requested to complete a questionnaire concerning their perceptions regarding instructional and research computing. The first objective of this survey was to identify perceptions concerning computing both at a general level for institutions and for majors at large. The second objective was to discover specifics about UTC and individual departmental needs. Previously, two other surveys were conducted dealing largely with these same issues. In 1974, as part of a report to the University of Chattanooga Foundation, Drs. Carney, Davis, Smullen, and Ward, all of UTC, reported similar analyses. In 1978 essentially the same study was replicated by Dr. Smullen.

Although the data reported will refer wherever possible to all three surveys (1974, 1978, 1981), attention will focus on the most current, which is 1981. For purposes of reporting, some departments have been coalesced into larger areas. These four larger areas are the Arts & Sciences; Business Administration and Economics; Engineering and Computer Science; and

Education, Nursing, Human Services, and miscellaneous. Although the data could be tabulated in many ways, this report will largely work with the positive responses "Strongly Agree" and "Mostly Agree", for the major areas outlined.

Responses to Survey

Fifty-seven percent of the teaching faculty responded to the 1981 survey as compared to 54 percent and 69 percent in 1978 and 1974, respectively. Table I shows the responses by area, and it is seen that the dominant areas are Humanities, Physical Science and Mathematics and Business Administration with 22, 20 and 12 percent of the total, respectively. No department or area was vastly under-represented or over-represented in the survey. The continuation of Table I breaks these larger areas into their major components for purposes of detailing the responses.

Computer Usage

Computer usage of the respondents indicates that the number of faculty who have used the computer moderately or extensively in the past has increased from 49% in 1974 and 48% in 1978 to 52% in 1981 (see Table II). The Humanities, Education, and Physical Science and Mathematics declined from 21%, 37% and 67% in 1978 to 19%, 32%, and 59%, respectively in 1981. However, Behavioral Science and Engineering grew from 44% and 55% in 1978 to 64% and 100% respectively in 1981, indicating heavy computer experience for those areas. The area which declined, surprisingly, was Business Administration and Economics, as it declined from 75% to 47% in the same period.

TABLE I

COMPOSITION OF THE GROUP OF RESPONDENTS

	1981	1978	1974
Total Responses	144	125	133
Total Faculty	253	231	194
Percentage Response	57%	54%	69%

By College/School/Division		Number of Responses	% of all Responses	Number of Faculty	Percentage Response
Arts and Sciences	1981	71	49%	142	50%
	1978	65	52%	130	50%
	1974	76	57%	127	60%
Engineering	1981	11	8%	21	53%
	1978	11	9%	11	100%
	1974	7	5%	8	88%
Business*	1981	17	12%	32	53%
Education*	1981	14	10%	29	48%
Nursing*	1981	7	5%	12	58%
Human Services*	1981	14	10%	17	82%

*Data not available for earlier years for these areas.

COMPOSITION OF THE GROUP OF RESPONDENTS IN 1981

By Area:	Frequency	Percentage of All Responses
Humanities	31	21.5
Physical Science & Mathematics	29	20.1
Behavioral Science	11	7.6
Engineering	7	4.9
Computer	4	2.8
Business	17	11.8
Education	14	9.7
Nursing	7	4.9
Human Services	14	9.7
Misc, Unknown	10	6.9
Total	144	100.0

TABLE II

UTILIZATION OF THE COMPUTER BY THE RESPONDENTS

		% Positive Responses by Area				% Responses		
		Arts & Sciences	School of Business	School of Engineering	All Others	Extensive	Moderate	Negligible or No Response
I have used the computer extensively in the past.	1981	42	47	100	31	17	35	48
	1978	37	**	55	**	18	30	53
	1974	38	**	100	**	11	38	51
I plan to use the computer extensively in my future classroom activities.	1981	45	70	91	47	17	35	48
	1978	54	**	64	**	18	50	31
	1974	74	**	100	**	26	54	20
I do not plan to use the computer in my future research.*	1981	58	82	91	70	51	17	32
	1978	49	**	55	**	34	26	41

*Disagreement with this statement has been treated as a positive response. No such question was asked in 1974.

**Not available.

TABLE IIA

PERCENTAGE UTILIZATION OF THE COMPUTER BY 1981 RESPONDENTS
(EXTENSIVE OR MODERATE POSITIVE RESPONSES ONLY)

By Area:	Past Usage Responses	Future Classroom Responses	Future Research Responses
Business, Economics	47	71	71
Physical Sciences, Mathematics	59	55	55
Engineering, CPSC	100	91	91
Behavioral Sciences	64	64	64
Education, Et al	32	46	46
Humanities	19	29	29

This can be explained, in great part, by the fact that the 1978 survey included Computer Science with Business while the 1981 survey included Computer Science with Engineering.

The faculty projections of future computer activities in their respective classrooms indicated that both the School of Business and the School of Engineering will in the future make much more extensive use of the computer in the classroom. The former will grow from 59% to 70% and the latter from 64% to 91%. When asked about the use of the computer in their future research all areas reported positive responses. Over 68% of the faculty indicated they expected to make extensive or moderate use of the computer in future research as opposed to 60% in 1978. Also noted was the large decrease in the neutral or no response category. The data generally supports the fact that UTC is acquiring either through recruitment or "in-house" training a computer literate faculty who generally are using the computer more and based on these experiences, will utilize it more in the future in both their classroom and research activities. Areas such as Engineering, Computer Science, Behavioral Sciences, and Business are leading this growth.

General Opinions

Faculty at UTC were asked to respond also to general questions concerning their perceptions regarding the computer and its effect upon society and education. Table III shows the data related to these questions.

There has been a small decrease in the positive responses to the statement that computers will improve education. Although in 1981 this proposition was agreed with either strongly or mostly by 86% of the faculty, in

TABLE III

GENERAL OPINIONS

		Percentage Positive Responses				All Responses				
		Arts & Sciences	Busi- ness	Engi- neering	Others	Agree Strongly	Agree Mostly	Neutral NA	Disagree Mostly	Disagree Strongly
Computers will improve university education.	1981	85	88	91	90	51	35	9	3	1
	1978	86	96	82	-	56	34	8	2	0
	1974	59	54	100	-	35	25	41*	1	1
Computers create an impersonal society.**	1981	55	59	64	44	20	33	28*	14	4
	1978	52	55	46	-	16	37	24*	18	5
	1974	43	64	57	-	22	28	25*	15	11
Computers are beyond the understanding of typical university undergraduates.**	1981	90	82	100	96	54	38	1	6	2
	1978	79	86	82	-	38	43	6	7	5
	1974	78	92	100	-	53	31	11	4	1
The computer is as important a resource as the library.	1981	47	88	82	60	29	29	8	19	15
	1978	46	80	64	-	30	31	7	24	10
	1974	28	58	71	-	20	22	15	29	15

*Large Neutral, No answer, or No Opinion response (20% or over)

**Disagreement with this statement has been treated as a positive response.

1978 this issue received 90% positive responses. On the issue of computers creating an impersonal society, 53% of the faculty disagreed. On several such questions disagreement was taken as a positive response. Clearly the faculty feels computers are well within the capability of understanding of typical undergraduates, as 92% responded positively in 1981, up from 81% in 1978. Response to whether the computer is as important a resource as is the library is much less positive. The data analysis indicates 58% of the faculty affirm this in 1981 as opposed to 61% in 1978. However, both the Engineering and Business areas grew from 65% and 80% in 1978 to 82% in 1981 respectively. Not surprisingly, Arts and Sciences disagreed with the library computer issue and affirmed this issue at only a 47% rate.

Computers in the Curriculum

This section deals with faculty perceptions of the use of the computer in specific subject matter areas. The responses to the issue of the necessity for computers in instruction in Natural Sciences and other sciences, the desirability of accessing computers in the Behavioral Sciences, the knowledge of and practice on computers for Business Administration, and the awareness of computing for students in the Humanities have not changed appreciably over the period 1978 to 1981 (see Table IV). That computers are necessary for instruction in areas such as Natural Science, Engineering and Mathematics is agreed with by 89% of the faculty with only small variance among the areas. That instruction for students in the Behavioral Sciences requires access to problem solving via computers is agreed with by 74% of the faculty and is as high as 91% in the Engineering area. Eighty-nine

TABLE IV

COMPUTERS IN THE CURRICULUM

		Percentage Positive Responses				All Responses				
						Agree		Neutral	Disagree	
		Arts & Sciences	Business	Engineering	Others	Strongly	Mostly	NA	Mostly	Strongly
Computers are necessary for today's instruction in areas such as natural science, engineering, and mathematics.	1981	86	88	91	93	59	30	8	3	0
	1978	89	88	91	-	57	32	6	5	0
	1974	88	84	100	-	56	30	9	1	4
Instruction for students in the behavioral sciences desirably requires access to computers for problem solving.	1981	85	77	91	80	35	39	21*	3	0
	1978	68	84	46	-	43	29	25*	3	0
	1974	79	74	100	-	38	41	17	5	0
Business Administration students must have knowledge of and practice in the uses of the computer in business applications.	1981	86	100	91	91	60	29	9	1	0
	1978	85	94	91	-	58	30	10	1	1
	1974	80	92	100	-	54	32	12	1	1
Students from the humanities should have an awareness of the influence of computers to today's society.	1981	89	65	91	93	44	44	11	1	0
	1978	88	88	91	-	45	43	10	1	1
	1974	82	92	100	-	40	47	10	3	0
The curriculum of UTC should have more breadth and depth of computer experience available to the students.	1981	44	77	64	58	16	38	38*	5	1
	1978	63	63	55	-	16	46	30*	6	2
	1974	66	78	86	-	29	41	24*	5	1

TABLE IV A

COMPUTERS IN THE CURRICULUM**

Responses of the various areas concerning the need for computers in their own curriculum from the 1978 and 1981 surveys.

		Agree		Neutral	Disagree	
		Strongly	Mostly	NA	Mostly	Strongly
Engineering & CPSC	1981	36	27	27	9	0
	1978	-	-	-	-	-
Science & Mathematics	1981	28	14	45	7	7
	1978	68	24	5	3	0
Behavioral Sciences	1981	36	36	18	9	0
	1978	44	44	11	0	0
Business, Economics	1981	53	29	12	0	0
	1978	84	11	5	0	0
Humanities	1981	26	13	32*	13	16
	1978	38	45	14	0	3

*Large Neutral, No Answer or No Opinion (20% or over)

**In 1978 CPSC was tabulated with Business and Engineering with Science.

percent of the faculty believe that students in Business Administration must have knowledge of, and practice in, computerized business applications. A robust 88% of the faculty believe that students in the Humanities should have awareness of computer influence on society. On the issue of whether the UTC curriculum should have more breadth and depth of computer experiences for students, affirmative responses declined from 62% in 1978 to 54% in 1981. Although the areas of Business and Engineering showed higher 1981 figures than the corresponding areas in 1978, Arts and Science faculty affirmed this by only 44% in 1981, a 19% decrease over 1978. This implies a general satisfaction with computing for this college rather than a decrease in interest directed towards computing. Correspondingly, in 1981 as compared to 1978, even larger numbers of faculty from the areas of Business and Engineering believed there should be more computing. This obviously indicates a major need for computing in these areas and a growth of computing.

When faculty were asked about the need for academic computing in their own departments, a high of 82% in Business felt their department should utilize the computer more in the curriculum than it presently does. The lower affirmative responses of 42% and 39% from the Sciences and Humanities respectively indicate a general satisfaction with what their areas are now doing, rather than dissatisfaction with the computer. This indicates a trend towards maturation in terms of computer utilization in these areas.

Computing Facilities

Table V presents data on faculty perceptions of UTC computer facilities. Many faculty in 1981, especially in the Engineering and Computer Science area,

TABLE V

PERCEPTIONS OF COMPUTING FACILITIES

		Percentage Positive Responses				All Responses				
		Arts & Sciences	Business	Engineering	Others	Agree Strongly	Agree Mostly	Neutral NA	Disagree Mostly	Disagree Strongly
UTC maintains and provides inadequate computer support for academic instruction and research.**	1981	42	53	9	49	15	28	35*	12	5
	1978	48	-	18	-	17	29	43*	8	3
	1974	32	-	100	-	16	23	48*	11	2
The present computer system and staff is conducive to faculty use for academic projects.	1981	56	71	36	60	19	40	25	10	5
	1978	54	-	55	-	17	38	34*	9	2
	1974	37	-	86	-	1	15	44*	29	10
UTC should provide short training courses for faculty on the computer and the available packages.	1981	85	82	82	96	44	44	9	2	1
	1978	88	-	73	-	47	39	13	0	1
	1974	93	-	71	-	54	38	8	0	1
The computer should be made more available to the faculty and students. +	1981	45	76	73	53	19	35	37	8	1
	1978	59	-	55	-	20	38	33*	9	1
The cluster concept is the most desirable means for providing terminals for student instruction. +	1981	44	47	82	44	15	32	46	6	1
The cluster concept is the most desirable means for providing terminals for faculty research.+	1981	13	12	36	27	4	15	52	17	12
Text processing should be provided by UTC to its students and faculty even if this requires major new resources. +	1981	44	53	73	53	22	26	35	8	8

*Large Neutral, No Answer, or No Opinion response (20% or over).
**Disagreement with this statement has been treated as a positive response.
+No such question was asked in 1974.

perceived computing facilities to be inadequate. Positive responses in this technical area have declined from 100% in 1974 to 18% in 1978 and 9% in 1981. Otherwise the overall faculty responses have been positive with 39% in 1974, 46% in 1978, and 43% in 1981 affirming UTC provides adequate instructional and research equipment.

On the issue that the present system and staff are conducive to faculty use, the positive responses have grown from 16% in 1974 to 55% in 1978 to 59% in 1981 . Again, as with the previous question, Engineering and Computer Science with only 36% positive responses perceive it less positively than do the other departments. The general dissatisfaction noted from Engineering and Computer Science is believed to be due to the lack of major computer systems that are local and the perceptions associated with computing at a remote site.

When questioned regarding short training courses for the faculty, 88% responded affirmatively with little variation over the respective areas. This service is well received by the general faculty.

The question "should the computer be made more available to the faculty and students" was answered "yes" by 54% of the faculty in 1981, down from 58% in 1978. Very high rates of 76% and 73% were seen in the areas of Business and Engineering, indicating a perception of greater need in these areas than in the university as a whole.

UTC utilizes the cluster approach to provide terminals for the students and faculty. Approving this cluster concept for student instruction was 47% of the faculty with a group of 46% being neutral. Engineering endorsed this at a 82% rate. On the issue of desirability of computer clusters for faculty research, only 19% endorsed this concept with the highs ranging from 36% in Engineering to a low of 12% in Business.

Text processing is being introduced gradually in UTC on the HP3000 through the package EDIT2. About 48% of the faculty believe this facility should be offered to faculty and students even if major new resources are required. Areas ranged from a high of 73% positive support in Engineering to 44% in Arts and Sciences. Certainly the UTC faculty are strongly behind the concept of text processing for classroom materials, reports, manuscripts and resumes.

In general, facilities are recognized as better than adequate by most areas of the university with the exception of the School of Engineering. General support for computing facilities and staff, instructional short courses, and text processing was demonstrated. Faculty research is seemingly not served well by the cluster concept but, in all, the computer systems are generally conducive to faculty use.

Emphasis and Rewards

When responding to the statement "your department should utilize computers more than it does now", 61% responded affirmatively in 1981 as opposed to 60% in 1978 (see Table VI). Interestingly, those strongly agreeing increased from 26% in 1978 to 34% in 1981. The area perceiving this need the most was Business with 82% positive responses and the least was 45% in the Arts and Sciences.

TABLE VI

EMPHASIS AND REWARDS

		Percentage Positive Responses				All Responses				
		Arts & Sciences	Business	Engineering	Others	Agree Strongly	Agree Mostly	Neutral NA	Disagree Mostly	Disagree Strongly
Your department should utilize computers more than it does now.	1981	45	82	64	78	34	27	26*	7	6
	1978	54	-	46	-	26	34	26*	13	1
	1974	62	-	86	-	30	36	21*	9	4
UTC has a reasonable emphasis on the uses of the computer in its educational processes.++	1981	38	65	45	44	11	33	46*	9	1
	1978	60	-	64	-	22	38	34*	5	2
	1974	26	-	71	-	11	21	51*	16	1
Within its role as a primarily undergraduate institution, UTC places too much importance on computing.** +	1981	62	82	91	69	35	34	26*	4	1
	1978	69	-	73	-	28	45	18	7	2
Faculty time spent on developing computer uses for the classroom is adequately recognized as a professional activity by the administration.+	1981	18	24	18	16	5	13	42*	24	17
	1978	15	-	27	-	6	12	48*	22	12

*Large Neutral, No Answer, or No Opinion response (20% or over)
**Disagreement with this statement has been treated as a positive response.
+No such question was asked in 1974.
++This question was preceded by "When compared to other primarily undergraduate institutions of its size."

The statement "UTC has a reasonable increase emphasis on the use of the computer in its educational processes", showed overall support declining from 60% in 1978 to 44% in 1981. The area of Business supported it with 65% positive responses; at the other end was the area of the Arts and Sciences which supported it at only 38%. This statistic is clouded by a large 64% undecided or neutral set of responses. Perhaps this is an indication of need for more emphasis rather than a dissatisfaction with the current emphasis as being "too much."

The statement "within its role as a primarily undergraduate institution, UTC places too much importance on computing", is disagreed with by 91% of the Engineering area, 82% of Business, and 62% of the Arts and Sciences. This indicates further that faculty perceive that either the same or more computing is required as opposed to less computing.

With regards to faculty computing activities being adequately recognized as professional activity by the administration, only 18% feel positive about this issue. The strongest response is that of Business which reports 24% positive responses. There is a large 42% neutral response category; this indicates a general perception that instructional computing does not count towards promotion and tenure in the same manner that scholarly writing does.

In-depth Analysis

A second survey was sent to those who were judged to be heavy users or who requested it. This survey asked for specific information regarding satisfaction with the systems, their components, the staff, and their functions; it contained several open ended questions regarding future desired computer acquisitions. Since this paper is designed for general opinions, most issues in this second survey are not included here.

Thirty-eight of the heaviest faculty users completed the questionnaire. Eighty-four percent were at least satisfied with our present systems. Nearly 53% of those responding felt that the 1981 computing environment (HP3000 and HP2000) was better than the 1978 computer system (HP2000 only). UTC consultants satisfied 90% of the faculty in terms of availability and 89% in terms of helpfulness. Software satisfied 63% of the faculty and dissatisfied 26%. System reliability was viewed favorably by 67% and negatively by 24% of the faculty responding. Relevance of UTC newsletters and helpfulness of UTC manuals were affirmed by 66% and 76% of the respondents, respectively.

The generally positive responses to these issues indicate that the HP 3000 system has been viewed favorably by a majority of its heavy users. Furthermore, the staff activities in terms of consulting, documentation, and overall effectiveness are considered satisfactory or better by a large majority of users. The Office of Academic Computing, which maintains most of these functions, is, therefore, by association viewed as a very positive factor in computing in UTC. The negative responses were mostly from Engineering and Computer Science; these were largely associated with the requirements or needs associated with large scale systems, major software packages in technical areas, and specific operating systems. Hence, Hewlett Packard equipment and software received a general vote of confidence.

Summary

Wide spread approval of UTC's instructional and research computing is voiced by the faculty. Faculty perceptions about societal implications of computing, the necessity for computing in various disciplines and student capacities for learning about computing were very favorable. Nearly all areas generally report affirmative attitudes towards computer associated activities in the curriculum. Minimal back-lash, if any, was evident towards university, area or departmental emphasis on computing. Several departments, namely Engineering and Computer Science, indicated the need for more computing that is probably of the non Hewlett-Packard nature and which is of a medium scale rather than a mini scale. It seems reasonable that the HP2000 and HP3000 have heightened the desire for timesharing, even on major systems, thus negating the perceived worth of remote systems that are largely batch entry.

Support facilities, staff and training were perceived in a positive manner by a majority of the respondents. The HP3000, augmented by the HP 2000 and IBM 370/3031 RJE, has created a viable, stable computing environment for university academic computing. Coupled with this is the Office of Academic Computing Services which has visibly and significantly assisted most academic areas with their instructional and research computing.

A famous economist once stated "Nature has no force as powerful as an idea whose time has come." Academic Computing's time has come and at UTC the idea and its manifestations are growing with adolescent fervor and are approaching intelligent, capable maturity.

JOBLIB/3000

By Esa A. Harjula

ABSTRACT

The JOBLIB/3000 system is now an advanced productivity tool, combining interactive macro processing techniques with new practical ideas for computer aided batch job preparation work. JOBLIB/3000 provides new approach to batch processing in on-line environment. It makes MPE even more friendly operating system.

DISTRIBUTED 6250 BPI TAPE

by Daniel R. O'Neill
Qualex Technology, Inc.

INTRODUCTION

One of the most frequently heard buzz words of the computer industry today is that of distributed processing - making a high level of data processing available to each user, but allowing each user to attach to and access a large data base.

This paper deals with another aspect of distributed computer power - that of distributing 6250 BPI tape systems between a number of CPU's. Qualex Technology, Inc. has formally announced "SMASH" SHARED MASS ARCHIVE STORAGE HOST at this HPGSUG user meeting and this paper will describe system operation, system configurations, etc. In addition, features of 6250 BPI technology will be high-lighted.

Qualex first introduced 6250 BPI tape technology and systems to HP end users in September of 1980. Prior to this date this field proven product (first installed in 1978) was only available to OEMs. Qualex's first tape systems provided users with a state-of-the-art technology solution to the frustrating backup, archive and interchange problems being experienced by the HP user. The Qualex Group 3000 tape systems provide the following features:

* Modern tape technology design
* 125 or 75 inches per second tape speed
* 60 sec rewind time for 2400 foot reel
* Triple density 800/1600/6250 BPI or Dual density 1600/6250 BPI
* Switch selectable density
* Automatic thread/load
* Use of Easy Load cartridges

* Extensive Diagnostic capability both within the drive and controller and via loadable software diagnostics
* Plug & program compatible with Series II & III
* Employs the most field proven 125 ips, small size 6250 drive in the industry

With the March 1, 1981 announcement by HP of the dedicated channel 7976A Tape Subsystem, for the Series 30, 33 and 44, and the HPIB interface module (STARFISH) for the HP3000 Series III, the pioneering efforts by Qualex to bring this state-of-the-art in tape technology to the Hewlett Packard community has been totally endorsed. The HP product does not match the performance, economics, configuration, serviceability and technology of the Qualex product, but at least now, the user knows that the use of 6250 is now acceptable in the HP world, will have two sources to choose from and can evaluate these sources based on price/performance.

It is appropriate at this point to review the advantages of 6250 BPI tape technology to the user.

* Higher Recording Rate for Greater Data Throughput
* Reel Storage capacity increased more than threefold
* Improved Read/write reliability with multi track error correction
* Quicker access to data
* Smaller IBG (0.3 inch vs 0.6 inch @ 1600 BPI)
* Shorter rewind time

Besides the above benefits/features, the 6250 technology provides additional advantages to HP users - when compared with hardware they are currently using. These advantages are:

* Auto thread/auto load
* High speed rewind - 500 ips
* 125 ips tape speed
* Quick rewind time - less than 1 minute
* Triple density

* Switch selectable density
* Automatic Hub
* Vastly improved operator features

THE RIGORS OF 6250 TAPE DESIGN

To meet these stringent requirements required a new generation of tape equipment encompassing significant breakthroughs in tape transport and capstan design as well as development of sophisticated controllers to meet all the format encoding/decoding, error correction and status requirements of the GCR code. The controller, to do this, is basically a complex computer in itself.

6250 BPI uses a recording technique known as Group Coded Recording (GCR). The formatter/controller "codes" bytes of user data into five bytes of coded data to be recorded such that there will be no more than two zero's in succession. This provides for an efficient code for recording the data without experiencing the long strings of ones often displayed in the 800 BPI or NRZI mode of recording. This coding technique also provides for a self clocking recording system which, when coupled with the multiple ECC characters built into the code, allows two tracks in error to be corrected "on the fly" (without stopping and re-reading).

The 6250 code is extremely powerful. An ECC character is inserted after each seventh user byte. Two additional ECC characters are inserted after the data portion of the block is complete. Due to the "overhead" of group coding and ECC characters, the actual recording density is 9042 BPI - not 6250. The user data comes to the tape system at a 6250 rate, but this data is actually put on tape in coded format at 9042 BPI. This high density not only required new head and read/write circuit technology, but also new sophistication in transport design to preclude data errors under stringent tape motion dynamics.

At 6250 BPI, the interrecord gap is cut in half from 0.6 inch, used at 1600 or 800 BPI, to 0.3 inches. The actual start and stop distances however were reduced by two thirds. At 1600 BPI, the normal start or stop distances is 0.190 inches whereas at 6250 the start distance is 0.075 inches. This provides quick access to data for the user, but represented stiff requirements for the tape drive designers.

The Qualex product meets all the requirements of the 6250 code. Qualex chose the Series 3000 transport it uses (manufactured by by Telex) because:

* the product is the only unit in it's size designed from the start for 6250 operation.
* the product had extensive field use with impressive reliability.
* the product was designed for serviceability and featured quality components and conservative design margins.

By way of contrast, the competitive product is actually a performance strained 75 ips, 1600 BPI machine modified to operate at 6250 BPI, but unable to write the IBM/ANSI standard 0.3 inch inter-record gaps on start/stop operation and also operates at excessively high tape tension.

The main thrust of the first part of this paper is to point out that meeting the challenge of 6250 BPI tape technology requires a more complex design with a corresponding increase in cost. This increase in cost translates into substantial benefits to the user in terms of a significant improvement in reliability; dramatically higher performance; along with the added benefits of ease of operation. The performance match between current disk drives and tape memory has now been satisfied and in the bargain the user has gained a more error tolerant product and the solution(s) to the previous tedious requirements, and resultant problems, associated with operator tape handling.

DISTRIBUTED 6250

The higher cost of this technology is what prompted Qualex to study methods of maximizing return on investment (ROI) to the the user. Qualex's current product is already lower cost and higher performance than the 7976A. However, to further enhance ROI, Qualex has designed SMASH (Shared Mass Archive Storage Host) to allow sharing of this state-of-the-art performance and technology over multiple CPU's.

SMASH allows the operator to switch the Qualex Group 3000 tape system between two, three or four CPU's. As the next slides will show, this system can be shared with various models of the HP3000 computers.

The first slide shows a single tape system tied to a Series III CPU. The next slide adds the Shared Mass Storage Feature option 002. The following slide shows option 004 which allows coupling four CPU's to the Group 3000 tape system. The CPU's can be a mix of Series II or III's and Series 44's.

The hardware elements of SMASH are housed in the Series 3000 tape controller/tape drive cabinet. No additional cabinet is required.

Selection of which CPU is connected to the tape system via SMASH is done by an operator activated switch.

A review of cost savings for the user is covered in the following charts.

As can be seen, this cost savings of the Qualex tape systems are significant, running from $20,000 to $194,000 depending on the configuration selected.

A significant dimension of the SMASH product for the single CPU user, making an investment in 6250 tape today, is the opportunity to enhance his return on investment when his site upgrades to a new computer and/or adds additional CPU capability to his site.

In summary, 6250 BPI technology is now a reality for HP3000 computers. The user has options to choose from with the most current being the SMASH capability introduced by Qualex. 6250 BPI technology is state-of-the-art in tape design - brings many benefits to the user and with the announcement of SMASH, provides a cost effective solution to a multiple CPU site.

A-7 - 07

DISTRIBUTED 6250 BPI TAPE

QUALEX

USER ADVANTAGES OF 6250

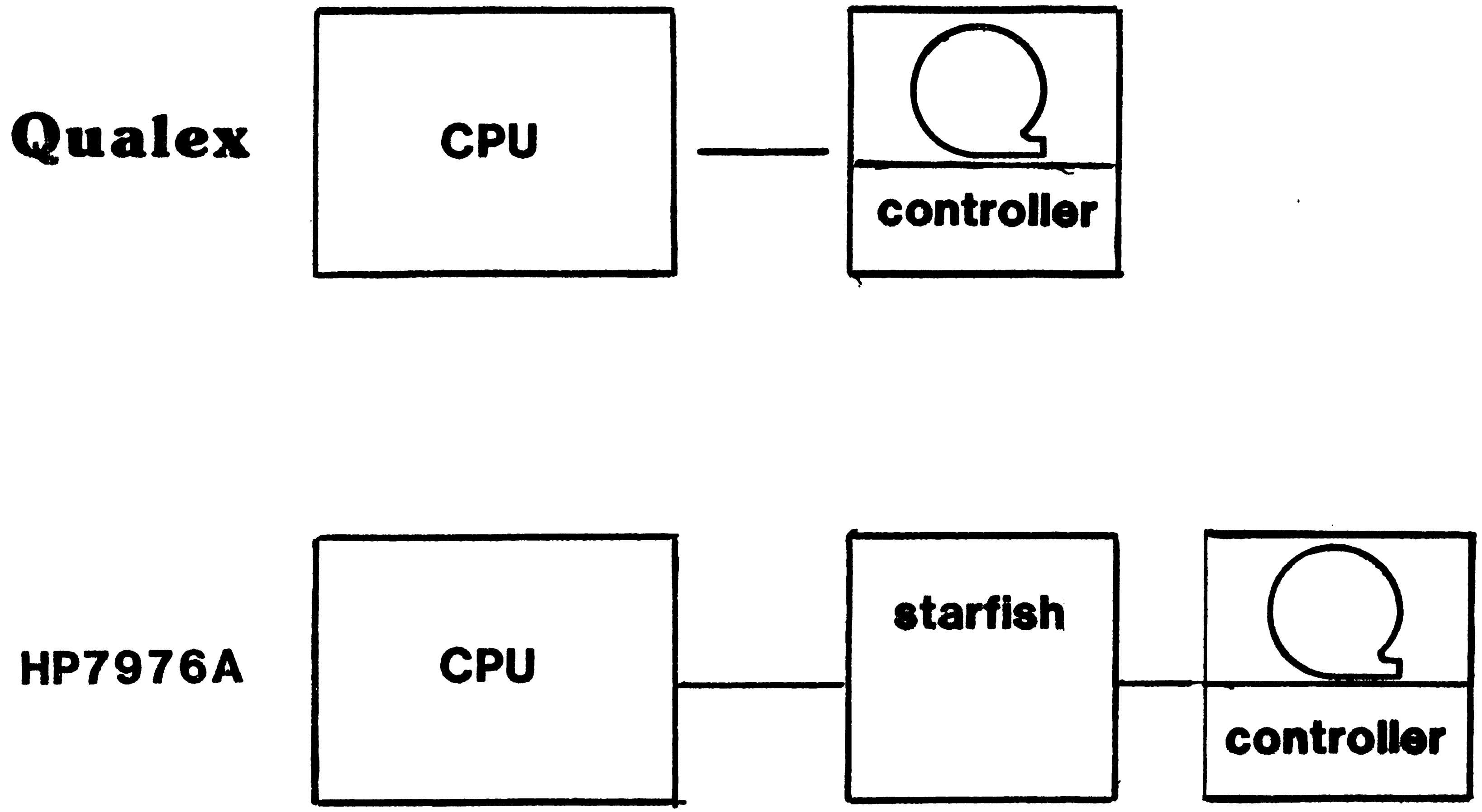
QUALEX

6250 TAPE PROVIDES UNIQUE ADVANTAGES TO HP USERS

- Auto Thread/Autoload
- Triple Density
- High Speed Rewind
- 125 ips Tape Speed
- Switch Selectable Density
- Automatic Hub
- Improved Operator Features

QUALEX

SINGLE TAPE SYSTEM



QUALEX

SMASH - ANNOUCEMENT

QUALEX

QUALEX INTRODUCES

"SMASH"

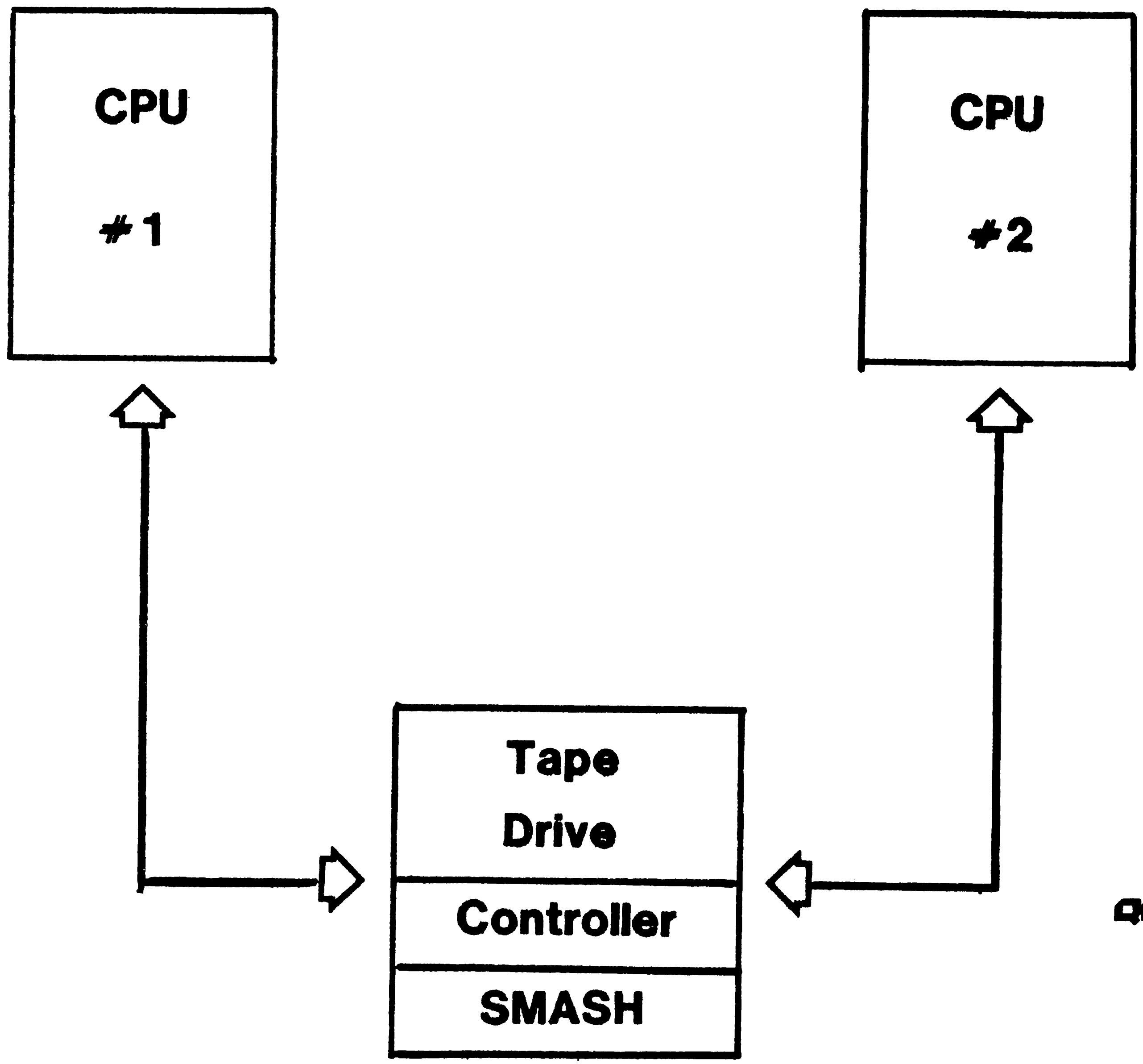
SHARED MASS ARCHIVE STORAGE HOST

QUALEX

SMASH (2 CPU'S)

QUALEX

SMASH - Shared Mass Archive Storage Host

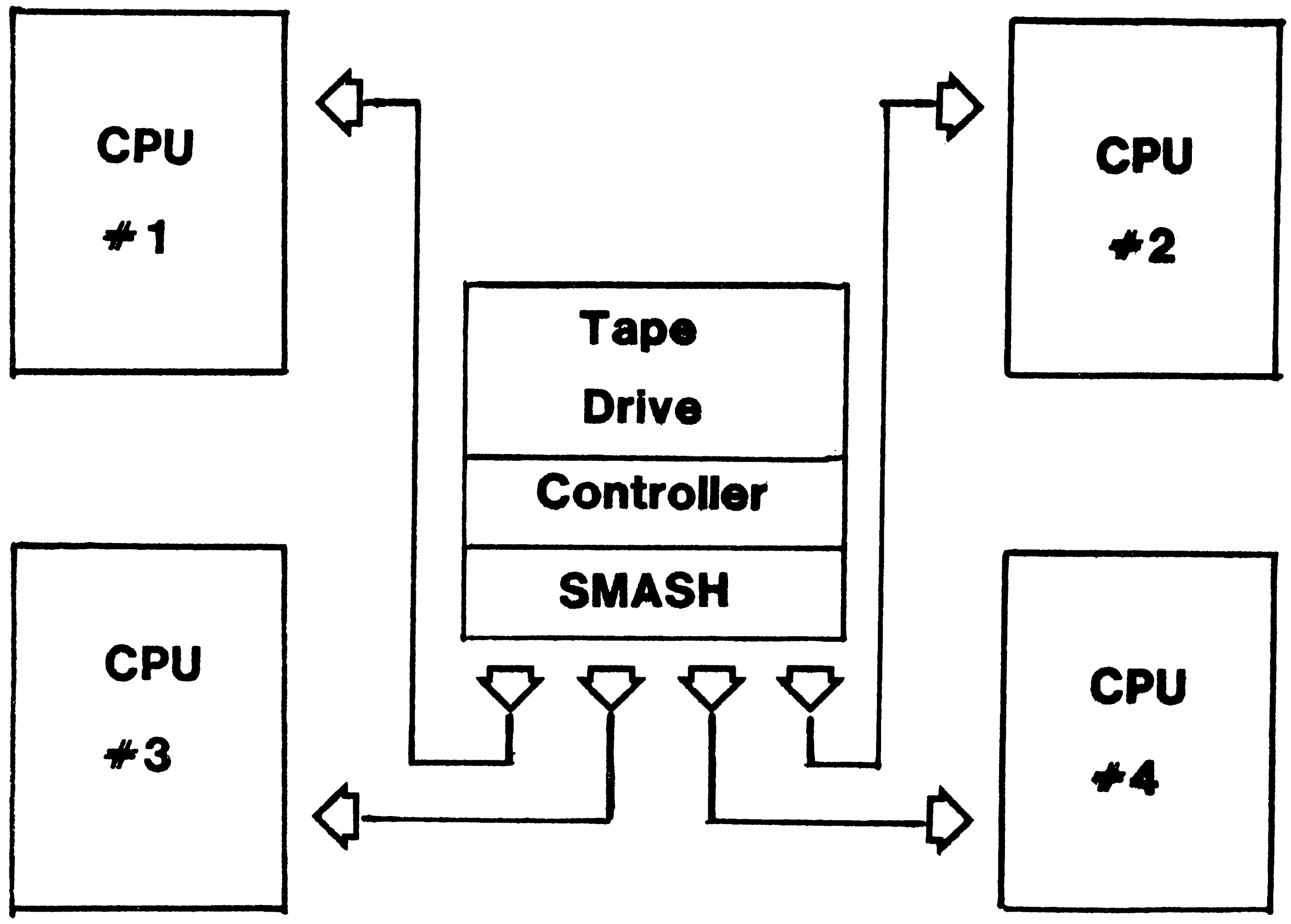


QUALEX

SMASH (4 CPU'S)

QUALEX

SMASH - 4 CPU Version



Note: CPU'S may be Series II, III or 44.

QUALEX

COST COMPARISON - SINGLE CPU SERIES III

QUALEX

COST COMPARISON

SINGLE SYSTEM on SERIES III

Qualex:	$47,500
HP:	$67,480
Savings:	$19,980

QUALEX

COST COMPARISON - SINGLE CPU SERIES 44

SINGLE SYSTEM on SERIES 44

Qualex:	$44,500
HP:	$52,250
Savings:	$7,750

QUALEX

COST COMPARISON - TWO CPU'S SERIES III

QUALEX

COST COMPARISON - TWO CPU'S SERIES III

Series III - without SMASH		Series III - with SMASH	
Qualex:	$95,000	Qualex:	$61,614
HP7976A:	$134,960	HP7976A:	$134,960
Savings:	$39,960	Savings:	$73,346

QUALEX

COST COMPARISON - TWO CPU'S SERIES 44

QUALEX

COST COMPARISON - TWO CPU'S SERIES 44

Qualex – with SMASH:	$61,614
HP7976A:	$104,500
Savings:	$42,886

QUALEX

COST COMPARISON - FOUR CPU'S
2 SERIES III
2 SERIES 44

QUALEX

COST COMPARISON - 4 CPU'S (2-III, 2-44)

Qualex - with SMASH:	$75,826
HP7976A:	$239,460
Savings:	$163,634

QUALEX

SOFTWARE MAINTENANCE AND SUPPORT IN THE DISTRIBUTED ENVIRONMENT

By:

Richard L. Foote
Systems Consulting Services

Distribution of processing power can pose new challenges for DP departments and users alike. Development, installation, training and all other aspects of DP become increasingly complex as the number of sites increases. A variety of methods are available for meeting this challenge. They typically involve changes in organizations, software and procedures. Flexibility and responsiveness are key ingredients. Traditional support techniques are reviewed and alternatives are explored. Case histories are used to illustrate ways of meeting the distributed support challenge.

QUERY - DIRECTIONS FOR THE 1980'S

By:

Orland Larson
Product Manager, HP

The purpose of this presentation will be to announce Hewlett-Packard plans for enhancements to QUERY. This will include a list and a brief description of these new enhancements.

DATACOM FOR FIRST TIME USERS

by

Tom Black
Marketing Manager, HP

&

Roselie Tobes
Sales Development Manager, HP

This presentation is intended for DATACOM neophytes. It covers the fundamental concept of computer datacommunication without using large munbers of "buzz words". The DATACOM products available on the HP 3000 computer family will be reviewed, and their overall capabilities discussed. In addition, some of the major considerations for a successful DATACOM installation, based on HP's extensive experience in this area, will be reviewed.

FULL TEXT WILL BE DISTRIBUTED AT THE SESSION

EVOLUTIONARY SYSTEMS DEVELOPMENT IN A DISTRIBUTED ENVIRONMENT

By A. Steven Wolf

Digital Communications Corporation

ABSTRACT

A philosophy and methodology of on-line systems evolutionary development is presented for a distributed HP 3000 environment. Tools to support this philosophy including HP products including IMAGE, V/3000, and DSG/3000 and non-HP products including Teleprocessing Monitors, Report Generators and data base utilities are discussed. Finally, examples of the usage of this philosophy and associated tools in the actual development of a marketing information system is discussed. This paper is intended for system managers, MIS directors and programmers interested in developing on-line systems that can be implemented without complex and time-consuming programming which may change as the needs of the users community changes.

APPLICATION SYSTEM OPERATION AND CONTROL

BY BARRY D. KURTZ
(DEVELOPMENT ENGINEER/MTS)

MANUFACTURING SYSTEMS OPERATION R&D
HEWLETT PACKARD COMPANY
CUPERTINO, CALIFORNIA

APPLICATION SYSTEM OPERATION AND CONTROL

TABLE OF CONTENTS

I. Introduction

II. Application Systems.

- A. Traditional Activities.
- B. User Interface.

III. HP's Recent Approach to Application System Management on the HP3000.

- A. The Application Monitor.
- B. Classes of Users.
- C. User Interface.
- D. Environment Definition.
- E. Application System Operation.
- F. System startup.
- G. Interactive Application Process Management.
- H. Background Job Management.

IV. Conclusion.

I. Introduction.

Application System Operation and Control is the daily execution of, and supervision of, application system functions. It includes initiating programs that provide application functions to users and scheduling and monitoring background processes performing batch processing.

Until now, application system operation and control has required considerable user intervention. Also, since most buisness computing machines are general-purpose in nature, the task of initiating application programs is frequently left to personnel who are not computer professionals. Thus the user of an application (a clerk, receiving dock worker, etc.) may have to learn the host computer's command language and error codes, which are optimized for general machine use instead of individual applications.

Recent developments in operating system software have lessened the burden on the non-DP professional. These include user-defined commands, comprehensive "help" facilities, and so on. However, these have solved only a part of the problem. The user of an application should perceive the computer system as a comprehensive solution to an application problem and not as a set of unrelated application tools.

Hewlett Packard's Application Monitor (developed as a component of HP's Materials Management/3000) has made a significant contribution to the ease of control and operation of application systems. By controlling and supervising application system activities, the Application Monitor greatly reduces the system management time required of the system administrator. This allows more time for management of external functions in the environment where the applications are used.

II. Application Systems.

A. Traditional Activities.

Most application systems involve the following activities:

- Initiation of on-line applications on user terminals.
- Scheduling and monitoring of background job processing.
- Initiating recovery and cleanup jobs.
- Supervising system operation in order to maintain consistency in the application system environment.

Traditionally, most of these activities were executed manually and required frequent human intervention. Control of the application environment was totally dependant upon the constant attention of the administrator or operator of the system. This practice has led to inconsistent data processing activities, sometimes resulting in the loss of critical management reports or accidential data file destruction.

B. User Interface.

User Interface is a very important area of application system management. The easier it is for a user to utilize the functions of an application system, the more productive each user will be.

In many cases, when a user desires to execute a particular application function, a program must be manually initiated through the use of operating system command language (whether it be an interactive process or background job). This requires a knowledge of some command language syntax and the capability to interpret the host computer's error codes (which may be difficult for the non computer professional).

The development of user-defined commands and comprehensive "help" facilities have certainly improved the user interface to application systems, but much more can be accomplished to ease the burden on the non-DP professional.

II. HP's Recent Approach to Application System Management on the HP3000.

A. The Application Monitor.

The Application Monitor was developed as a component of HP's Materials Management/3000 product. It is an integral part of an application system. The monitor controls the execution of, and provides services to, applications running under its control. It schedules, initiates and controls all interactive and batch job activities in an application system. The monitor takes a major step towards operatorless, abortless, application systems by providing automatic application scheduling, control, and recovery services.

B. Classes of Users.

Two main classes of users make use of the services provided by the monitor:

o System Administrators

o End users

The system administrator is an individual who has global responsibility for the application system. This individual supervises all application system activities (i.e., background job scheduling, on-line application scheduling and control, etc.).

The end users of an application include all those who use application programs functions (i.e., clerks, managers, receiving dock personnel, stores or inventory personnel, etc.). End users benefit from good application system management, which leads to consistent data processing, reports that are on time, and other benefits. However, they do not usually get involved in actual operation and control.

C. User Interface

The user interface utilized by the Application Monitor is a friendly fill-in-the-form CRT interface. This allows the user or manager of the system to perform comprehensive system control without having to learn a complex command language. Each screen presented utilizes the CRT terminal's function keys. The functions are described in eight shaded boxes at the top of the screen. Each box corresponds to a single function key (the leftmost box corresponding to function key 1 and so on.).

A command window is also presented on monitor menu selection screens. This area is utilized for special system control functions that may not be executed by a simple function key signal.

Sample user interface screens will be presented as part of this discussion.

D. Environment Definition.

The application system environment is defined through the use of the Application Customizer (also a component of Materials Management/3000). The customizer allows the user to define the following information for system operation:

- o Terminal configuration
- o Device configuration
- o Schedule of when interactive applications are to run and their associated CRT terminals.
- o Schedule of when batch jobs are to be initiated.

This information is stored in the customizer's application data dictionary for later user by the monitor. When the data in the dictionary is to be applied to the current application system environment, a process is executed that will copy this information into a "prepared" run time version of the dictionary.

The customizer's data dictionary contains other information which is application subsystem dependent (i.e., data item definitions, screen formats, data base formats, etc.) and will not be discussed in this paper.

E. Application System Operation.

The Monitor is composed of eight seperate programs. These programs run simultaneously and work together to monitor and control the application system (Figure 1 depicts the application system environment.).

```
         ----------------------------------
         | Host Computer Operating System |
         ----------------------------------
                        |
           ------------------------------
           |  Monitor Control Process   |
           ------------------------------
                        |
                  ------------------------
                  |                      |
                  |       ------------------------------------
                  |       | Interactive Application Programs |
                  |       ------------------------------------
                  |
Monitor Services  |
---------------------------------------------------------
|  SIP  |  SAI  |  SMP  |  SAP  |  PSM  |  JSP  |  BJP  |
---------------------------------------------------------
```

Figure 1. Application System Environment.

The application system environment, showing the relationship of the Application Monitor to other programs. The Application Monitor initiates, monitors, and controls run time application activities. Monitor services are provided to application programs and users to initiate additional processes and supervise system activity.

The following is a brief description of the major processes that make up the Application Monitor.

Monitor Control Process (MCP)

The Monitor Control Process is the parent of all programs in the application system environment. Once the application system is active, it is this program which initiates, monitors and controls application program activities.

The MCP utilizes a memory table to track program activity throughout the system. This table is called the Application Control Table (ACT). Each program in the system has an entry in the ACT called a Process Information Block (PIB). The information kept in each PIB is sufficient for the MCP to monitor current status of each program and provide any services that are determined necessary.

System Initialization Process (SIP)

The System Initialization Process is initiated at system startup time and executes once a day. This program reads a prepared application data dictionary generated by the Application Customizer. Based on the information read from this dictionary, the SIP initializes global tables to be used by the application system.

System Administrator Interface (SAI)

This program is the system administrator's "window" to the Application Monitor. The SAI allows the user to select various functions to review and control application system operation.

Figure 2 depicts the user interface for the System Administrator Process.

Job Scheduler Process (JSP)

The Job Scheduler Process automatically schedules background jobs that were specified to be run on the current day. These jobs are pre-defined by the system administrator through the customizer.

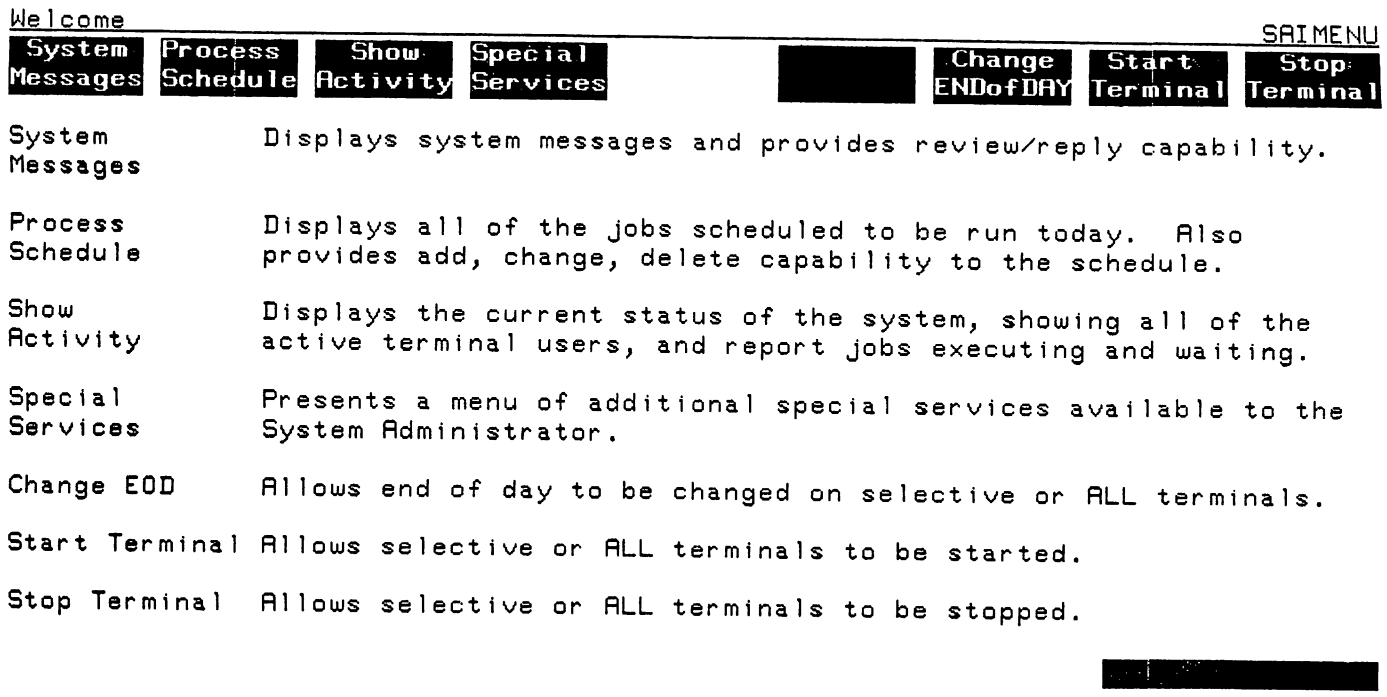
Background Job Processor (BJP)

A background job that has been scheduled for execution will be processed by the Background Job Processor. This program executes background job commands and provides comprehensive job restart/recovery capability.

System Activity Process (SAP)

This program allows the system administrator to review and control system activities. Through the SAP user interface, the system administrator can review the current activity of all interactive and background jobs and control background job processing concurrency.

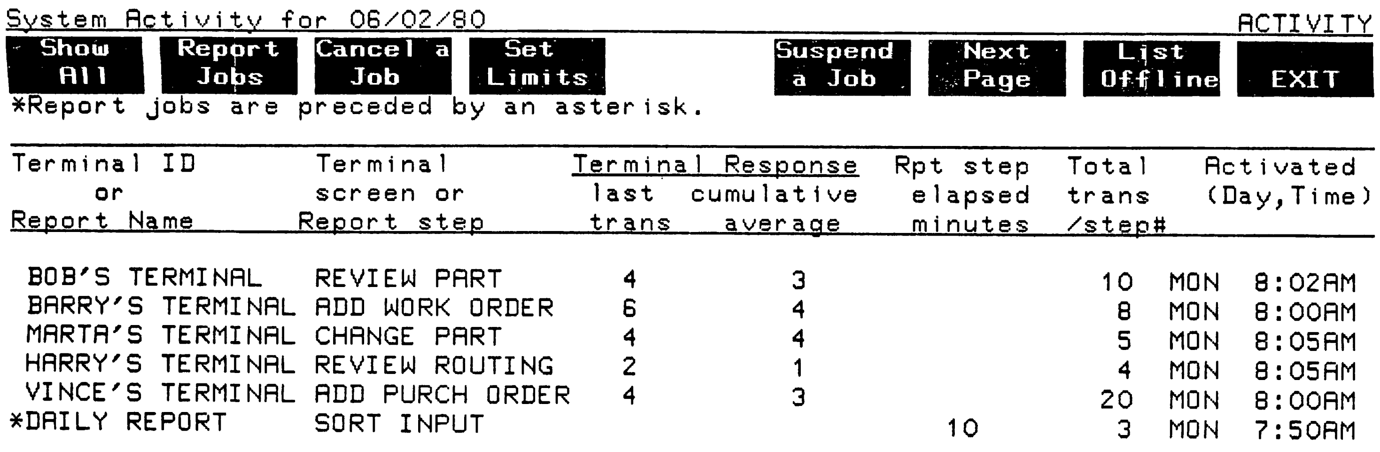
Figure 3 depicts the System Activity user interface.



System Messages	Displays system messages and provides review/reply capability.
Process Schedule	Displays all of the jobs scheduled to be run today. Also provides add, change, delete capability to the schedule.
Show Activity	Displays the current status of the system, showing all of the active terminal users, and report jobs executing and waiting.
Special Services	Presents a menu of additional special services available to the System Administrator.
Change EOD	Allows end of day to be changed on selective or ALL terminals.
Start Terminal	Allows selective or ALL terminals to be started.
Stop Terminal	Allows selective or ALL terminals to be stopped.

Figure 2. System Administrator Interface.

The System Administrator Interface is the administrator's "window" to the application system. The functions which may be initiated from this screen are represented in eight shaded boxes at the top of the screen. These functions are initiated by depressing one of eight CRT terminal function keys. The first shaded box corresponds to function key 1, the second to function key 2, and so on. This relieves the system administrator of the need to memorize a system command language, and allows functions to be initiated quickly and accurately.



```
System Activity for 06/02/80                                          ACTIVITY
 Show    Report  Cancel a   Set            Suspend    Next    List
 All     Jobs    Job       Limits          a Job      Page    Offline   EXIT
*Report jobs are preceded by an asterisk.

Terminal ID        Terminal          Terminal Response   Rpt step   Total     Activated
    or             screen or         last  cumulative    elapsed    trans    (Day,Time)
Report Name        Report step       trans   average     minutes    /step#

 BOB'S TERMINAL    REVIEW PART         4        3                    10   MON   8:02AM
 BARRY'S TERMINAL  ADD WORK ORDER      6        4                     8   MON   8:00AM
 MARTA'S TERMINAL  CHANGE PART         4        4                     5   MON   8:05AM
 HARRY'S TERMINAL  REVIEW ROUTING      2        1                     4   MON   8:05AM
 VINCE'S TERMINAL  ADD PURCH ORDER     4        3                    20   MON   8:00AM
*DAILY REPORT      SORT INPUT                             10          3   MON   7:50AM
```

Figure 3. System Activity Process User Interface.

The System Activity Process allows the system administrator to review current system activity. Interactive application transactions may be tracked and background job execution may be controlled. In this example, BOB'S TERMINAL is executing the REVIEW PART transaction (ten transactions have been executed, and the terminal has been active since 8:02 AM.). A backgound job (DAILY REPORT) is running and is presently executing the SORT INPUT step.

Processing Schedule Maintenance (PSM)

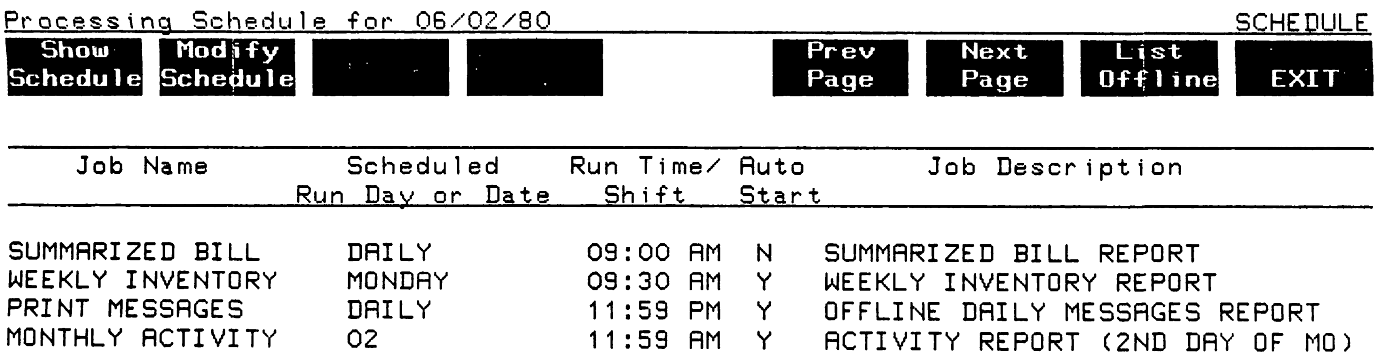
The Processing Schedule Maintenance process allows the system administrator to review and modify the background job processing schedule for the current day.

Figures 4 and 5 depict the user interface for the Processing Schedule Maintenance process.

System Messages Process (SMP)

Messages informing the system administrator of the specifics of system activity may be reviewed with the System Messages Process. Certain messages may require a reply from the system administrator. The reply may be processed utilizing the SMP user interface.

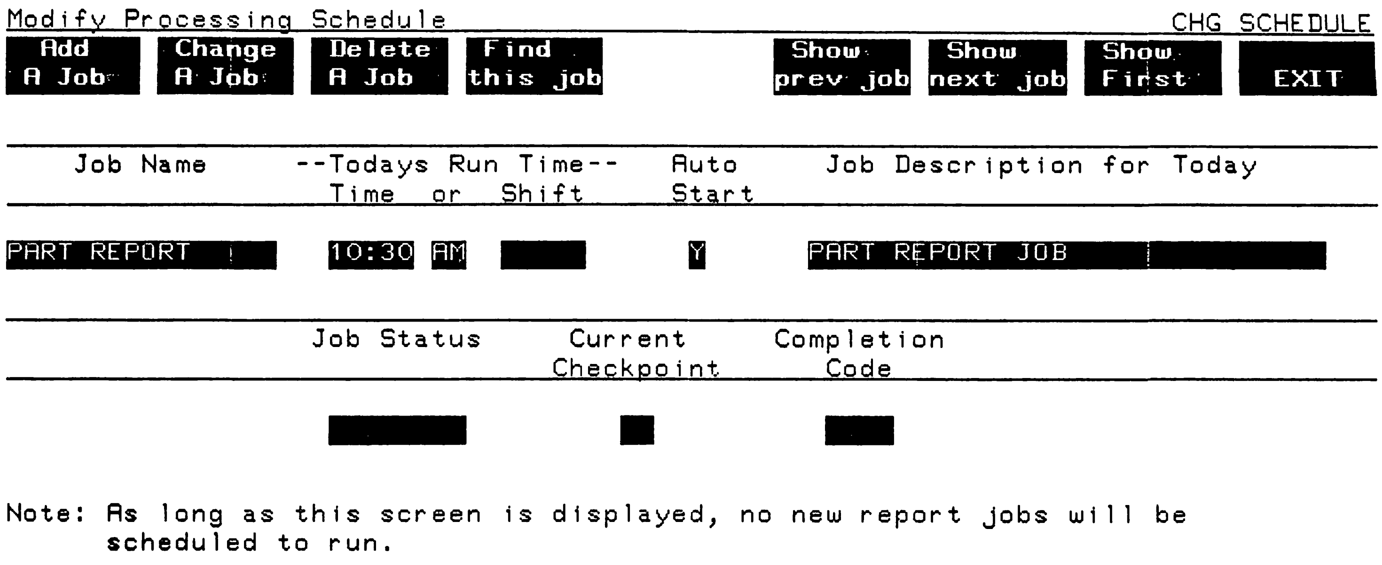
Figure 6 depicts the user interface for the System Messages Process.



```
Processing Schedule for 06/02/80                                      SCHEDULE

     Job Name           Scheduled      Run Time/ Auto         Job Description
                    Run Day or Date    Shift    Start

SUMMARIZED BILL      DAILY              09:00 AM  N   SUMMARIZED BILL REPORT
WEEKLY INVENTORY     MONDAY             09:30 AM  Y   WEEKLY INVENTORY REPORT
PRINT MESSAGES       DAILY              11:59 PM  Y   OFFLINE DAILY MESSAGES REPORT
MONTHLY ACTIVITY     02                 11:59 AM  Y   ACTIVITY REPORT (2ND DAY OF MO)
```

Figure 4. Processing Schedule Maintenance User Interface

The Processing Schedule Maintenance process allows the system administrator to review the background job schedule for the current day. In this example, three jobs are scheduled to be run. The first job (SUMMARIZED BILL) has its AUTO START flag set to "N". This means that it will require manual intervention to run. The administrator must set the flag to "Y" in order for the job to run. The other jobs will run automatically when their time comes up.



Modify Processing Schedule CHG SCHEDULE

Add A Job | Change A Job | Delete A Job | Find this job | Show prev job | Show next job | Show First | EXIT

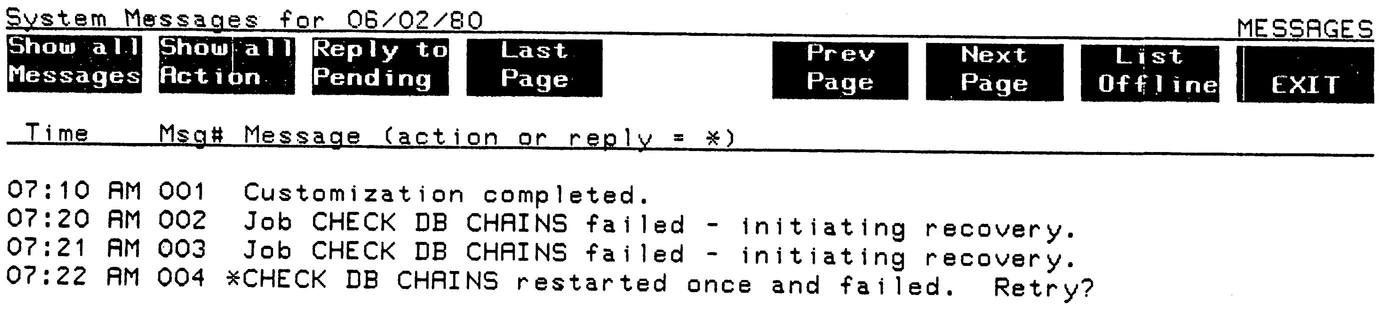
Job Name	--Todays Run Time-- Time	or Shift	Auto Start	Job Description for Today
PART REPORT	10:30 AM		Y	PART REPORT JOB

Job Status	Current Checkpoint	Completion Code

Note: As long as this screen is displayed, no new report jobs will be scheduled to run.

Figure 5. Processing Schedule Maintenence User Interface.

The system administrator may modify the current day's background job schedule through the use of the Processing Schedule Maintenance process. In this example, a job (PART REPORTS) is being added to the schedule. If this job is to be run at regular intervals, the system administrator may add it to the permanent job schedule in the application data dictionary via the customizer. The job will then automatically be scheduled to run daily, weekly, monthly, or yearly as specified.



```
System Messages for 06/02/80

 Time     Msg# Message (action or reply = *)

07:10 AM 001  Customization completed.
07:20 AM 002  Job CHECK DB CHAINS failed - initiating recovery.
07:21 AM 003  Job CHECK DB CHAINS failed - initiating recovery.
07:22 AM 004 *CHECK DB CHAINS restarted once and failed.  Retry?
```

Figure 6. System Messages Process User Interface.

The System Messages Process allows review of system informational messages and messages requring action from the system administrator. In this example, a background job has been restarted and failed on the restart. The Background Job Processor is asking the system administrator if the job should run again. The system administrator may initiate the reply action to this messages by depressing the appropriate CRT terminal function key which corresponds to a shaded box at the top of the screen.

F. System Startup

The Monitor Control Process is the initial program to be run in the application system. The MCP is initiated via a user defined command supplied with the installation software. This relieves the user from having to know the physical file name of the MCP and any parameters that must be supplied.

The MCP launches the System Initialization Process. The SIP reads the prepared application data dictionary generated by the customizer. This dictionary contains information critical to system operation and control. Utilizing this information, the System Initialization Process builds global tables to be used during that day's operation of the system. When initialization is complete, the MCP begins normal execution.

Figure 7 depicts the application system environment during system start-up time.

G. Interactive Application Process Management

Application processes are launched according to values initialized in the ACT. When the MCP initiates an application program it sends the program its PIB entry number. This PIB entry number corresponds to a physical PIB in the ACT. By utilizing this number in Monitor supplied intrinsics, the application may perform the following functions:

- o Obtain information regarding which interactive terminal to use
- o Start and communicate with a concurrent process to facilitate simultaneous processing of data
- o Send a message to and start a successive process suspending execution of the application until the successive process completes
- o Log transaction response time for review by the system administrator

Once the application system is in operation, interactive applications are automatically initiated according to the schedule defined by the system administrator via the Customizer. If the schedule defined for application initiation is accurate, no user interaction is necessary to gain access to application program functions (The appropriate application will be presented to the appropriate set of users at the appropriate time.).

If special needs arise, the system administrator may modify interactive application activity through the System Administrator Interface.

Figure 8 depicts the MCP's management of interactive applications.

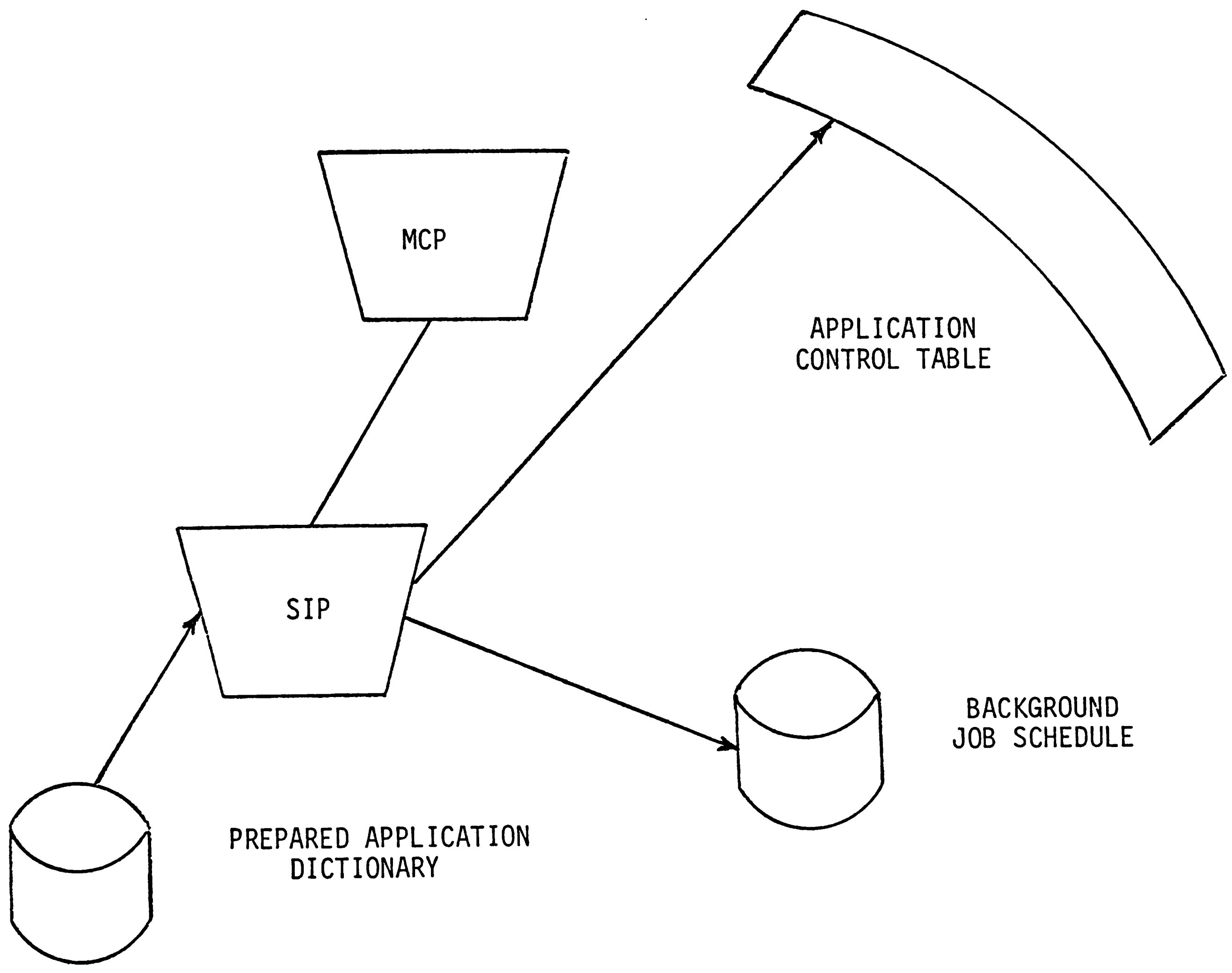


Figure 7. Application System Startup

The Monitor Control Process (MCP) is the first program to run in the application system. Its first step is to launch the System Initialization Process (SIP). The SIP reads a prepared application data dictionary generated by the Application Customizer. From this dictionary, the SIP builds system-wide tables used in operation and control of the application system. The two main tables generated are the Application Control Table (ACT) and the background job schedule. Once the tables are initialized, the MCP begins normal operation and continues execution indefinitely.

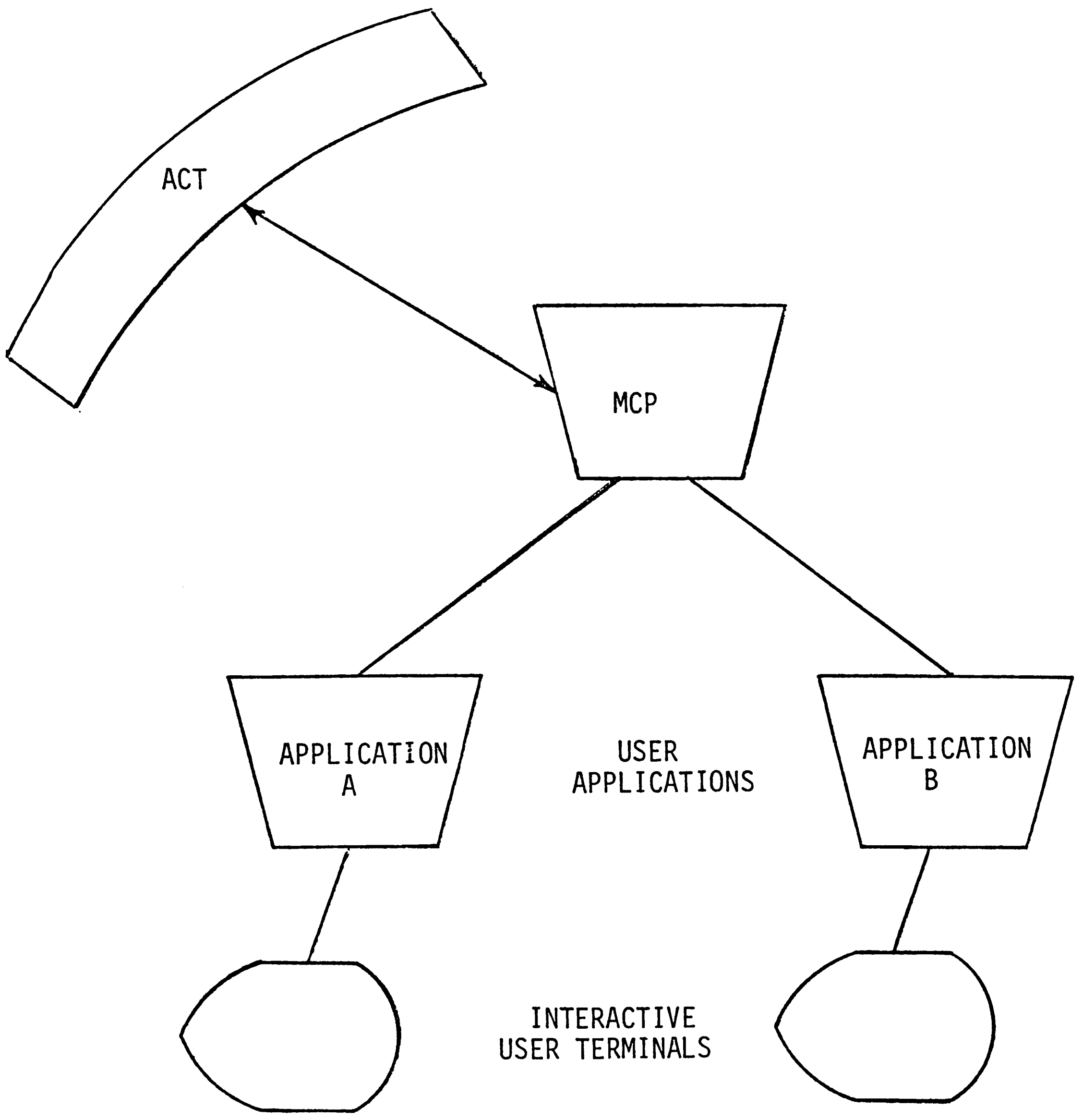


Figure 8. Interactive Application Management

The Monitor Control Process (MCP) reads the Application Control Table (ACT) built by the System Initialization Process. According to values initialized in the ACT the MCP launches application programs for presentation of application functions to users. The application programs utilize monitor intrinsics to request services and allow the system administrator to review and control application activities.

H. Background Job management

Background jobs are scheduled for execution according to a job list created by the system administrator via the customizer. This list is read each day by the System Initialization Process to determine the jobs to be run for the current day.

The jobs are automatically executed at the time specified in the list. If a job fails, predefined recovery procedures are executed. When intervention is necessary, the system administrator may modify the execution sequence of background jobs via the Processing Schedule Maintenance process.

Figure 9 depicts Monitor's management of background jobs.

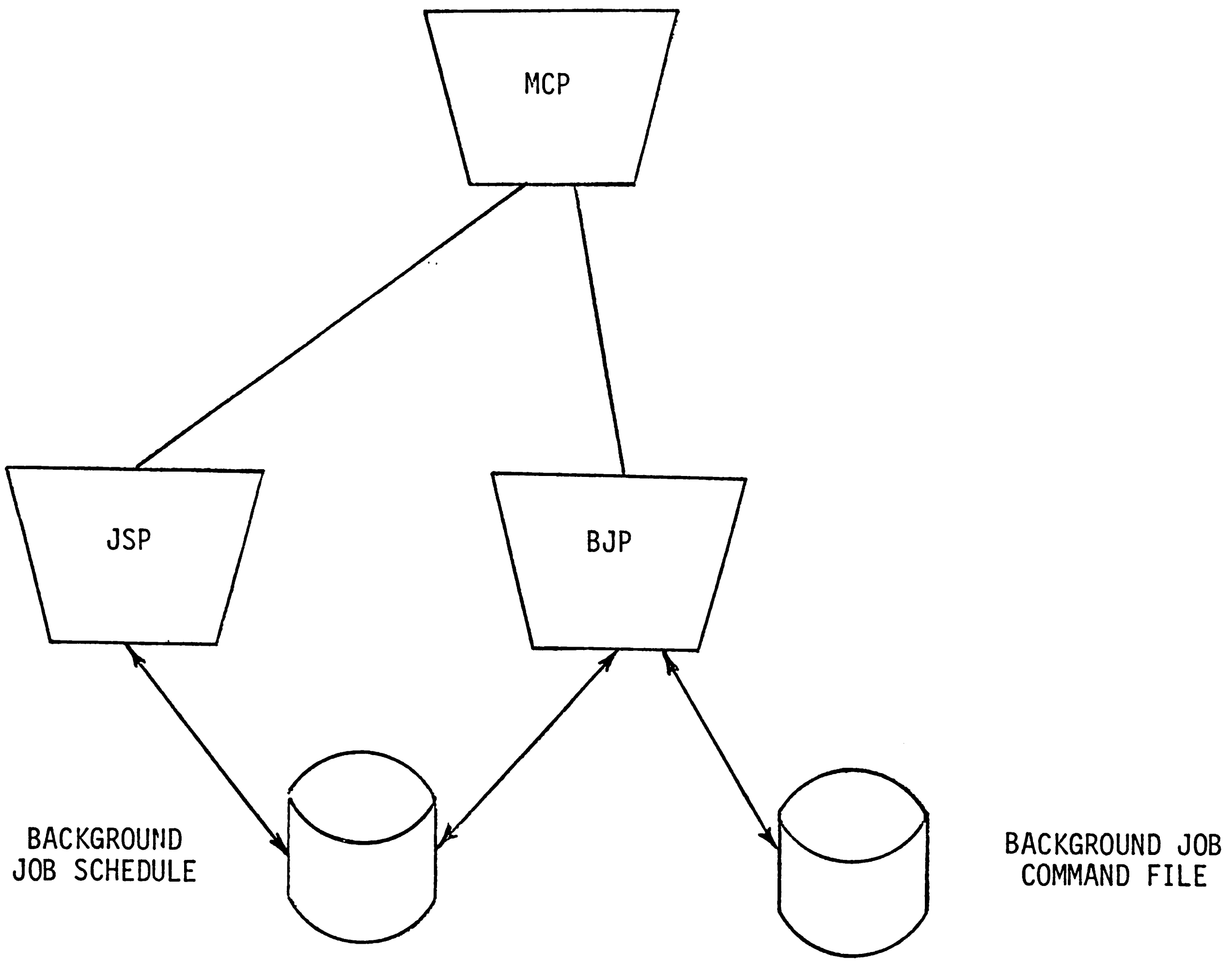


Figure 9. Background Job Management.

The Job Scheduler Process (JSP) reads the Background Job Schedule to determine which jobs should be scheduled for execution at the current time. All jobs that have run times on or before the current time will be scheduled for execution. The Background Job Processor (BJP) executes a job by processing that job's command file. Up to three BJPs may run concurrently allowing concurrent execution of up to three background jobs. Each job may have recovery procedures and checkpoints defined. If a job fails, the recovery procedures are automatically executed and the job is restarted at the appropriate checkpoint or step.

IV. Conclusion

The automation of an application system can significantly reduce the management time which is required of the administrator of an application system. This allows more time for management of external functions in the environment where the applications are used.

The Application Monitor is a major step in HP's long term commitment to increase the ease of use of application systems.

MANUFACTURING CONTROL, PLANNING AND FEEDBACK IN A DISTRIBUTED PROCESSING ENVIRONMENT

By: Mick Belcham
Martin Marietta Data Systems

For a multi-plant manufacturing company the decision to implement a Distributed Processing environment is obviously important. However, it only really becomes significant if the company's management team is capable of recognizing and implementing the associated required changes in management practice and control.

Here I am not referring as much to data processing management as I am to the company's operations management -- those that control what each plant manufactures and the resources required to do so -- money, people, plant, inventory, etc. To them, the decision is not so much one of Distributed Processing, not even one of Distributed Systems, but more importantly one of Distributed Management Control. It is the formalizing of the levels of responsibility (and hopefully, authority) that each autonomous production facility has in establishing what it makes and when it makes it.

Let's take as an example a company that manufactures power tools and other related equipment.

For the sake of the example let us assume that its headquarters and its major final assembly operations are in the Chicago vicinity and that over the years it has grown and built other manufacturing facilities in the Mid-west and more recently in two of the Southern states. To date it has done little to change management policies to reflect this multi-plant situation -- certainly nothing to take advantage of this "distributed manufacturing" environment. All it has seen has been the increased complexity of making shipment schedules from the main assembly plant when much of what is shipped is dependent upon the performance of remote fabrication and sub-assembly plants; plants that suffer not only from being managed independently of the main plant but also from being "buffered" from the shipment schedule by inter-plant transportation problems.

What has it tried to do? The same thing that all of us would do if we were to go after the symptoms and not the cause. It has attempted to tighten centralized control over the remote plants. Just as it was getting used to the idea of "distributed manufacturing" -- even dreaming of installing small computers at each plant site! -- it has had to reverse its posture and make each plant more (rather than less) dependent upon centralized schedules. All plans for remote computer sites have been forgotten, the central computer has been upgraded and the old system (originally designed to support a single plant environment) has been significantly modified to fit the new ideology.

And how have things improved? . . . you guessed it. They haven't. If anything they are worse. They can't even rely upon the "goodwill" of the remote plants any more. Everybody is blaming everybody else. There is even talk at the main assembly plant of <u>purchasing</u> some of those parts previously fabricated at one of the plants. "How else can we meet schedules? At least we will have a better chance of getting what we want when we want when we want it from an 'outside vendor'." Et cetera, et cetera, et cetera.

What was their mistake? Blindness. Blindness to the basics of all good management practice -- accountability and insulation. <u>Accountability</u> for one's own performance, <u>insulation</u> against everybody else's.

They simply went half way. Each plant was theoretically held accountable for its inventory levels, its production efficiencies and its ability to satisfy the main plant's requirements. The fact was however that each of these factors was more dependent upon the abilities of the main plant to assemble to a reasonably-stable production schedule that it was upon the quality of management control at the remote plant itself. Accountability without insulation.

Before examining the impacts of this scenario (and many like it) upon the requirements of a Distributed Processing environment let us take it one step further; take it to its eventual almost suicidal conclusion (all in the name of good traditional management practice, mind you!).

It had to go this one further step before the company's management could be jolted into asking the question that it should have asked all along, "How do I have to change my management philosphies in a distributed manufacturing environment? And what should be my information control mechanisms to support it?".

What was this one final step. Well, think about it? What would you do? Your company's falling apart by all accounts. You still make the highest quality power tools in the business, your company name is as well known as ever in the retail market place, etc,

etc. However two problems are demanding more and more of your time -- manufacturing costs are rising alarmingly and your distributors are complaining increasingly about your shipment performance.

You have done all you can to upgrade your control mechanisms -- in fact, in the last three years you've doubled the size of your data processing department to improve communications between the various manufacturing facilities. And yet the same old problem occurs time and again -- the remote plants cannot satisfy the assembly plant's requirements. When the remote plants are asked for their comments, the answer is always the same, "They tell us what they want. We assume that they are right. We begin to buy and to make according to their requirements and priorities -- and then they change them upon us. We have what they don't want, they want what we don't have."

As company management you always have three options; do nothing, do something, or do nothing and make it look like something!

Let us assume that the "do something" option prevails, and that you feel an element of sympathy with the position of the guys at the remote plants. What can you do further? The answer appears simple. Establish a new management policy -- assembly production schedules are to be re-established monthly for the next six months, but the schedule for the next two months is to be frozen; within that two month's time frame nothing changes. We'll make what we said we would make.

Now things begin to get really wild. The distributor's complaints are getting worse because they cannot react to the market place. Your own warehouse inventories increase because they now have to hedge against unexpected demand during those two months. Your assembly plant is now second-guessing both the distributors and the warehouses as to what is real demand and what is only destined for "hedge" inventory. Your remote fabrication and sub-assembly plants become outwardly complacent (after all they have a stable schedule). Inwardly however, they are bracing themselves for the inevitable re-direction of management policy once the euphoric honeymoon is over; once everbody recognizes that the company which once had an indisputable reputation for service has now become inflexible and un-reactive to the market place it serves.

Back to the management drawing board! The next step? Who knows? Let us hope however that at least some consideration is given to the potential advantages of "distribution" -- a Distributed Processing environment, using Distributed Systems under the over-all auspices of Distributed Management Control.

The definition of this last term - Distributed Management Control - is the crux of the whole matter. If implemented correct-ly it becomes the backdrop for all management control mechanisms and therefore for all related systems work, particularly in the area of Manufacturing Control.

As an example let's look at the interface between the main plant and the remote plants in the earlier example. In the context of management control it went through three distinct stages:

Stage 1: Very informal, lacking in commitment from either side, unpredictable. Each manager knowing what he is being held responsible for (and pleasing the assembly plant is not necessarily it!)

Stage 2: Very straight forward, anything the assembly plant wants immediately becomes a mandatory requirement upon the remote plant whether it's feasible or not. Each remote plant becomes a "puppet", accountable for what it cannot control.

Stage 3: Very structured, "you can only have what you thought you wanted 2 months ago"; total accountability.

How would it look with Distributed Management Control? In the simplest terms it would be like a mature customer/vendor relationship; formal when required, but flexible wherever possible. More specifically it would be like a high-volume customer/vendor relationship -- the sort typically controlled by a formal blanket purchase order with a series of releases against it.

The most important characteristic of such a relationship is that it can be monitored. The "bounds of reasonableness" are established so that both "sides" can see an exception as it occurs and examine its desirability prior to its becoming critical. Each has the "right" to refuse or accept an exceptional situation (as defined by their "blanket" agreement) making each accountable for his own performance. Equally, each recognizes the desirability of avoiding such exceptional circumstances and (hopefully) sees improved information and communication as the means by which this may be achieved.

Enter the world of Distributed Sustems. To quote from the APICS (American Production and Inventory Control Society) Dictionary, the term Distributed Systems "refers to computer systems in multiple locations throughout an organization, working in a cooperative fashion, with the system at each location primarily serving the needs of that location but also able to receive and supply information from other systems within the network."

In terms of the earlier discussion on Distributed Management Control, I would like to concentrate on the implied stand-alone characteristics of such systems in such a network. Obviously intercommunication is important and each system has to recognize the level of its dependence upon outside sources. However, the whole essence of Management Control such as we have discussed in this paper is its ability to operate in spite of everybody else. Whereas the system should be capable of communication with others, it should not be dependent upon it.

And this leads right into the second half of this paper -- how do today's on-line inter-active systems relate to such a management environment.

I would like first to dispense with a semantics problem - "Closed Loop Systems". It seems likely that this term is to win the buzz-word title of all time - more people appear to have listened, for longer, to more other people giving more different definitions of this one term than any other since Materials management became a science rather than a fall guy!

I hope you will bear with my own version of it.

I see it as being comprised of three functions - evaluation, feedback, and commitment. As such it represents a significant step forward in materials management practice. (after all it is not long since the only available words were plan, expedite, and smoke screen!).

What of these three more contemporary words:

Evaluation: Gone are the days (I hope) when the feasibility of a given production plan was merely a coincidental characteristic of it! Company managements have become increasingly more interested in Manufacturing's view of their proposed sales (shipment) schedules and, as such, are expecting more refined and provable answers. Meanwhile manufacturing now recognizes that the only way to avoid being "dumped on" is by quickly identifying problem areas - production bottlenecks, inventory restrictions, etc.

Feedback: This is the second "third" of a closed loop system - equally important but far less common that "evaluation". It is amazing how many companies that, having identified a production problem, limit their reaction to one of "gritting teeth"! Why? For at least two system-related reasons; the first being that their systems were designed only to identify the effect of the problem not the cause, and the second being that even if the cause <u>was</u> identified the production schedule cannot be juggled within the computer system to correctly reflect what manufacturing will eventually do to avoid it. Classic cases, always resulting in negligible feed-back.

Commitment: This third constituent of a Closed Loop environment is always the most difficult to obtain. However without it, the other two are useless. It is the difference between driving a manufacturing facility from a "statement of desire", and driving it from an agreed and feasible schedule. If a vendor cannot deliver on time, change the purchase order date so that the system can re-evaluate the impact. If we cannot ship to a customer on time, again, change the date on the order. Change anything that conflicts with reality. Drive the system off what we think we can do, not what we would have liked to have happened if everything had gone right. If a discrepancy occurs, evaluate it, feed it back to the point at which it can be averted, and commit to it. Buffer manufacturing against desirable schedules, give them something they can do, something they can commit to.

That's called closing the loop". As can be seen from the system descriptions that follow, the basic "closed loop" philosophies have done much to influence the design of today's state-of-the-art manufacturing systems.

What are the "components" of such a manufacturing system? (see Figure 1)

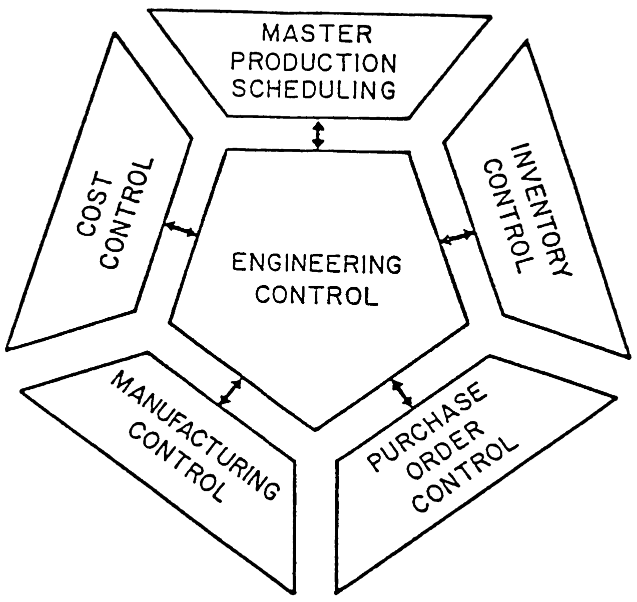


Figure 1 - Components of a Manufacturing System.

Engineering Control: This system should perform all maintenance and reporting functions concerning the system's six prime data bases:

-Item Master: One record for every part number relevent to the plant in question. It should contain all descriptive and policy information related to that part.

-Product Structure: A series of records for each manufactured item describing the lower-level items (raw material, purchased components, etc.) from which this item is made.

-Routing: A series of records for each manufactured item describing operation-by-operation how this item is to be made on the shop floor.

-Process: Descriptive information about each "process" identified on the routing data base, together with a list of tools required to complete that "process".

-Tool: One record for each tool referred to in the Process data base. It should contain descriptive and policy data related to the use and maintenance of that tool.

-Work Center: One record per production "center" in the shop. It should contain scheduling and efficiency data, capacity details, and cost rates.

Master Production Scheduling: This system should embody three functions - Production Planning, Resource requirements Planning, and Master Production Scheduling function itself. Specifically:

- Production Planning: The development of an overall statement of production and its "explosion" to the more detailed master scheduling level.

-Resource Requirements Planning: A broad-brush review of the likely impacts of a given master schedule upon critical production resources.

-Master production scheduling: The development of a realistic and detailed Master Schedule based both upon the exploded Production Plan and upon evaluation of Resource Requirements.

Inventory Control: This is the system that should translate the master schedule into a detailed replenishment plan and monitor progress and activity levels against it. Specifically:

-Planning: The development of the replenishment plan both for purchased and for manufactured items.

-Releasing: The preparation of a

replenishment order for release -- either for its placement with a vendor or for its dispatch to the shop floor for picking and manufacture.

-Expediting: The monitoring of the overall status of purchase and manufacturing orders.

-Recording: The recording of activity against each purchased or manufactured part -- issues, receipts, shipments, scrap, etc.

-Accounting: The development of all associated accounting transactions and the analysis of these and their resulting inventory levels.

-Management: The summarization of activities and performance of the planning-releasing-expediting -recording functions and their presentation in a form which management can interpret and act upon.

<ins>Manufacturing Control</ins>: (or more precisely, Shop Floor

Control) the further translation of the manufacturing replenishment plan (as developed in inventory control) into a detailed operation-by-operation statement of work, and the monitoring of status and performance against it. Specifically:

-Releasing: The identification of the appropriate routing for each production order, and the printing of its shop floor packet.

-Scheduling: The development of each production order's schedule, and the printing of each order's dispatch lists.

-Reporting: The gathering and recording of labor and other data from the shop floor.

-Expediting: The reporting of each order's status and any related exception conditions.

-Analysis: The periodic review and summarization of shop floor activity and the reporting of overall and detailed performance.

-Capacity Requirements Planning: The development of a long-term work plan for each work center and the comparison of this with the expected available capacities.

Cost Control: This component of the system should develop the appropriate standard costs used by the accounting function (discussed earlier under Inventory Control) as well as monitor actual purchasing and manufacturing costs against these standards. Specifically:

-Cost Generation: The build-up of tentative and/or current and/or firm standard costs for each part using the system's Product Structure and Routing data bases.

-Standard Costing: The translation of activity against purchase and production orders into their appropriate dollar equivalents, and the reporting both of performance variances and work in process levels.

Purchase Order Control: This system should perform the traditional Purchasing Department functions associated with the placing of purchase orders and the monitoring of vendor performance against them. Specifically:

-Placement: The printing of the

purchase order document itself and where relevent, the individual release schedules for each open blanket order.

-Expedite: The tracking of each order's status and likely "dock" dates

-Analysis: The monitoring of actual vendor performance.

-Reconciliation: The comparison of vendor invoices with their asso-ciated purchase order information.

Now that we have defined these system "components" we can take another look at "closing the loop". Figure 2 shows how some of these individual components tie together, not only from the point of view of logical information flow "downwards", but also from that of feedback (dotted lines) "upwards".

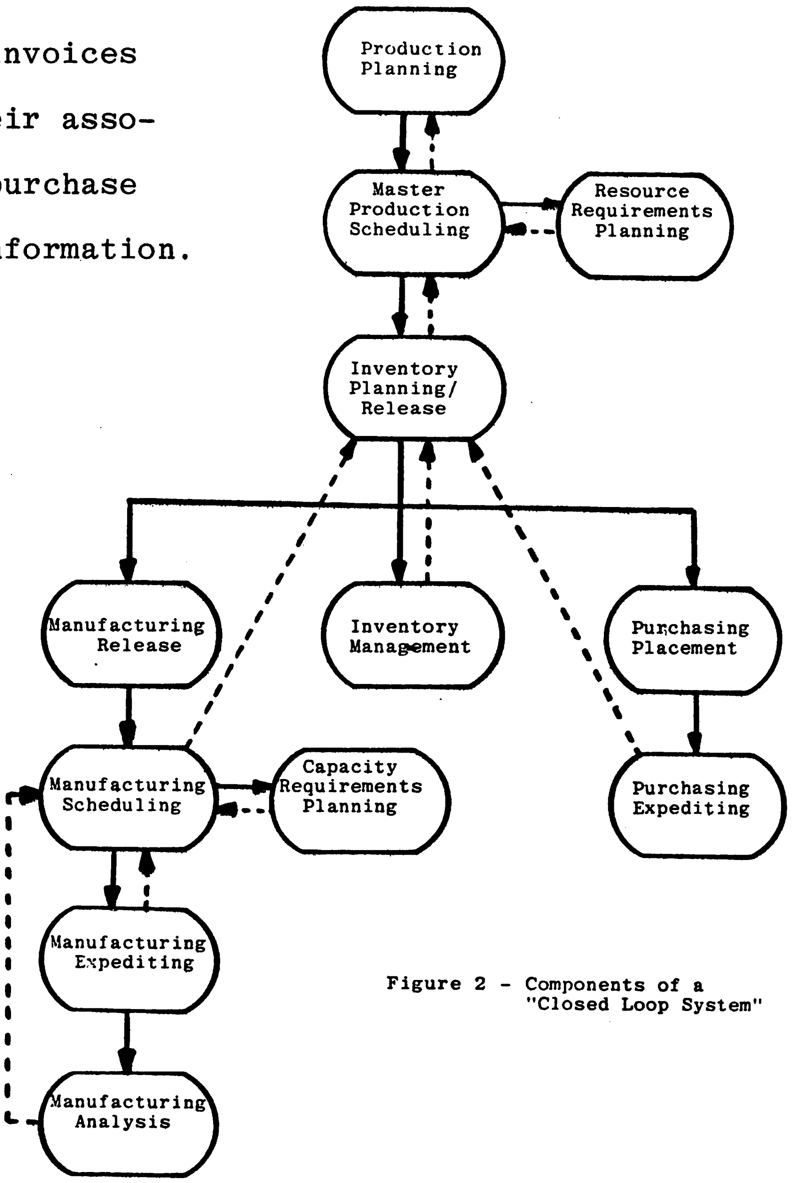


Figure 2 - Components of a "Closed Loop System"

So far we have:

-defined the concepts of Distributed Management Control

-defined a system model suitable for such an environment and shown how it should hang together.

Nexi we will describe some of the required system features in detail, but once again let us precede the discussion with another go at semantics - this time concerning the term "on line interactive systems:"

The only obvious consistency between the alternative definitions of this term is that none include the words "punched card"! Everything else is up for grabs! For the sake of the remainder of this paper I feel we should standardize on one view - mine, naturally!

Perhaps the best approach is to look at what I see it as <u>not</u> being. In the first case I see it as a blantant exageration of a system's capabilities, and in the second I see it as an understatement (incredible as that may seem!).

First as an exageration; I do not see the term applying to:

-a system with on-line editing but only batch updating capabilities.

-nor to a system limited only to on-line inquiry.

-nor to a combination of the two.

Equally as an understatement; I do not see the term applying to:

-a system that, with one on-line transaction, can evaluate a series of different situations and options, report back (still on line) an optimum result, and, when requested to do so, update a number of file records to reflect that conclusion.

In its simplest terms, therefore, I see the term referring to a system's ability to update files with transactions in full on-line mode and to be able to process an inquiry against those updated files with the immediately succeeding transaction.

What does all this mean to the specification of a good manufacturing control system? A lot. Simply specifying it as "on-line and interactive" is not sufficient. A good system will embody a large spectrum of environments - all the way from pure batch to what I described earlier as an "understatement" (and which I would like to more specifically entitle "interpretively interactive"). Let me give you three examples. One for each of three given environments.

(1) Pure Batch

ABC analysis and reclassification: rarely is a part's ABC classification used to drive any on-line decision-making activities. Its constant update therefore is hardly worth the effort; certainly not worth the overhead associated with on-line recalculation.

(2) <u>On-line/Interactive</u>

Issues and Receipts: in a dynamic manufacturing environment the movement of inventory is a constant activity. Many decisions are made only on the basis of the current availability of the part concerned. Without these movements being "on-line and interactive" the system would be incapable of estimating a part's current availability.

(3) <u>Interperetively Interactive</u>

Production Order Release: as a new order is forced into production to satisfy an urgent requirement the two most critical concerns of an inventory planner are (1) "have I enough inventory of each of the required components?" and (2) "What have I just done to the shop?". Neither are easy questions, and certainly not easy answers. They involve research but it is important enough to know these answers, that a system should handle the entire release in interactive mode, and report back interpretively the results. An example of such a result would be the system returning the message, "The release of the total order will cause shortages. However halve the order quantity and we should be OK".

As we go through the description of the system's on-line/ interactive features examples of each processing environment will appear; each, I hope demonstrating that the type of processing used is always a compromise between the need for constantlv uo-to-date data, and the overhead associated with processing in full interactive mode.

ENGINEERING CONTROL (see Figure 3)

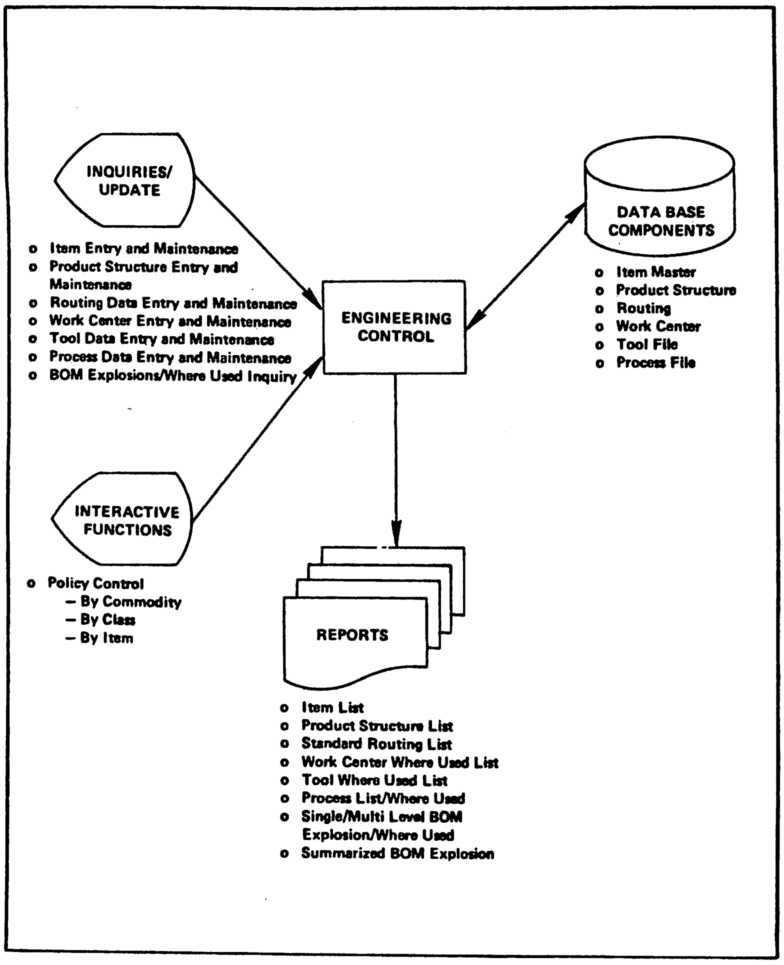

INQUIRIES/ UPDATE
o Item Entry and Maintenance
o Product Structure Entry and Maintenance
o Routing Data Entry and Maintenance
o Work Center Entry and Maintenance
o Tool Data Entry and Maintenance
o Process Data Entry and Maintenance
o BOM Explosions/Where Used Inquiry
ENGINEERING CONTROL
DATA BASE COMPONENTS
o Item Master
o Product Structure
o Routing
o Work Center
o Tool File
o Process File
INTERACTIVE FUNCTIONS
o Policy Control
– By Commodity
– By Class
– By Item
REPORTS
o Item List
o Product Structure List
o Standard Routing List
o Work Center Where Used List
o Tool Where Used List
o Process List/Where Used
o Single/Multi Level BOM Explosion/Where Used
o Summarized BOM Explosion

Figure 3 - Engineering Control Overview

This is the system that maintains the six prime data bases listed earlier - Item Master, Product Structure, Routing, Process, Tool and Work Center. Each should be capable of on-line interactive (not necessarily interpretive) update and inquiry. An example is shown in Figure 4 (Screen IM001)

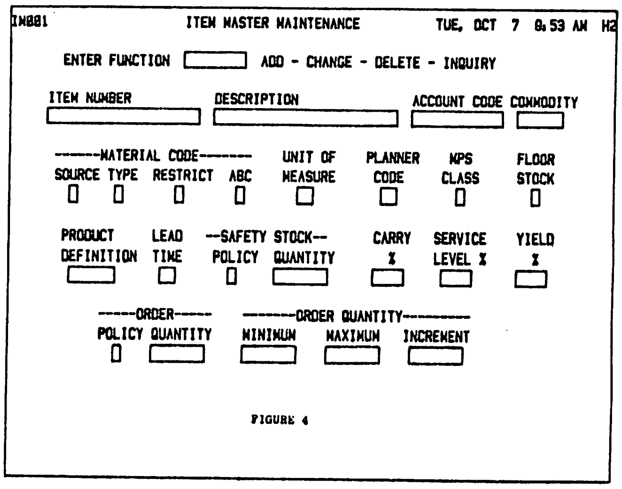

```
IM001                    ITEM MASTER MAINTENANCE            TUE, OCT  7  8:53 AM  H2

     ENTER FUNCTION [    ] ADD - CHANGE - DELETE - INQUIRY

   ITEM NUMBER            DESCRIPTION                ACCOUNT CODE  COMMODITY
   [          ]           [            ]             [        ]    [    ]

   -------MATERIAL CODE-------     UNIT OF    PLANNER    MPS     FLOOR
   SOURCE  TYPE  RESTRICT  ABC     MEASURE     CODE     CLASS    STOCK
     [ ]   [ ]     [ ]     [ ]       [ ]        [ ]      [ ]      [ ]

   PRODUCT     LEAD   --SAFETY STOCK--     CARRY   SERVICE    YIELD
   DEFINITION  TIME   POLICY  QUANTITY       %     LEVEL %      %
   [    ]      [ ]     [ ]    [      ]     [   ]   [   ]      [   ]

        -----ORDER-----      -------ORDER QUANTITY---------
        POLICY  QUANTITY     MINIMUM    MAXIMUM    INCREMENT
         [ ]    [      ]     [      ]   [      ]   [      ]
```

FIGURE 4

This would be the screen through which an Item Master record may be added to the data base (provided it passed some basis validity checks), changed, inquired of, or deleted (provided also that it passed its validity checks).

Altogether there would be seven Item Master screens, some specifically for inquiry purposes only, some for adding or changing other data fields, and some for a combination of the two. However only this screen (IM001) would be used for adding or deleting Item Master records.

Another example of a combination Add-Change-Inquire-Delete screen is shown in Figure 5 (screen PS002). This is used for transactions against the Product Structure and allows specific parent-component links to be updated, etc. The validity checks are more demanding in this case because of the natural structure of the data base. Both Parent and Component Item Numbers must match already existing Item Master records. Otherwise any attempted update will be rejected. The same would be true of the Substitute Item.

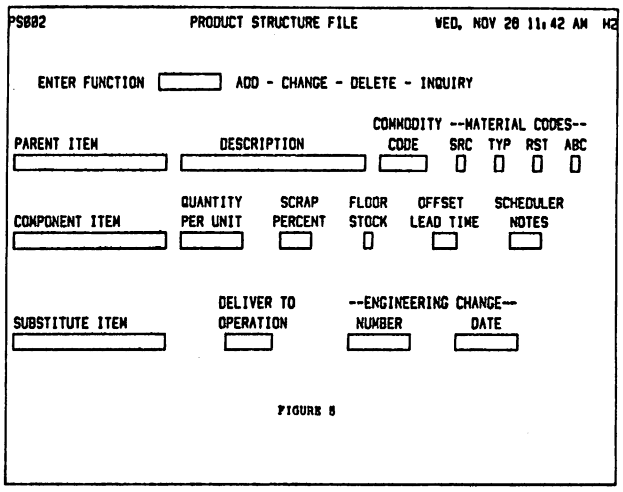

```
PS002               PRODUCT STRUCTURE FILE          WED, NOV 28 11:42 AM  H2

  ENTER FUNCTION [      ] ADD - CHANGE - DELETE - INQUIRY

                                              COMMODITY --MATERIAL CODES--
PARENT ITEM            DESCRIPTION               CODE    SRC  TYP  RST  ABC
[              ]  [                    ]      [     ]    [ ]  [ ]  [ ]  [ ]

                  QUANTITY    SCRAP     FLOOR   OFFSET     SCHEDULER
COMPONENT ITEM    PER UNIT    PERCENT   STOCK   LEAD TIME  NOTES
[              ]  [      ]    [    ]    [ ]     [   ]      [    ]

                      DELIVER TO      --ENGINEERING CHANGE--
SUBSTITUTE ITEM       OPERATION        NUMBER       DATE
[              ]      [     ]         [      ]    [      ]
```

FIGURE 5

Another aspect of interactive processing is also demonstrated by this screen. The six data fields between "Description" and "ABC" are not in fact on the Product Structure data base. They are instead held on the Item Master record for the parent item and are returned by the system for information purposes only at the time of transaction entry.

Two other facets of interactive processing are also applicable to the Product Structure - List-Type inquiries, and "same-as-except" processing. Figure 6 (Screen PS001) shows an example of each. By specifying the Inquiry function and entering a Parent Item Number the system will "return" a list of all associated parent-component link records. It provides therefore the on-line equivalent of a bill-of-material. The second feature here (same-as-except) is an extension of this. Rather than having to specify each parent-component link for a new product structure, an engineer might elect to specify it based upon its similarity with an already existing bill. By using the Inquiry function, the existing bill of material can be listed on the screen. Then having:

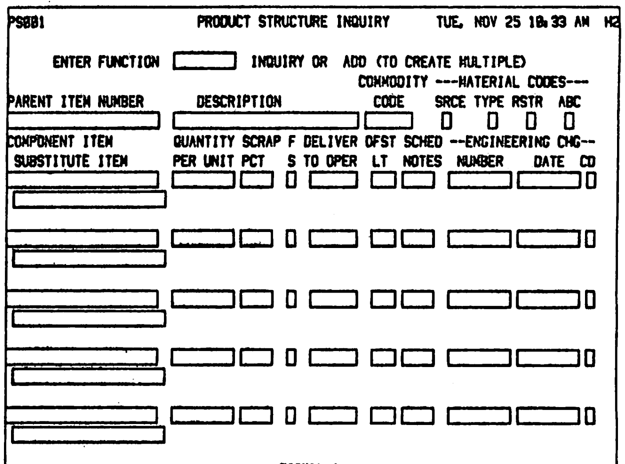

```
PS001                 PRODUCT STRUCTURE INQUIRY      TUE. NOV 25 10:33 AM  H2

    ENTER FUNCTION [     ] INQUIRY OR  ADD (TO CREATE MULTIPLE)
                                          COMMODITY ---MATERIAL CODES---
PARENT ITEM NUMBER      DESCRIPTION          CODE    SRCE TYPE RSTR  ABC

COMPONENT ITEM          QUANTITY SCRAP F DELIVER OFST SCHED --ENGINEERING CHG--
 SUBSTITUTE ITEM        PER UNIT PCT   S TO OPER LT   NOTES  NUMBER    DATE  CD
```

FIGURE 6

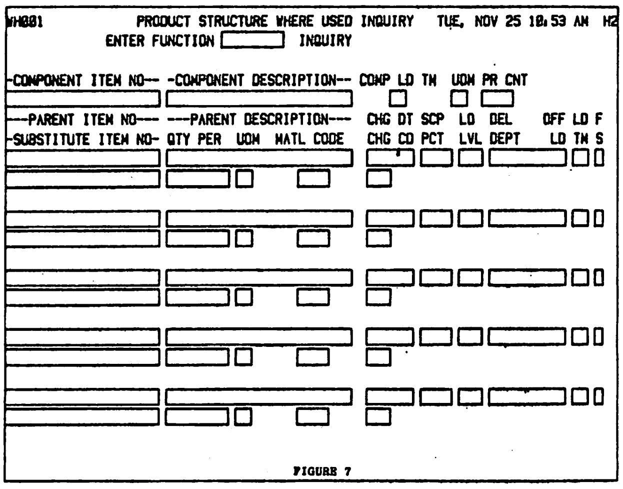
-made any adds/changes/deletes to the existing structure to make it specific to the new

-changed the Parent Item Number to reflect the new structure's parent, and

-changed the function from Inquiry to add

The new product structure can be automatically written to the data base.

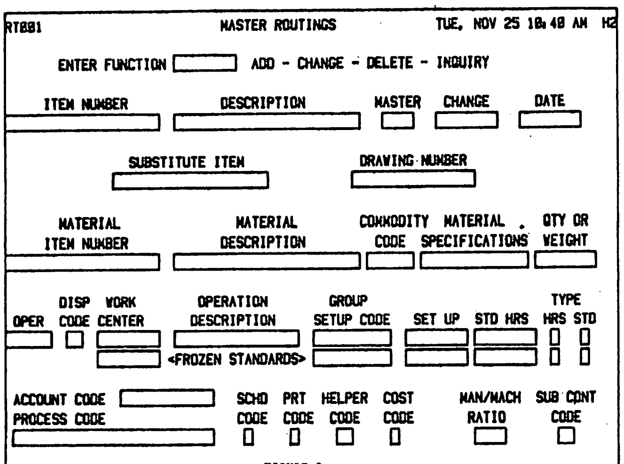
Figure 7 (screen WH001) shows Where-Used feature. Entering a specific, say, purchased part in the Component Item Number field will cause the system to list all parent links associated with it; or, in other words, all manufactured items that have this component in their single-level bill of material. It is very important for an engineer to have access to this type of information - particularly when designing a replacement for an existing part. It effectively tells him which product structure links he should amend to reflect the replacement.



```
WH001          PRODUCT STRUCTURE WHERE USED INQUIRY   TUE, NOV 25 10:53 AM  H2
            ENTER FUNCTION [      ]  INQUIRY

-COMPONENT ITEM NO-- -COMPONENT DESCRIPTION-- COMP LD TM  UOM PR CNT
---PARENT ITEM NO--- ---PARENT DESCRIPTION---  CHG DT SCP  LD  DEL     OFF LD F
-SUBSTITUTE ITEM NO- QTY PER  UOM  MATL CODE   CHG CD PCT  LVL DEPT     LD TM S
```

FIGURE 7

The four manufacturing engineering data bases - Routing, Process, Tool and Work Center would also be available for on-line update and inquiry - specifically for add-change-delete inquiry functions.

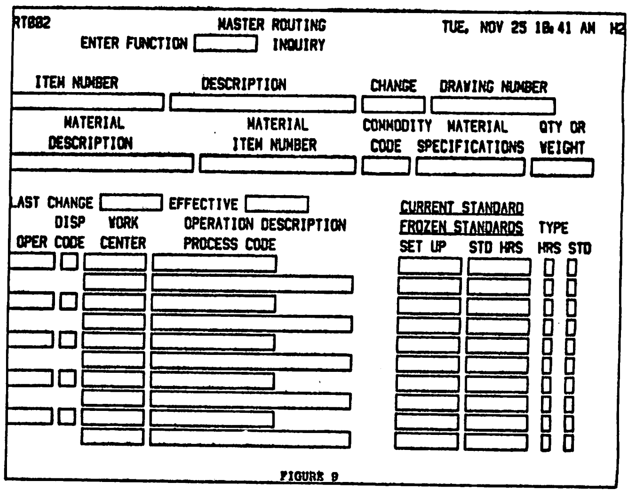


```
RT001                    MASTER ROUTINGS              TUE, NOV 25 10:40 AM  H2

     ENTER FUNCTION [      ] ADD - CHANGE - DELETE - INQUIRY

   ITEM NUMBER           DESCRIPTION        MASTER  CHANGE    DATE

              SUBSTITUTE ITEM          DRAWING NUMBER

    MATERIAL             MATERIAL        COMMODITY  MATERIAL     QTY OR
   ITEM NUMBER          DESCRIPTION        CODE   SPECIFICATIONS WEIGHT

      DISP WORK     OPERATION       GROUP                          TYPE
OPER  CODE CENTER  DESCRIPTION   SETUP CODE   SET UP  STD HRS   HRS STD
                 <FROZEN STANDARDS>

ACCOUNT CODE [        ]  SCHD  PRT  HELPER  COST     MAN/MACH  SUB CONT
PROCESS CODE             CODE  CODE  CODE   CODE      RATIO      CODE
```

FIGURE 8

Figure 8 (screen RT001) shows the equivalent Routing screen. Once again the system "returns" the heading information. The only updateable information is from the "OPER" (Operation Number) field and on. One point worth noting is the "Frozen Standard" line. This would maintain the standards that existed for this operation at the time that Accounting last "froze" their standards. It would be maintained for as long as required to insure correct reporting of variances.

The operation number shown on this screen assumes a 6-character length. This is to allow the system to maintain both prime and alternate routings on the data base. The first four characters would be a normal sequential operation. The last two indicate the operation's prime/ alternate condition.



```
RT002                  MASTER ROUTING                    TUE, NOV 25 10:41 AM  H2
       ENTER FUNCTION [     ] INQUIRY

 ITEM NUMBER           DESCRIPTION            CHANGE  DRAWING NUMBER

   MATERIAL                 MATERIAL        COMMODITY  MATERIAL       QTY OR
  DESCRIPTION              ITEM NUMBER        CODE  SPECIFICATIONS   WEIGHT

LAST CHANGE [     ] EFFECTIVE [     ]          CURRENT STANDARD
      DISP    WORK     OPERATION DESCRIPTION   FROZEN STANDARDS  TYPE
OPER  CODE   CENTER    PROCESS CODE            SET UP   STD HRS  HRS STD
```

FIGURE 9

Figure 9 (Screen RT002) demonstrates the equivalent list-type inquiry.

At this point it is probably appropriate to discuss the functional inter-relationships between these four data bases. Figure 10 (Manufacturing Engineering Data Base Relationships) depicts this. In each operation record on the routing data base there is reference to a work center (mandatory) and a process (optional).

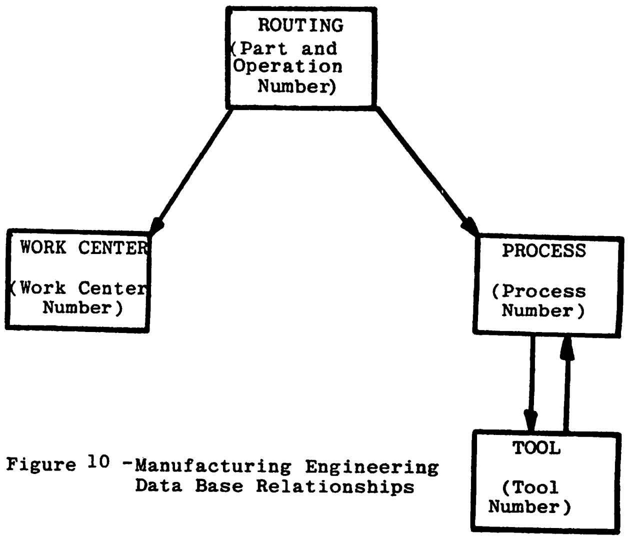
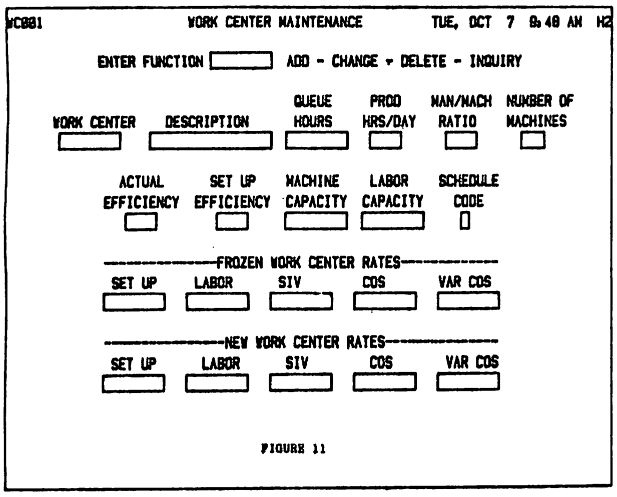


Figure 10 -Manufacturing Engineering Data Base Relationships

These provide the necessary ties-in to these two other data bases. In addition, each process data base record can contain one or more references to the tools required for that process. These references provide the ties-in to the tool data base, as well as providing the necessary where-used references.

Figures 11 through 13 (screens WC001, PM001, and TM001) show some of the data maintained in on-line mode in each of these three additional bases.


WC001 WORK CENTER MAINTENANCE TUE, OCT 7 9:40 AM H2

ENTER FUNCTION ADD - CHANGE - DELETE - INQUIRY

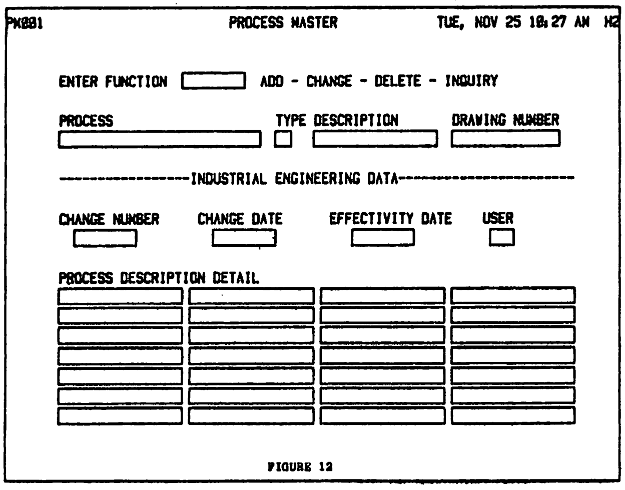
WORK CENTER | DESCRIPTION | QUEUE HOURS | PROD HRS/DAY | MAN/MACH RATIO | NUMBER OF MACHINES

ACTUAL EFFICIENCY | SET UP EFFICIENCY | MACHINE CAPACITY | LABOR CAPACITY | SCHEDULE CODE

FROZEN WORK CENTER RATES: SET UP | LABOR | SIV | COS | VAR COS

NEW WORK CENTER RATES: SET UP | LABOR | SIV | COS | VAR COS

FIGURE 11


PM001 PROCESS MASTER TUE, NOV 25 10:27 AM H2

ENTER FUNCTION ADD - CHANGE - DELETE - INQUIRY

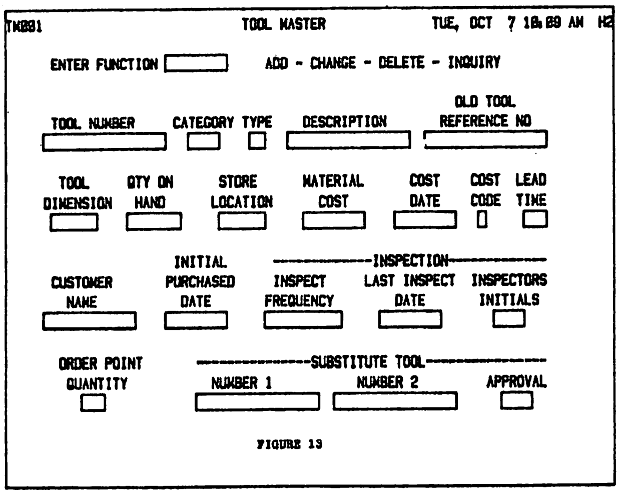
PROCESS | TYPE | DESCRIPTION | DRAWING NUMBER

INDUSTRIAL ENGINEERING DATA

CHANGE NUMBER | CHANGE DATE | EFFECTIVITY DATE | USER

PROCESS DESCRIPTION DETAIL

FIGURE 12


TM001 TOOL MASTER TUE, OCT 7 10:09 AM H2

ENTER FUNCTION ADD - CHANGE - DELETE - INQUIRY

TOOL NUMBER | CATEGORY | TYPE | DESCRIPTION | OLD TOOL REFERENCE NO

TOOL DIMENSION | QTY ON HAND | STORE LOCATION | MATERIAL COST | COST DATE | COST CODE | LEAD TIME

CUSTOMER NAME | INITIAL PURCHASED DATE | INSPECTION: INSPECT FREQUENCY | LAST INSPECT DATE | INSPECTORS INITIALS

ORDER POINT QUANTITY | SUBSTITUTE TOOL: NUMBER 1 | NUMBER 2 | APPROVAL

FIGURE 13

One final comment on the Engineering Control System; should any of these six data bases be "distributed"? Or in other words, is it likely that one of the following is true:

-the data originates in one (plant) location and is used by another?

-the data is used by multi (plants) locations?

Typically the only data to which this may be applicable would be:

-the Item Master Description field

-the Product Structure.

If this is so then consideration must be given to the need for inter-location communication of this information. There are really two options to consider:

-frequent copies of this data being made available (in batch mode) to each appropriate location

-remote on-line access to another distributed processor.

Normally the first would be the least complex from a control point of view, but possibly the least desirable as far as a user is concerned.

MASTER PRODUCTION SCHEDULING (See Figure 14)

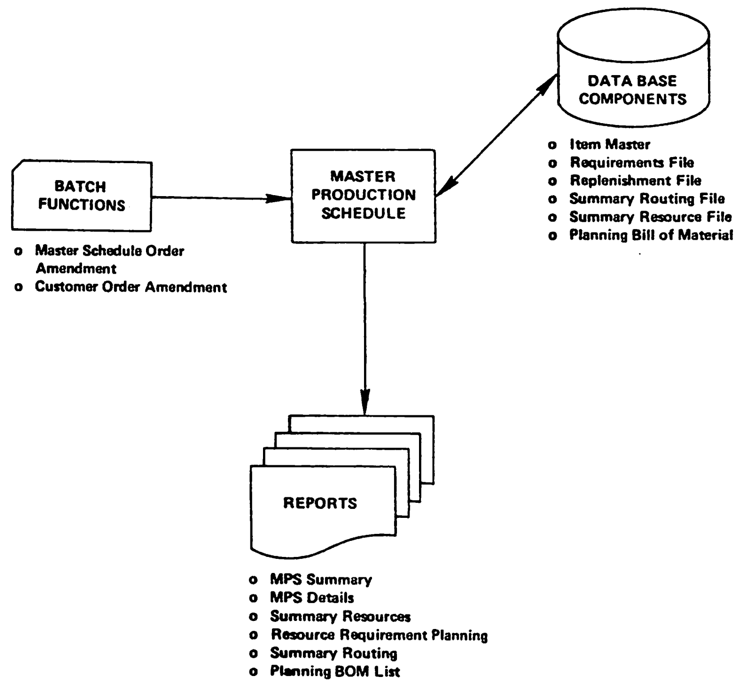


Figure 14 - Master Production Scheduling Overview

As can be seen from the overview this would frequently be viewed as a batch function. This does not mean however that nothing could interact with the Master Schedule in an on-line mode. It is just that neither the development of forecast demands from the Production Plan nor the re-iterative simulation capabilities of Resource Requirements Planning are seen as desirable/practicable on-line interactive functions.

The Master-Schedule-related functions which would be on-line however would be included in the Inventory Control module, just as the same functions would be available for all "inventory" items. These on-line functions are:

-the addition, ammendment or deletion of a production order (or in the case of an MPS item, of the Master Schedule itself).

-the development of lower level component demands as a result of such a change.

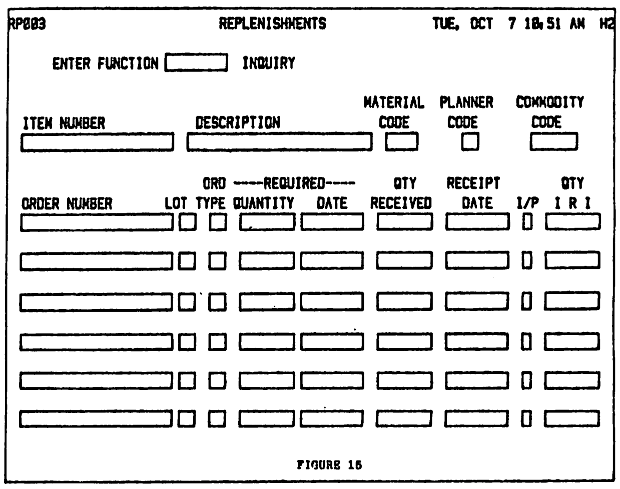
-the entry of customer orders against the production schedule

-the ability to inquire the status of a production order (or schedule)

-the ability to time-phase inventory availability, with specific reference to monitoring the "consumption" of a Master Schedule.

The screens required to satisfy these last two items are shown in Figures 15 and 16.

The first (Figure 15) shows Screen RP003. With the entry of the Inquiry function and the appropriate master scheduled item, the system will return a time

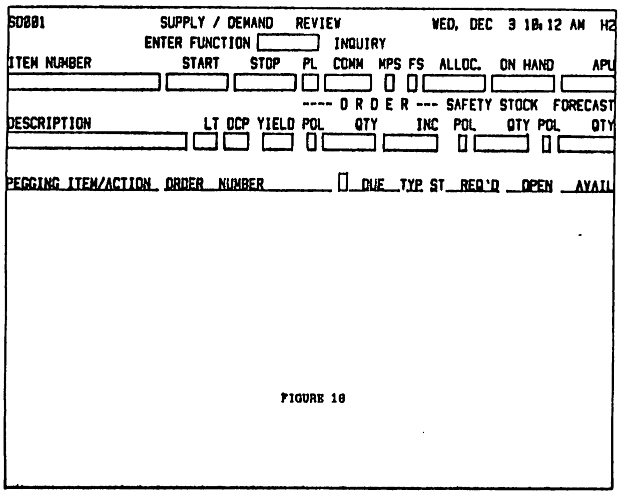


```
RP003                      REPLENISHMENTS                TUE, OCT  7 10:51 AM  H2

   ENTER FUNCTION [    ]  INQUIRY

                                              MATERIAL  PLANNER  COMMODITY
ITEM NUMBER          DESCRIPTION                CODE      CODE      CODE

                        ORD ----REQUIRED----   QTY     RECEIPT        QTY
ORDER NUMBER       LOT TYPE QUANTITY  DATE   RECEIVED   DATE    I/P  I R I
```

FIGURE 15

phased list of all scheduled production for that item, and the individual status of each production "batch.

Figure 16 shows Screen SD001. The same entry is required (Inquiry plus Item Number) but, additonally, a date range (start/stop) can be specified. This allows the Master Scheduler to concentrate only upon projected activity during the time period under consideration. The first two lines provide details from the part's Item Master data base, one element of which is the part's current balance on hand (if any). This is the starting point of the time-phasing. Each projected activity (either production or usage) is listed in due date sequence and the effect of such activity shown under the heading AVAIL (available).



```
SD001                SUPPLY / DEMAND   REVIEW            WED, DEC  3 10:12 AM  H2
                 ENTER FUNCTION [    ]  INQUIRY
ITEM NUMBER               START    STOP   PL  COMM  MPS FS  ALLOC.  ON HAND     APU
[              ]  [      ] [      ] [ ] [     ] [] [] [      ] [      ] [      ]
                                   ---- O R D E R --- SAFETY STOCK  FORECAST
DESCRIPTION               LT DCP YIELD POL    QTY      INC  POL    QTY POL     QTY
[                  ] [  ] [  ] [   ] [] [     ] [      ] [] [     ] [] [     ]

PEGGING ITEM/ACTION  ORDER  NUMBER        [] DUE  TYP ST  REQ'D  OPEN   AVAIL
```

FIGURE 16

INVENTORY CONTROL (see Figure 17)

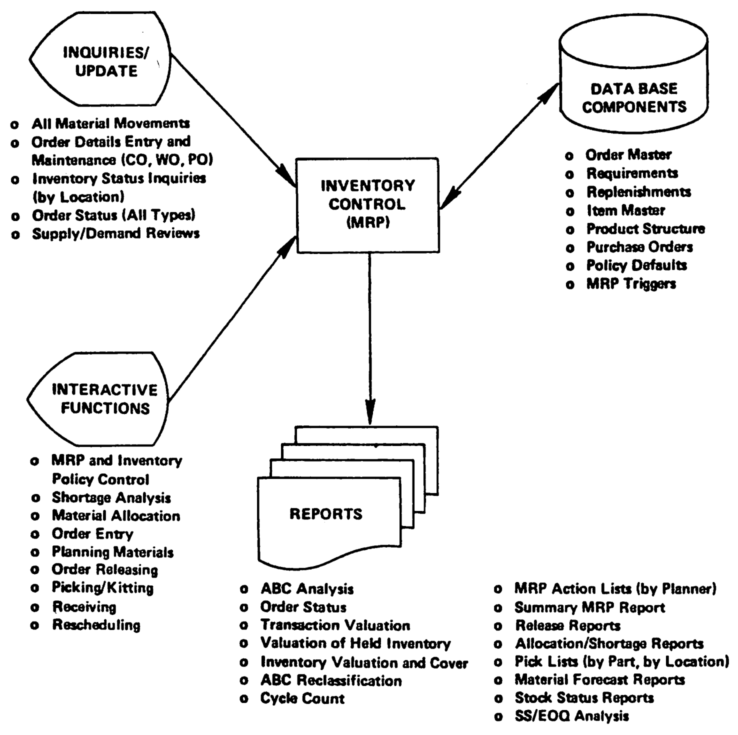


Figure 17 - Inventory Control Overview

As can be seen from the overview this part of the system is highly interactive in nature. Let us review some of these functions under the same headings as we used earlier-planning, releasing, expediting, recording, accounting and management.

(1) Planning: The ability of the system to develop replenishment plans (both purchasing and production) for each recognized inventory item is the nucleus of the whole system's operation. The

technique used - MRP- is a method of developing a time-phased picture of each item's expected usage and suggesting the appropriate replenishment plan to avoid potential stock outs. This type of review is conducted in a logical sequence dictated by each part's position in the product structure. The highest level items are tackled first so that their replenishment plans can be translated into projected component demands prior to that level itself being reviewed by MRP. In the simplest terms the system recognizes two record types - supplies and demands. Each of these are split again:

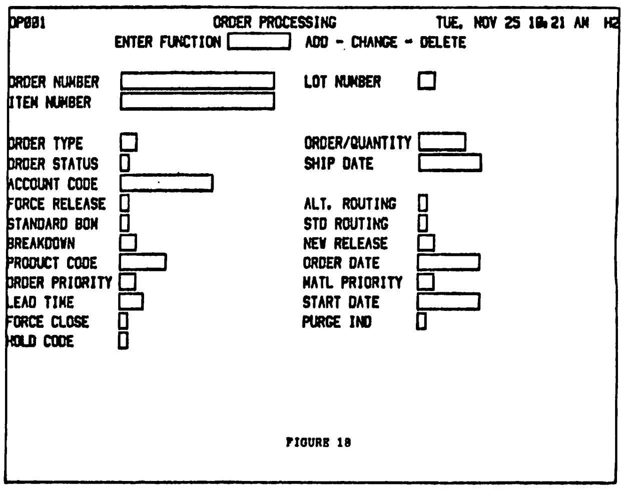
-supplies:	Production orders
	Purchase orders
-demands:	customer orders
	forecast orders
	dependent demands

The latter two are typically created automatically by the system - forecast orders from a forecasting algorithm, dependent demands from the explosion of production orders created at the next higher level in the product structure. The other three would be input to the system in on-line mode:

Production orders:	through screen OP001 (see Figure 18)
Purchase orders:	through screens PO001 and PO003 (see later purchasing discussion)
Customer orders:	through screens OP001 and OM001 (See Figures 18 and 19)

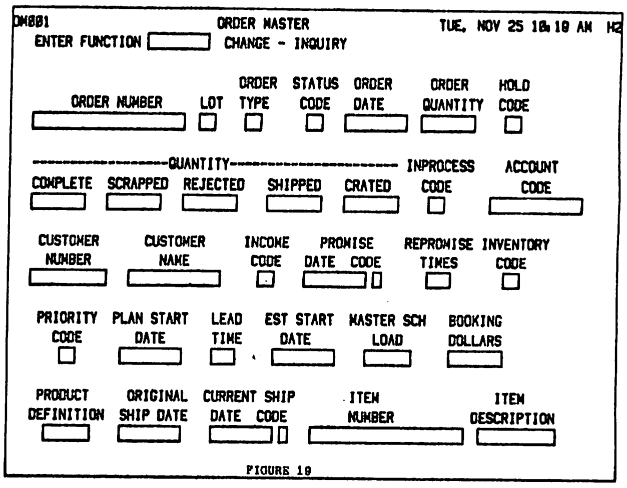
Two points are worth noting regarding the OP001 screen. Firstly that the order type field dictates whether the order being entered is a production or a customer order - or a <u>combination of both</u>. This feature allows for special customer orders to be entered that are for non-inventory items. They do not have to match a record on the Item Master; they are planned for shipment directly from work in process.

The second point relates to the field "Standard



```
OP001                    ORDER PROCESSING              TUE, NOV 25 10:21 AM  H2
               ENTER FUNCTION [    ]  ADD - CHANGE - DELETE

ORDER NUMBER   [            ]      LOT NUMBER       [ ]
ITEM NUMBER    [            ]

ORDER TYPE     [ ]                 ORDER/QUANTITY   [   ]
ORDER STATUS   [ ]                 SHIP DATE        [     ]
ACCOUNT CODE   [       ]
FORCE RELEASE  [ ]                 ALT. ROUTING     [ ]
STANDARD BOM   [ ]                 STD ROUTING      [ ]
BREAKDOWN      [ ]                 NEW RELEASE      [ ]
PRODUCT CODE   [   ]               ORDER DATE       [     ]
ORDER PRIORITY [ ]                 MATL PRIORITY    [ ]
LEAD TIME      [  ]                START DATE       [     ]
FORCE CLOSE    [ ]                 PURGE IND        [ ]
HOLD CODE      [ ]
```

FIGURE 18



```
OM001                    ORDER MASTER                  TUE, NOV 25 10:10 AM  H2
  ENTER FUNCTION [    ]  CHANGE - INQUIRY

                                ORDER  STATUS  ORDER    ORDER     HOLD
        ORDER NUMBER      LOT   TYPE   CODE    DATE     QUANTITY  CODE
  [              ]        [ ]   [ ]    [ ]     [    ]   [    ]    [ ]

  ------------------QUANTITY----------------------  INPROCESS   ACCOUNT
  COMPLETE  SCRAPPED  REJECTED  SHIPPED  CRATED     CODE        CODE
  [    ]    [    ]    [    ]    [    ]   [    ]     [ ]         [       ]

  CUSTOMER    CUSTOMER     INCOME   PROMISE      REPROMISE  INVENTORY
   NUMBER       NAME       CODE     DATE  CODE   TIMES      CODE
  [      ]   [       ]     [ ]      [     ][ ]   [ ]        [ ]

  PRIORITY  PLAN START  LEAD   EST START  MASTER SCH  BOOKING
    CODE      DATE      TIME     DATE       LOAD      DOLLARS
    [ ]      [    ]     [ ]     [    ]     [    ]     [    ]

  PRODUCT     ORIGINAL   CURRENT SHIP      ITEM             ITEM
  DEFINITION  SHIP DATE  DATE  CODE        NUMBER           DESCRIPTION
  [    ]      [    ]     [    ][ ]    [            ]     [      ]
```

FIGURE 19

BOM". The entry of an "N" (No) here (whether for a special customer order or for a non-standard production order) allows the system, using another Order Processing screen to present the same-as-except option to the inventory planner. Using this feature, a planner can either override the system-suggested bill-of-material, or simply substitute a part, or change a "Quantity per".

(2) Releasing: One of the system's most powerful "interpretively interactive" functions would be covered by screen OP004 (See Figure 20). This would allow a user to:

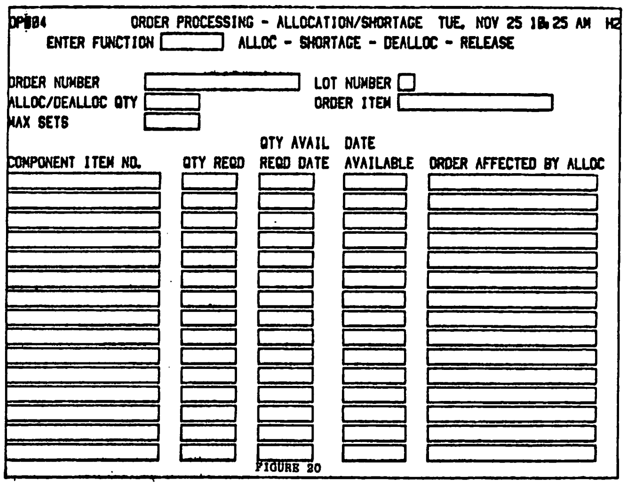
-check for potential shortages prior to the release of an order

-force-allocate inventory to an order

-force-de-allocate inventory from an order

-force-release an order.

The Shortage function amply shows this ability. With the single entry of an Order Number (typically for an existing production order) the system:



```
OP004          ORDER PROCESSING - ALLOCATION/SHORTAGE  TUE. NOV 25 10:25 AM  H2
    ENTER FUNCTION [     ]  ALLOC - SHORTAGE - DEALLOC - RELEASE

ORDER NUMBER        [            ]   LOT NUMBER [ ]
ALLOC/DEALLOC QTY   [    ]           ORDER ITEM [            ]
MAX SETS            [    ]
                                QTY AVAIL  DATE
COMPONENT ITEM NO.     QTY REQD REQD DATE  AVAILABLE  ORDER AFFECTED BY ALLOC
```

FIGURE 20

-ascertains the component items that need to be available.

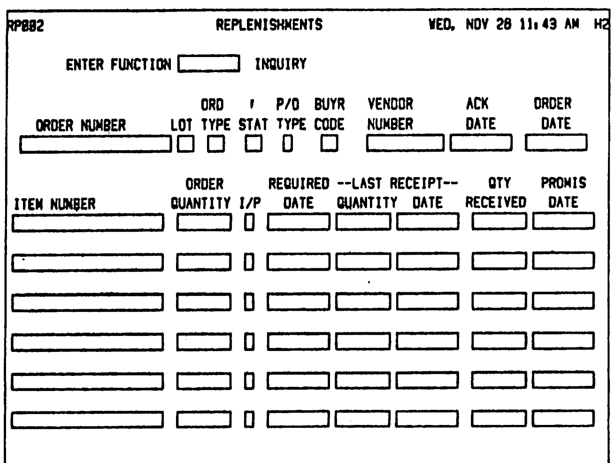
-checks for any potential shortage condition that may exist on these items.

-displays on the screen these potential shortages.

-ascertains the date on which these shortages should go away if all goes according to plan.

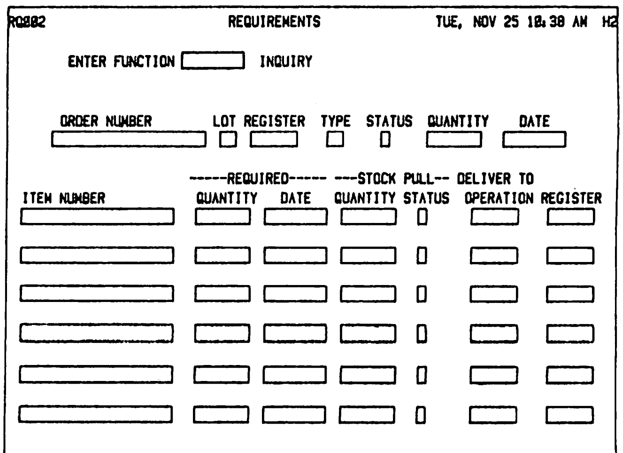
-calculates the maximum partial quantity (if any) that this order could be re-leased for while still avoiding the shortage.

(3) <u>Expediting</u>: the system would provide total on-line reference capability for existing production, purchase and customer orders (see Figures 21 and 22 - screens RP002 and RQ002).



```
RP002                    REPLENISHMENTS              WED, NOV 28 11:43 AM  H2

      ENTER FUNCTION [     ]  INQUIRY

                            ORD   '   P/O  BUYR  VENDOR       ACK      ORDER
   ORDER NUMBER         LOT TYPE STAT TYPE CODE  NUMBER       DATE     DATE

                     ORDER       REQUIRED --LAST RECEIPT--   QTY      PROMIS
 ITEM NUMBER         QUANTITY I/P  DATE   QUANTITY  DATE   RECEIVED   DATE
```

FIGURE 21

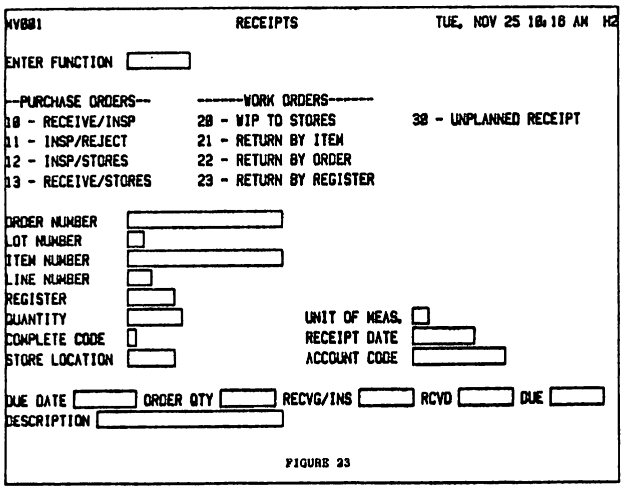


```
RQ002                      REQUIREMENTS               TUE, NOV 25 10:38 AM  H2

      ENTER FUNCTION [     ]  INQUIRY

   ORDER NUMBER        LOT REGISTER  TYPE  STATUS  QUANTITY   DATE

                       -----REQUIRED----- ---STOCK PULL-- DELIVER TO
 ITEM NUMBER           QUANTITY   DATE    QUANTITY STATUS OPERATION REGISTER
```

FIGURE 22

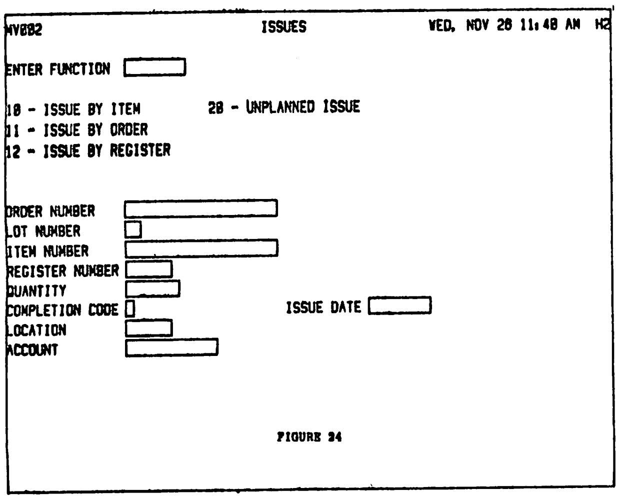
The first shows the status of the order itself, the second shows the status of the components.

(4) Recording: All receipts and issues, would be on-line transactions as would the inquiry on an item's balance on hand and its one or more stocking locations. The three screens (MV001, MV002, and IM005) are shown in Figures 23 through 25. A number of points are worth noting regarding the two movement



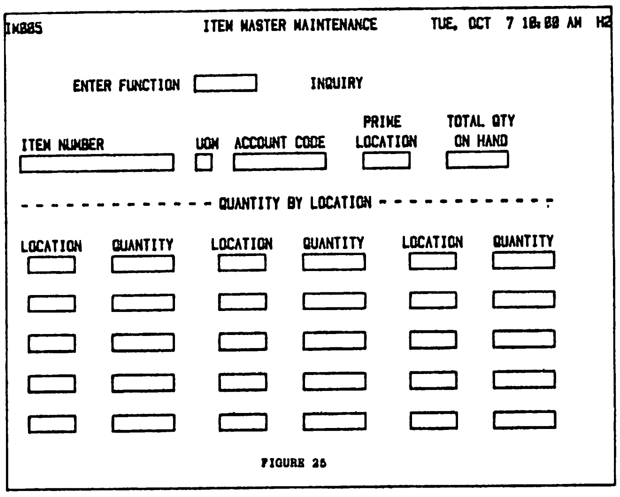
```
MV001                    RECEIPTS              TUE, NOV 25 10:10 AM  H2

ENTER FUNCTION [      ]

--PURCHASE ORDERS--      -------WORK ORDERS-------
10 - RECEIVE/INSP        20 - WIP TO STORES        30 - UNPLANNED RECEIPT
11 - INSP/REJECT         21 - RETURN BY ITEM
12 - INSP/STORES         22 - RETURN BY ORDER
13 - RECEIVE/STORES      23 - RETURN BY REGISTER

ORDER NUMBER   [              ]
LOT NUMBER     [ ]
ITEM NUMBER    [              ]
LINE NUMBER    [  ]
REGISTER       [    ]
QUANTITY       [     ]                UNIT OF MEAS. [ ]
COMPLETE CODE  [ ]                    RECEIPT DATE  [     ]
STORE LOCATION [    ]                 ACCOUNT CODE  [        ]

DUE DATE [     ] ORDER QTY [     ] RECVG/INS [     ] RCVD [     ] DUE [     ]
DESCRIPTION [                ]
```

FIGURE 23



```
MV002                  ISSUES           WED, NOV 26 11:48 AM  H2

ENTER FUNCTION [      ]

10 - ISSUE BY ITEM         20 - UNPLANNED ISSUE
11 - ISSUE BY ORDER
12 - ISSUE BY REGISTER

ORDER NUMBER      [              ]
LOT NUMBER        [ ]
ITEM NUMBER       [              ]
REGISTER NUMBER   [    ]
QUANTITY          [     ]
COMPLETION CODE   [ ]                 ISSUE DATE [      ]
LOCATION          [    ]
ACCOUNT           [        ]
```

FIGURE 24



```
IM005          ITEM MASTER MAINTENANCE      TUE, OCT  7 10:00 AM  H2

      ENTER FUNCTION [      ]        INQUIRY

                                                PRIME       TOTAL QTY
 ITEM NUMBER                 UOM  ACCOUNT CODE   LOCATION    ON HAND
 [              ]            [ ]  [         ]   [     ]     [      ]

 - - - - - - - - - - - - - QUANTITY BY LOCATION - - - - - - - - - - - -

 LOCATION  QUANTITY   LOCATION  QUANTITY   LOCATION  QUANTITY
 [     ]   [      ]   [     ]   [      ]   [     ]   [      ]
 [     ]   [      ]   [     ]   [      ]   [     ]   [      ]
 [     ]   [      ]   [     ]   [      ]   [     ]   [      ]
 [     ]   [      ]   [     ]   [      ]   [     ]   [      ]
 [     ]   [      ]   [     ]   [      ]   [     ]   [      ]
```

FIGURE 25

screens.

-unless a transaction is catagorically stated as unplanned (functions 20 and 30) it will always be validated against an existing supply or demand record. If one cannot be found, or there is some other discrepancy, the transaction is rejected

-a completion code can be set if the transaction is to close-short the supply or demand record.

-if a store location is not specified it will assume the part's prime location.

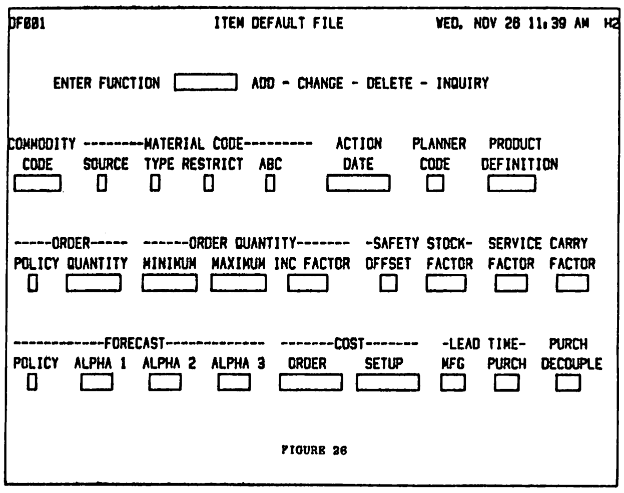
-the issue transaction can be "interpretively interactive". Functions 11 and 12 will automatically create issue transactions by "implication".

-receipts against a purchase order can be two-staged - firstly into inspection, then into stores.

(5) Accounting: Each Item Master record contains an account code. A debit or credit transaction is created (at full frozen standard costs) wherever a receipt or issue (respectively) is processed.

The contra-entry is also created based upon the transaction's own account code. This may be specified in the transaction itself (for instance, for a scrap transaction) or in the original order record against which it is being processed.
The system also keeps track of the inventory value and analyzes this by product line, commodity code, etc.

(6) Management: One of the most powerful features in an inventory control system is the ability to drive the inventory management policies from "default" records. This enables the inventory planner, (as can be seen from figure 26) to specify his safety stock, order quantities etc at a "generic" level (possibly commodity code) and with one small change, radically affect the individual replenishment plans being developed for each item within that commodity code. The end effect obviously is to impact the projected inventory levels by reflecting changes in management policy.



```
DF001                     ITEM DEFAULT FILE               WED. NOV 28 11:39 AM  H2

    ENTER FUNCTION [      ] ADD - CHANGE - DELETE - INQUIRY

COMMODITY --------MATERIAL CODE---------    ACTION    PLANNER  PRODUCT
  CODE    SOURCE  TYPE RESTRICT  ABC        DATE      CODE     DEFINITION
 [    ]    [ ]    [ ]    [ ]     [ ]      [       ]   [  ]     [     ]

-----ORDER-----  ------ORDER QUANTITY-------  -SAFETY STOCK-  SERVICE CARRY
POLICY QUANTITY  MINIMUM  MAXIMUM INC FACTOR  OFFSET  FACTOR  FACTOR  FACTOR
 [ ]   [     ]   [     ]  [     ]  [    ]      [  ]   [    ]  [    ]  [    ]

------------FORECAST-------------  -------COST-------  -LEAD TIME-  PURCH
POLICY  ALPHA 1  ALPHA 2  ALPHA 3   ORDER     SETUP    MFG   PURCH  DECOUPLE
 [ ]    [   ]    [   ]    [   ]    [      ]  [      ]  [  ]  [  ]   [  ]
```

FIGURE 26

MANUFACTURING CONTROL (See Figure 27)

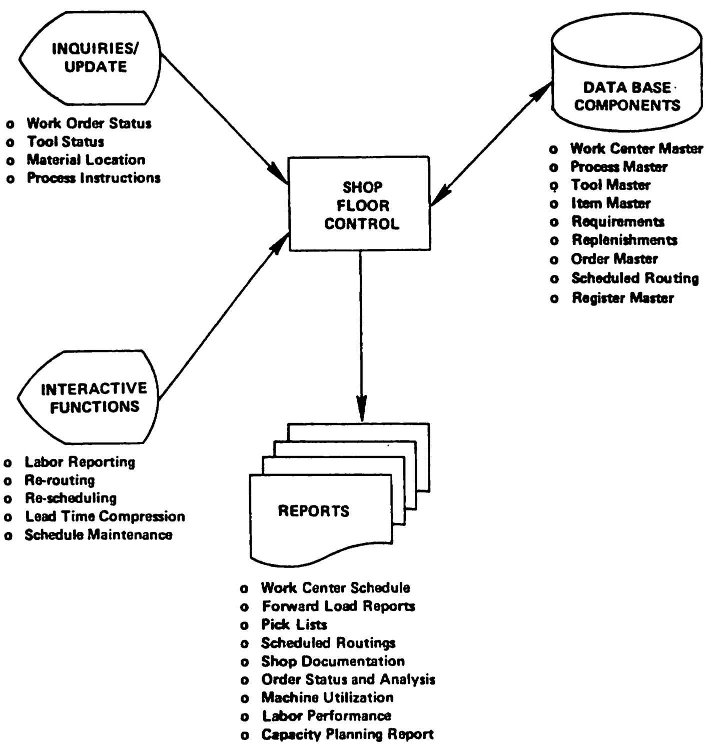


Figure 27 Manufacturing Control Overview

Four of the earlier-listed functions of the Manufacturing Control system should be on-line and/or interactive. They are: Order Release, Order Scheduling, Shop Floor Reporting and Order Expediting (or Status Reporting). The other two, Analysis and Capacity Requirements Planning, would normally be pure batch.

In interactive mode an order's release and the development of its operation-by-operation schedule should be simultaneous. This would be instigated by the OP001 screen described earlier --

specifically by the Force Release indicator. The end effect would be a completely-developed back schedule of all work based upon

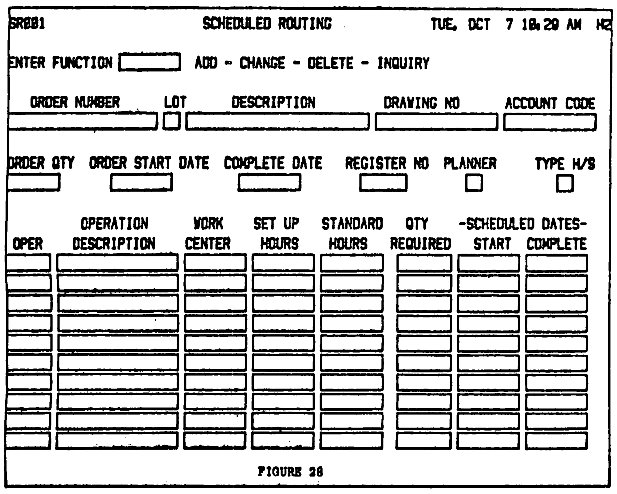
-the order's due-for-complete date

-the order's size

-the routing used

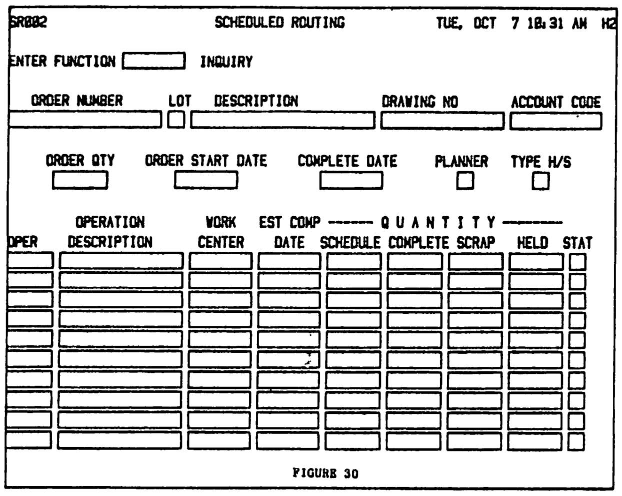
-the projected move and queue times betweens operations

It should be available for on-line Inquiry - see Figure 28, screen SR001.



```
SR001                 SCHEDULED ROUTING              TUE, OCT  7 10:29 AM  H2

ENTER FUNCTION [    ]  ADD - CHANGE - DELETE - INQUIRY

  ORDER NUMBER    LOT    DESCRIPTION        DRAWING NO     ACCOUNT CODE

ORDER QTY  ORDER START DATE  COMPLETE DATE  REGISTER NO  PLANNER   TYPE H/S

          OPERATION     WORK   SET UP  STANDARD   QTY    -SCHEDULED DATES-
OPER     DESCRIPTION   CENTER  HOURS    HOURS   REQUIRED   START  COMPLETE
```

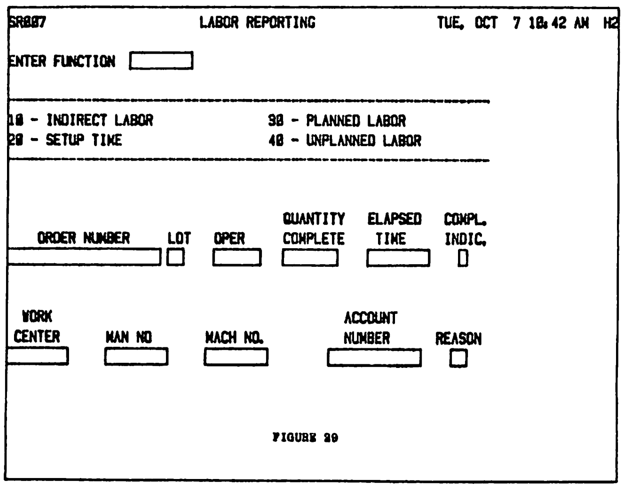
FIGURE 28



```
SR002                 SCHEDULED ROUTING              TUE, OCT  7 10:31 AM  H2

ENTER FUNCTION [    ]  INQUIRY

  ORDER NUMBER    LOT  DESCRIPTION          DRAWING NO      ACCOUNT CODE

   ORDER QTY   ORDER START DATE   COMPLETE DATE    PLANNER   TYPE H/S

        OPERATION       WORK  EST COMP ----- Q U A N T I T Y -------
OPER   DESCRIPTION     CENTER   DATE  SCHEDULE COMPLETE SCRAP   HELD  STAT
```

FIGURE 30

As labor transactions are posted - see Figure 29, screen SR007- each operations status should be maintained and available for on-line inquiry see Figure 30, screen SR002.



```
SR007                 LABOR REPORTING                TUE, OCT  7 10:42 AM  H2

ENTER FUNCTION [    ]
------------------------------------------------------------------
10 - INDIRECT LABOR                   30 - PLANNED LABOR
20 - SETUP TIME                       40 - UNPLANNED LABOR
------------------------------------------------------------------

                                       QUANTITY   ELAPSED   COMPL.
   ORDER NUMBER     LOT    OPER        COMPLETE    TIME     INDIC.

 WORK                                         ACCOUNT
CENTER      MAN NO       MACH NO.             NUMBER     REASON
```

FIGURE 29

COST CONTROL (See Figures 31 and 32)

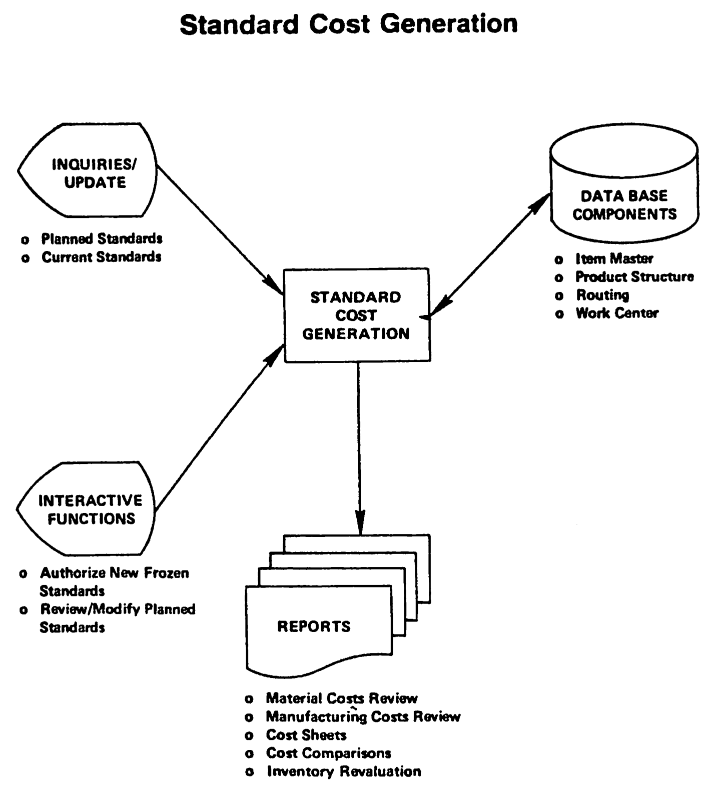


Figure 31 - Cost Control Overview (1)

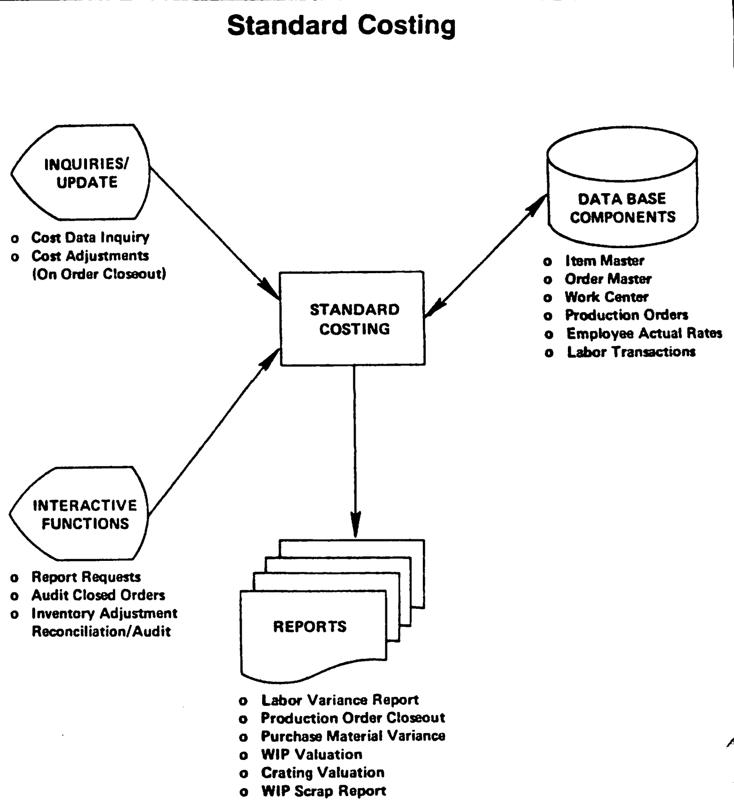
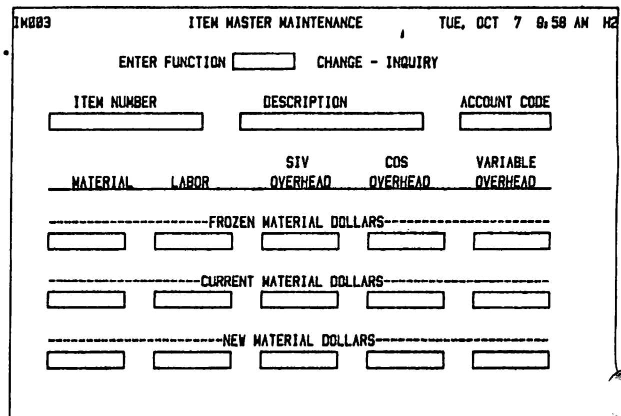


Figure 32 - Cost Control Overview (2)

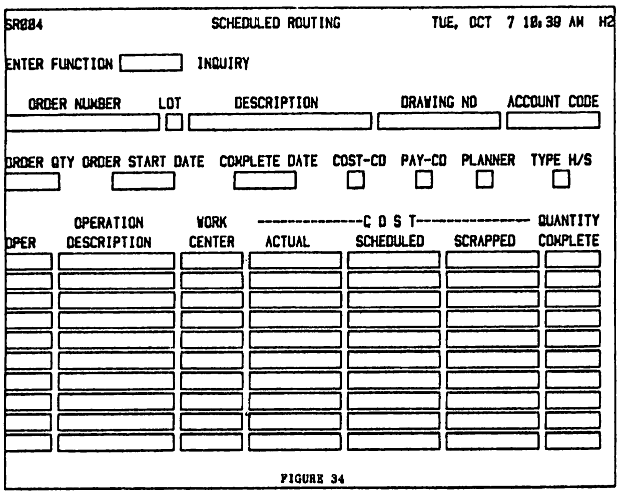
The two overviews describe most of the required system features. Only one screen relates to the cost generation function and that is shown in Figure 33 (Screen IM003). The Change function allows the interactive update of the New and Current costs; it does not allow for that of Frozen. These have either to be



```
IM003              ITEM MASTER MAINTENANCE          TUE. OCT  7  8:58 AM H2

          ENTER FUNCTION [    ]  CHANGE - INQUIRY

   ITEM NUMBER              DESCRIPTION                    ACCOUNT CODE
[              ]      [              ]                  [        ]

                                    SIV          COS        VARIABLE
   MATERIAL     LABOR           OVERHEAD     OVERHEAD      OVERHEAD

------------------------FROZEN MATERIAL DOLLARS------------------------
[      ]      [      ]      [      ]      [      ]      [      ]

------------------------CURRENT MATERIAL DOLLARS-----------------------
[      ]      [      ]      [      ]      [      ]      [      ]

-------------------------NEW MATERIAL DOLLARS--------------------------
[      ]      [      ]      [      ]      [      ]      [      ]
```

FIGURE 33

"rolled" from the Current or New costs, or to be amended in batch mode. Three screens relate to the Standard Costing function; each being an inquiry about an order's costs to date. they are "hierarchical" in design - one provides costs for the order as a whole, another a summary of costs by operation, and the third the back-up detail of an individual operation's costs. The second such screen (SR004) is shown as an example in Figure 34.



```
SR004                    SCHEDULED ROUTING              TUE, OCT  7 10:39 AM  H2

ENTER FUNCTION [    ]  INQUIRY

   ORDER NUMBER      LOT      DESCRIPTION          DRAWING NO     ACCOUNT CODE

ORDER QTY ORDER START DATE  COMPLETE DATE  COST-CO  PAY-CO  PLANNER  TYPE H/S

          OPERATION       WORK   ---------------C O S T--------------- QUANTITY
OPER     DESCRIPTION     CENTER    ACTUAL     SCHEDULED    SCRAPPED    COMPLETE
```

FIGURE 34

PURCHASE ORDER CONTROL (See Figure 35)

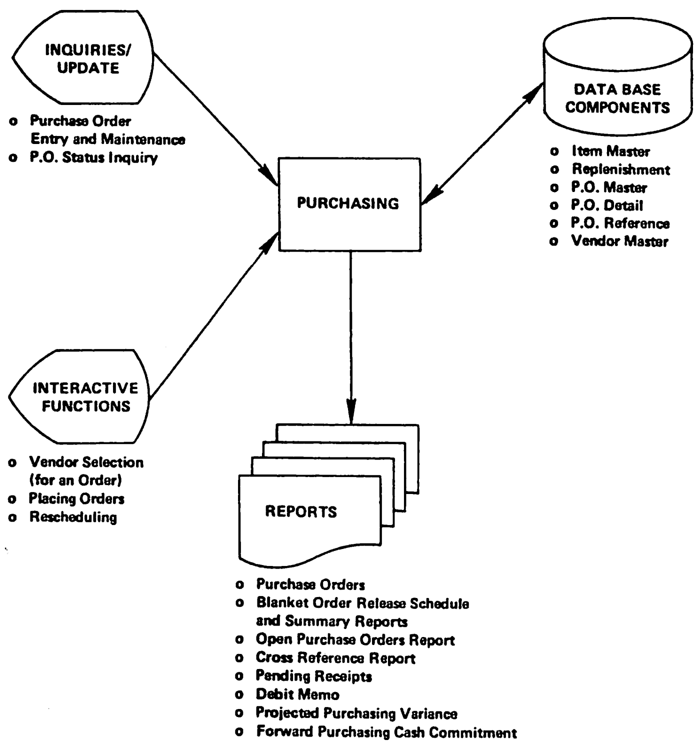
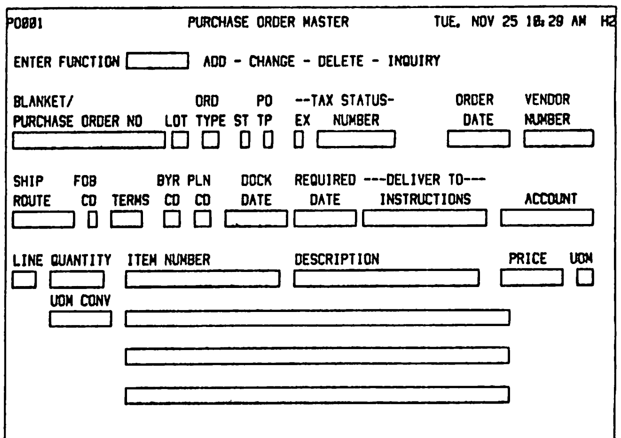


Figure 35 - Purchasing Systems Overview

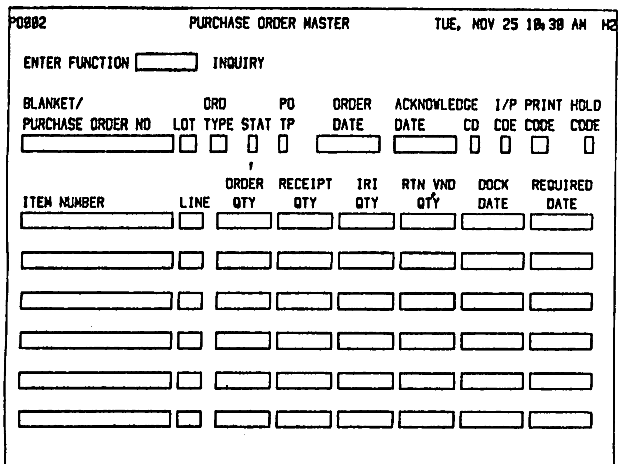
As was stated earlier this system component would provide help for the purchasing department in their placing and expediting of purchase orders. In all other respects -- purchase order printing, invoice reconciliation, vendor analysis, etc- the system would perform in batch mode. Figures 36 through 38 show the screens that would be used


```
PO001                 PURCHASE ORDER MASTER          TUE. NOV 25 10:29 AM  H2

ENTER FUNCTION [    ]  ADD - CHANGE - DELETE - INQUIRY

BLANKET/                    ORD     PO  --TAX STATUS-       ORDER    VENDOR
PURCHASE ORDER NO     LOT  TYPE  ST  TP  EX   NUMBER          DATE     NUMBER

SHIP      FOB          BYR PLN    DOCK    REQUIRED ---DELIVER TO---
ROUTE      CD  TERMS   CD  CD     DATE      DATE     INSTRUCTIONS          ACCOUNT

LINE QUANTITY  ITEM NUMBER        DESCRIPTION                   PRICE   UOM
     UOM CONV
```

FIGURR 36

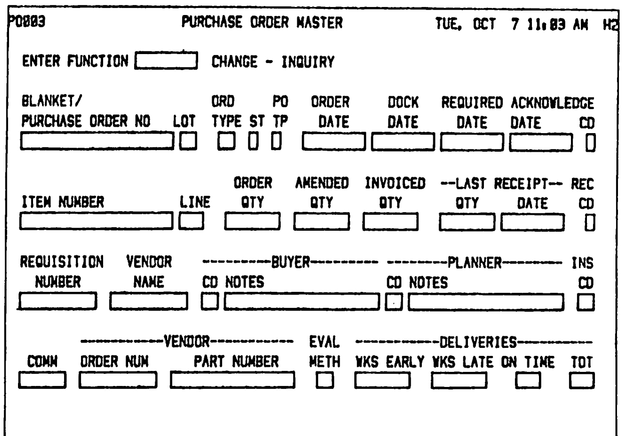
to iniate an order (screens P0001 and P0003) and to inquire as to its current status (P0002 and again P0003).

The system should allow for purchase orders to be placed both for discoete quanties and as a series of blanket order releases.



```
P0002                PURCHASE ORDER MASTER          TUE, NOV 25 10:30 AM  H2

ENTER FUNCTION [    ]  INQUIRY

BLANKET/                      ORD        PO    ORDER   ACKNOWLEDGE  I/P PRINT HOLD
PURCHASE ORDER NO        LOT TYPE STAT  TP     DATE    DATE      CD  CDE CODE  CODE

                                 ORDER  RECEIPT  IRI   RTN VND  DOCK   REQUIRED
ITEM NUMBER              LINE    QTY     QTY     QTY     QTY    DATE     DATE
```

FIGURE 37



```
P0003                PURCHASE ORDER MASTER          TUE, OCT  7 11:03 AM  H2

ENTER FUNCTION [    ]  CHANGE - INQUIRY

BLANKET/                     ORD     PO   ORDER    DOCK   REQUIRED ACKNOWLEDGE
PURCHASE ORDER NO       LOT  TYPE ST TP    DATE    DATE     DATE    DATE     CD

                               ORDER  AMENDED  INVOICED  --LAST RECEIPT-- REC
ITEM NUMBER             LINE    QTY     QTY      QTY      QTY      DATE   CD

REQUISITION  VENDOR   ---------BUYER---------  --------PLANNER--------  INS
  NUMBER      NAME    CD NOTES                 CD NOTES                  CD

     -----------VENDOR-----------   EVAL   -----------DELIVERIES-----------
COMM   ORDER NUM     PART NUMBER    METH   WKS EARLY WKS LATE  ON TIME   TOT
```

FIGURE 38

SUMMARY

So that's it; an on-line interactive system capable of handling a distributed environment. Obviously it has to be far more capable than just this brief overview can depict. However, much of the remainder of the system would almost follow by implication from the nucleus discussed here.

Let us finally return to the more conceptual requirements that we reviewed earlier and hopefully agreed.

(1) everything hangs upon management's ability to implement new techniques of management control.

(2) the new "techniques" are nothing more than a logical extension of some very basic management practices. The difference is that these have to be formalized in order that they may be "distributed" and implemented.

(3) in a distributed environment the majority of control problems arise in the interfaces not in the manufacturing plants themselves.

(4) a good system will not only be dependent upon but also positively encourage, Evaluation, Feedback and Commitment.

(5) as representatives of a service organization within our own company it is <u>our</u> responsibility to help our management understand these aspects of a distributed environment.

NOTE

The screen formats reproduced in this paper represent a selection of those available in Martin Marietta Data System's MAS-H application system. For further information concerning this system please contact:

Richard M. Nemesson
Director of Marketing
Martin Marietta Data Systems
6301 Ivy Lane
Suite 300
Greenbelt, Md. 20770

TRANSACTION LOGGING AND ITS USES

By Dennis Heidner

Boeing Aerospace Company

ABSTRACT

For some time data-base users have been concerned about the integrity of their data-bases and methods to prevent them from being corrupted. Another concern is performance measurement. When H-P introduced MIT-1918, they also introduced "Transaction Logging". Transaction logging is intended to provide a means of repairing data-bases which are either damaged or are suspected of being so. There are however many additional benefits to be derived from transaction logging including, automatic audit trails, historical records of the data-base users, and information on the data-base performance.
The purpose of the paper is to discuss the basic concepts of transaction logging, its benefits, and its drawbacks.

Various logging schemes, such as long logical blocks, concurrent transactions, multiple IMAGE data-bases, and user-written application programs are examined. Several different data-base logging cycles and H-P recommended recovery procedures are discussed, and a method of recovering and synchronizing multiple data-bases is proposed.

Finally the paper covers how the transaction log has been used to monitor the system performance (as seen by the data-base user), to validate and debug new user-written application software, and provide an complete audit trail for future reference.

DESIGNING FRIENDLY INTERACTIVE SYSTEMS

A presentation on financial systems design to be given at the HPGSUG North American meeting in Orlando, Florida, in April 1981, by Mr. Jack Damm of The Palo Alto Group.

DESIGNING FRIENDLY INTERACTIVE SYSTEMS

By Jack Damm, Principal, The Palo Alto Group, Sunnyvale, Calif. (408) 735-8490

Good afternoon. I am going to talk about financial planning on the HP3000 with the Dollar-Flow planning language. My discussion will focus on three areas: 1) What financial planning is, and why there is a need for computerized planning; 2) Design considerations for "friendly" user-oriented applications; and 3) How the language Dollar-Flow is used for applications like profit planning.

THE NEED FOR FINANCIAL PLANNING

First, let's start with two questions: What is financial planning? And why is it necessary? Financial planning is making decisions about allocating the scarce resources of an organization so as to best achieve its goals. In the private sector, this usually means how best to allocate money and people to achieve profitability goals. In the public sector, it may mean how best to allocate people and dollars to provide a desired level of service. The main idea here is that the resource is scarce and, as a manager, hard decisions have to be made about how to use it. More specifically, financial planning is setting budgets, making pricing decisions, and estimating future demand for products and services, in order to achieve profit and/or performance goals.

Why is formal planning necessary? First, of course, because a scarce resource (typically money) is involved. If we had enough money for everything, then we could simply raise our salaries and retire early. Secondly, it is very important to have general agreement within an organization about how goals are to be achieved. No assumptions should be made without clearly stating and documenting them. With a good financial plan, trouble signs can be spotted earlier and corrective action taken sooner. Businesses which fail to plan effectively are the best illustration of the need for planning.

Let me offer one last reason why planning is important. For many companies, planning is a necessity because of the complexity of their operations. A typical manufacturing company may purchase thousands of parts for use in a vast array of products, and assemble them in many different locations. They cannot wait until there is no money in the till to decide that it's time to raise prices. And the current rates of inflation make this an even more important consideration.

THE TYPICAL PLANNING PROCESS

Okay, let's assume that one accepts the need for financial planning. So what's the big deal? Well let's look at the typical planning process and I'll show you.

First, planning involves lots of numbers. And these numbers change often. Financial planning involves projections into the future and is a very uncertain process. When you're uncertain, then you have to do contingency planning. Play "what if" games. What if sales are 20% higher than planned? What if the cost estimates are too optimistic? What if our product sales mix is different? Because of uncertainty, alternative plans are necessary, increasing the amount of work required to plan several times over.

And that´s not all. The attempt to reach a targeted objective such as profit adds to the work. It may take several passes before all of the budgets combined with the sales estimates, cost estimates, and so forth, sum up to the desired results. The task soon becomes monumental.
company? Try changing every format statement in the model in an hour. And add to that the bother of documentation.

To summarize, manually prepared plans can be flexible, but they take a long time to do and lots of effort, especially if several passes are done. They often lack documentation. Planning with traditional programming languages takes too long to set up, is inflexible, and requires the services of a programmer.

PROBLEM ORIENTED LANGUAGES

Let me digress for a moment. For several decades now, computer scientists have been searching for a "universal" programming language. ALGOL? PL/I? APL? PASCAL? The search goes on. Each has its merits, each its disadvantages. But these "procedure oriented" languages have one thing in common: You have to be a programmer to use them. And it is altogether too easy to include bugs in even the simplest of programs. As long as there is a programmer acting as middleman between the user (or analyst) and the computer there are going to be communication problems. Maintenance problems. Resource and priority problems.

What´s the answer? A planning oriented application language which incorporates the good aspects of traditional programming, but eliminates the problems. Where plans can be set up and revised easily, without having to be a programmer. What I am describing here is one example of another class of programming languages, "problem oriented" languages. Languages which have been designed to provide solutions in a general way to classes of problems. Simple enough to be used by non-programmers. Easier to debug. Self-documenting. QUERY is an example of a problem oriented language. It provides access to IMAGE data bases in a fashion simple enough to be used by non-programmers. Dollar-Flow is a problem oriented language, designed as a tool for non-programmers who want to set up tabular planning reports.

Financial planning is an area well suited to problem oriented languages. There is a considerable amount of generality in what planners do, although no two plans are the same. A financial plan typically involves mathematical operations on rows and columns of numbers. With well defined rules for the calculations. And the burden of planning in any other way gives the financial planner considerable incentive to try new appproaches.

This is a good start. But we still have to get the planner onto the terminal and communicating with the computer. How is this done? By giving him an effective tool. One which is both friendly and enables him to get the job done in a way that he understands.

DESIGNING FRIENDLY SYSTEMS

This leads us to the next point: What makes a system "friendly"? How can a system be designed so the novice or non-computer type feels comfortable with it? I ofer here a few of my ideas and techniques for developing friendly systems.

SIMPLICITY

Keep the system simple at all cost. Do not let the internal structure on the computer dictate how a system looks to the user. Let him express his ideas in his own terms. For example, the original design for the Dollar-Flow language was based on a set of documentation which I prepared for a group of accounting types. This documentation described the workings of a particular customized model on a line by line basis. I figured: What could be a better set of design specifications for a language than actual documentation? As you document your model you are also writing your program! Another example. Dollar-Flow re-orders calculation rules automatically. Thus, line 1 on a report can reference data on line 10, which, in turn, can reference data on line 20. Dollar-Flow automatically figures out the proper sequence for calculations (calculate 20, then 10, then 1) without any intervention by the user.

The following is not an uncommon occurrence: You work many hours preparing budgets and doing sales forecasts. With a board meeting just a few days away, you finish your plan. The company president takes one look at the results of the combined numbers and gives it back, requesting a 15% cut in the budget. You prepare a revised budget, repeat all of the calculations, this time under increasing pressure to get the job done fast. The day before the board meeting, marketing revises the forecast. All of the budgets must be revised again. And now it is getting late into the evening the day before the meeting. The planning process finally ends. With a good plan? No, with exhaustion. Does this seem like a doomsday tale? It's not. I've seen this happen many times. No wonder people dread budgeting time.

Combine the sheer effort required to plan effectively with the requirements for a good plan: It must be TIMELY. In a dynamic, growing company, a plan must reflect today's expectations, not yesterday's. It must be ERROR FREE. Late-night, reworked plans suffer from simple calculation errors. Errors due to using the wrong set of estimates, because they keep on changing. Imagine the embarrassment of a summation error. And with all this, the plan must remain FLEXIBLE. I worked on a profit plan for a company a few years ago which added an entire product line between iterations of the plan. And finally, when you are all done, a good plan must be WELL DOCUMENTED. What factors were used for overhead? What was the basis for the final sales figure? How was a particular number calculated? All too often, there is little documentation on how a plan was actually prepared.

To summarize: A typical financial plan involves lots of numbers, which change often. The need for many iterations makes this process time consuming and exhausting. At the same time, the plan must be timely, error free, and well documented. In short, good financial planning is not easy.

WHAT IS THE BEST WAY TO PLAN?

Given that this is the nature of planning, what is the best way to plan? How can it be done with a minimum of difficulty? Traditionally, there have been two ways of planning. Planning by hand (and calculator) and planning using the computer. Let's take a look at both of these methods and evaluate the pluses and minuses of each.

Preparation of plans manually has several drawbacks. First, because of the amount of data involved and the number of iterations, it is slow and time consuming. After many iterations, accuracy becomes a problem. The wrong estimates may be used, particularly if they keep changing. Calculation errors seem to increase with each iteration. And documentation is usually not very good.

On the other hand we have financial planning on the computer using the traditional programming languages like BASIC, FORTRAN, or COBOL. Once set up, a model written in one of these languages will run on the computer in a matter of minutes or seconds. Great! But here´s the catch. The model will run very quickly once it has been set up, but it may take months to get it developed. And you need a programmer. Let´s see what can happen. You start your plan well in advance of the next budgeting cycle. With six months lead time you give a precise set of specifications to an enthusiastic programmer who dutifully sets about coding your model. At the end of the first three months, he comes back to you with his first try. You patiently point out where the model is not consistent with the specifications, settle on a set of revisions, and the model is reprogrammed to your satisfaction. All set, right? No. As you begin using the model, the company president starts to change his mind (even though he reviewed the original specifications). Add a decimal place here, another line item there. Why aren´t all twelve columns of data on the first page? Frustration.

What is the moral of our story? Programming a planning application with the traditional programming languages lacks flexibility. The programmer needs lead time to set up the application and has difficulty in reacting to short term changes. How about adding another division to a multi-divisional

It is important that the application be self documenting. For example, Dollar-Flow is a menu driven system. At each step of operation, the user knows his alternatives. There is little need for a "pocket guide" to the language. This is not to say that there is no need for manuals. A good manual is important. But it is a fact that few people actually read manuals. The less a system forces a user to read the manual, the more usable it will be.

Not only should the user be told what his alternatives are, the system should also help him to choose the proper response. Throughout the Dollar-Flow prompts, the most likely response is shown in brackets as the "default" response. In some cases, he can use the default response without bothering to even understand the question! For example, the prompt:

```
USE STANDARD OVERALL REPORT FORMAT (<Y>,N,W-WIDE PAGE)?
```

In one brief prompt, the user can see his options and pick one. A simple carriage return will cause the system to use the default response. And his entire report format is set up. No PRING USING or FORMAT statements. Very simple. And it can be changed easily. As the user becomes more familiar with the language, he can begin to exercise more options. With an ´N´ response, Dollar-Flow leads the user through a review of the many formatting alternatives. Report formatting can even be done on a trial and error basis. Start off with the standard format, then change the column width or number of decimal places shown as needs require.

As I already mentioned, the design for the Dollar-Flow calculation rules was based on a set of user oriented documentation. Ask a user to describe how the values on the report are to be calculated in his own terms. With the addition of a few quote marks here and there, he has already written a program in the Dollar-Flow language. Self-documenting languages not only save the effort required for documentation, but make debugging easier as well.

One last comment about simplicity. Save the user concerns about internal structure through structure independent (or data base) approaches data relationships. One of the beauties of QUERY is that the user doesn't have to concern himself with all of the details of the data base to get a simple report. In Dollar-Flow, all reports are programs, all saved programs are files, and all save files contain reports. To reference data on a saved Dollar-Flow report, simply indicate the line name and the report save file name:

```
MARKETING BUDGET = 'BUDGET' OF 'MKTG';
```

There is no need for the user to know how the data is stored or even which line on the 'MKTG' report is the 'BUDGET' line which he is using.

ERROR HANDLING

Okay, so let's say you have implemented a simple system. Does this mean that users won't make mistakes? Of course not. In fact, the friendlier a system is, the greater the likelihood that the users will not be computer types. So, keep in mind that "too err is human, to forgive is good systems design." Of course, you must edit all inputs. But then use a friendly approach when the user has made an error. Because Dollar-Flow is menu driven, simple typing errors cause the system to repeat the prompt. Errors of a more complex nature, such as where a report is referenced but does not exist, generate intelligible error messages. Along with each error message give a message number. And provide a glossary with the documentation which gives even greater detail on the possible cause of the problem.

At the same time that it is informative, a system should help the user to work around problems. For example, in the case of an invalid report reference in Dollar-Flow, the user can interactively specify a different report name, or values, or zeroes. He can also indicate that computation should cease after a scan for further errors. Again, unless a particular error is extremely serious, warn the user and proceed (with his permission). Another example. As far as the mathematician is concerned, division by zero gives unworkable results. In Dollar-Flow, division by zero yields 'invalid' numbers (which print as asterisks), but doesn't stop computation. It's amazing how much more satisfying a user finds a report filled with asterisks than just a list of error messages. At least he can look at the format to see if it's to his liking.

If you must tell the user that he has made an error, tell him as early as possible. One of the most enlightened things done by the MPE operating system is to edit the job statement when a job is being streamed from an interactive session. It sure is better to find out right away than waiting for the job to begin execution to find out that a simple error has been made on the JOB statement. Report development in Dollar-Flow is completely interactive. If a user is setting up a report and he enters a calculation rule with invalid syntax, the system responds with a message immediately, and permits him to edit his error (not unlike the BASIC interpreter). It is not necessary to go into the computation step to find many errors.

MAINTENANCE AND SUPPORT

Let us assume that as an enlightened designer of friendly systems you have now designed and implemented your masterpiece. Are you done? Of course not. This is only the first step. There are two more important aspects which are critical for good, friendly systems: Continuing improvement and good support. Let me talk about continuing development first. No system is great on the first try. I am a believer in the iterative approach to systems development, if you can afford it. I am not talking about sloppy design. I am talking about the tremendous wealth of ideas that you can get from your users, AFTER you have implemented a system. Try to be receptive to the suggestions of your users (even if they are infeasible). Never give a critical user the impression that you think he has just offered a bad idea. Go out of your way to solicit ideas from your users. If the situation merits it, get involved in several of their applications. You can learn about ways the system is being used that you never thought about. Ways in which its use may be awkward. Which messages are more annoying than useful. Which features are badly needed. I send periodic questionnaires to my users (some of them even respond). This helps to prioritize new features. And users group meetings are a great boon to information flow.

How should this wealth of new ideas be integrated into an already developed system? Carefully. Do not rush a new version of a system out to users just because they need a particular feature. You must let a new version of a system be "burned in" first by a test site. Software bugs cost you credibility. Once lost, credibility is very difficult to reestablish, so reliability is extremely important. After all, would a user prefer a system with the bells and whistles he wants but doesn't work, or one which works with a few less features?

Speaking of bugs and user suggestions leads me to the question of support. There is nothing more frustrating to a user than to get 95% of the way to his computer solution only to be stopped by the application package he is using. For any reason. If you can afford to do it, good support pays great dividends. Dollar-Flow is supported in an "on-line" fashion. This means that if a user has a problem, he picks up the telephone and calls. If his problem is with an existing report, we may even log onto his system and take a look at that report. This kind of support not only helps to find and eliminate system problems quickly, but we also find out about areas where the documentation may be confusing (or incorrect). Where another feature might simplify the user's application. In short, on-line support can be another source of good ideas from users.

Let me summarize these techniques for creating friendly systems. First KEEP IT SIMPLE. Try to think like the user instead of a computer expert. Use his terms. Assume that he won't read the manual. Try to make it self-explanatory. Second, be INFORMATIVE but FORGIVING with your error handling. Edit all inputs, but don't bother the user with minor errors. When the application merits, CONTINUING ENHANCEMENT will make a much more usable system. Respond to user suggestions. But exercise good judgment in the trade-off between adding new features and degrading SYSTEM RELIABILITY.

'IT PLANNING

I am not going to take too much time on the last part of my talk. I am just going to show you a few sample reports prepared using Dollar-Flow. At the risk of violating my agreement not to make a sales pitch, I invite you to .sit the PALO ALTO GROUP's booth during the vendor exhibits for a demonstration of Dollar-Flow in action.

Let me first describe the typical company profit planning cycle and the environment in which a planning tool like Dollar-Flow is used. The typical Dollar-Flow user is the accountant or company controller who is responsible for preparing the reports. Not a programmer. Most users are working on in-house HP3000 systems. With access to CRT's and a system line printer nearby. Reports are written interactively, and manual inputs are also entered via the terminal. Usually, reports are printed on the CRT for review then saved when the user is satisfied with the report. If hard copy is desired, the reports can be routed to the line printer. For generating large numbers of reports, the "batch command mode" is used, where with very little terminal input a large number of reports can be generated.

Profit planning typically begins with a preliminary sales forecast. Preliminary. Sales forecasts always change. And at the last minute, too. Often the sales forecast is done on a product-by-product basis for the first year or so, then combined with overall dollar sales projections further in the future. The near term unit forecasts are sometimes adjusted based on an overall dollar figure. The forecast is iterated several times. To make a change, the product manager just runs Dollar-Flow, inputs whichever figures have changed, pushes a few buttons, and the new sales forecast is ready. Since many parts of the profit plan depend on this sales forecast, the typical plan is usually set up with reports referencing the sales forecast report. If the figures are changed on the sales forecast, these changes will be automatically reflected on the other reports the next time they are run. Some manufacturing companies even use a multi-level sales forecast step, where a build plan (or production plan) is generated from the sales forecast.

Meanwhile, departmental budgets are prepared. Some Dollar-Flow users centralize the budgeting function and only distribute budget worksheets to each department or location. This is usually done if there are only one or two budget iterations. On the other hand, some of our customers distribute the budget preparation, with each location setting up its own budget in Dollar-Flow. In this case, figures can be input to Dollar-Flow, changes can be made, and several iterations of the budget can be done all in a matter of minutes. And budget consolidations are fun! With a few simple commands to Dollar-Flow, a whole series of budgets can be consolidated into a departmental or divisional budget. When changes are made to the low level budgets, they automatically are reflected on the consolidated budget the next time it's run.

The profit/loss projection is next. Using the data from the sales forecast, the build plan, and the budgets, and adding factors for items like sales discounts and returns, a pro forma operating statement is prepared. Often, the bottom line (profit) on this report determines what (if any) changes need to be made to the budgets. With a flexible tool like Dollar-Flow, a financial executive can even do sensitivity analysis: What if sales are 20% lower then forecast? What if our discount schedule is more aggressive and our volume is larger?

Some companies that rely on substantial amounts of debt to finance their operations combine the profit/loss projection with a cash flow projection. This is because interest paid (an item of expense on the profit/loss statement) has an impact on the amount of money required to run the business. This determines the level of borrowing, which, in turn, affects the amount of interest which is paid. Dollar-Flow, and most good financial planning languages, can solve the "simultaneous equations" this circular logic represents, and determir a level of debt and debt service which are consistent with each other. This is far more difficult when done manually.

Another procedure which is laborious when done by hand is the aging of accounts receivable and accounts payable projections. Using Dollar-Flow, once the rules for aging have been set up, a change in the sales forecast or the build plan will automatically be reflected in new receipts and payables projections.

And, finally, some companies prepare pro forma balance sheets as the last step in their profit planning cycle. This is not necessarily the way all companies plan. Or even the way all Dollar-Flow users plan. In fact, many Dollar-Flow users are not even responsible for profit planning. Instead, the system is used for a wide variety of ad hoc applications involving calculations on rows and columns of numbers. It is even used as a design tool for systems which will later be hard-coded in COBOL, FORTRAN, or BASIC.

Some of the other applications of Dollar-Flow that I am aware of include:

Product pricing. Comparing alternative prices for a single product (the plotting capability is great for comparisons). Or comparing profit percentage across an entire product line. Financial ratio analysis. Comparing selected financial ratios against industry standards or company objectives. Capital budgeting. Rates of return and discounted cash flows can be calculated easily using built-in financial functions.

Performance reporting. Variance reports showing actual budgets or profits versus plan. How sales are doing against target. (One Dollar-Flow user generates 500 graphs every month showing product line sales performance for every branch of every distributor who markets his products!)

SUMMARY

Let me leave you with a few parting thoughts. Financial planning is not an easy process. Figures change. The whole approach to a plan may change. And you need your results yesterday. Traditional systems design and programming methods are not going to be effective in this kind of situation. Use a better approach. With a friendly, problem oriented planning language like Dollar-Flow, applications nightmares can become applications successes,

SOFTWARE CONTRACTS: PREVENTATIVE MAINTENANCE TO ENSURE SUBSEQUENT QUALITY SERVICES

BY

WILLIAM F. LEONARD, JR.

Software Contracts: Preventative Maintenance to Ensure Subsequent Quality Services

First time users of a computer system in most instances tend to be small to medium sized organizations overwhelmed by the complexity and choice of available options for hardware and software acquisition. The hardware decisions are usually made with a great deal of preparation, analysis and comparison. However, the software issues tend to involve even more difficulties. When an eventual decision has been made, the acquisition typically consists of a single software vendor.

In many cases, a contract is not involved. A simple hand shake, signifying the importance of trust and good faith, is considered to be a professional understanding between two organizations. In other instances, the buyer signs a contract proposed by the vendor and prepared in accordance with the vendor's best interests. Even when a contract is used, the buyer may enter into a complex business arrangement without the benefit of professional advice available from the organization's legal counsel. It appears totally incongruous that an organization approaching the acquisition of a complex computer system in a business-like manner will carefully research the purchase of hardware but enter into a software commitment without careful preparation and scrutiny. Standard advice implied in the phrase "caveat emptor," or let the buyer beware, certainly would appear appropriate in this type of situation. An ancillary rule would be that unless a written contract is prepared and signed, an informal understanding is totally worthless.

This may sound like a rare harsh piece of advice to offer for a professional commitment between a software vendor and a buyer but it is the only solid insurance policy that can be relied upon when difficult situations develop. As indicated earlier, many small to medium sized organizations are first time users of a computer system. As such, they lack any previous experience with a software vendor. This absence of previous experience or a successful relationship must be offset with a clear cut definition of the responsibilities for both parties in the development, implementation and maintenance phases of a long term commitment. Hence, a contract mutually developed and approached with the goal to be as comprehensive as possible is the only satisfactory mechanism for ensuring success. A perspective emphasizing the characteristics of pre-planning, mutual understanding, clearly worded documentation, timetables for delivery and legal review represents a business-like concern for a smooth and orderly project. This type of "preventative maintenance" approach can only assist in ensuring a high level of quality services following contract initiation and at the same time minimizing potential problems.

How should this approach be initiated? What topics represent the highest priorities for eventual agreement? At what point does a desire to formulate a comprehensive agreement become counter productive in terms of time, effort and other limited resources? These questions are extremely important and relevant to a contracting organization, especially first-time users and small to medium sized firms.

In the case of the Alexandria school system, with a total enrollment of approximately 11,000 students, a first-time user situation applied. The administration determined that the organization's size was insufficient to justify an internal software development approach with significant resources expended for such an undertaking. Therefore a decision was made to find the most appropriate software package that was available and to plan on customizing it to the degree necessary. Unfortunately, such a package could not be found. The school system decided to enter into a contract with a new software vendor located in the same metropolitan area and to develop a system in accordance with a set of specifications. The contract was developed quickly but was reviewed by school management, the vendor and legal counsel.

The basic components of the total system are now in place. The vendor and the school system are continuously refining the system as would be expected. Some work remains to be completed but firm plans for doing so are not yet finalized.

In spite of some extensive preparation and the existence of a contract, problems have and are continuing to occur. Many areas that ideally should be incorporated in an agreement were not addressed prior to contract initiation; the absence of written understandings in these areas represents the major explanation for many of the problems that did develop. An example is hardware availability for development work. In the initial several months after contract initiation, an extensive amount of software development work, out of necessity at the time, was completed on the user's equipment. Most of the activity occurred during periods when normal production work was not involved. However, the lack of a clearly worded clause in the contract on this matter resulted in occasional situations where the vendor and user had to work out solutions on a case-by-case basis.

The existence of a "right to modify" clause and an agreement pertaining to "final acceptance/completion of project" have also created some difficulties that could have been avoided or reduced in magnitude if additional time for review had been allowed. It is now expected that the vendor and the school system will soon be entering into negotiations to address the remaining provisions of the original contract that have not been fulfilled and at the same time develop a mutually agreeable contract for on-going maintenance support. Both parties are interested in doing so and the end product of these deliberations will hopefully be an agreement considered to be satisfactory for a permanent, support relationship.

Alexandria's experience to date has reaffirmed the original view that a contract was necessary and needed to be as specific as possible. However, in true hindsight form, this experience has identified a number of potentially vulnerable areas that should have been addressed in the contract from the beginning.

Alexandria has learned from its experience but how do other similar organizations adequately approach the task? Is it absolutely necessary that this invaluable information be obtained through experience? Certainly not! A number of approaches are available that could be utilized.

Initially, it is suggested that a potential user carefully consider some of the broad management-related issues that typically occur in software acquisition. An example relates to whether or not the software vendor is new to the business or is embarking on a new application. In such a situation, the potential user must be careful to review such decision-influencing factors as the quality and depth of the vendor's staff as well as the vendor's computer resources and overall financial condition.

The following list of key factors should be considered:

1. Degree of system customization expected.
2. Software vendor's capabilities.
3. Possible effect of hardware upgrades in the future.
4. Management philosophy on single versus multiple vendors.
5. Existing package or development contract.
6. Degree of eventual independence expected of user in software operation.
7. Degree of specificity in development and implementation expected - documentation standards, installation requirements, programming efficiency, and others.

Other factors could easily be added. However, the nature of the potential concerns should be apparent from the examples listed. The user's reactions to these factors should indicate the extent of emphasis to be placed on specific components of any proposed contract.

With these factors identified, the potential user can approach a number of available sources for additional information and advice. Professional colleagues in a users' association, existing agreements developed by other organizations and a temporary consultant retained for third party objectivity and technical assistance represent examples of existing sources. Regardless of the approach utilized, the most important advice to a potential user pertains to the need to allow adequate time for pre-contract planning with the likely vendor and the user's legal counsel. The provision of adequate time for informal discussions, negotiations, research and review cannot be easily substituted.

In this regard, the attached software checklist is offered as a partial index of questions a potential user could consider prior to contract initiation. Additional items for the checklist could be identified but in its present form the list represents a compilation of the most significant areas to be addressed in an agreement. The potential user, working in cooperation with legal counsel, is the most appropriate party for determining the relevance of each item to the particular organization. The list can assist in identifying those areas that need to be dealt with in greater detail when the contract is finalized.

When contracting organizations begin to emphasize in their overall project activities the need for comprehensive pre-planning and protection through the existence of well-prepared agreements, then the software industry will benefit from the additional professionalism and good will created. A "preventative maintenance" perspective, almost defensive in nature, will assist the users in obtaining a level of service necessary for long-term success.

SOFTWARE CONTRACT REVIEW

A CHECKLIST APPROACH

ITEM NO.	HEADING	CHECKLIST ITEM	Has item been completed? YES	NO
		I. PLANNING:		
1	Identification of System Components	Does the contract include a section describing in detail all components expected from the software system?	___	___
2	Cost of System Components	Does the contract specify the individual prices for all components in the software system?	___	___
3	Legal Review	Has the user reviewed the proposed contract with the organization's legal counsel to ensure its best interests?	___	___
4	Consultant Review	Has a data processing consultant been retained to assist in negotiations with the vendor or review the proposed contract for possible changes?	___	___
5	Establishment of Contract Priorities	Has the user prepared for negotiations by attaching a priority value to all required or desired contract provisions in an effort to obtain agreement on the most important needs?	___	___

SOFTWARE CONTRACT REVIEW

A CHECKLIST APPROACH

ITEM NO.	HEADING	CHECKLIST ITEM	Has item been completed? YES	NO
	II. PERFORMANCE:			
6	Assignment of Responsibility	Have the vendor and user listed the tasks to be performed by each to achieve implementation on time?	___	___
7	Schedule of Responsibility	Have the vendor and user each prepared a schedule for determining the completion date for all assigned tasks to achieve implementation on time?	___	___
8	Documentation and Related Standards	Does the contract require that complete documentation be provided to the user and are minimal standards identified for ensuring an acceptable level of quality?	___	___
9	Installation Standards	In the case of software development, does a requirement exist to guarantee that the software system will comply with whatever installation standards are acceptable to the user?	___	___
10	Reproduction of Documentation	Has the user received permission to reproduce any part of the system documentation with a proviso that it be used internally for operational success?	___	___
11	Corrections and Upgrades	Does the contract indicate that the user can obtain without charge any corrections or upgrades to the software system that are intended to improve its quality?	___	___

SOFTWARE CONTRACT REVIEW

A CHECKLIST APPROACH

ITEM NO.	HEADING	CHECKLIST ITEM	Has item been completed? YES	NO
	II. PERFORMANCE (Continued):			
12	Source Code Accessibility	Has the user obtained access to source code for the software system?	___	___
13	Right to Future Options	Does the contract include the right to obtain future options or changes in the software system at the same price offered future users?	___	___
14	Right to Modify	Has the user obtained in the contract the right to modify the software system, with a waiver of maintenance indicated?	___	___
	III. INSTALLATION:			
15	Test in Actual Environment	Has the user stipulated that the final purchase of the software is conditional on its acceptable operation in the user's actual environment?	___	___
16	Acceptance Criteria	Have the user and vendor specifically listed the acceptance criteria needed to validate the software system?	___	___
17	Basis for Criteria Identification	Have the acceptance criteria been determined on the basis of the requirements and specifications section used for initial system design?	___	___

SOFTWARE CONTRACT REVIEW

A CHECKLIST APPROACH

ITEM NO.	HEADING	CHECKLIST ITEM	Has item been completed? YES	NO
		III. INSTALLATION (Continued):		
18	Level of Operational Availability	Has a specific requirement been formulated which states the level of operational availability expected of the software system over a period of time?	___	___
19	Run Time Requirements	Have specific expectations been developed that pertain to the run time requirements of the software system when fully operational?	___	___
20	Hardware Requirements	Have specific expectations been developed that indicate the degree of hardware requirements to be used by the software system when fully operational?	___	___
21	Acceptance Period	Does the contract include a requirement that a specific acceptance period is necessary for validating the software system prior to final purchase?	___	___
22	Delivery Timetable	Has the user specified delivery dates with an additional reasonable period of time indicated as a contingency for unanticipated problems or delays?	___	___
23	Acceptance Procedures	Has the user specified the acceptance and approval mechanism to be utilized when work products are delivered?	___	___

ITEM NO.	HEADING	CHECKLIST ITEM	Has item been completed? YES	NO
	III. INSTALLATION (Continued):			
24	Software Support	Are on-going support requirements after installation carefully defined and priced?	___	___
	IV. STAFF:			
25	Project Staff of Vendor	Prior to contract approval, is the vendor willing to provide a synopsis of his staff resources - in terms of size, training, experience and ability to provide adequate support services?	___	___
26	Full-Time Employee Requirement	Does the contract state that the vendor's project staff should be full-time employees, not part-time "moonlighters," and that they are bonded?	___	___
27	Agreement on Hiring Employees	Is there an agreement in the contract that the vendor and the user will refrain from hiring away from each other employees currently under contract?	___	___
28	Hiring Employees in Termination	Does a agreement exist that permits the user to hire the employees of the vendor in the event of business termination?	___	___
29	Project Staff Substitutions	Does the user have the option of demanding that substitutions be made in the vendor's project staff if personnel or work problems infringing on the project's success indicate the need to do so?	___	___

SOFTWARE CONTRACT REVIEW

A CHECKLIST APPROACH

IV. STAFF (Continued):

ITEM NO.	HEADING	CHECKLIST ITEM	Has item been completed? YES	NO
30	Security Requirements	In the case of software development, especially work on-site, has the vendor agreed to comply with the user's internal security provisions?	___	___

V. FINANCIAL CONSIDERATIONS:

ITEM NO.	HEADING	CHECKLIST ITEM	Has item been completed? YES	NO
31	Recovery of Progress Payments	Does the contract provide protection to the user in the form of recovery of progress payments from the vendor if an acceptable software system cannot be developed or delivered?	___	___
32	Conditional Acceptance of Hardware	Has the user related the final acceptance of the hardware to the performance of the software system?	___	___
33	Guarantees on Price Movement	Does the contract protect the user with specific guarantees on increases and/or decreases in price, with changes occurring only when authorized by the user?	___	___
34	Delivery of Finished Product on Partial Basis	If progress payments are made over an extended period of time, does the contract require an equivalent portion of finished product on a phasing basis as a fair exchange to the user?	___	___

SOFTWARE CONTRACT REVIEW

A CHECKLIST APPROACH

V. FINANCIAL CONSIDERATIONS (Continued):

ITEM NO.	HEADING	CHECKLIST ITEM	Has item been completed? YES	NO
35	Comprehensive Coverage on Costs	Does the contract identify all possible charges related to the successful completion of the project, including personnel costs, external purchases, use of machine time, maintenance, training installation support, documentation and similar items?	___	___
36	Schedule for Progress Payments	Is a specific allocation formula developed for approaching progress payments?	___	___
37	Hold Back Payment Provision	Is a specific percentage of total payment withheld by the user until project completion to ensure a successful conclusion to the contract?	___	___
38	Term of License	Has adequate attention been focused on specifying the term of license and a renewal notification procedure by the vendor?	___	___
39	Business Termination	Has the user incorporated adequate protection in the contract in the event that the vendor undergoes business termination?	___	___
40	Vendor's Financial Condition	Is the vendor obligated to provide a copy of a current financial statement to the user at the time of contract approval?		

SOFTWARE CONTRACT REVIEW

A CHECKLIST APPROACH

VI. WARRANTIES:

ITEM NO.	HEADING	CHECKLIST ITEM	Has item been completed? YES	NO
41	Warranty Test	Does the contract state that the software system will perform in accordance with the user's specifications?	___	___
42	Cancellation Option	Does the contract indicate that cancellation without penalty to the user exists as an option if performance guarantees are not met?	___	___
43	Periodic Progress Review	Are the expectations, requirements and responsibilities contained in the contract formally identified as the basis to be used in periodic progress meetings to determine that all obligations are being fulfilled?	___	___
44	Guarantee of Ownership	Is the vendor willing to include a statement in the contract guaranteeing ownership of the software system and that the user enjoys "hold harmless" protection from third parties?	___	___
45	Free Maintenance During Warranty	Does the contract protect the user during the specified warranty period from all possible maintenance charges?	___	___
46	Freedom of Use Provision	Does the contract extend freedom of use to the user as long as the software system is utilized within the user's organization?	___	___

DISTRIBUTED CONTROL USING ONE CPU

by

John Beckett

Director of Computer Services
Southern Missionary College
Collegedale, TN 37315

Industry trade journals these days seem to be bristling with stories of the pros and cons of distributed processing. Central to these discussions is the assumption that distributed intelligence risks distributed control, and distributed control will bring chaos.

Our site has only one multi-use computer, an HP 3000 Series III. We have chosen to distribute control of our system to the users because we find it does not compromise anything we find important, and because we get more out of our computer thereby.

Our implementation of distributed operations uses three fundamental concepts:

1. The function of the Computer Center is to provide computer "power" and expertise in harnessing that

power. Operational control of the application of that power is at the clients' discretion.

2. Shared devices such as line printers are operated by people at the Computer Center. The ONLY reason for this is the expense of such devices and their operators. If a client office could afford its own line printer, there is no reason they could not run it themselves.
3. Programming standards are still maintained by the central DP department. In our case, this means that all programmers work for us and are rented out to clients as they need work done.
4. An oversight committee composed of our users allocates the computer resources (disc space, CPU time, terminals and ports) available. This committee meets approximately once per month. It provides the DP manager with policies and guidelines for day-to-day allocation decisions. One of the primary functions of the DP manager is to report to this committee and to serve them as a technical consultant. In a larger organization these two functions might be separated.

What we have done is to deliberately sacrifice control over our processing, giving it to our users. We consider ourselves to be providers of power rather than service. Only in the area of programming do we get involved in any direct decision making by our own authority.

A key element in distributing control is providing a meaningful heirarchy of service levels. We have defined the following:

1. Sessions, CS priority. Used for instances where people are doing small bites (spelled with an "i") of processing such as data entry or inquiry.
2. Sessions, DS priority. Used where terminal access requires demands for non-trivial amounts of processing. Examples include serial searches in QUERY, compiles, etc.
3. Jobs, daytime. Used for timely reports and updates.
4. Jobs, non-daytime. Used for everything possible.

Sessions at CS priority is the processing mode with which HP users are most familiar. Sessions at DS priority are easy to accomplish. In this paper I wish to discuss our approach to batch processing, which has proved to be a key element in our Distributed Control concept.

Why do we use batch access at all in an interactive system? This question we researched when our HP 3000 was first acquired in 1977. We found the reasons alluring:

1. Batch jobs automatically execute at low priority. Thus they don't impact resonse time as do long programs run from terminals.
2. Batch jobs are detached from terminals. Nobody has to stay around to reply to the next request for input, or log off when they are done.
3. The $STDLIST log is available for later inspection should anything go wrong.

After seeing the potential, I immediately ordered that everything possible be done on our computer by batch jobs. That was in 1977. We have since solved the following problems:

1. You can't file a batch job stream with documentation, since it contains passwords. I know you can blank them out, but that gets forgotten.
2. You don't dare change passwords (even though they have been compromised by a programmer who ran a listing of a stream file), because for the next business cycle all jobs will fail until you get the passwords onto their stream files, which effort in itself will probably compromise them, and so on.
3. Job stream files don't adjust themselves to changing conditions. Little things like "we need three copies this time instead of two" and "please do a rerun using the OLDTX file; there was a bug I've fixed now" can all-too-easily leave you with stream files that won't work next business cycle. You have to choose between:
    A. Writing programs to take into account every possible operational exigency, and adjust automatically ($$$$$!)
    B. Trust your operators to use a text editor to change the stream files each time for that run (danger! EDITOR has no means of validating those changes.)
4. HP's job stream "hopper" is an extremely rudimentary device.

It is little better than a card reader, differing mostly in that you can see its shortcomings better. There is no way to actually set a specific date and time for a job. Nor is there any way to schedule a job for execution every week (or month, quarter, year, or any other interval).

For a year or so we tried writing little programs here and there, to combat these problems. Some of the programs became embedded in applications. Others became utilities used by more than one application. Then one day we woke up to the fact that we had a class of problems that could be solved by a single solution: a comprehensive interface to the :STREAM command in MPE. Hence was born SLS.

SLS got its name from the original name of the program file on our system: STREAM.LIB.SYS. One of our programmers got me to put in a system UDC with its initials because he was a slow typist. The first time I contributed it to the library I didn't want to name it after an MPE command, so I just called it SLS. Now it resides in SLS.SLS.SYS on our system.

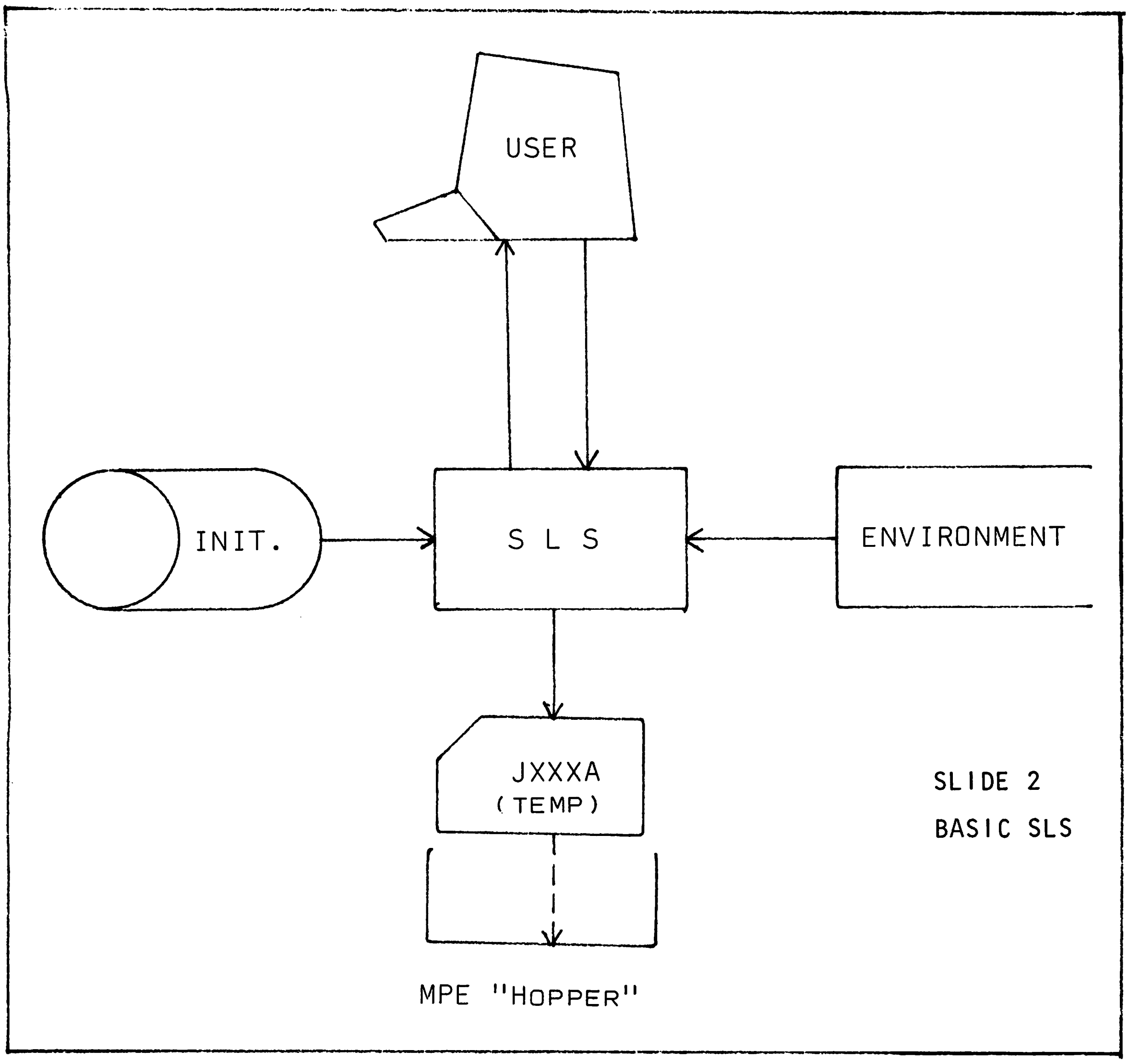
SLS got one of its prime characteristics from MPE: friendliness. For example, it is designed to install itself with almost no effort on your part.

```
:HELLO MANAGER.SYS
:NEWACCT ORLANDO,MGR;CAP=AM,AL,GL,ND,SF,IA,BA,PH

:HELLO MGR.ORLANDO
:NEWGROUP DATA
:NEWGROUP DOC
:NEWGROUP SOURCE
:NEWGROUP JOB
:FILE TAPE;DEV=TAPE
:RESTORE *TAPE;SLS@.@;SHOW
:FILE STREAMI=SLS.JOB
:RUN SLS1.PUB;PARM=1

PLEASE ENTER PASSWORD FOR MANAGER.SYS (DEFAULT: NONE)?______
PLEASE ENTER PASSWORD FOR SYS ACCOUNT (DEFAULT: NONE)?______

IN ORDER TO USE THE JSNUM FUNCTION, YOU MUST ALLOW SLS
TO USE PRIVELEGED MODE.
SELECT PRIV OR NOPRIV VERSION?JSNUM   <------ oops!
SELECT PRIV OR NOPRIV VERSION?NOPRIV
DO YOU WISH TO HAVE THE PROGRAMS COMPILED AS A PART OF
THE INSTALLATION PROCEDURE (YOU MUST HAVE SPL)?NO
DO YOU HAVE SLS ON YOUR SYSTEM ALREADY?NO
HOW MANY COPIES OF THE USER MANUAL SHALL I RUN?5

STREAM JXXXA
```

#J59   SLIDE 1

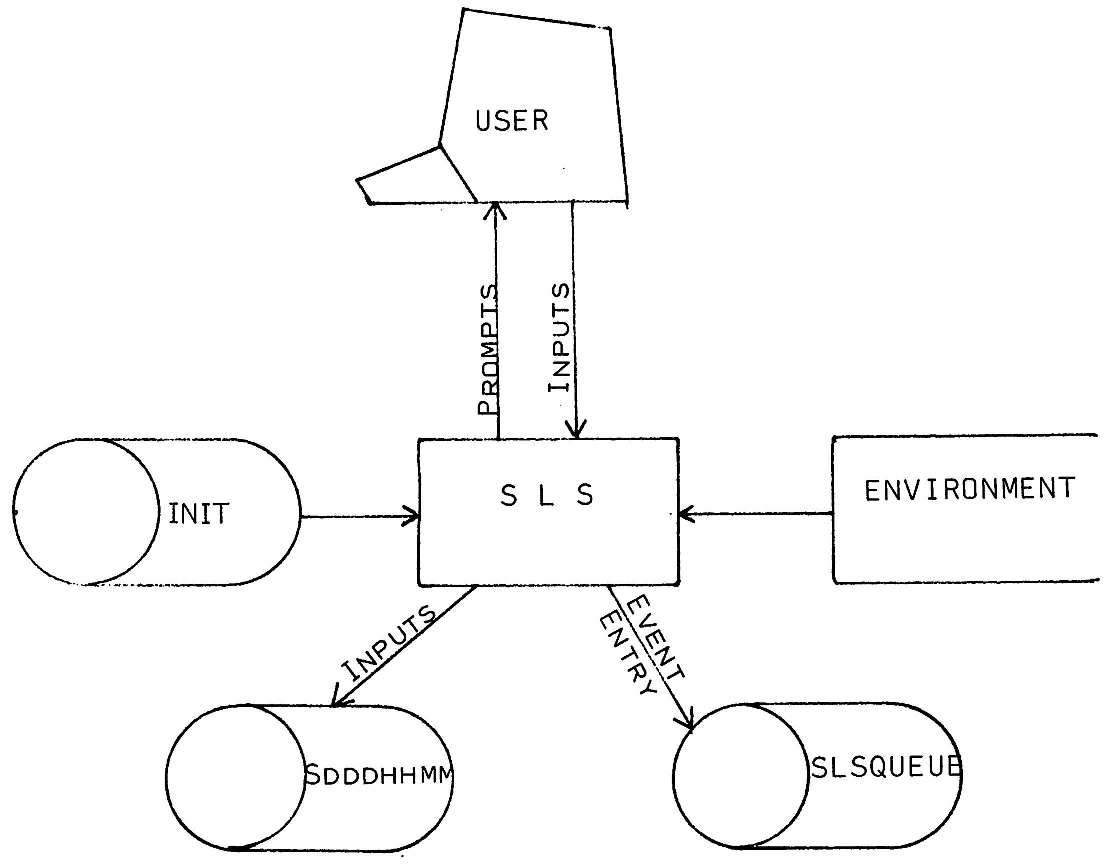
I would like you to note at this point some features of SLS:

1. It produces a batch job, so the work will not detract from system response.
2. It obtained several variables from me. In one case my answer was not appropriate. SLS handled the problem on-the-spot.
3. When it is done, SLS will have produced for me the documentation I wanted. I didn't have to remember to ask--SLS asked me.



SLIDE 2
BASIC SLS

In the mode illustrated by the installation procedure, SLS operates under control of an initiator (template) file. SLS may obtain additional information from the computer environment or from the user. The final result is a configured job stream in a temporary file which SLS then streams.

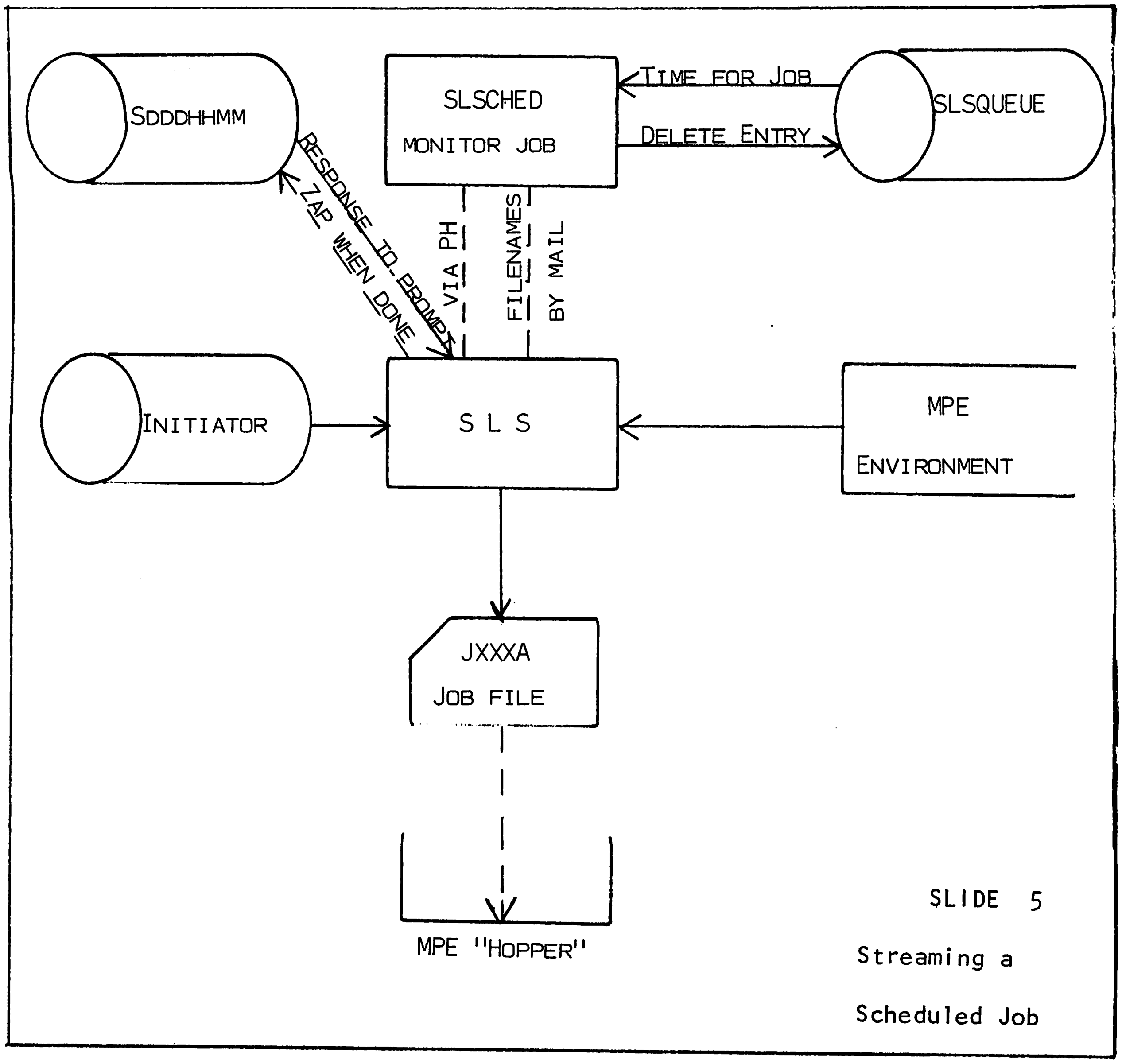


SLIDE 3

SCHEDULING A JOB

```
:RUN SLSCHED.SLS.SYS


SLS SCHEDULEING QUEUE UTILITY

>S @.ADMIN

JOB DATE     TIME USER           INITIATOR                INPUT FILE

122 03/05/81 2310 ADMREC.ADMIN   ROSSLSI.PUB.ADMIN        S0641017.ADMREC

127 03/05/81 2310 ADMREC.ADMIN   ROSSLSI.PUB.ADMIN        S0641119.ADMREC

106 03/06/81 0100 FINPROG.ADMIN  DAYJOBI.FINANCE.ADMIN

111 03/08/81 0330 A900.ADMIN     UPDATEI.A900.ADMIN       S0071710.A900

18  03/09/81 0130 FINPROG.ADMIN  AWLETERI.FINANCE.ADMIN

37  03/10/81 0130 FINPROG.ADMIN  VOUCHI.FINANCE.ADMIN

75  03/11/81 0300 DWOMEN.ADMIN   PHONDIRI.DWOMEN.ADMIN    S2451623.DWOMEN

77  03/11/81 0330 DWOMEN.ADMIN   ROOMCHKI.DWOMEN.ADMIN    S2401458.DWOMEN

108 03/12/81 0130 FINPROG.ADMIN  INAUDITI.FINANCE.ADMIN

20  03/15/81 0030 CAFSYS.ADMIN   MIDMONI.CAFSYS.ADMIN

25  03/15/81 0040 STAFFPAY.ADMIN INDEXI.STAFFPAY.ADMIN

26  03/15/81 2200 FINPROG.ADMIN  SELM018I.FINANCE.ADMIN

27  03/25/81 0400 PAYROLL.ADMIN  STUWORKI.PAYROLL.ADMIN

29  04/01/81 0015 FINPROG.ADMIN  SELM008I.FINANCE.ADMIN

30  04/01/81 0030 CAFSYS.ADMIN   ENDMONI.CAFSYS.ADMIN

31  04/01/81 0040 CSHOP.ADMIN    CSRECAPI.CSHOP.ADMIN

>EXIT
END OF PROGRAM
:
```

SLIDE 4



SLIDE 5

Streaming a Scheduled Job

In an alternative mode, the job file is discarded. Instead, a permanent disc file with all terminal input is saved. An entry regarding the date and time the job is to run is made in a file named SLSQUEUE.SLS.SYS. At the proper time, a monitor job will find the entry, run SLS against the initiator specified using the input file saved for parameters, and stream the job.

Before we get any deeper into technical details, let's look at the seven functions of SLS:

1. Display of information on the user's terminal. The usual purpose is prompting. Any information to which SLS has access may be displayed.
2. Interrogation of user at the terminal, for information to be used in constructing the job.
3. Interrogation of disc files for information needed in constructing the job. An excellent application for this function is passwords. Passwords used in an application need only be kept in one place, with SLS looking them up as needed.
4. Interrogation of the operating environment. You can check for the existence of files, check sizes and empty space in them, and verify that there is sufficient room in a file for the data you propose to put in it. You also have access to many items in the WHO intrinsic. Optionally (uses priveleged mode), you may have access to the job/session ID of the perpretrator of the job.
5. Manipulation of information gathered in the steps above. This can include validation, translation, arithmetic, and many other functions normally associated with programming languages.
6. Preparing the actual job stream file. If we are merely putting it into the SLSQUEUE file, this product will be

discarded.

7. Submitting the job either:

   A. Directly, through the programmatic :STREAM command.

   B. Deferred, by adding to the SLSQUEUE file an entry specifying the initiator, date/time, and (if any input was requested of the operator of the terminal) parameters to be input.

These functions are performed under control of the initiator file, which is very similar to a computer program. SLS functions much like a language interpreter.

Now I'd like to show you how to use SLS. I'll use the quickest way I know--examples. Let's trace a universal application, system backup, through the various stages from where most sites are at to a fully SLS'd job that requires no operator intervention other than mounting tapes (and authorizing them). First, an HP-styled job stream:

```
BACKUPJ.MGR.SYS
    !JOB BACKUP,MANAGER/ALPHA.SYS
    !FILE TAPE;DEV=TAPE
    !SYSDUMP *TAPE
    NO                    << CHANGES >>
    0                     << DATE    >>
    !EOJ
                                    SLIDE 6
```

The first thing we need to do is get the password out of this job. So we will create an additional file (IPASS.MGR.SYS) with the password to MANAGER.SYS on the first line:

```
                    IPASS.MGR.SYS
                       ALPHA

                                          SLIDE 7
```

Now we will rewrite the job stream so it will get the password from this file:

```
BACKUPI.MGR.SYS
   << SYSTEM BACKUP INITIATOR VERSION 0.0 6/3/80 >>
   FILE IPASS.MGR MGRPASS 1 1 8
   PRINT !JOB BACKUP,MANAGER/<MGRPASS>.SYS
   PRINT !FILE TAPE;DEV=TAPE
   PRINT !SYSDUMP *TAPE
   PRINT NO                 << CHANGES >>
   PRINT 0                  << DATE    >>
   PRINT !EOJ                             SLIDE 8
```

Note the following differences:

1. The file name ends with I instead of J.

2. Commands to be submitted are preceded with PRINT.

3. The variable MGRPASS is assigned a value by getting it from the file IPASS, and is referenced by <MGRPASS>.

4. You can give a copy of the initiator to anybody without fear, because they don't have your password.

5. Changing all jobs run by MANAGER.SYS for a new password can be accomplished by changing only the one password file.

Of course, it may be that you don't always want to do a full backup. The next version prompts the operator for the date to use:

```
BACKUP.MGR.SYS
    << SYSTEM BACKUP INITIATOR VERSION 1.0 6/3/80 >>
    << 0.0 06/03/80 INITIAL VERSION               >>
    << 1.0 06/03/80 ADDED "DATE" PROMPT           >>
    FILE IPASS.MGR MGRPASS 1 1 8
    3 INPUT DATE ENTER BACKUP DATE (DEFAULT: FULL BACKUP)?
    NULL 1
    YES 1 SET DATE 0
    YES 1 DISPLAY FULL BACKUP HAS BEEN SELECTED
    YES 1 GOTO 5
    DATECHK MM/DD/YY
    NO DISPLAY  INVALID DATE. ENTER IN FORMAT MM/DD/YY OR
    NO DISPLAY   CARRIAGE RETURN ONLY TO THIS PROMPT.
    NO GOTO 3
    5 PRINT !JOB BACKUP,MANAGER/<MGRPASS>.SYS
    PRINT !FILE TAPE;DEV=TAPE
    PRINT !SYSDUMP *TAPE
    PRINT NO                   << CHANGES >>
    PRINT <DATE>               << DATE    >>
    PRINT !EOJ
```

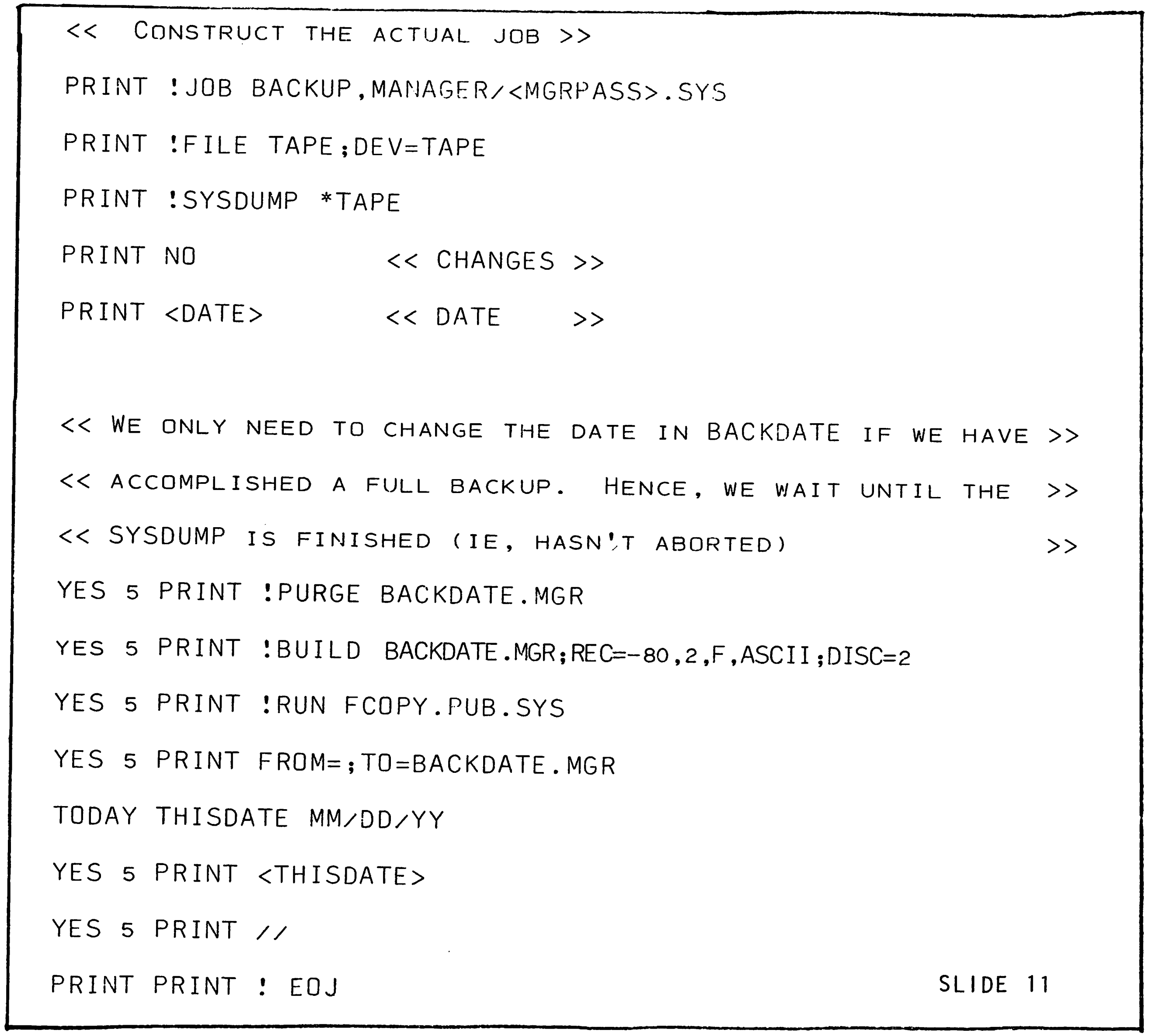
SLIDE 9

Example of use:

```
:RUN SLS.SLS.SYS
JOB NAME?BACKUP.MGR
ENTER BACKUP DATE (DEFAULT: FULL BACKUP)?05/21/80
STREAM JXXXA
   #J28
END OF PROGRAM
:
                                              SLIDE 10
```

A nice "wrinkle" might be to add automatic selection of date in the case of relative backups. This is easy to do, using TODAY to get the date and FCOPY to put it on a file for subsequent retrieval by the FILE statement:

```
BACKUPI.MGR.SYS

    << SYSTEM BACKUP INITIATOR VERSION 2.0 6/3/80 >>
    << HISTORY:                                   >>
    <<  0.0 06/03/80 INITIAL VERSION              >>
    <<  1.0 06.03.80 ADDED "DATE" PROMPT          >>
    <<  2.0 06/03/80 AUTOMATIC DATE FOR RELATIVE  >>

    << FLAG USEAGE:
    <<  1  RELATIVE BACKUP                        >>
    <<  5  FULL BACKUP                            >>
    FILE IPASS.MGR MGRPASS 1 1 8
    INPUT DATE ENTER BACKUP TYPE: FULL OR REL?
    NULL 5
    YES 5 SET DATE FULL
    MATCH 5 FULL
    MATCH REL
    NO 1 NO 5 DISPLAY I NEED DEFAULT, "FULL", OR "REL".
    NO 1 NO 5 REPEAT

    << SET UP SYSDUMP DATE IN DATE VARIABLE    >>
    NO 5 FILE BACKDATE.MGR DATE 1 1 8
    NO 5 DISPLAY I AM INITIATING BACKUP RELATIVE TO <DATE>.
    YES 5 SET DATE 0
```



```
<<  CONSTRUCT THE ACTUAL JOB >>

PRINT !JOB BACKUP,MANAGER/<MGRPASS>.SYS

PRINT !FILE TAPE;DEV=TAPE

PRINT !SYSDUMP *TAPE

PRINT NO              << CHANGES >>

PRINT <DATE>          << DATE    >>


<< WE ONLY NEED TO CHANGE THE DATE IN BACKDATE IF WE HAVE >>

<< ACCOMPLISHED A FULL BACKUP.  HENCE, WE WAIT UNTIL THE  >>

<< SYSDUMP IS FINISHED (IE, HASN'T ABORTED)               >>

YES 5 PRINT !PURGE BACKDATE.MGR

YES 5 PRINT !BUILD BACKDATE.MGR;REC=-80,2,F,ASCII;DISC=2

YES 5 PRINT !RUN FCOPY.PUB.SYS

YES 5 PRINT FROM=;TO=BACKDATE.MGR

TODAY THISDATE MM/DD/YY

YES 5 PRINT <THISDATE>

YES 5 PRINT //

PRINT PRINT ! EOJ                                   SLIDE 11
```

The file BACKDATE.MGR.SYS will be used as a repository of the last backup date.

Note that as initiators get longer, the need for documentation increases.

For the final version, we will schedule this job to run every night at 10 PM. It will do a full backup on Sunday night, and relative on others. For further interest, we'll skip Saturday nights.

BACKUPI.MGR.SYS

```
    << SYSTEM BACKUP INITIATOR VERSION 3.0 6/9/80 >>
    << HISTORY:                                   >>
    <<  0.0 06/03/80 INITIAL VERSION              >>
    <<  1.0 06/03/80 ADDED "DATE" PROMPT          >>
    <<  2.0 06/03/80 AUTOMATIC DATE FOR RELATIVE  >>
    <<  3.0 06/09/80 SCHEDULE NIGHTS EXCEPT SAT   >>


    << FIRST FIGURE OUT WHEN TO RUN NEXT >>
    DATECNV RUNDATE MM/DD/YY ; ; ; 1D
    DATECNV RUNDAY WWW; <RUNDATE>;MM/DD/YY
    LOAD <RUNDAY>
    MATCH 5 SUN
    MATCH 1 SAT
    << WE WILL NEED TO ADJUST BY 1 DAY >>
    YES 1 DATECNV RUNDATE MM/DD/YY; <RUNDATE>; MM/DD/YY; 1D
    SUBMIT <RUNDATE> 22:00
    FILE IPASS.MGR MGRPASS 1 1 8
    NO 5 FILE BACKDATE.MGR DATE 1 1 8
    YES 5 SET DATE 0
    PRINT !JOB BACKUP,MANAGER/<MGRPASS>.SYS
    PRINT !FILE TAPE;DEV=TAPE
    PRINT !SYSDUMP *TAPE
    PRINT NO                << CHANGES >>
    PRINT <DATE>            << DATE    >>
    YES 5 PRINT !PURGE BACKDATE.MGR
    YES 5 PRINT !BUILD BACKDATE.MGR;REC=-80,2,F,ASCII;DISC=2
    YES 5 PRINT !RUN FCOPY.PUB.SYS
```

```
YES 5 PRINT FROM=;TO=BACKDATE.MGR

TODAY FULLDUMP MM/DD/YY

YES 5 PRINT <FULLDUMP>

YES 5 PRINT //

YES 5 PRINT EXIT

PRINT !EOJ

SCHED

NO STOP
```

SLIDE 12

The last two statements bear comment. You don't want the job to be streamed when you use SLS to put it into the scheduler queue. So you use the SCHED statement to check if this execution of SLS is resulting from a schedule entry in SLSQUE. If not, just STOP to avoid streaming the job.

There are other capabilities of SLS. It is quite possible, for instance, to use SLS in an environment where terminals are managed by process handling. We provide SPL procedures packaged for non-SPL programmers, which do all the interaction with SLS for you. You need only code up a simple module in your terminal-handling program, and you can henceforth implement new job initiators your client can use without recompiling your program or even taking their process structure down!

The scheduling mode of SLS can be used for processing "un-jobs" we call events. An event is simply a processing of an SLS initiator that does things via the COMMAND intrinsic, and never actually streams a job. Every hour, for instance, we have an SLS event that sends beeps to a list of users who have requested that they be put on the "bong" list. Another SLS event is used at the end of the month to produce month-end reports and clear out useage counters.

After over two years of use (only one year with scheduling), what have been our results with SLS? We think it has been very worthwhile.

1. No longer do we forget to run repetitive tasks. When the student labor clerk comes in to do her payroll on the 25th of the month, her worksheet is already printed. That job

never runs while people are using the computer.

2. Our users are weaned from hovering over terminals from which they have run critical executions of programs. They put them into the job stream, then go off and do something productive.

3. With the reliability that comes from SLS's ability to check parameters for reasonableness, our clients are more willing to defer tasks so that they run outside of peak hours. This has helped our mid-afternoon response time greatly.

4. Forty percent of CPU time used by our administrative users in the last six months was done while there was NOBODY doing operations at the computer center. That, friends, was a person and associated family we didn't have to feed. That was a $10 reduction in the per-student cost of attending our school this year. That is a measurable difference in our organization's competitive position.

5. SMC has many dedicated workers. Some of them come in at strange hours during peak periods. I once caught the head of one of our user departments coming in at 3:30 AM. In former times, they had to have a computer person help if they were to get much work done. Now, the college benefits directly from such incidents--SLS preserves their productivity whenever they come in to work. That spreads out the load on our computer, in turn improving response time.

The results of our Computer Service Users Committee have been likewise encouraging. This committee started working in the fall of

1980, and has already accomplished a number of tasks:

1. Developed a working understanding among responsible people in our user departments, of events which affect response time. We have seen substantial efforts in the user departments as a result, to help us maintain good response time.
2. Given the DP management invaluable help in assigning priorities to pending software projects. This has been a two-way street. In some cases we have asked their indulgence while we delay technically demanding projects in favor of some less important but achievable.
3. Reallocated certain resources such as terminals, based on changing needs over our yearly cycle.
4. Begun to lay the groundwork for guaranteeing to the administration that a second CPU if purchased, would be effectively utilized but not "used up" until its depreciation cycle is finished.
5. Guided a subcommittee in the development of a resource allocation plan for students.

Terminal Spooling

In the abstract I promised to describe our experiences with terminal spooling. This topic is closely related to SLS and distributed operations. The final step in giving control to our users is giving them each all the visible elements of a computer. By installing printers in our major client departments, we have completed the task. The key was finding affordable printers. This was done by converting existing printing terminals to spooled devices.

The only change necessary was in the MPE software. Although this feature was announced for the Bruno release of MPE, we have been using it since release 1906. Thereupon hangs a tale...

One day a programmer at American Management Systems was talking to an SE about the problem of getting spool files out on terminals. He asked for some help in using SPOOK to capture spoofles on a printing terminal. The SE thought that was a strange way to do it. Why not just spool the terminal? That was a very significant question. I know a few other people who have asked it, with no response from the MPE lab. This time the right person knew the right thing. The result was a program, contributed on the San Jose swap tape last year, called SPOOLTRM.

There was quite some shock in some of our user departments when we asked if we could fix printing terminals so they couldn't be used to log onto the computer. When they found out that the result was the ability to send output from any other terminal to it, with all the advantages of the spooler, they tried it. We did learn several lessons. As HP begins to support spooling of terminals in the Bruno release, you may wish to take these into account:

1. The payroll people LOVE the fact that they now know that nobody can snoop at their printouts--which never leave the office.
2. They LOVE even more, the convenience. Our computer center is in a building separated by a courtyard from most administrative offices. Now they don't have to go outside to get three lousy pages.

3. We wish there were a simple method of limiting spoofle size to a specific device. But clients can learn pretty fast when their precious office printer is once tied up all afternoon, to route things properly. Again, SLS helps with this by reminding them.
4. Don't let it trouble your conscience if you tell them that a spooled terminal can only run one kind of paper. They will think they are really smart when they get it to do otherwise, and you won't have to put up with complaints when they blow it. But PLEASE don't expect most office workers to be able to understand special forms procedures in the spooler.
5. There is a price to pay--CPU cycles. It takes more of the CPU's time to service a request for another character on a terminal, than it does for that same character on a parallel-interfaced line printer (ie, 2613-2617-2619-2608). This factor is a particular problem for Series II and III users. In these systems, the CPU overhead for character transmission is higher because the ATC is a "dumber" interface than the ADCC used in the series 30/33/44. In a Series III running four spooled terminals at 2400 baud, this could result in a 15% overhead factor for handling terminal interrupts alone. To this one must add MPE overhead, swapping, and disc I/O.

Conclusions

Recently, a department which has the idea it is "terminal-poor" had to give up one of its terminals for a few months. Their first reaction was to unspool the printer. After a staff meeting, however, they elected instead to make one of their CRT's mobile and retain the spooled state of the printer. They would rather share a CRT and occasionally have to go across the office for a lookup or update, than have to run to the computer center for every little printout. The day of the hard copy is by no means over.

Southern Missionary College, though it has only one CPU, has chosen to distribute control of the system to its users. We have done so because we feel there are advantages inherent in such a system. The DP management "control" functions are limited to executing procedures and policies set up by users for their mutual benefit, and packaging the system so that it is understandable to users. We feel this is the best way to get the most for our DP dollar.

We do not know what implications our experience has for systems with distributed intelligence. We see several factors that might encourage us to distribute our processing to multiple CPU's:

1. Physical lack of proximity. This is not currently a problem for us. If, for instance, we were to expand to our Orlando campus, this could be a significant factor.
2. Security. This is currently a prominent driving force towards acquisition of a second CPU. However efficient MPE may be in segregating users, it is impossible to assure our users that mere software can isolate applications as well as an electrical insulator such as two CPU's not interconnected would provide.

3. Dissimilarity of resource needs. However much we may prefer the HP 3000 for our administrative data processing, we are not convinced that it is necessarily the best system for academic processing. The 32,132 limit on adressing is a severe limitation compared to other systems of comparable cost. If the HP 3000 did not come with IMAGE, it would very possibly not be in the running.
4. Dissimilarity of useage patterns. Due to the way system resources are handled in the HP 3000, mixtures of dissimilar use can create severe response time degradation if certain users do certain things (like getting stuck in program loops). We feel that users who insure that they will not do such things, should not be subject to delays because others don't.
5. The HP 3000 can only handle so much of certain things, period. Examples include: directory limitations and excessive overhead in process creation. If you want over 6,000 sectors in your directory, you have to buy another system.

As things now stand, we feel that the HP 3000 as we are currently using it is an effective way of using the financial resources available to us for Data Processing. We have further discovered that a moderate amount of programming (creating SLS) and management effort (the Computer Service Users Committee) have resulted in multiplying its usefulness to our organization.

Programming Standards:

A Tool for Increased Programmer Productivity

Delores R. Payne
Programmer/Analyst
The EMI Companies

Wayne E. Holt
Director, Computer Services
Whitman College

The Orlando Meeting of the HP3000 Users Group

April 1981

Key words: Standards, COBOL, Programming, Productivity

I. Overview

II. The Whitman Approach

III. Conclusions

Supplement. COBOL Coding Standards

Appendix

I Overview

We are, according to the September special edition of ComputerWorld, entering the "Software Decade", a decade that will be marked by the passage of layered software technologies, and replaced by integrated methods that vastly increase productivity and simplify use of the computer.

An ambitious vision of the future to be sure. In the HP3000 world, precompilers, code-generators, ad-hoc report writers, procedural sub-languages, and the like are already appearing on the market. Certainly, this lends credence to the vision of the 1980's. But is that era truly upon us, and will programming as we know it quickly disappear from the face of the earth? we doubt it.

All too often, one of the decisions that upper management will dictate to the DP establishment is, "Thou shalt develop all programs in XXXXX", where XXXXX is frequently COBOL, RPG, or Fortran. They do this in a non-spiteful way, since they believe that they are looking out for the best interests of the company.

After all, COBOL programmers are, for instance, "easy" to obtain on the open market. If your software investment has been in COBOL, then you stand a good chance of keeping it running with people who are "off the street" if a real crisis erupts.

As for the "new software productivity tools", a frequent attitude seems to be, "too new", and, "not as easy or productive as they are advertised to be". Now, we won't debate the merit or spirit of these arguments in this paper - the situation is a fact, and should be coped with intelligently if possible. In fact, coping with this situation is one purpose of this paper.

The title of the paper would have you believe that a programmer productivity tool has been within our grasp since the beginning. That tool is the often maligned creature, the "programming standard". Almost everyone will agree that standards are important, but few can point to a real implementation that actually worked. In fact, many will argue that standards are impossible to create, maintain, or police, and are therefore counterproductive. We take umbrage with this line of reasoning. As will be explained in Section II, the situation at Whitman dictated a serious look at this issue. A comprehensive, 30-month experiment has led us to the conclusion that programming standards, regardless of style, will improve quality and quantity of code, and lower overall costs of development.

The Whitman College Computer Center has been in operation since July of 1977, when the College recognized the need for modern data processing in both the Academic and Administrative areas. The decision was not lightly considered. At that point, one office of the Administration had an NCP 101 that was meagerly shared with several other offices and one Academic department had an IBM 1130 that also served the sciences in a small way.

The HP3000 was purchased with the idea of consolidating services and eliminating old equipment. Being a small college, with a very conservative fiscal policy, a deliberate effort was made to control program development from the very beginning. Vendor-supplied application software was judged inadequate at that time and a decision was made to develop a major area of the Administration "in-house".

In order to deflate the total overall cost of such a decision, a variety of techniques were adopted to expedite coding and assure quality. The results, as evaluated after 30 months, were even better than expected. In summary, a time-honored industry standard of 4 lines of code per hour was decimated by comparison to the effective coding rate of 34 lines of code per hour that was achieved by our staff.

As noted, there were several techniques utilized to achieve this rate. However, the dominant factor was the adoption, at a very early stage, of a set of programming standards. In the four years of the Center's existance, these standards have grown and matured. In other words, they have changed. Yet, this change has not caused the development effort to speed up or slow down. Thus our statement to the effect that having standards is more important than what is in the standards. As a measure of the universality of such a set of standards, ours are distributed to a dozen other sites in the Pacific Northwest on a regular basis.

II The Whitman Approach

Before details of the project at Whitman can be explained, a bit of background on the environment should be related. The College is a four-year, liberal arts institution founded in a very conservative American tradition. This conservatism is pervasive from finances to curriculum. Thus, a project of the magnitude that will be presented was not undertaken lightly.

Some of the constraints on the project are unique to the academic environment. Salaries, for instance, run 30-40% below the average in most metropolitan business sectors. The programming staff at the Computer Center has been characterized as both young and without experience. In context, this is an accurate assessment; it would be erroneous to deduce that these two characteristics imply a lack of either skill or finesse in their chosen profession, however.

For the most part, the staff is, in fact, fairly untrained when hired by the College. They are picked for the job based upon potential rather than experience, since the College is not competitive with industry in the job market and can rarely attract experienced people. Criteria for selection include above-average intelligence, high self-motivation, and goal orientation. So, while the staff may have obvious drawbacks, it nonetheless is primed to produce an above-average quality of program code within minimum time constraints.

In a way, this has been an asset rather than a liability. Staff training has little wasted motion, and the integration into the shop is sped by having no pre-conceived notions and habits to be altered to fit within the College's operating philosophy. Strong standards make such training possible.

In 1973, Datamation published an article, "Chief-Programmer Team for the New York Times", in which it was indicated that about 29 lines of code per manday was an optimum rate for COBOL coding in a batch environment. This is slightly less than 4 lines of code per manhour. This rate has been reaffirmed since then, but always in the batch world.

There are no reputable studies for development on interactive machines, such as the HP3000. There are many issues hidden in the batch vs. interactive environment that make our study invalid as a scientific experiment. However, we will attempt to enumerate the various factors and let you decide which is the most significant.

Many a manager has chewed out the programmer caught keypunching his own cards, regardless of how persuasively the programmer argued that "it was faster to do it myself". "We aren't paying you to keypunch!" is the usual opening line. However, this attitude just doesn't pay when carried over to the interactive environment. At Whitman, this approach was rejected in favor of having a terminal for every person developing code.

In the batch environment, many sharp programmers will duplicate program cards in order to "cut and paste" new programs quickly. In the interactive environment, text editors carry this art to new heights. Well-written programs are ideal templates for building similar software. This not only increases productivity, it also helps train new programmers in proper programming techniques.

This leads of course to programming standards. If everyone follows basically the same rules in designing and building software, then communication among the programmers is greatly facilitated for both training and maintenance.

Communication is a vital link in the development process. A very interesting study in the March 1980 issue of Datamation, "Software Manpower Costs: A Model", states that one programmer working 60 hours a week can complete a project in the same calendar time as two others, but at three-quarters the cost. A bold claim, but one that has some basis in fact. Communication between humans slows down productivity! The more people on a project, the slower each increment of progress for each person. Good standards can bridge the gap in communications. If everyone talks the same lingo, then the need for redundant communication is eliminated.

Whitman has in fact also subscribed to the "work longer" theory in a production mode. This is a further complication in evaluating the reason for rate of code development.

In Figure 1, a breakout is presented for program development between July 1977 and December 1979. During that period, 289 programs were developed in 9595 manhours, consisting of 328,305 lines of program code. This is an average of 34 lines per manhour, or 274 lines per manday. The 30 months were equivalent to 4.6 manyears of effort. Each of the tenniques outlined above were utilized and contributed to the rates achieved.

Whitman College Computer Services

Software Development July 1977-December 1979

	#PROG	#LINES	#HOURS
ADMISSIONS OFFICE			
AD Admissions System	43	33896	980.0
COMPUTER CENTER			
BG Beacon/Guardian	09	13125	338.5
JA Job Accounting	09	11210	292.5
UT Utilities	33	10693	233.5
FINANCIAL AID OFFICE			
FA Financial Aid	14	19542	904.5
FINANCIAL DEVELOPMENT			
AL Alumni Affairs	13	18217	428.5
FD Financial Dev			204.0
GR Gift Records	15	27936	810.5
ML Central Mailing	10	10258	229.0
HOUSING OFFICE			
SH Student Housing	28	25931	727.0
PROVOSTS OFFICE			
BD Budget Status	10	11065	609.0
ER Employee Records			1017.0
REGISTRAR			
CG Class Grading	22	29199	818.5
CR Class Records	45	59335	2029.1
SR Student Records	29	51013	994.0
TREASURERS OFFICE			
SC Securities	08	6885	400.0
< TOTAL >	289	328305	9594.6

Figure 1

III Conclusions

The project at Whitman took advantage of several techniques for increasing programmer productivity, including:

- o programmers have their own terminal for online text editing of source code,
- o cut and paste template techniques were perfected within certain COBOL standards,
- o programmers were encouraged to spend more contiguous time on projects, and
- o comprehensive programming standards were established early in the project.

Without a doubt, programming standards played a vital role in the success of the other three techniques. The standards defined the boundaries of programmer communication, and set the stage for role models and templates to be established.

The standards created for COBOL have been carried over to other languages, such as SPL and Fortran. In the early months of the project, cut and paste models were "whatever could be found" that worked out well previously. Now, formal standard skeletons have been created for all types of standard programs. These skeletons ensure that all code is developed uniformly, and cuts down on re-inventing programming methods.

These techniques can increase productivity in virtually any shop. They meet the needs of those managers who must intone the gospel of the "standard language" to data processing, and assure a more reasonable expenditure for code generated.

When coupled with the new productivity tools, projects developed in the coming decade will certainly live up to the bright promises written in the current journals. At Whitman, this marriage is occurring today. We continue to develop COBOL code for our primary needs, but make strong use of ad-hoc report writers (such as QUIZ) for unique user-requested reports. Small systems of limited life are quickly developed without programming by creating data bases, maintaining them with HPGSUG Contributed software such as DBENTRY, modifying them with ADAGER as needed, and generating reports with QUERY, QUIZ, or REX.

In short, the old adage "use the right tool for the job" will take on more meaning in the future, as more tools are developed to service our needs. We would hope that one of the tools selected to enhance programmer productivity is the creation of shop standards, no matter what size the shop is. It is one of the most effective means available, when one realizes that the cost is, simply, better planning and more efficient usage of resources. That is a price that all shops can afford to pay.

SUPPLEMENT. COBOL Coding Standards

COBOL programming has always emphasized clarity and ease of maintenance, and these objectives form the basis for most standards manuals in the industry today, Careful attention should be paid in their development, however, to avoid standards that are too lax and inconsistent or too restrictive and inflexible. Such an attempt has been made at Whitman College as the standards presented here are designed to be flexible and yet insure some degree of continuity between programs.

The following discussion of Whitman COBOL Coding Standards is divided into two major sections, General Standards and Standards by Division. Those topics that do not fit within a particular COBOL division are included as general standards and they are as follows:

* Compilation Techniques
* Model "cut and paste" Skeletons
* COBOL Copy Library
* Subsystem Calls
* Common Code Table (CCT)
* Error Handling
* Abort Processing
* Console Posting

The four major COBOL divisions contain standards that are specific to a particular division, and they will be included where appropriate.

General COBOL standards will be discussed first, followed by a presentation of the standards by division. Appendix A includes ALL of the examples noted in this section, and will be referenced by the letter A and a number indicating the page of the appendix containing the figure (Ex - see page A-7). All COBOL examples used are from a production program, CG228, that was modified to test the current set of programming standards, and from the V/3000 skeleton program.

General Standards

* Compilation Techniques. In many shops, the days of the interactive compiles are over. The demands on system resources are too great to allow programmers the luxury of a "quick" compile to get a new, fresh listing. This is precisely the case at Whitman. To help insure that compiles are in fact streamed, a new utility program, BANNER (included in the Contributed Library), has been developed to process COBOL compiles. It works in most other languages as well.

  BANNER must be run in a batch mode in order to obtain the information required on the cover sheet about the source, and then insure that it be printed with the source listing. The file equations required for BANNER may be set up in a UDC (see page A-1) to facilitate its use. It creates a cover sheet with large block letters indicating the PROGRAM-ID, and many other pieces of information about the compile including the actual file name used and the User who streamed the job (see page A-1). Actually two identical headers are produced so that cover sheets need never be folded! The information on this cover sheet has also proved invaluable in helping operators deliver the source compile to the proper individual since in many cases involving maintenance the person working on the program is not the original author.

* Model "cut and paste" Skeletons. In order to assist the programmers at Whitman College, special model programs (known as skeletons) have been developed to serve as a base starting point for all new development (see page A-2). The first two pages of a skeleton program are included in the appendix. Note the use of special abbreviations and X's to indicate values that must be entered by the programmer. These skeletons are clean-compiled programs containing each of the four COBOL divisions with all necessary elements and copylib routines normally found in the particular type of program (report, prompt/answer, V/3000, etc). Consequently no one actually ever writes a program from "scratch." A solid foundation is laid for the programmer before any attempt at adding the new program logic is made. This helps keep existing employees on top of the current set of standards in the shop, and helps new employees become familiar with the acceptable coding standards at a much faster rate. These skeletons may also serve as a guide when older programs are modified and standardized, and as a guide to the acceptable level of programming expected in this shop.

* COBOL Copy Library. Whenever possible and practical, COPYLIB routines are created to standardize a set of common variables in working storage or establish a standard paragraph for the procedure division. Special naming conventions are used when naming the routine so that it may be easily identified (see page A-3). Certain standard headings are also required of the COPYLIB routines to insure consistency in the COBOL copy

library, and these fall into several different categories (see page A-4) depending on the type of working storage area. To help insure that COPYLIB standards are followed and understood, one individual in the shop is assigned the responsibility for the maintenance of this file. It is intended that these Copylib routines make the code more consistent and easier to maintain, and reduce as much duplicate effort as possible by providing standard procedures for every programmer to use.

* Subsystem Calls. Special attention is given to standardization of HP specific software subsystems, such as KSAM, V/3000, and IMAGE. The reason most obvious is that the shop cannot afford the time for a "learning curve" for new employees. No matter how well trained in COBOL, few new employees know HP subsystems! The copylib, combined with the skeleton "cut and paste" method, enforces standards while propping up productivity.

  - KSAM. All working storage areas for production KSAM files are standardized in COBOL copylib routines. A standard file format is created for each using proper descriptive prefixes. All production programs access at least one KSAM file, the Common Code Table (CCT), to obtain values for report and screen headings as well as a variety of other codes. Standard lookup procedures have been developed for this special KSAM file since it is so heavily used (see pages A-12, A-13). A sample table, 901 (see page A-11), is included as an example of the file contents. All other KSAM files are referenced in a similar manner, and are built using the same standard format in working storage.

  - V/3000. V/3000 standard screen techniques are heavily used in this shop. As time permits, DEL/3000 programs are being rewritten using the new set of V/3000 standards and all new screen handling programs are being built from a skeleton. A standard V/3000 communications area is included in the COBOL Copylib, and a special VIEW-DATA-BUFFER structure has been developed as well (see page A-5 for a partial picture). The VIEW-DATA-BUFFER has three distinct breakouts. The first contains the system and office names, and the third contains the form name. These three fields must be numbered 1, 2, and 3 on each V/3000 screen in order for the standard heading routines to work properly. That leaves VIEW-DATA-BUFFER-2 as the area for all other screen fields. Using this VIEW-DATA-BUFFER, standard procedures get, display, and read V/3000 screens. The initialization routine is included in the appendix (see page A-6) to show how the first screen is processed. From that point on, standard individual procedures are called to manipulate the V/3000 screens.

  - IMAGE. One of the most heavily used HP subsystems is IMAGE. Fairly specific standards have been established for working storage areas as well as procedure division calls since IMAGE is used in many production programs. Examples of the standard IMAGE working storage buffer and the list and buffer areas for two Student data base data sets are included in appendix A (see pages A-7 and A-8) to give you an idea of the conventions used.

The IMAGE calls used in the procedure division have recently undergone a major change. In the past the practice has been to use ALL-ITEMS as the list item in an IMAGE call. When the most recent set of standards were modified, this practice came under scrutiny and to the most part has been abandoned in favor of the PREVIOUS-LIST option due to greater processing efficiency and ease of maintenance.

Initially the working storage areas of the program must be created containing lists of the desired data elements and the corresponding data buffers containing the same elements (see page A-8). Each IMAGE list is then ready for "initialization". This "initialization" involves a serial read of one record in each data set using the actual LIST as it is defined in working storage (see page A-9). Care must be taken to properly process the first record, however, should any data set actually be read serially. All necessary information is obtained from the IMAGE root file with the read of these initial records, and is stored away for future use. From this point on, all IMAGE calls involving a LIST item should use the IMAGE-PREVIOUS-LIST value (see page A-7) to avoid the overhead of going to the root file to obtain the LIST information with each call (see page A-10).

There are tradeoffs between the ALL-ITEMS and PREVIOUS-LIST options, and they must be considered when deciding how to best use IMAGE in a given program. The following criteria may be used as a guide when a new program is being developed or a program is being modified:

1. execution - by priming all data sets used with the proper IMAGE lists (manually selected lists containing only the data elements necessary) and using the PREVIOUS-LIST option mentioned above, time will be saved during program execution since the root file information is only obtained once.

2. maintenance - by using manually prepared lists instead of ALL-ITEMS, only programs containing affected data items will require recompilation when a data base is rebuilt.

3. additions - when records are being added to a data base, it takes less overhead to use the ALL-ITEMS option since most of the data items are usually included in an add mode anyway,

4. working storage requirements - only one Copylib routine for each data set would be required for the ALL-ITEMS option, but these buffers would be the largest possible and fixed in size. The PREVIOUS-LIST option would necessitate the creation of more Copylib areas, but the buffer sizes in the programs would be optimized.

* Common Code Table. A special KSAM file has been created at Whitman College to house certain "Common Codes". This file contains over fifty tables which house codes from several different systems. These codes are unique and have the same meaning no matter what system they are found in. Some examples are Admissions Office Test Codes, Registrar Student Status Codes, Financial Aid Adjustment Codes, and United States Postal Codes (see page A-11). This standardization of codes simplifies maintenance, and since these are external tables changes do not usually affect the programs. The standard working storage area for the Common Code Table (CCT) and the common routines required to obtain information from this "table" are included in the appendix to show how the CCT is normally accessed (see pages A-12 and A-13).

* Error Handling. Another Common Code Table is used by programmers in the development of error handling procedures. This is a table of standard Error Messages (a partial list is on page A-14). All programs requiring User error messages must access this table as no program-imbedded error messages are allowed. If a programmer does need a new error message, then the analyst responsible for the Common Code Table maintenance must approve the request and make sure that it is entered properly.

To help describe how errors are handled in a screen program, a few sample V/3000 error handling paragraphs are located on page A-15 of the appendix. When an error occurs in a typical V/3000 program, the CCT-ERROR-KEY is primed with the proper error number, the VIEW-FIELD-NUM is primed with the number of the field in error, and error handling procedures are performed. Screen error handling is controlled basically from paragraph P625-VIEW-SECONDARY-ERROR (see page A-15). All routines required to lookup the error message, display it to the User, and flash the field in error are performed from this standard routine. Errors are processed in this manner until all have been corrected.

One key reason for the standardizing of error messages is to facilitate User awareness and understanding of their particular system. Consistent error messages make it easier for the User to become familiar with the types of errors and problems that may occur. Each error message is given a unique number to increase the ease of lookup, and to avoid unnecessary duplication. Unique error messages also insure program and programmer consistency with regard to error handling.

* Abort Processing. The same philosophy of consistency holds for abort messages as well. Standard IMAGE, KSAM, and V/3000 abort COPYLIB routines have been developed to standardize abort procedures and messages. Refer to pages A-16 and A-17 of the appendix to find samples of the general abort working storage area, a typical paragraph calling an abort procedure, and the abort procedure itself. In this case an IMAGE abort is used to illustrate the steps required to perform a standard abort. The abort messages come from the same Common Code Table mentioned above, and are written to the Users screen or job execution report when an abort condition occurs. The messages are designed to help the programmers pinpoint the actual line within a particular paragraph that the abort occurred, and give the User some indication of the problem so that they can notify the computer center with meaningful information. This speeds the identification of the problem, and hence the solution since finding out exactly where something went wrong is often over half the battle.

  In addition to notifying the User of an abort condition, each standard abort routine sends a very obvious message to the operator's console (an entire line of lozenges) so that they are made aware of a problem (see page A-18). This is especially helpful when a batch job aborts since the printer may be busy for hours, thus delaying the printout of a job execution report that tells the operator that the job "blew up". Messages of this type necessitate the existence of a hard copy console, however, to insure that the abort message is "heard" and not lost on a screen that "rolled up" and disappeared. This instant operator notification is another mechanism designed to insure that exception conditions are noticed as early as possible so that proper actions may be taken to recover.

* Console Posting. In another effort to assist the operator in day-to-day activities, a standard set of "TELL" procedures have been developed and are installed in all production programs. These special procedures identify three points in each job process: 1) TELLSTART, 2) TELLFORM and 3) TELLSTOP. Using a special subroutine, these messages are printed in columns 81-132 of the operators console to help distinguish them from MPE messages. Other information such as the User, the file Output number (#O), the form name, and the job/session number are also included in these operator messages to help them readily identify jobs and form requirements (see page A-19).

Standards by Division

* IDENTIFICATION DIVISION

The first three pages in every source program should look fundamentally the same. Special standards have been established for all three pages so that by the time these pages have been read specific pieces of information about the program will be known in some detail. It is too easy to specify a PROGRAM-ID and go onto the ENVIRONMENT DIVISION without the inclusion of any other pertinent information.

To avoid this all too common occurence, a special Title page is included (see page A-20). Using the $TITLE command, the title of the program is printed on the left side of each compile and the PROGRAM-ID and revision level are printed on the right. Every subsequent page will have the title and program number on it unless overridden by a $PAGE heading. Following this important piece of information are:

PROGRAM-ID	program name <revision level>
DATE-WRITTEN	month, year
INSTALLATION	Whitman College
SECURITY	Public, Restricted, or Confidential

To identify as many of the people involved with a particular program as possible, a special author/analyst section is included. It lists the programmer(s), systems analyst(s), and system designer(s). To complete the title page, a brief summary of the program's purpose is given, along with all necessary file descriptions and non-standard subroutine calls.

The second page of a COBOL program (see page A-21) contains detailed compilation instructions which are especially important to programmers unfamiliar with the program. The MAXDATA information is a very important piece of information, and it should be updated properly so that programs are compiled correctly. A maintenance log is also included to identify all of the modifications that have been made and by whom (initials). This can be helpful in understanding the extent of modifications that have been made to a particular program and learning some of its history as well.

One particular piece of information that is often left to the last minute and then left out due to time pressure is the synopsis of program logic (see page A-22). This in reality should be one of the first things written in a program as it can help organize the programmer's logic and possibly pinpoint special problem areas or give insight to better solutions. At any rate, this is a very important standard as it can really help a maintenance programmer get a better feel for what is going on in a program as well as refresh the memory of the author who may have written the program some time ago. Finally, this synopsis should be written in enough detail to give a good logic flow of the program, but be simple enough so that someone unfamiliar with the program can derive a basic understanding from it.

* ENVIRONMENT DIVISION

This division is always the smallest, but nevertheless it contains some fundamental information that should not be overlooked (see page A-23). First of all a $PAGE is used to isolate the division and make it easier to find. Some standard special names are used to refer to the console and for top-of-page.

The Input-Output Section contains the select and assign clauses. A special format is assigned to printer files so that they are easily recognizable on the console during a :SHOWOUT. All other files described in this division will require MPE file equations.

* DATA DIVISION

The most common problem found in data divisions is the inability to find variables without searching the entire working storage section. One of the basic standards used is the $PAGE. It helps delineate particular variables or copylib routines from one another so as to enhance readability. The file and working storage sections are also separated by a $PAGE to further separate them for easier reference.

Another standard developed to increase the ability to find things is the requirement that each variable in working storage be assigned a meaningful prefix (suffixes are acceptable but not preferred) to associate it with other common variables. Each area containing common variables should then be preceded by a header statement (see page A-24) describing what kind of variables are included. Some common areas of differentiation could be FLAGS, HOLD AREAS, TABLES, or other such breakouts.

All working storage copylib routines use the prefix notion in that all variables used in these routines should contain proper three-character prefixes. The copylib routines themselves are then given a file name that also contains the prefix. All of these special naming conventions are used to help keep common variables together to enhance readability, increase variable location, and make the procedure division logic easier to follow with the use of meaningful data names.

In order to make the working storage section even easier to read, all copylib routines and other variable areas should fall in a similar sequence. The following guidelines are used in the order given to arrange all working storage elements:

* all general copylib routines and general variables (flags, counters, save areas),

* communications areas,

* all file descriptions (KSAM, V/3000, IMAGE),

* all copylib report routines should be followed by report format areas and be the last contents of working storage.

A fairly common COBOL standard found in this division is the practice of aligning PICTURE clauses on some predetermined column. The standard used at Whitman stresses that PICTURE clauses line up in column 40 whenever possible. In conjunction with this standard is the practice of indenting data hierarchies in units of 4, with even levels being skipped. Indentation beyond the 05 level is not necessary.

* PROCEDURE DIVISION

At Whitman College, a modular approach to logic coding is used, with heavy emphasis on the PERFORM verb. One major mainline is included at the beginning of the Procedure Division, and the logic here controls the rest of the program. In fact, the program starts, stops, and/or aborts from this single mainline (see page A-25). This approach is used to add an extra measure of control, and to help organize the rest of the program around one common point.

The logic sections within the Procedure Division are meaningful paragraph names that are composed of both sequential numbers and titles. The range is:

001-099	Mainline Logic
100-199	Data File Input
200-299	Data File Output
300-399	Screen Procedures
400-499	Edits and Field Processing
500-599	(Program Specific)
600-699	Error Procedures
700-799	(Program Specific)
800-899	Report Procedures
900-999	Housekeeping.

Each of these logic sections should be preceded by a $PAGE and begin with the three-line standard header which indicates what routines are being performed (see page A-26).

Major program segments should be kept to a minimum, and the logic should remain in a segment as long as possible. This is done to reduce the amount of swapping that occurs each time a different segment has to be brought into memory. Traditionally a COBOL sort program contains three major segments or sections: Mainline Logic Section, Sort Section (input procedure), and Report Section (output procedure). A data entry program often can perform all of its functions out of one section, the Mainline.

As in the Data Division, readability is of key importance. In order to improve readability of this division, only one logic statement should be coded per line. Also standard indentations should be observed in statements with multiple verbs. Arguments in a USING clause are listed on separate lines underneath the first argument to further enhance the code.

Each logic module should be organized into a paragraph which has only one exit and one entrance. Each paragraph should have an enclosed comment preceding it that summarizes the logic found in that logic module. A $PAGE is used to separate the paragraph for better readability.

The use of GO TO's in programming has created much controversy among programmers and proponents of different styles of COBOL coding. The standards developed at Whitman College allow GO TO statements to be used, but only with certain restrictions:

1. One Exit,
2. One Entrance,
3. Intermediate logic leading to 1 or 2,
4. Abort.

No other logic transfers should take place. "Fall through" should always transfer back to the Mainline. Rampant GO TO's that go to different paragraphs and even different sections are not permitted. Instead a controlled use of GO TO statements is proposed to preclude the use of unnecessary nested IF statements, and to make the code simpler and easier to follow.

APPENDIX A

Page

* Compilation Techniques
- BANNER UDC.................................. A-1
- BANNER Cover Sheet for CG228 .... A-1

* Model "cut and paste" Skeleton
- Skeleton Title Page A-2
- Skeleton Second Page A-2

* COBOL Copy Library
- Standard Report Format A-3
- Sample Working Storage Headings............. A-4

* Subsystem Calls

- V/3000 Partial Working Storage Areas A-5
- V/3000 Initial Display Paragraph A-6
- IMAGE Working Storage Area A-7
- Student Data Base Lists and Areas A-8
- IMAGE List Initialization Routine A-9
- Sample IMAGE Calls A-10

* Common Code Table
- Sample Common Code Table - 901 .... A-11
- CCT Working Storage Area A-12
- CCT Input Paragraphs A-13

* Error Handling
- Partial CCT Error Table - 980 .... A-14
- V/3000 Standard Error Paragraphs A-15

* Abort Procedures
- Console Abort Working Storage Area A-16
- IMAGE Paragraph - Calls Abort Procedure ... A-16
- IMAGE Standard Abort Paragrph A-17
- Console Abort Display A-18

* Console Posting
- Console TELLSTART, TELLFORM, TELLSTOP A-19

IDENTIFICATION DIVISION
- Title Page A-20
- Compilation Inst and Maintenance Log A-21
- Partial Synopsis of Program Logic A-22

ENVIRONMENT DIVISION
- Standard Select Clauses A-23

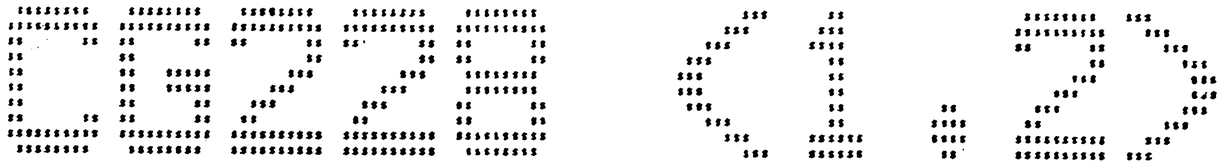
DATA DIVISION
- Standard Variable Naming Convention A-24

PROCEDURE DIVISION
- Mainline Logic A-25
- Sample Paragraph A-26

```
:JOB COMPILE,USER/PASS,ACCT
:FILE COPYLIB=COBOLIB,PUB
:COBBANNER CG228C
:PURGE CG228
:PREP $OLDPASS,CG228
:SAVE CG228
:EOJ
```

```
COBBANNER TEXT,USL=$NEWPASS,LIST=$STDLIST,MAST=$NULL,NEW=$NULL
FILE COBTEXT=!TEXT
FILE COBUSL=!USL
FILE COBLIST=!LIST
FILE COBMAST=!MAST
FILE COBNEW=!NEW
RUN BANNER,UTIL,IRIS;PARM=0
RESET COBTEXT
RESET COBUSL
RESET COBLIST
RESET COBMAST
RESET COBNEW
```

```
                    W H I T M A N   C O L L E G E   C O M P U T E R   C E N T E R
```



```
                                   COBOL Compile for: MANAGER.WCCS.PUB

                                    MON, MAR  2, 1981, 10:39 AM

                                   Source File Name: MODELC.PUB.WCCS



          Compiler File Statistics                                        Source File Statistics

      Formal Name            Actual Name                          Name          : MODELC.PUB.WCCS

      COBTEXT           MODELC.PUB.WCCS                           Creator       : MANAGER

      COBUSL            $NEWPASS                                  LDEV          : 3

      COBLIST           $STDLIST ($STDLIST)                       Filecode      : 1052

      COBMAST           $NULL                                     File Limit    : 1113

      COBNEW            $NULL                                     EOF Pointer   : 1113

                                                                  Max Extents   : 15

                                                                  Ext Size      : 25 Sectors

                                                                  Block Size    : 1280 Bytes

                                                                  Record Size   : 80 Bytes


                                 - A  M.A.R.C.U.S.  PRODUCTION -
```

```
001000$CONTROL USLINIT,SOURCE
001100$TITLE " >> .........1.........2.........3.........4 << ",&
001200$"                                     PPPPP <RRR>"
001300 IDENTIFICATION DIVISION.
001400 PROGRAM-ID.                          "PPPPP <RRR>".
001500 DATE-WRITTEN.                         XXXXXXXX, XXXX.
001600 INSTALLATION.                         WHITMAN COLLEGE.
001700 SECURITY.                             SSSSSSSSSSSS.
001800*
001900*      *--------------------------------------------------------*
002000*      |                 >> NOTICE OF COPYRIGHT <<              |
002100*      *--------------------------------------------------------*
002200*      |    PPPPP is Whitman College Proprietary software       |
002300*      *========================================================*
002400*      |   Programming   | Systems Analysis  | Systems Design   |
002500*      *--------------------------------------------------------*
002600*      |     AAAAA       |      BBBBBB       |     CCCCCCC      |
002700*      *--------------------------------------------------------*
002800*
002900*      *--------------------------------------------------------*
003000*       SUMMARY:  PPPPP XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
003100*                 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
003200*
003300*                 (1)  File Descriptions:
003400*                      UT145K01   Common Code Table
003500*                      FFFFWIDE   Report Print File
003600*
003700*                 (2)  Subroutine calls:
003800*                      UT817      Obtains JOB/USER Name
003900*                      UT818      Console Tell Form Routine
004000*
004100*      *--------------------------------------------------------*
```

```
PAGE 0002   PPPPP <RRR>        >> .........1.........2.........3.........4 <<

004300*      *--------------------------------------------------------*
004400*       Compilation Instructions:
004500*
004600*            :FILE P;DEV=COMLP
004700*            :FILE COPYLIB=COBOLIB,PUB,DEV
004800*            :COBOL PPPPPC,SOURCE,DEV,,*P
004900*            :PREP SOLDPASS,PPPPP,XXXXXXXX,ADMIN;MAXDATA=XXXX
005000*
005100*      *--------------------------------------------------------*
005200*
005300*      *--------------------------------------------------------*
005400*        DATE     REV    BY     MAINTENANCE LOG
005500*      XX/XX/XX <1.0> XXX Primary Installation of software.
005600*
005700*      *--------------------------------------------------------*
```

```
          034600 01  RPT-DUMMY                        COPY RPT021WS.
RPT021WS  001000     .
RPT021WS  001100     03  COPY-CONTROL-BYTE            PIC X(01).
RPT021WS  001200*    *-----------------------------------------------------------*
RPT021WS  001300*          Standard Report Print Routine Common Formats
RPT021WS  001400*    *-----------------------------------------------------------*
RPT021WS  001410 01  RPT-STD-LOGON.
RPT021WS  001420     03  RPT-STD-SESSION-NBR          PIC X(06).
RPT021WS  001430     03  FILLER                       PIC X(01)  VALUE "/".
RPT021WS  001440     03  RPT-STD-JOB-NBR              PIC X(06).
RPT021WS  001500 01  RPT-SUB-HEAD-1                   PIC X(132) VALUE SPACES.
RPT021WS  001600 01  RPT-SUB-HEAD-2                   PIC X(132) VALUE SPACES.
RPT021WS  001700 01  RPT-SUB-HEAD-3                   PIC X(132) VALUE SPACES.
RPT021WS  001800 01  RPT-COL-HEAD-1                   PIC X(132) VALUE SPACES.
RPT021WS  001900 01  RPT-COL-HEAD-2                   PIC X(132) VALUE SPACES.
RPT021WS  002000 01  RPT-COL-HEAD-3                   PIC X(132) VALUE SPACES.
RPT021WS  002100 01  RPT-PRINT-LINE-OUT.
RPT021WS  002200     03  RPT-NARROW-PRINT-LINE        PIC X(96)  VALUE SPACES.
RPT021WS  002300     03  FILLER                       PIC X(36)  VALUE SPACES.
RPT021WS  002400 01  RPT-HOLD-PRINT-LINE.
RPT021WS  002500     03  RPT-NARROW-HOLD-LINE         PIC X(96)  VALUE SPACES.
RPT021WS  002600     03  FILLER                       PIC X(36)  VALUE SPACES.
RPT021WS  002700 01  RPT-PAGE-COUNT                   PIC 9(05)  VALUE 0.
RPT021WS  002800 01  RPT-LINE-COUNT                   PIC 9(02)  VALUE 99.
RPT021WS  002900 01  RPT-LINE-SLEW                    PIC 9(02)  VALUE 1.
RPT021WS  003000 01  RPT-HOLD-SLEW                    PIC 9(02)  VALUE 0.
RPT021WS  003100 01  RPT-TOP-CODE                     PIC X(01)  VALUE SPACE.
RPT021WS  003200     88  RPT-TOP-OF-PAGE-NEEDED                  VALUE "T".
RPT021WS  003300 01  RPT-SWITCHES.
RPT021WS  003400     03  RPT-SW-SUB-1                 PIC 9(01)  VALUE 0.
RPT021WS  003500         88  RPT-SUB-HEAD-1-NEEDED               VALUE 1.
RPT021WS  003600     03  RPT-SW-SUB-2                 PIC 9(01)  VALUE 0.
RPT021WS  003700         88  RPT-SUB-HEAD-2-NEEDED               VALUE 1.
RPT021WS  003800     03  RPT-SW-SUB-3                 PIC 9(01)  VALUE 0.
RPT021WS  003900         88  RPT-SUB-HEAD-3-NEEDED               VALUE 1.
RPT021WS  004000     03  RPT-SW-COL-1                 PIC 9(01)  VALUE 0.
RPT021WS  004100         88  RPT-COLUMN-LINE-1-NEEDED            VALUE 1.
RPT021WS  004200     03  RPT-SW-COL-2                 PIC 9(01)  VALUE 0.
RPT021WS  004300         88  RPT-COLUMN-LINE-2-NEEDED            VALUE 1.
RPT021WS  004400     03  RPT-SW-COL-3                 PIC 9(01)  VALUE 0.
RPT021WS  004500         88  RPT-COLUMN-LINE-3-NEEDED            VALUE 1.
```

```
         027200 01   CCT-DUMMY                          COPY CCT001WS.
CCT001WS 001000          .
CCT001WS 001100          03  COPY-CONTROL-BYTE          PIC X(01).
CCT001WS 001200*          *-------------------------------------------------*
CCT001WS 001300*           FILE FORMAT: UT145K01--Common Codes Table file
CCT001WS 001400*          *-------------------------------------------------*
CCT001WS 001500 01   CCT-BASIC-RECORD.
CCT001WS 001600          03  CCT-KEY                    PIC X(15).
CCT001WS 001700          03  CCT-VALUE                  PIC X(50).


         028400 01   CDB-COM-DUMMY                      COPY CDB001WS.
CDB001WS 001000          .
CDB001WS 001100          03  COPY-CONTROL-BYTE   PIC X(01).
CDB001WS 001200*         *--------------------------------------------------------*
CDB001WS 001300*                   Data Base COMMON -- Communications Area
CDB001WS 001400*         *--------------------------------------------------------*
CDB001WS 001500 01   CDB-COMMON-BASE-INFO.
CDB001WS 001600      03  CDB-BASE-NAME          PIC X(16) VALUE "  COMMON.COMMON;".
CDB001WS 001700      03  CDB-PASSWORD           PIC X(08) VALUE ";        ".
CDB001WS 001800 01   CDB-COMMON-DATA-SETS.
CDB001WS 001900      03  CDB-UIC-MST            PIC X(16) VALUE "UIC-MASTER;     ".
CDB001WS 002000      03  CDB-WID-MST            PIC X(16) VALUE "WID-MASTER;     ".
CDB001WS 002100      03  CDB-ATTRIBUTE-MST      PIC X(16) VALUE "ATTRIBUTE-MASTER".
CDB001WS 002200      03  CDB-ZIP-CODE-MST       PIC X(16) VALUE "ZIP-CODE-MASTER;".
CDB001WS 002300      03  CDB-ADDRESS-DTL        PIC X(16) VALUE "ADDRESS-DETAIL; ".
CDB001WS 002400      03  CDB-ATTRIBUTE-DTL      PIC X(16) VALUE "ATTRIBUTE-POINTR


         028500*
         028600*     *--------------------------------------------------------*
         028700*                          CDB001L2 --- LIST #2
         028800*       WID-MASTER Master Data Set              Data Base COMMON
         028900*     *--------------------------------------------------------*
         029000 01   CDB001L2-LIST.
         029100      03  FILLER                 PIC X(50) VALUE
         029200          "ID-PREFIX;                                             "
         029300 01   CDB001L2-AREA.
         029400      03  CDB-ID-PREFIX          PIC 9(03) COMP.




         038000*     *--------------------------------------------------------*
         038100*                     Report CG228/F482 Format Area
         038200*     *--------------------------------------------------------*
         038300 01   F482-SYSTEM                        PIC X(50).
         038400 01   F482-REPORT                        PIC X(10) VALUE "CG228/F482".
         038500 01   F482-PRIVACY                       PIC X(14) VALUE "RESTRICTED".
         038600 01   F482-TITLE-3.
         038700      03  F482-TITLE-LITERAL             PIC X(30).
         038800      03  FILLER                         PIC X(05) VALUE " for ".
         038900      03  F482-LITERAL                   PIC X(15) VALUE SPACES.
         039000 01   F482-OFFICE                        PIC X(50).
```

```
          013900*     *--------------------------------------------------------*
          014000*                  VIEW/3000 Screen Buffers
          014100*     *--------------------------------------------------------*
          014200 01   VIEW-DATA-BUFFER.
          014300      02  VIEW-DATA-BUFFER-1.
          014400          03  DATA-SYSTEM-NAME       PIC X(50).
          014500          03  DATA-OFFICE-NAME       PIC X(50).
          014600      02  VIEW-DATA-BUFFER-2.
          014700          03  DATA-ACTION            PIC X(01).
          014800              88  END-OF-JOB                   VALUE "!".
          014900              88  VALID-ACTION                 VALUE "A" "C" "D".
          015000              88  ADD-ACTION                   VALUE "A".
          015100              88  CHANGE-ACTION                VALUE "C".
          015200              88  DELETE-ACTION                VALUE "D".
          015300          03  DATA-ID-CODE           PIC X(06).
          015400      02  VIEW-DATA-BUFFER-3.
          015500          03  DATA-FORM-TITLE        PIC X(50).
          015600          03  DATA-VALIDATE          PIC X(01).
          015700              88  DATA-IS-VALID                VALUE "/".

VEW021WS  001200*     *--------------------------------------------------------*
VEW021WS  001300*          VIEW/3000 Communications and Parameter Area
VEW021WS  001400*     *--------------------------------------------------------*
VEW021WS  001500 01   VIEW-ERROR-MSG.
VEW021WS  001600      03  VIEW-ERROR-CODE             PIC X(15) VALUE SPACES.
VEW021WS  001700      03  FILLER                      PIC X(57) VALUE SPACES.
VEW021WS  001800 01   VIEW-FIELD-NUM                  PIC S9(04) COMP VALUE 0.
VEW021WS  001900 01   VIEW-NEXT-FLD-NUM               PIC S9(04) COMP VALUE 0.
VEW021WS  002000 01   VIEW-ERROR-FIELD-NUM            PIC S9(04) COMP VALUE 0.
VEW021WS  002100 01   VIEW-LENWINDOWMSG               PIC S9(04) COMP VALUE 0.
VEW021WS  002200 01   VIEW-ACTUAL-LEN                 PIC S9(04) COMP VALUE 0.
VEW021WS  002300 01   VIEW-LENFLDBUFF                 PIC S9(04) COMP VALUE 0.
VEW021WS  002400 01   VIEW-LENDATABUFF                PIC S9(04) COMP VALUE 0.
VEW021WS  002500 01   VIEW-LENERRMSG                  PIC S9(04) COMP VALUE 0.
VEW021WS  002600 01   VIEW-LENERRBUFF                 PIC S9(04) COMP VALUE 0.
VEW021WS  002700 01   VIEW-MESSAGE-BUFFER.
VEW021WS  002800      03  FILLER                      PIC X(13) VALUE
VEW021WS  002900          " X c &a23r20C".
VEW021WS  003000      03  VIEW-MSG-ENHANCEMENT        PIC X(04) VALUE SPACES.
VEW021WS  003100      03  VIEW-MSG-BUFF               PIC X(50) VALUE SPACES.
VEW021WS  003200      03  FILLER                      PIC X(04) VALUE " b W".
VEW021WS  003300 01   VIEW-MESSAGE-BUFFER-2.
VEW021WS  003400      03  VIEW-MSG-ENHANCEMENT-2      PIC X(04) VALUE SPACES.
VEW021WS  003500      03  VIEW-MSG-BUFF-2             PIC X(72) VALUE SPACES.
VEW021WS  003600 01   VIEW-STANDARD-ENHANCEMENT       PIC X(04) VALUE " &dJ".
VEW021WS  003700 01   VIEW-ERROR-ENHANCEMENT          PIC X(04) VALUE " &dF".
VEW021WS  003800 01   VIEW-CLEAR-ENHANCEMENT          PIC X(04) VALUE " &d@".
VEW021WS  003900 01   VIEW-DUMMY-PARAM                PIC X(02).
VEW021WS  004000 01   VIEW-TERM-NAME                  PIC X(09) VALUE "TERMINAL".
VEW021WS  004100*
VEW021WS  004200 01   VIEW-COM-AREA.
VEW021WS  004300      03  VIEW-STATUS                 PIC S9(04) COMP VALUE 0.
VEW021WS  004400          88  VIEW-OP-VALID          VALUE 0.
VEW021WS  004500      03  VIEW-LANGUAGE               PIC S9(04) COMP VALUE 0.
VEW021WS  004600      03  VIEW-COM-AREA-LENGTH        PIC S9(04) COMP VALUE 60.
VEW021WS  004700      03  FILLER                      PIC S9(04) COMP VALUE 0.
VEW021WS  004800      03  VIEW-CURRENT-MODE           PIC S9(04) COMP VALUE 0.
VEW021WS  004900      03  VIEW-LAST-KEY               PIC S9(04) COMP VALUE 0.
```

```
         036700*----------------*----------------------------------------*--------------
         036800*----------------*       Screen Procedures               *--------------
         036900*----------------*----------------------------------------*--------------
         037000*
         037100 P300-DUMMY. COPY VEW300P2.
VEW300P2 001000*
VEW300P2 001100*     *-----------------------------------------------------------*
VEW300P2 001200*       This routine displays the View/3000 form to the screen
VEW300P2 001300*       and performs the initial read of the screen.
VEW300P2 001400*     *-----------------------------------------------------------*
VEW300P2 001600 P300-VIEW-DISPLAY-FORM.
VEW300P2 001700     CALL "VGETNEXTFORM" USING VIEW-COM-AREA.
VEW300P2 001800     IF  NOT VIEW-OP-VALID
VEW300P2 001900         MOVE "925??P300A" TO GEN-ABORT-BREAKOUT,
VEW300P2 002000         PERFORM P658-VIEW-STANDARD-ERROR THRU P658-EXIT,
VEW300P2 002100         GO TO P099-ABORT.
VEW300P2 002200 P300-INIT-FORM.
VEW300P2 002300     CALL "VINITFORM" USING VIEW-COM-AREA.
VEW300P2 002400     IF  NOT VIEW-OP-VALID
VEW300P2 002500         MOVE "925??P300B" TO GEN-ABORT-BREAKOUT,
VEW300P2 002600         PERFORM P658-VIEW-STANDARD-ERROR THRU P658-EXIT,
VEW300P2 002700         GO TO P099-ABORT.
VEW300P2 002800 P300-SHOWFORM.
VEW300P2 002900     PERFORM P375-VIEW-SCREEN-TITLES THRU P375-EXIT.
VEW300P2 003000     CALL "VSHOWFORM" USING VIEW-COM-AREA.
VEW300P2 003100     IF  NOT VIEW-OP-VALID
VEW300P2 003200         MOVE "925??P300C" TO GEN-ABORT-BREAKOUT,
VEW300P2 003300         PERFORM P658-VIEW-STANDARD-ERROR THRU P658-EXIT,
VEW300P2 003400         GO TO P099-ABORT.
VEW300P2 003500 P300-READ-FORM.
VEW300P2 003600     CALL "VREADFIELDS" USING VIEW-COM-AREA.
VEW300P2 003700     IF  NOT VIEW-OP-VALID
VEW300P2 003800         MOVE "925??P300D" TO GEN-ABORT-BREAKOUT,
VEW300P2 003900         PERFORM P658-VIEW-STANDARD-ERROR THRU P658-EXIT,
VEW300P2 004000         GO TO P099-ABORT.
VEW300P2 004100 P300-GET-BUFFER.
VEW300P2 004200     CALL "VGETBUFFER" USING VIEW-COM-AREA
VEW300P2 004300                             VIEW-DATA-BUFFER
VEW300P2 004400                             VIEW-LENDATABUFF.
VEW300P2 004500     IF  NOT VIEW-OP-VALID
VEW300P2 004600         MOVE "925??P300E" TO GEN-ABORT-BREAKOUT,
VEW300P2 004700         PERFORM P658-VIEW-STANDARD-ERROR THRU P658-EXIT,
VEW300P2 004800         GO TO P099-ABORT.
VEW300P2 004900 P300-EXIT.
VEW300P2 005000     EXIT.
```

```
         028200 01  IMAGE-COM-DUMMY                COPY IMG021WS.
IMG021WS 001000
IMG021WS 001100    03  COPY-CONTROL-BYTE            PIC X(01).
IMG021WS 001200*   *-----------------------------------------------------------*
IMG021WS 001300*          IMAGE/3000 Communications and Parameter Area
IMG021WS 001400*   *-----------------------------------------------------------*
IMG021WS 001500 01  IMAGE-LENERRBUF                 PIC S9(04) USAGE COMP.
IMG021WS 001600 01  IMAGE-STATUS.
IMG021WS 001700         03  IMAGE-CONDITION         PIC S9(04) USAGE COMP.
IMG021WS 001800             88  IMAGE-OP-VALID                 VALUE  0.
IMG021WS 001900             88  IMAGE-BAD-PASSWORD             VALUE -21.
IMG021WS 002000             88  IMAGE-EOF                      VALUE 11.
IMG021WS 002100             88  IMAGE-RECORD-NOT-FOUND         VALUE 17.
IMG021WS 002200             88  IMAGE-END-OF-CHAIN             VALUE 15.
IMG021WS 002300             88  IMAGE-BEGINNING-OF-CHAIN       VALUE 14.
IMG021WS 002400         03  IMAGE-WORD2             PIC S9(04) USAGE COMP.
IMG021WS 002500         03  IMAGE-WORD3-4           PIC S9(09) USAGE COMP.
IMG021WS 002600         03  IMAGE-WORD5-6           PIC S9(09) USAGE COMP.
IMG021WS 002610             88  IMAGE-NO-DETAILS-FOUND         VALUE 0.
IMG021WS 002700         03  IMAGE-WORD7-8           PIC S9(09) USAGE COMP.
IMG021WS 002800         03  IMAGE-WORD9-10          PIC S9(09) USAGE COMP.
IMG021WS 002900 01  IMAGE-MODES.
IMG021WS 003000         03  IMAGE-MODE1             PIC S9(04) USAGE COMP VALUE 1.
IMG021WS 003100         03  IMAGE-MODE2             PIC S9(04) USAGE COMP VALUE 2.
IMG021WS 003200         03  IMAGE-MODE3             PIC S9(04) USAGE COMP VALUE 3.
IMG021WS 003300         03  IMAGE-MODE4             PIC S9(04) USAGE COMP VALUE 4.
IMG021WS 003400         03  IMAGE-MODE5             PIC S9(04) USAGE COMP VALUE 5.
IMG021WS 003500         03  IMAGE-MODE6             PIC S9(04) USAGE COMP VALUE 6.
IMG021WS 003600         03  IMAGE-MODE7             PIC S9(04) USAGE COMP VALUE 7.
IMG021WS 003700         03  IMAGE-MODE8             PIC S9(04) USAGE COMP VALUE 8.
IMG021WS 003800         03  IMAGE-MODE9             PIC S9(04) USAGE COMP VALUE 9.
IMG021WS 003900 01  IMAGE-ERROR-MSG                 PIC  X(72).
IMG021WS 004000 01  IMAGE-DUMMY-PARAM               PIC S9(04) USAGE COMP.
IMG021WS 004100 01  IMAGE-ALL-ITEMS                 PIC  X(02) VALUE "@;".
IMG021WS 004200 01  IMAGE-NULL-ITEMS                PIC  X(02) VALUE "0;".
IMG021WS 004300 01  IMAGE-PREVIOUS-LIST             PIC  X(02) VALUE "*;".
IMG021WS 004400 01  IMAGE-DSET-NAME                 PIC  X(16) VALUE SPACES.
IMG021WS 004500 01  IMAGE-KEY                       PIC  X(16) VALUE SPACES.
IMG021WS 004600 01  IMAGE-ARGUMENT                  PIC  X(10) VALUE SPACES.
```

```
*       *-------------------------------------------------------*
*                            STU008L2 --- LIST #2
*        SEM-CREDIT-DTL Detail Data Set          Data Base STU
*       *-------------------------------------------------------*
 01     STU008L2-LIST,
        03  FILLER                    PIC X(50) VALUE
            "SEMESTER-KEY,CUM-F-CREDITS,CUM-GPA,TOTL-CREDIT-EAR",
        03  FILLER                    PIC X(50) VALUE
            "N;                                                ",
 01     STU008L2-AREA.
        03  STU-SEMESTER-KEY          PIC 9(03) COMP.
        03  STU-CUM-F-CREDITS         PIC 9(02) COMP.
        03  STU-CUM-GPA               PIC 9V999 COMP.
        03  STU-TOTL-CREDIT-EARN      PIC 9(04) COMP.
*       *-------------------------------------------------------*
*                            STU012L2 --- LIST #2
*        REG-STUDENT-DET Detail Data Set         Data Base STU
*       *-------------------------------------------------------*
 01     STU012L2-LIST.
        03  FILLER                    PIC X(50) VALUE
            "WHITMAN-ID,STUDENT-LEVEL,NAME,STUDENT-STATUS,PROJ-",
        03  FILLER                    PIC X(50) VALUE
            "GRAD-MO,PROJ-GRAD-YR,CLASS-RANK,CLASS-SIZE,UPDATE-",
        03  FILLER                    PIC X(50) VALUE
            "DATE,UPDATE-TIME;                                 ",
 01     STU012L2-AREA.
        03  STU-WHITMAN-ID            PIC X(06).
        03  STU-STUDENT-LEVEL         PIC X(01).
        03  FILLER                    PIC X(01).
        03  STU-NAME                  PIC X(30).
        03  STU-STUDENT-STATUS        PIC X(02).
        03  STU-PROJ-GRAD-MO          PIC X(02).
        03  STU-PROJ-GRAD-YR          PIC X(02).
        03  STU-CLASS-RANK            PIC 9(03) COMP.
        03  STU-CLASS-SIZE            PIC 9(03) COMP.
        03  STU-UPDATE-DATE           PIC X(08).
        03  STU-UPDATE-TIME           PIC X(06).
 01     STU-REG-LOCK-DESCRIPTOR.
        03  STU-REG-LOCK-DESC-NBR     PIC S9(04) COMP VALUE 1.
        03  STU-REG-LOCK-DESC-1.
            05  STU-REG-LOCK-LENGTH  PIC S9(04) COMP VALUE 21.
            05  STU-REG-LOCK-DSET    PIC X(16)  VALUE
                "REG-STUDENT-DET;".
            05  STU-REG-LOCK-ITEM    PIC X(16)  VALUE
                "WHITMAN-ID;     ".
            05  STU-REG-LOCK-RELOP   PIC X(02)  VALUE "= ".
            05  STU-REG-LOCK-VALUE   PIC X(06)  VALUE SPACES.
```

```
*         *-------------------------------------------------------*
*          This routine establishes the IMAGE lists for the
*          following data sets:  WID-MASTER, REG-STUDENT-DET, and
*          SEM-CREDIT-DTL.  The previous list option will then
*          be used in all subsequent IMAGE calls.
*         *-------------------------------------------------------*
 P951-ESTABLISH-IMAGE-LISTS.
     CALL "DBGET"   USING CDB-BASE-NAME
                          CDB-WID-MST
                          IMAGE-MODE2
                          IMAGE-STATUS
                          CDB001L2-LIST
                          CDB001L2-AREA
                          IMAGE-DUMMY-PARAM.
     IF  NOT IMAGE-OP-VALID
         MOVE "92598P951A" TO GEN-ABORT-BREAKOUT,
         GO TO P951-ERROR-OUT.
     CALL "DBGET"   USING STU-BASE-NAME
                          STU-SEM-CRED-DTL
                          IMAGE-MODE2
                          IMAGE-STATUS
                          STU008L2-LIST
                          STU008L2-AREA
                          IMAGE-DUMMY-PARAM.
     IF  NOT IMAGE-OP-VALID
         MOVE "92598P951B" TO GEN-ABORT-BREAKOUT,
         GO TO P951-ERROR-OUT.
     CALL "DBGET"   USING STU-BASE-NAME
                          STU-REG-STUD-DTL
                          IMAGE-MODE2
                          IMAGE-STATUS
                          STU012L2-LIST
                          STU012L2-AREA
                          IMAGE-DUMMY-PARAM.
     IF  NOT IMAGE-OP-VALID
         MOVE "92598P951C" TO GEN-ABORT-BREAKOUT,
         GO TO P951-ERROR-OUT.
     GO TO P951-EXIT.
 P951-ERROR-OUT.
     PERFORM P645-IMAGE-STANDARD-ERROR THROUGH P645-EXIT.
     GO TO P099-ABORT.
 P951-EXIT.
     EXIT.
```

```
*      *--------------------------------------------------------*
*           This routine locks the STUDENT RECORDS Data Base.
*      *--------------------------------------------------------*
 P250-LOCK-STU.
     CALL "DBLOCK" USING STU-BASE-NAME
                         STU-REG-LOCK-DESCRIPTOR
                         IMAGE-MODE5
                         IMAGE-STATUS.
 P250-EXIT.
     EXIT.
*
*      *--------------------------------------------------------*
*          This routine unlocks the STUDENT RECORDS Data Base.
*      *--------------------------------------------------------*
 P255-UNLOCK-STU.
     CALL "DBUNLOCK" USING STU-BASE-NAME
                           IMAGE-DUMMY-PARAM
                           IMAGE-MODE1
                           IMAGE-STATUS.
 P255-EXIT.
     EXIT.
*
*      *--------------------------------------------------------*
*          This routine updates the CLASS-RANK and the CLASS-
*          SIZE on the REG-STUD-DTL.
*      *--------------------------------------------------------*
 P270-UPDATE-STU-DTL.
     CALL "DBUPDATE" USING STU-BASE-NAME
                           STU-REG-STUD-DTL
                           IMAGE-MODE1
                           IMAGE-STATUS
                           IMAGE-PREVIOUS-LIST
                           STU012L2-AREA.
 P270-EXIT.
     EXIT.
```

COMMON CODE TABLE 901
United States Postal Codes

KEY	BASIC VALUE
901AK	ALASKA
901AL	ALABAMA
901AR	ARKANSAS
901AZ	ARIZONA
901CA	CALIFORNIA
901CO	COLORADO
901CT	CONNECTICUT
901DC	DISTRICT OF COLUMBIA
901DE	DELAWARE
901FL	FLORIDA
901GA	GEORGIA
901HI	HAWAII
901IA	IOWA
901ID	IDAHO
901IL	ILLINOIS
901IN	INDIANA
901KS	KANSAS
901KY	KENTUCKY
901LA	LOUISIANA
901MA	MASSACHUSETTS
901MD	MARYLAND
901ME	MAINE
901MI	MICHIGAN
901MN	MINNESOTA
901MO	MISSOURI
901MS	MISSISSIPPI
901MT	MONTANA
901NC	NORTH CAROLINA
901ND	NORTH DAKOTA
901NE	NEBRASKA
901NH	NEW HAMPSHIRE
901NJ	NEW JERSEY
901NM	NEW MEXICO
901NV	NEVADA
901NY	NEW YORK
901OH	OHIO
901OK	OKLAHOMA
901OR	OREGON
901PA	PENNSYLVANIA
901RI	RHODE ISLAND
901SC	SOUTH CAROLINA
901SD	SOUTH DAKOTA
901TN	TENNESSEE
901TX	TEXAS
901UT	UTAH
901VA	VIRGINIA
901VT	VERMONT
901WA	WASHINGTON
901WI	WISCONSIN
901WV	WEST VIRGINIA
901WY	WYOMING

```
         027200 01   CCT-DUMMY                        COPY CCT001WS.
CCT001WS 001000      .
CCT001WS 001100      03  COPY-CONTROL-BYTE            PIC X(01).
CCT001WS 001200*      *-----------------------------------------------*
CCT001WS 001300*       FILE FORMAT: UT145K01--Common Codes Table file
CCT001WS 001400*      *-----------------------------------------------*
CCT001WS 001500 01   CCT-BASIC-RECORD.
CCT001WS 001600          03  CCT-KEY                  PIC X(15).
CCT001WS 001700          03  CCT-VALUE                PIC X(50).
CCT001WS 001800 01   CCT-FILE-TABLE.
CCT001WS 001900          03  CCT-FILE-NUMBR        PIC S9(04) VALUE 0 USAGE COMP.
CCT001WS 002000          03  CCT-FILE-NAME         PIC  X(08) VALUE "UT145K01".
CCT001WS 002100          03  CCT-FILE-I-O          PIC S9(04) VALUE 0 USAGE COMP.
CCT001WS 002200          03  CCT-FILE-ACCESS       PIC S9(04) VALUE 2 USAGE COMP.
CCT001WS 002300          03  CCT-FILE-PREV-OP      PIC S9(04) VALUE 0 USAGE COMP.
CCT001WS 002400 01   CCT-KEY-LENGTH          PIC S9(04) VALUE 15 USAGE COMP.
CCT001WS 002500 01   CCT-REC-LENGTH          PIC S9(04) VALUE 65 USAGE COMP.
CCT001WS 002600 01   CCT-KEY-LOCATION        PIC S9(04) VALUE  1 USAGE COMP.
CCT001WS 002700 01   HARDWIRED-LOOKUP-ERRORS.
CCT001WS 002800      02  CCT-KSAM-GENERAL-ERROR.
CCT001WS 002900          03  FILLER                   PIC X(34) VALUE
CCT001WS 003000             "ERROR 801 - KSAM FILE SYSTEM ERROR".
CCT001WS 003100          03  CCT-KSAM-ERROR-NBR       PIC 9(04).
CCT001WS 003200          03  FILLER                   PIC X(12) VALUE SPACES.
CCT001WS 003300      02  TERM-ACCESS-ERROR.
CCT001WS 003400          03  FILLER                   PIC X(34) VALUE
CCT001WS 003500              "Error 707 - Terminal Access Error ".
CCT001WS 003600          03  ERROR-CODE-1             PIC -9(04).
CCT001WS 003700          03  FILLER                   PIC X(11) VALUE SPACES.
CCT001WS 003800      02  FORM-FILE-ERROR.
CCT001WS 003900          03  FILLER                   PIC X(12) VALUE
CCT001WS 004000              "Error 701 - ".
CCT001WS 004100          03  FORM-FILE-MSG            PIC X(14).
CCT001WS 004200          03  FILLER                   PIC X(24) VALUE
CCT001WS 004300              " is not a form file      ".
CCT001WS 004400      02  FILE-FORM-ERROR.
CCT001WS 004500          03  FILLER                   PIC X(28) VALUE
CCT001WS 004600              "Error 712 - Form File Error ".
CCT001WS 004700          03  ERROR-CODE-2             PIC -9(04).
CCT001WS 004800          03  FILLER                   PIC X(17) VALUE SPACES.
CCT001WS 004900      02  CCT-EMERGENCY.
CCT001WS 005000          03  FILLER                   PIC X(32) VALUE
CCT001WS 005100              "Error 924 - Error Msg not in CCT".
CCT001WS 005200          03  FILLER                   PIC X(18) VALUE SPACES.
CCT001WS 005300 01   CCT-ERROR-MSG                    PIC X(50).
CCT001WS 005400 01   CCT-MATCH-KEY                    PIC X(15)      VALUE SPACES
CCT001WS 005500 01   CCT-REDEFINE-KEY-1 REDEFINES CCT-MATCH-KEY.
CCT001WS 005600      03   CCT-ERROR-KEY               PIC X(08).
CCT001WS 005700      03   FILLER                      PIC X(07).
         027300 01   CCT-REDEFINE-KEY-2 REDEFINES CCT-MATCH-KEY.
         027400      03  CCT-TABLE-KEY                PIC 9(03).
         027500      03  CCT-2-CHAR-KEY               PIC X(02).
         027600      03  FILLER                       PIC X(10).
         027700 01   CCT-REDEFINE-KEY-4 REDEFINES CCT-MATCH-KEY.
         027800      03  FILLER                       PIC X(03).
         027900      03  CCT-4-CHAR-KEY               PIC X(04).
         028000      03  FILLER                       PIC X(08).
```

```
*-----------------*-------------------------------*---------------
*-----------------*         INPUT ROUTINES        *---------------
*-----------------*-------------------------------*---------------
*
*    *------------------------------------------------------------*
*      This routine will look up the necessary titles from
*      the following Common Code Table: 983, 985, and 987.
*      These titles will be used to prime the report headings.
*    *------------------------------------------------------------*
 P100-READ-CCT-TITLES.
     MOVE 983                      TO CCT-TABLE-KEY.
     MOVE "CG"                     TO CCT-2-CHAR-KEY.
     PERFORM P105-CCT-INPUT THROUGH P105-EXIT.
     IF  GEN-ERROR-FOUND
         MOVE 0                    TO GEN-ERROR-FLAG,
         MOVE SPACES               TO CCT-VALUE.
     MOVE CCT-VALUE                TO F482-SYSTEM.
     MOVE 985                      TO CCT-TABLE-KEY.
     MOVE "64"                     TO CCT-2-CHAR-KEY.
     PERFORM P105-CCT-INPUT THROUGH P105-EXIT.
     IF  GEN-ERROR-FOUND
         MOVE 0                    TO GEN-ERROR-FLAG,
         MOVE SPACES               TO CCT-VALUE.
     MOVE CCT-VALUE                TO F482-OFFICE.
     MOVE 987                      TO CCT-TABLE-KEY.
     MOVE "F482"                   TO CCT-4-CHAR-KEY.
     PERFORM P105-CCT-INPUT THROUGH P105-EXIT.
     IF  GEN-ERROR-FOUND
         MOVE 0                    TO GEN-ERROR-FLAG,
         MOVE SPACES               TO CCT-VALUE.
     MOVE CCT-VALUE                TO F482-TITLE-LITERAL.
 P100-EXIT.
     EXIT.
 P105-DUMMY. COPY CCT105P2.
CCT105P2  *    *------------------------------------------------------------*
CCT105P2  *      This routine will read the Common Code Table based
CCT105P2  *      on the key fields defined during data field edits.
CCT105P2  *    *------------------------------------------------------------*
CCT105P2   P105-CCT-INPUT.
CCT105P2       MOVE ZERO TO GEN-ERROR-FLAG.
CCT105P2       CALL "CKREADBYKEY" USING CCT-FILE-TABLE
CCT105P2                                KSAM-FILE-STATUS
CCT105P2                                CCT-BASIC-RECORD
CCT105P2                                CCT-MATCH-KEY
CCT105P2                                CCT-KEY-LOCATION
CCT105P2                                CCT-REC-LENGTH.
CCT105P2       MOVE SPACES TO CCT-MATCH-KEY.
CCT105P2       IF  NOT KSAM-SUCCESSFUL-OPERATION
CCT105P2           MOVE "???" TO CCT-VALUE,
CCT105P2           MOVE 1 TO GEN-ERROR-FLAG,
CCT105P2           GO TO P105-EXIT.
CCT105P2   P105-EXIT.
CCT105P2       EXIT.
```

COMMON CODE TABLE 980
Error Codes

KEY	BASIC VALUE (Error Message)
98000101	Error 001 - Must be Alphabetic
98000102	Only alphabetic characters are allowed.
98000201	Error 002 - Must be alphabetic or space fill
98000202	Only alphabetic characters are allowed, as well as
98000203	spaces (to the right of the alpha data ONLY)
98000301	Error 003 - Must be alphanumeric or space-fill
98000302	Alphabetic, numeric, or spaces (including those
98000303	embedded in data)are allowed-NO special characters
98000401	Error 004 - Must be numeric
98000402	Only numeric characters are allowed.
98000501	Error 005 - Must be numeric or zero-fill
98000502	Numeric characters only, with the exception of
98000503	spaces before and/or after the data only, NOT
98000504	embedded. These spaces are converted to zeros.
98000601	Error 006 - Must be numeric within range
98000602	Only a range of numeric values are valid. See the
98000603	User Guide for this format for further information
98001001	Error 010 - No data has been entered.
98001002	ENTER has been pressed, but no data has been input
98001101	Error 011 - Must be entered as space or -
98002501	Error 025 - City must not be left blank
98002701	Error 027 - Zip Prefix invalid, see Table 910
98002801	Error 028 - Mail Flag Invalid
98002901	Error 029 - State Code invalid, see Tables 901/903
98003201	Error 032 - Telephone must be blank or numeric
98003202	The only valid combinations of blanks and numerics
98003203	are as follows: bbb-bbb-bbbb
98003204	bbb-nnn-nnnn
98003205	nnn-nnn-nnnn.
98006001	Error 060 - Name must not be left blank
98006101	Error 061 - Surname must be followed by a comma
98006201	Error 062 - Name must not be changed
98006202	The name is used as an IMAGE/3000 search item
98006203	You MUST NOT change its value while correcting
98006204	other data on the screen.
98006501	Error 065 - Data item must NOT be changed
98006901	Error 069 - Action must NOT be changed
98007001	Error 070 - Invalid ID
98007801	Error 078 - Major Connector must NOT be blank
98007901	Error 079 - Major Connector must be -, ;, or space
98008001	Error 080 - Social Security Number must be numeric

```
          060400 P615-DUMMY. COPY VEW615P2.
 :W615P2  001000*
VEW615P2  001100*          *-------------------------------------------------*
VEW615P2  001200*             This routine displays the CCT error message in the
VEW615P2  001300*             form name box on the Whitman Standard View screen.
VEW615P2  001400*          *-------------------------------------------------*
VEW615P2  001600 P615-DISPLAY-ERROR.
VEW615P2  001700           MOVE CCT-VALUE TO VIEW-MSG-BUFF.
VEW615P2  001800           MOVE VIEW-ERROR-ENHANCEMENT TO VIEW-MSG-ENHANCEMENT.
VEW615P2  002000           DISPLAY VIEW-MESSAGE-BUFFER.
VEW615P2  003000           PERFORM P322-VIEW-SHOWFORM THRU P322-EXIT.
VEW615P2  003100 P615-EXIT.
VEW615P2  003200           EXIT.
          060500 P617-DUMMY. COPY VEW617P2.
VEW617P2  001000*
VEW617P2  001100*          *-------------------------------------------------*
VEW617P2  001200*             This routine displays the form title in the form
VEW617P2  001300*             name box on the Whitman Standard View screen.
VEW617P2  001400*          *-------------------------------------------------*
VEW617P2  001600 P617-RESET-ERROR.
VEW617P2  001700           MOVE FORM-TITLE TO VIEW-MSG-BUFF.
VEW617P2  001800           MOVE VIEW-STANDARD-ENHANCEMENT TO VIEW-MSG-ENHANCEMENT.
VEW617P2  002000           DISPLAY VIEW-MESSAGE-BUFFER.
VEW617P2  003100           PERFORM P322-VIEW-SHOWFORM THRU P322-EXIT.
VEW617P2  003200 P617-EXIT.
VEW617P2  003300           EXIT.
VEW625P2  001100*          *-------------------------------------------------*
VEW625P2  001200*           This routine flags a particular field on the View/3000
VEW625P2  001300*           screen with an error condition.  If the CCT-ERROR-KEY
VEW625P2  001400*           is not spaces, a lookup is done to find the proper CCT
VEW625P2  001500*           error message and display it to the User's screen.
VEW625P2  001600*          *-------------------------------------------------*
VEW625P2  001800 P625-VIEW-SECONDARY-ERROR.
VEW625P2  001900           MOVE -1 TO VIEW-LENERRMSG.
VEW625P2  002000           CALL "VSETERROR" USING  VIEW-COM-AREA
VEW625P2  002100                                   VIEW-FIELD-NUM
VEW625P2  002200                                   VIEW-DUMMY-PARAM
VEW625P2  002300                                   VIEW-LENERRMSG.
VEW625P2  002400           IF  NOT VIEW-OP-VALID
VEW625P2  002500               MOVE "925??P625A" TO GEN-ABORT-BREAKOUT,
VEW625P2  002600               PERFORM P658-VIEW-STANDARD-ERROR THRU P658-EXIT,
VEW625P2  002700               GO TO P099-ABORT.
VEW625P2  002800           IF  CCT-ERROR-KEY EQUAL TO SPACES
VEW625P2  002900               GO TO P625-EXIT.
VEW625P2  003000           PERFORM P680-ERROR-LOOKUP THRU P680-EXIT.
VEW625P2  003100           PERFORM P615-DISPLAY-ERROR THRU P615-EXIT.
VEW625P2  003200 P625-EXIT.
VEW625P2  003300           EXIT.
KSM680P2  001000*          *-------------------------------------------------*
KSM680P2  001100*           This routine will look up the appropriate error
KSM680P2  001200*           message on the CCT file (UT145K01) on request.
KSM680P2  001300*          *-------------------------------------------------*
KSM680P2  001400 P680-ERROR-LOOKUP.
KSM680P2  001500           MOVE 0 TO GEN-ERROR-FLAG.
KSM680P2  001600           PERFORM P105-CCT-INPUT THROUGH P105-EXIT.
KSM680P2  001700           IF  GEN-ERROR-FOUND
KSM680P2  001800               MOVE CCT-EMERGENCY TO CCT-ERROR-MSG,
KSM680P2  001900               GO TO P680-EXIT.
KSM680P2  002000           MOVE CCT-VALUE TO CCT-ERROR-MSG.
KSM680P2  002100 P680-EXIT.
KSM680P2  002200           EXIT.
```

```
        01  CONSOLE-COPY-DUMMY            COPY GEN022WS.
GEN022WS        .
GEN022WS        03  COPY-CONTROL-BYTE             PIC X(01).
GEN022WS *     *-------------------------------------------------------*
GEN022WS *                         Standard ABORT Format Area
GEN022WS *     *-------------------------------------------------------*
GEN022WS    01  GEN-ABORT-BREAKOUT.
GEN022WS            03  GEN-ABORT-ERROR           PIC X(03).
GEN022WS            03  GEN-ABORT-SECTION         PIC X(02).
GEN022WS            03  GEN-ABORT-PARAGRAPH       PIC X(04).
GEN022WS            03  GEN-ABORT-POINT           PIC X(01).
GEN022WS    01  GEN-ABORT-MESSAGE.
GEN022WS            03  FILLER                    PIC X(09) VALUE "CKPOINT ('
GEN022WS            03  GEN-ABORT-SECTION         PIC X(02).
GEN022WS            03  FILLER                    PIC X(01) VALUE ")".
GEN022WS            03  GEN-ABORT-PARAGRAPH       PIC X(04).
GEN022WS            03  FILLER                    PIC X(01) VALUE "-".
GEN022WS            03  GEN-ABORT-POINT           PIC X(01).
GEN022WS            03  FILLER                    PIC X(02) VALUE ": ".
GEN022WS            03  GEN-ABORT-LITERAL         PIC X(50) VALUE SPACES.
GEN022WS    01  GEN-ABORT-KEY.
GEN022WS            03  FILLER                    PIC X(03) VALUE "980".
GEN022WS            03  GEN-ABORT-ERROR-KEY       PIC X(03).
GEN022WS            03  FILLER                    PIC X(09) VALUE "01        "


       *     *-------------------------------------------------------*
       *         This routine reads the WID-MST based on the WID-VALUE
       *         from the STUDENT-DTL.
       *     *-------------------------------------------------------*
        P130-READ-WID-MST.
            MOVE STU-WHITMAN-ID OF STU012L2-AREA
                 TO CDB-WID-VALUE, STU-WID-VALUE.
            PERFORM P165-GET-WID-MST THROUGH P165-EXIT.
            IF  IMAGE-RECORD-NOT-FOUND
                MOVE 0 TO CDB-ID-PREFIX,
                GO TO P130-EXIT.
            IF  NOT IMAGE-OP-VALID
                MOVE "92548P130A" TO GEN-ABORT-BREAKOUT,
                GO TO P130-ERROR-OUT.
            GO TO P130-EXIT.
        P130-ERROR-OUT.
            PERFORM P645-IMAGE-STANDARD-ERROR THROUGH P645-EXIT,
            GO TO P099-ABORT.
        P130-EXIT.
            EXIT.
```

```
          121400 P645-DUMMY, COPY IMG645P2.
IMG645P2  001000*     *--------------------------------------------------*
IMG645P2  001100*                Standard IMAGE/3000 ABORT routine
IMG645P2  001200*     *--------------------------------------------------*
IMG645P2  001300 P645-IMAGE-STANDARD-ERROR.
IMG645P2  001400      MOVE CORR GEN-ABORT-BREAKOUT TO GEN-ABORT-MESSAGE.
IMG645P2  001500      MOVE GEN-ABORT-ERROR TO GEN-ABORT-ERROR-KEY.
IMG645P2  001600      MOVE GEN-ABORT-KEY TO CCT-ERROR-KEY.
IMG645P2  001700      PERFORM P680-ERROR-LOOKUP THROUGH P680-EXIT.
IMG645P2  001800      MOVE CCT-VALUE TO GEN-ABORT-LITERAL.
IMG645P2  001900      DISPLAY " X H J".
IMG645P2  002000      DISPLAY GEN-ABORT-MESSAGE.
IMG645P2  002100      MOVE SPACES TO IMAGE-ERROR-MSG.
IMG645P2  002200      CALL "DBERROR"   USING IMAGE-STATUS,
IMG645P2  002300                             IMAGE-ERROR-MSG,
IMG645P2  002400                             IMAGE-LENERRBUF.
IMG645P2  002500      CALL "DBEXPLAIN" USING IMAGE-STATUS.
IMG645P2  002600      MOVE IMAGE-ERROR-MSG TO GEN-ABORT-LITERAL.
IMG645P2  002700      MOVE GEN-ABORT-MESSAGE TO GEN-TELL-ABORT-MSG.
IMG645P2  002800      CALL "TELLABORT" USING GEN-TELL-JOB-NAME
IMG645P2  002900                             GEN-TELL-USER,
IMG645P2  003000                             GEN-TELL-DUMMY,
IMG645P2  003100                             GEN-TELL-DUMMY,
IMG645P2  003200                             GEN-TELL-ABORT-MSG.
IMG645P2  003300      GO TO P645-EXIT.
IMG645P2  003400 P645-EXIT.
IMG645P2  003500      EXIT.
          121500 P680-DUMMY, COPY KSM680P2.
KSM680P2  001000*     *--------------------------------------------------*
KSM680P2  001100*       This routine will look up the appropriate error
KSM680P2  001200*       message on the CCT file (UT145K01) on request.
KSM680P2  001300*     *--------------------------------------------------*
KSM680P2  001400 P680-ERROR-LOOKUP.
KSM680P2  001500      MOVE 0 TO GEN-ERROR-FLAG.
KSM680P2  001600      PERFORM P105-CCT-INPUT THROUGH P105-EXIT.
KSM680P2  001700      IF  GEN-ERROR-FOUND
KSM680P2  001800          MOVE CCT-EMERGENCY TO CCT-ERROR-MSG,
KSM680P2  001900          GO TO P680-EXIT.
KSM680P2  002000      MOVE CCT-VALUE TO CCT-ERROR-MSG.
KSM680P2  002100 P680-EXIT.
KSM680P2  002200      EXIT.
          121600 P999-END-SECTION-05.
          121700      EXIT.

    DATA AREA IS %006303 WORDS.
    CPU TIME = 0:01:06.  WALL TIME = 0:03:35.
END COBOL/3000 COMPILATION.  NO ERRORS.  NO WARNINGS.
```

```
[illegible]S305[illegible]02/11/81[illegible]13:15:35[illegible]------------------------(KELSEY)----#S305---[illegible]   Begin AD016 (2.1)  AD016
[illegible]S305[illegible]02/11/81[illegible]13:15:36[illegible]------------------------(KELSEY)----#S305---[illegible] AD016 A B O R T J O B [illegible]
[illegible]S305[illegible]02/11/81[illegible]13:15:36[illegible]------------------------(KELSEY)----#S305---[illegible] AD016 A B O R T J O B [illegible]
[illegible]S305[illegible]02/11/81[illegible]13:15:36[illegible]CKPOINT (98)P915-A: ERROR 801 - KSAM FILE SYSTEM ERROR0052
[illegible]S305[illegible]02/11/81[illegible]13:16:05[illegible]------------------------(KELSEY)----#S305---[illegible]   Begin AD016 (2.1)  AD016
[illegible]S305[illegible]02/11/81[illegible]13:16:05[illegible]------------------------(KELSEY)----#S305---[illegible] AD016 A B O R T J O B [illegible]
[illegible]S305[illegible]02/11/81[illegible]13:16:05[illegible]------------------------(KELSEY)----#S305---[illegible] AD016 A B O R T J O B [illegible]
[illegible]S305[illegible]02/11/81[illegible]13:16:05[illegible]CKPOINT (98)P915-A: ERROR 801 - KSAM FILE SYSTEM ERROR0052
13:16/#S306/22/LOGOFF
[illegible]S305[illegible]02/11/81[illegible]13:16:59[illegible]------------------------(KELSEY)----#S305---[illegible]   Begin AD016 (2.1)  AD016
13:17/#S308/29/LOGON FOR: GALPIN,MANAGER.WCCS,PUB ON LDEV #23
[illegible]S305[illegible]02/11/81[illegible]13:20:04[illegible]------------------------(KELSEY)----#S305---[illegible]     End AD016 (2.1)  AD016
13:20/15/STANDARD FORMS ON LDEV#6
13:20/#S280/45/LOGOFF
?13:20/15/SP#6/IS #S300; F969WIDE ON LDEV#6 (Y/N)?
13:20:47[illegible]02/11/81[illegible]------------------------------------------------#S300-----[illegible]
13:20/#S309/45/LOGON FOR: OH,PAINE.WCCS,P4 ON LDEV #46
OK
REPLY 15,Y
:
13:21/#S308/29/LOGOFF
```

```
####S100#03/02/81#13:17:20#--------------------------(POLZIN)----#S100---#    Begin ML705 (1.0) .ML705
13:17/#S102/33/LOGOFF
SO

DEV/CL    DFID     JOBNUM   FNAME    STATE FRM SPACE RANK PRI #C
COMLP     #0206    #J32     $STDLIST OPENED      1024         8   1
COMLP     #0241    #S94     F969WIDE OPENED      1024         8   1
COMLP     #0125    #S38     F260SPCL READY  F       8     D 1   1
COMLP     #0242    #S94     F222SPCL OPENED F    1024     D 1   1

4 FILES (DISPLAYED):
    0 ACTIVE
    1 READY; INCLUDING 1 SPOOFLES, 1 DEFERRED
    3 OPENED; INCLUDING 3 SPOOFLES
    0 LOCKED; INCLUDING 0 SPOOFLES
    4 SPOOFLES:  3000 SECTORS
OUTFENCE = 6
OUTFENCE = 10 FOR LDEV 24
.
####S94#03/02/81#13:23:24#--------------------------(COMFORAB)--#S94----#                                           #0241        STD WIDE <---F969
####S94#03/02/81#13:23:24#--------------------------(COMFORAB)--#S94----#                                           #0242   8.5 X 11 WHITE <---F222
13:23/15/STANDARD FORMS ON LDEV#6
####S94#03/02/81#13:23:25#--------------------------(COMFORAB)--#S94----#    End AD110 (2.2)
?13:23/15/SP#6/IS #S94; F969WIDE ON LDEV#6 (Y/N)?
####S94#03/02/81#13:23:40#--------------------------(COMFORAB)--#S94----#   Begin AD105 (2.0)  AD105
13:24/#S103/33/LOGON FOR: DIETZ,MGR.CONTACT,G3K ON LDEV #45
13:25/12/MISSING USER FOR "MATH.CLASS,         " ON LDEV "50"
OK
REPLY 15,Y
:
####S100#03/02/81#13:26:09#-------------------------(POLZIN)----#S100---#    End ML705 (1.0)  ML705
13:26/#J36/40/LOGON FOR: ML107CPL,KELSEY.WCCS,K1 ON LDEV #10
13:26/#S100/30/LOGOFF
13:27/#S104/31/LOGON FOR: CL408,NOEL.WCCS,N2 ON LDEV #23
13:28/#J36/40/LOGOFF
13:32/#S105/43/LOGON FOR: RICHMOND,ECON39.CLASS,ECON39 ON LDEV #27
13:32/#J37/29/LOGON FOR: CL408CPL,NOEL.WCCS,N2 ON LDEV #10
```

```
$CONTROL USLINIT,SOURCE
$TITLE ">>    Senior Rank in Class Update         <<            ",&
$"                                  CG228 <1,2>"
 IDENTIFICATION DIVISION.
 PROGRAM-ID.                        "CG228 <1,2>".
 DATE-WRITTEN.                       JUNE, 1979.
 INSTALLATION.                       WHITMAN COLLEGE.
 SECURITY.                           CONFIDENTIAL.
*
*      *--------------------------------------------------------*
*      |                >> NOTICE OF COPYRIGHT <<               |
*      *--------------------------------------------------------*
*      |    CG228 is Whitman College Proprietary software       |
*      *========================================================*
*      | Programming     | Systems Analysis  | Systems Design   |
*      *--------------------------------------------------------*
*      | Delores Payne   | Delores Payne     | Wayne Holt       |
*      *--------------------------------------------------------*
*
*      *--------------------------------------------------------*
*       SUMMARY:  CG228 ranks graduating seniors by cumulative
*                 GPA, and updates the Registrar Student Detail
*                 with their rank and class size when an update
*                 run is requested.  Only seniors that satisfy
*                 the following criteria will be ranked:
*
*                     * STU-STUDENT-LEVEL of 9,
*                     * STU-STUDENT-STATUS of B3, B4, B5, B6,
*                                             B8, C2, or C3,
*                     * PROJ-GRAD-DATE of PARAM-WINTER-DATE,
*                                         PARAM-SPRING-DATE,
*                                         PARAM-FALL-DATE.
*
*                 A standard narrow report, F482, is produced
*                 of those seniors meeting the above criteria
*                 and is sorted by descending cumulative GPA.
*
*                 (1)  File Descriptions:
*                      CG228S16   Parameters for CG228
*                      UT145K01   Common Code Table
*                      COMMON     Data Base Common
*                      STU        Student Records Data Base
*                      F482NARO   Report Print File
*
*                 (2)  Subroutine calls:
*                      UT817      Obtains JOB/USER Name
*
*      *--------------------------------------------------------*
*      *-------------------------------------------------------*
*       Compilation Instructions:
*
*            :FILE P;DEV=COMLP
*            :FILE COPYLIB=COBOLIB,PUB,WCCS
*            :COBOL CG228C,SOURCE,IRIS,,*P
*            :PREP $OLDPASS,CG228,REGSTRAR,ADMIN;MAXDATA=8000
*
*      *-------------------------------------------------------*
*
*      *-------------------------------------------------------*
*       DATE      REV   BY     MAINTENANCE LOG
*       6/29/79 <1,0> DRP Primary Installation of software.
*
*       6/09/80 <1,1> DRP Added statuses B8 and C3 to the list
*                         of valid senior student status codes.
*
*       6/12/80 <1,2> DRP Brought up to programming standards
*                         established May 30, 1980.
*      *-------------------------------------------------------*
```

```
008000*-----------------*-----------------------*----------------*
008100*------------------* SYNOPSIS OF PROGRAM LOGIC *-----------------*
008200*-----------------*-----------------------*----------------*
008300*
008400*   LOGIC DESCRIPTION:
008500*
008600*  I.   MAINLINE SECTION.
008700*
008800*       A. Displays the program number and revision level.
008900*       B. Performs all file opens and initial reads:
009000*          1.  Parameter file,
009100*          2.  CCT KSAM file,
009200*          3.  IMAGE files,
009300*          4.  Report files.
009400*       C. Performs the TELL-START routine.
009500*       D. Performs the LEGEND SECTION which produces a front
009600*          page legend on the report identifying the parameters
009700*          chosen by the User.
009800*       E. Performs the SORT-LOGIC SECTION.
009900*       F. Sorts the students by descending CUM-GPA.
010000*       G. Performs the REPORT-LOGIC SECTION.
010100*       H. Performs file closes:
010200*          1.  CCT KSAM files,
010300*          2.  Parameter file,
010400*          3.  Report files,
010500*          4.  IMAGE files.
010600*       I. Print a last page message on the report, CG228/F482.
010700*       J. Perform the TELL-FORM routine which posts a forms
010800*          message to the system console for a STD NARROW RPT.
010900*       K. Performs the TELL-STOP routine.
011000*       L. Stops the program.
011100*
011200* II.   SORT-LOGIC SECTION.
011300*
011400*       A. Serially reads the Registrar Student Detail on the
011500*          STU Data Base (REG-STUDENT-DET).
011600*          1.  Checks for valid senior STU-STUDENT-LEVEL (9 only)
011700*              and STU-STUDENT-STATUS (B3-B6,B8,C2, and C3).
011800*          2.  Checks for acceptable Projected Graduation Dates
011900*              input by the User in the paramater interface:
012000*              a.  WINTER-DATE - December graduates,
012100*              b.  SPRING-DATE - Spring graduates,
012200*              c.  FALL-DATE   - Fall graduates.
012300*          3.  If the student does NOT satisfy the above criteria,
012400*              another REG-STUDENT-DET will be read (return to A).
012500*       B. Perform a calculated read on the WID-MST on the COMMON
012600*          Data Base to obtain the ID-PREFIX for the WHITMAN-ID.
012700*       C. Perform a backwards chained read on the SEM-CREDIT-DTL
012800*          on the STU Data Base to obtain the record matching the
012900*          input academic term (contains the lastest grade data).
013000*       D. If the student does not have the correct SEM-CREDIT-DTL,
013100*          an error is flagged and another student record is read
013200*          (return to A).
013300*       E. The student has passed all edits so the necessary data
013400*          is now moved to the SORT-REC, and the record is released
013500*          to the COBOL SORT.
013600*       F. Repeat steps A through E until all students are read.
```

```
018700 ENVIRONMENT DIVISION.
018800 CONFIGURATION SECTION.
018900 SOURCE-COMPUTER.                    HP-3000.
019000 OBJECT-COMPUTER.                    HP-3000.
019100 SPECIAL-NAMES.
019200     TOP IS TOP-OF-PAGE,
019300     CONSOLE IS MASTER-CONSOLE.
019400 INPUT-OUTPUT SECTION.
019500 FILE-CONTROL.
019600     SELECT PRINT-FILE   ASSIGN TO "F482NARO,UR,,COMLP(CCTL)".
019700     SELECT PARAM-FILE   ASSIGN TO "CG228S16".
019800     SELECT SORT-FILE    ASSIGN TO "SORTFILE,DA".
```

```
         024300 WORKING-STORAGE SECTION.
         024400 01  GEN-COPY-DUMMY              COPY GEN021WS.
GEN021WS 001000     .
GEN021WS 001100     03  COPY-CONTROL-BYTE             PIC X(01).
GEN021WS 001200*    *-----------------------------------------------------------*
GEN021WS 001300*                  General Constants and Variables
GEN021WS 001400*              Console Communication and Message Parameters
GEN021WS 001500*    *-----------------------------------------------------------*
GEN021WS 001600 01  GEN-ERROR-FLAG                    PIC 9(01) VALUE 0.
GEN021WS 001700         88  GEN-ERROR-FOUND                     VALUE 1.
GEN021WS 001800 01  GEN-EDIT-FLAG                     PIC 9(01) VALUE 0.
GEN021WS 001900         88  GEN-NO-DATA-EDITED                  VALUE 0.
GEN021WS 002000 01  GEN-LENGTH                        PIC S9(04) COMP.
GEN021WS 002100 01  GEN-RUN-TIME.
GEN021WS 002200         03  GEN-RUN-HOUR              PIC 9(02).
GEN021WS 002300         03  GEN-RUN-MIN               PIC 9(02).
GEN021WS 002400         03  GEN-RUN-SEC               PIC 9(02).
GEN021WS 002500 01  GEN-JOB-SESSION-INFORMATION.
GEN021WS 002600         03  GEN-JOB-SESSION-NBR       PIC X(06).
GEN021WS 002700         03  GEN-JOB-NUMBER-HYPH       PIC X(06).
GEN021WS 002800         03  GEN-JOB-NUMBER-BLANK      PIC X(06).
GEN021WS 002900         03  GEN-JOB-NAME              PIC X(08).
GEN021WS 003000         03  GEN-JOB-USER-NAME         PIC X(08).
GEN021WS 003100         03  GEN-JOB-COMBINED-NAME     PIC X(15).
GEN021WS 003200         03  GEN-JOB-LDEV              PIC S9(04) COMP SYNC.
GEN021WS 003300 01  GEN-CONSOLE-TELL-INFORMATION.
GEN021WS 003400         03  GEN-TELL-JOB-NAME         PIC X(06).
GEN021WS 003500         03  GEN-TELL-USER             PIC X(08).
GEN021WS 003600         03  GEN-TELL-PROGRAM          PIC X(12).
GEN021WS 003700         03  GEN-TELL-OUTPUT-NBR       PIC X(08).
GEN021WS 003800         03  GEN-TELL-FORM-NAME        PIC X(04).
GEN021WS 003900         03  GEN-TELL-FORM-MESSAGE     PIC X(14).
GEN021WS 004000         03  GEN-TELL-DUMMY            PIC X(02).
GEN021WS 004100         03  GEN-TELL-ABORT-MSG        PIC X(70).
         024500 01  GEN-SORT-COUNT                    PIC 9(03) VALUE 0.
         024600 01  GEN-WORK-WID.
         024700     03  GEN-WORK-WID-PREFIX           PIC 9(03).
         024800     03  GEN-WORK-WID-SUFFIX           PIC X(06).
         024900 01  GEN-FIRST-TIME-FLAG               PIC 9(02) VALUE 0.
         025000         88  GEN-FIRST-TIME                      VALUE 0.
         025100*    *-----------------------------------------------------------*
         025200*                           Hold Area
         025300*    *-----------------------------------------------------------*
         025400 01  HOLD-NEW-RANK                     PIC 9(03) VALUE 0.
         025500 01  HOLD-SAME-RANK                    PIC 9(03) VALUE 0.
         025600 01  HOLD-STUDENT-LEVEL                PIC X(01).
         025700         88  HOLD-VALID-SENIOR-LEVEL             VALUE "9".
         025800 01  HOLD-CUM-GPA                      PIC 9V999 COMP.
         025900 01  HOLD-STUDENT-STATUS               PIC X(02).
         026000         88  HOLD-VALID-SENIOR-STATUS            VALUE "B3" "B4"
         026100                                                       "B5" "B6"
         026200                                                       "B8" "C2"
         026300                                                       "C3".
         026400 01  HOLD-PROJ-DATE.
         026500     03  HOLD-PROJ-MO                  PIC X(02).
         026600     03  HOLD-PROJ-YR                  PIC X(02).
```

```
 PROCEDURE DIVISION.
*        >>>>>>>>>>--------------------------<<<<<<<<<<
                      MAINLINE SECTION 98.
*        >>>>>>>>>>--------------------------<<<<<<<<<<
*
*     *-----------------------------------------------------*
*       SUMMARY: This Section of code is designed to be
*       the Main Driver for all program logic modules. It
*       contains all major file I/O overhead and general
*       housekeeping routines, and is the primary module.
*     *-----------------------------------------------------*
*
 P015-INITIALIZE.
     DISPLAY "CG228 <1,2>".
     PERFORM P912-OPEN-PARAM-FILE THROUGH P912-EXIT.
     PERFORM P131-READ-PARAM-FILE THROUGH P131-EXIT.
     PERFORM P915-OPEN-KSAM-FILES THROUGH P915-EXIT.
     PERFORM P100-READ-CCT-TITLES THROUGH P100-EXIT.
     PERFORM P975-OPEN-IMAGE-BASES THROUGH P975-EXIT.
     PERFORM P951-ESTABLISH-IMAGE-LISTS THROUGH P951-EXIT.
     PERFORM P917-OPEN-REPORT THROUGH P917-EXIT.
     PERFORM P995-TELL-START THROUGH P995-EXIT.
 P015-MAINLINE.
     PERFORM P030-LEGEND-DRIVER THROUGH P999-END-SECTION-25.
     SORT SORT-FILE
          ON DESCENDING KEY SORT-CUM-GPA,
             INPUT PROCEDURE IS SORT-LOGIC
             OUTPUT PROCEDURE IS REPORT-LOGIC.
 P090-STOP.
     PERFORM P925-CLOSE-KSAM-FILES THROUGH P925-EXIT.
     PERFORM P965-CLOSE-PARAM-FILE THROUGH P965-EXIT.
     PERFORM P969-CLOSE-REPORT-FILE THROUGH P969-EXIT.
     PERFORM P985-CLOSE-CDB THROUGH P985-EXIT.
     PERFORM P996-TELL-STOP THROUGH P996-EXIT.
     STOP RUN.
 P099-ABORT.
     CALL "ABORTRUN".
```

```
*----------------*------------------------------------------*----------------*
*----------------*          OUTPUT ROUTINES                 *----------------*
*----------------*------------------------------------------*----------------*
*
*         *-----------------------------------------------------------*
*          This routine will update the STU Data Base with the
*          rank in class for seniors and the size for that class.
*          The data will be put on the REG-STUD-DTL only for a
*          live run. Otherwise this paragraph will be omitted.
*         *-----------------------------------------------------------*
 P220-UPDATE-RECORD.
     MOVE SORT-WID                   TO GEN-WORK-WID.
     MOVE GEN-WORK-WID-SUFFIX        TO STU-WID-VALUE.
     MOVE SORT-WID                   TO STU-WID-VALUE.
 P220-DBLOCK.
     PERFORM P250-LOCK-STU THROUGH P250-EXIT.
     IF  NOT IMAGE-OP-VALID
         MOVE "92556P220A" TO GEN-ABORT-BREAKOUT,
         GO TO P220-ERROR-OUT.
 P220-DBFIND.
     PERFORM P160-FIND-STU-DTL THROUGH P160-EXIT.
     IF  IMAGE-WORD5-6 EQUAL TO 0
         GO TO P220-EXIT.
     IF  NOT IMAGE-OP-VALID
         MOVE "92556P220B" TO GEN-ABORT-BREAKOUT,
         GO TO P220-ERROR-OUT.
 P220-DBGET.
     PERFORM P165-READ-STU-DTL THROUGH P165-EXIT.
     IF  IMAGE-END-OF-CHAIN
         GO TO P220-EXIT.
     IF  NOT IMAGE-OP-VALID
         MOVE "92556P220C" TO GEN-ABORT-BREAKOUT,
         GO TO P220-ERROR-OUT.
 P220-DBUPDATE.
     MOVE HOLD-NEW-RANK  TO STU-CLASS-RANK OF STU012L2-AREA.
     MOVE GEN-SORT-COUNT TO STU-CLASS-SIZE OF STU012L2-AREA.
     PERFORM P270-UPDATE-STU-DTL THROUGH P270-EXIT.
     IF  NOT IMAGE-OP-VALID
         MOVE "92556P220D" TO GEN-ABORT-BREAKOUT,
         GO TO P220-ERROR-OUT.
 P220-DBUNLOCK.
     PERFORM P255-UNLOCK-STU THROUGH P255-EXIT.
     IF  NOT IMAGE-OP-VALID
         MOVE "92556P220E" TO GEN-ABORT-BREAKOUT,
         GO TO P220-ERROR-OUT.
     GO TO P220-EXIT.
 P220-ERROR-OUT.
     PERFORM P645-IMAGE-STANDARD-ERROR THROUGH P645-EXIT.
     GO TO P099-ABORT.
 P220-EXIT.
     EXIT.
```

ARCHIVE RETRIEVAL SYSTEM

By George McCauley

Grumman Aerospace, Research Dept.

ABSTRACT

Archive retrieval system provides a disc space management system that is user oriented and convenient for the SM. Files are periodically purged on a least used basis and a library perasal and retrieval system is established. User has created in his behalf a stream job that restores his files for him. AM can restore across groups and SM can restore across groups and accounts. Handles awkward syntax problems for user and produces listings of backed up files as well as parged files.

A DATABASE TEST AND REPAIR FACILITY

BY DUANE SOUDER

A Database Structure Test and Repair Facility

Databases may be corrupted in many ways. However, the most common way for a database to become corrupted is for a System Failure to occur during a DBPUT or a DBDELETE. IMAGE does not cause system failures by itself. The operating system will interrupt the execution of an IMAGE intrinsic for other processes (usually system processes). During the interruption, the operating system can encounter a situation from which it cannot recover. The result of course, is a system failure. There is a possibility that DBPUT or DBDELETE was doing internal pointer manipulation. If internal pointers were being changed, you have a physically corrupted database.

There is no current way to tell if the internal pointers were being changed at the time of the system failure. When the machine is restarted, the database is in an unknown state. The user has only a few choices available:

A) Just continue running and hope that there was no damage to the internal pointers, or

B) Use a program from the contributed library like DBCHECK (or DBGROOM, etc.) to try and find any problems, or

C) Do a DBUNLOAD followed by a DBLOAD.

Today, databases are being built larger and larger to meet the increasing demands of data processing. DBUNLOAD and DBLOAD are not reasonable solutions unless the database is damaged beyond the "Point of No Return". This "Point of No Return" has been a gray area for sometime. This new tool (or utility) called DBTEST, will attempt to address this gray area and help to define its boundaries by recommending DBUNLOAD/DBLOAD only if the problems are severe enough to warrant this action.

DBTEST was designed to examine the internal structures of the database and if necessary, to repair any damage found. Database down time will be significantly reduced by not having to unload and then reload the database. The user does not need to know anything about IMAGE internals, how chains work or any of the internal pointers. The user needs only the schema of the database and some working knowledge of their applications. The user interface was made simple and there are helping routines throughout the program to explain situations and aid the user in making decisions. If internal damage is found, the user does not have to allow the program to repair the database. The user has the final word and the program will not modify any internal structures unless the ok has been obtained from the user.

IMAGE uses a simple data structure called a Doubly Linked List for building chains in DETAIL data sets and Synonym Chains in MASTER data sets. Synonym chains are formed in MASTER data sets when different key values hash to the same location. During the addition or deletion of data records, the doubly linked lists are modified to reflect the operation performed on the data. A damaged database is the result of the modifications not being completed on the doubly linked lists. This is true for MASTER data sets as well as DETAIL data sets. DBTEST operates in maintenance mode like the other utility programs and it uses the IMAGE intrinsics along with privileged mode to examine and repair the internal chains.

DBTEST's internal pointer checking is simple and straight forward. For example, when you request to check a detail data set, DBTEST will examine two (2) things:

First, DBTEST will scan what is called the free record list. This list is a singly linked list of records that were deleted from the detail data set using DBDELETE. This free record list is the garbage collection mechanism that IMAGE uses during a call to DBPUT. The bit map is tested and the record itself is checked to be sure that it truly is a free record. Any necessary repairs to this list are attempted without user intervention. If a record is found that does not look like a free record, it will be displayed to the user with an appropriate message so that the user may recover any data in the record by examining the appropriate search items. This record will be delinked from the free chain list (so DBPUT will not use this record). If the free chain list cannot be repaired, an appropriate message will be displayed and the user will be given some alternatives based on the severity of the problem encountered.

Secondly, DBTEST will then ask the user to input the search item name and its value (also called the argument). DBINFO is then called to find out if the search item is valid. Assuming it is vaild, DBFIND will verify the argument and return the first and last record number in the chain along with the number of entries in the chain. DBTEST will then find out which master set and internal path is associated with that search item through a series of calls to DBINFO. (The record in the master set is obtained with DBGET to verify the internal pointers and the record number is kept for later access if necessary.) Using calls to DBGET, the chain is examined by making sure that the current record points back to the previous record that was examined. This process is continued until the end of the chain is reached or until a hole is found. (A hole is nothing more than a pointer to an invalid record.) While the links are being examined, a counter is incremented to compare against the master's entry count. The bit map is tested and the key is compared against the users input to make sure that a record with a different key value has not been linked into this chain. If a hole is found, DBTEST will then start from the bottom of the chain making the same checks above to insure consistency. The chain will be examined until another hole is found or the last record is reached while going forward. (In this case, the user will be told what has happened and will be asked if the chain should be repaired.) The counter will be compared against the masters record of the number of entries and if they do not match, the user will be asked if this should be repaired also. If there was a hole, the user will be told how many entries were lost/gained based on the master's record of the number of entries in the chain.

When a MASTER data set is to be examined, DBTEST needs only to access that data set. The doubly linked lists are used only for synonym chain construction as described earlier. The master data set is then read serially until a record is found that qualifies as a potential primary entry. The count, beginning and end of the synonym chain are remembered and the synonym chain is then checked for consistency. (This examination includes bit map checking, a running count of the entries and backward/forward pointer checking.) If there are any holes, the user will be told of the problem and DBTEST will automatically start from the end of the synonym chain making all consistency checks as described above until another hole is found or the last record going forward is reached. Then the user will be asked to allow DBTEST to repair any damage after an explanation has been given. DBTEST's synonym count is then compared to the primary's count of synonyms. The result of this comparision will be displayed to the user.

These descriptions are the methods used in verification of DETAIL chains and MASTER synonym chains in IMAGE databases by DBTEST. The checking is simple and the resultant patches are seen to be both useful and powerful. DBTEST was originally designed to examine internal chain structures and provide useful reports for further investigation. When it found internal problems, it was simple to implement code to fix the broken chains. This is how DBTEST evolved into a repair facility. With minor knowledge of IMAGE and a database schema, the user has a powerful tool which will help repair the database with minimum down time. This down time will of course be dependent on the size of the database and the extent of the user's knowledge of the applications.

There are some slides which show the useage of DBTEST and its user interface. All error conditions are not presented here but we feel that a reasonable amount are present to provide the user with a good idea how this tool may be used to help them should the situation ever arise.

THIS IS

DBTEST

A.03.00 MON, APR. 27, 1981, 1:00 PM

B-11 - 09

DATABASE STRUCTURE (CHECK/REPAIR) TOOL

```
HELP(H) OR WHICH BASE OR 'CR' ? MFG

                 / APPEARS TO BE OK.
DATA BASE MFG <--- WAS BEING ERASED.
                 \ WAS BEING MODIFIED WITH
                      OUTPUT DEFERRED !

OUTPUT TO TERMINAL (Y/N) ?  Y
```

```
HELP(H) OR LOOK(L) OR 'CR' ?  L

HELP(H) OR WHICH DATA SET OR 'CR' ? SUP
```

EXAMINATION OF DETAIL DATA SETS

```
HELP(H) OR SERIAL(S) OR CHAINED(C)
     READS ON THIS DETAIL ?  C

HELP(H) OR SEARCH ITEM NAME ? ACCT

HELP(H) OR ARGUMENT ? 153405

PRINT OUT ENTRIES IN CHAIN (Y/N) ? Y
```

```
WE COULD NOT GET THE N'TH FORWARD ENTRY IN THE CHAIN.
WE HAVE ENCOUNTERED WHAT APPEARS TO BE A BROKEN
DETAIL CHAIN WHILE USING THE FORWARD POINTERS.  WE
SHALL ATTEMPT TO RUN THE BACKWARD POINTERS TO
RETRIEVE ANY OTHER ENTRIES IN THE CHAIN.


WE COULD NOT GET THE N'TH BACKWARD ENTRY IN THE CHAIN.
THE DETAIL CHAIN IS BROKEN IN BOTH DIRECTIONS.


NUMBER OF ENTRIES IN CHAIN ACCORDING TO MASTER = 10


HELP OR PATCH ENTRY OR DO NOTHING (H/P/N) ? P
```

```
YOU HAVE TWO (2) OPTIONS:
1)  DO NOTHING AND CONTINUE SEARCHING
2)  PATCH
    NOTE:  THIS PATCH WILL DO THE FOLLOWING:
    a)  PATCH ENTRY
        FORWARD = LAST GOOD ENTRY (GOING BACKWARD)
    b)  PATCH ENTRY
        BACKWARD = LAST GOOD ENTRY (GOING FORWARD)
    c)  PATCH PRIMARY
        ENTRY COUNT = CURRENT NUMBER OF ENTRIES

WHICH OPTION ? 2

ARE YOU SURE ? Y

   ENTRY COUNT KEPT IN PRIMARY IS CORRECT.
OR
   ENTRY COUNT KEPT IN PRIMARY IS BAD.

                      GAINED
   NUMBER OF ENTRIES           = 1
                      LOST
```

EXAMINATION OF MASTER DATA SETS

```
HELP(H) OR CHECK A PARTICULAR ENTRY(PE)
   OR CHECK ALL SYNONYM CHAINS(SC)
   OR PERFORM SERIAL READ(SR) ? PE

PRINT OUT PRIMARY (Y/N) ? Y

PRINT OUT ALL SYNONYMS (Y/N) ? Y
```

DATABASE THERAPY: A practitioner's experiences

F. Alfredo Rego

Rego Software Pty
Calle del Arco 24
Antigua, GUATEMALA

Telephone (502-2) 324336
Telex 4192 TELTRO GU

ABSTRACT

I will describe my first-hand experiences on some fundamental aspects of database therapy:

Prevention: A bit of prevention can save many megabytes worth of reloading.

Periodic checkups: There are procedures to test, diagnose and report database errors and faults. And your own users may very well be your best detectors.

Treatment: There are good procedures for the correction of database errors and faults. They are still primitive but can be very effective in those cases they can now handle. And they are impressively learning to deal with new cases!

Follow-up. You cannot afford to lower your guard. You must follow up. Always.

INTRODUCTION

I have had the privilege to visit hundreds of HP3000 computer installations in all continents and in many islands. There are "radical" differences in applications, people, software, hardware, and environment. But all of these HP3000 computer installations have the very same purpose: To maintain a bunch of bits. The specific patterns formed by their pet configurations of such zillions of bits are as varied as they can be, but our fellow HP3000 users all over the world are doing exactly the same thing. It is very important to remember this!

A bunch of bits: That is all that we really have in any computer system in general and in any database in particular. A bunch of bits, some which are supposed to be ON and some which are supposed to be OFF.

To be ON, or not to be ON: that is the BIT question!

Who is authorized to decide which bits are supposed to be on or off? Who is responsible for detecting flip-flopped bits? Who is able to correct runaway bits? Let's share some thoughts and feelings on these challenging subjects.

PREVENTION

A bit of prevention can save many megabytes worth of reloading. My own personal bias favors PREVENTION ABOVE ANYTHING ELSE. I believe prevention has the greatest possible payoff, especially when the stakes are high.

Databases face serious health hazards. Let's discuss a few, beginning with the more innocent-looking ones. And let's see what preventive action you can take.

Sloppy and/or disgruntled people are the worst possible hazard to a database and to the whole computer system. If you cannot keep these jewels under control, you might as well forget everything that follows.

People can (and do) misuse software. They can use QUERY to find all the entries that meet certain criteria and then delete them. They can accomplish the same (good? bad?) thing with a ten-line BASIC program or with the equivalent 100-line structured-COBOL program. It does not matter HOW they do it. If those entries are supposed to be there (according to you), then YOU should take some preventive action. For instance:

- Use IMAGE LOGGING and Bob Green's DBAUDIT (see reference 1) to find out who did what, when, to which entries.

- Use DBUTIL to disable QUERY-B write access. Or assign a password to QUERY. Or remove QUERY from your system.

- Take advantage of the powerful password-security mechanism provided by IMAGE to restrict write-access to sensitive items or sets. Change these passwords every now and then. Be sure to write them down and store them in a cool, dry place which is NOT accessible to your enemies and/or friends.

- Assign a maintenance word to every database with DBUTIL (at creation time, or later on). Change this maintenance word periodically. Assign a password to DBUTIL itself, so people will have a harder time if/when they decide to ERASE or PURGE your databases.

Innocent-looking application programs, as you well know, are one of the worst threats to the consistency of a database. Your best strategy is to build a set of application programs that spend their lives checking the application-dependent consistency of your database. You will have to face some painful decisions regarding tradeoffs here. But something and somebody must check things occasionally just to be sure that obvious errors have not crept in.

Remember that your own applications software is responsible for keeping the semantic consistency of your database. Nobody else except you and your staff knows anything about your specific design and implementation. Therefore, _you_ must make sure that the effect of your applications software is constantly monitored by your end-users, by your quality-control and auditing staff, and (last but not least) by your programmers.

Remember also that a database is an integrated entity that has two distinct elements: the BASE system (IMAGE in our case) and the DATA that lives there. The best database management system in the world will be useless if the data you keep in it has no meaning. And it is the main responsibility of all your application programs to maintain the validity and usefulness of that data!

People can use ACCOUNT MANAGER capabilities to purge the database's group. Even worse, they can use SYSTEM MANAGER capabilities to purge the database's account. You can run LISTDIR2.PUB.SYS to list all the accounts and all the users in your system, to check their capabilities. To protect yourself, stand right at the line printer when the printout comes out. Otherwise, somebody will see it and everybody will know everybody's capabilities! Be sure that no one has more MPE capabilities than the strict minimum necessary to operate.

Privileged users (and the system operator) can use SYSDUMP, STORE and RESTORE with IMAGE databases. As long as they know what they are doing and as long as they store/restore fully-consistent collections of privileged IMAGE MPE files, everything will be all right. But the moment they make one mistake, NOTHING will be right! This is why the modules DBSTORE and DBRESTOR were designed by Hewlett-Packard specifically to store/restore IMAGE databases. They do some reasonableness checks to increase the probability that you are backing up or restoring a consistent database. And they mark the database on disc indicating the backup date and time (an important thing for logging and recovery purposes). ADAGER's BACKUP module is functionally equivalent to DBSTORE and DBRESTOR, but it uses just a fraction of their time and tape resources. As an added preventive measure, ADAGER encrypts the root file as it backs it up to tape so that nobody can FCOPY it to the line printer to find out all the user-class passwords. (MORAL: lock up your database backup tapes just as if they were cash. Never forget that your database may be much more valuable than cash!)

People can store things on the wrong set of tapes, clobbering whatever data was on the tapes! And people can restore from the wrong set of tapes, clobbering whatever data was on disc! A well-organized tape-library system is a good investment, especially if it is itself computerized (after all, you are trying, precisely, to protect yourself from sloppy operators...)

People can physically keep your backup tapes in a hostile environment, thereby rendering them useless. I recommend professional handling of your off-site tape storage. And I recommend that you periodically check the validity of the tapes kept both on-site and off-site. What would happen if you had to restore some file from some tape that was physically impossible to read?

People can fail to backup the system at all. Whether they do it intentionally or innocently is irrelevant: the catastrophic consequences are exactly the same. How do you know that the tapes that are supposed to contain your full sysdumps really contain them? Backing things up is a chore. How do you know that your people are not taking shortcuts? I know a user who has an HP3000 machine dedicated 8 hours a day to just a single purpose: a RELOAD from the sysdump tapes produced by other computers. If any reload has any difficulties whatsoever, he takes immediate action to correct the problem while it is IMPORTANT but not URGENT. Most people wait until a problem is IMPORTANT, URGENT, and IMPOSSIBLE to solve within the given time/resource constraints. This user was one of these people. After a near-catastrophe, he learned that any investment in prevention pays handsome dividends in terms of everyday health. Both his and his database's.

Please keep in mind that computerized databases are not exempt, by any means, from ENTROPY: Any system degrades to an ultimate state of inert uniformity. But please also keep in mind that you can delay your database's inevitable failure and decline. You can keep your database in a good state of repair, efficiency, validity and effectiveness. But you must be willing to invest in PREVENTIVE MAINTENANCE. Otherwise, you and your database are doomed.

Your problem, as a manager (of the whole universe, of a country, of a company, of a department, of a computer system, of a program, of even one bit), is always this: You must first choose, out of an unlimited collection of possible objectives, the ONE goal that you want to reach; and then you must also choose, out of a very limited collection of resources, those few resources that will help you reach your goal in a finite time.

These choices are extremely difficult. There is no question about that! But you can make your life easier if you <u>decrease</u> your collection of possible goals, if you <u>increase</u> your collection of resources, or if you combine <u>both</u> approaches.

In the specific case of your HP3000 computer system, you are very fortunate. There are many good people (in the Hewlett-Packard Company, in the HP3000 vendor community, in the HP3000 user community and in the directors and staff of the Users Group). These people have done, are doing and will continue to do excellent work, whose end result means that you can concentrate a large number of extended resources on YOUR objectives. There are many "possible goals" that you do NOT have to worry about unless you really want to. If you want an operating system, a database management system, a report writer, a word processor, an editor, etc., SOMEBODY else has already spent endless hours and lots of valuable resources to make these things available to you. Likewise, if you want some warnings about things that you should treat with care and respect, somebody else has spent endless hours and lots of valuable resources <u>and</u> has published these warnings for everybody's benefit.

Two recent examples are Gerald W. Davison's article on <u>IMAGE LOCKING and Application Design</u> (see reference 2) and my article on <u>Design and Maintenance Criteria for IMAGE Databases</u> (see reference 3). If you want an extended bibliography and a list of reference materials, please write to me and I will be happy to send you a printout of my latest computerized information. Rick Bergquist, the author of DBLOADNG and the User's Group Interface Committe member for IMAGE would also like to hear from you (see reference 4).

A computer (HP3000 or otherwise) is just as good as the electrical power that feeds it. My company has only a small Series 30 (Koala) but we have cheerfully invested in an uninterruptible power supply that costs as much as one disc drive. I personally authorized that procurement since I firmly believe in the value of PREVENTION. Instead of having one more disc drive, I prefer to preserve whatever we have! I <u>will not</u> accept growth at the expense of reliability.

A computer room is NOT a social club. Keep traffic to a minimum. Do not use it to store things which may be harmful to your equipment. Make your operators strictly accountable for everything that moves (and does not move) in the computer room, just as if they were bank tellers. After all, your information may be easily worth millions of Krugerrands.

Most systems software is amazingly water-tight. Systems software people tend to be careful to the point of being paranoid. And the good systems software available for the HP3000 is truly outstanding. Just check the DATAPRO ratings for Hewlett-Packards systems like IMAGE and for independent systems like QEDIT and ADAGER or talk to the person sitting next to you. What is not so water-tight is the sloppy or malicious use of systems software.

Take QEDIT as an example. If somebody deletes a few lines from some of your source programs (especially lines that contain "IF" statements), your whole computer system may produce strange results. Nevertheless, QEDIT is completely innocent. You must develop procedures to decrease the probability of this unpleasant happening. You may want to use Larry Simonsen's program to compare source files (see reference 5). And you may want to implement a computerized system to keep track of changes to your source files. We have IMAGE logging. We should also have SOURCE-FILE logging.

Take ADAGER as another example. If somebody adds a field to a data set or reshuffles the fields in a data set without recompiling ALL affected application programs to update their buffer definitions, your whole computer system may produce some unusual results also. Nevertheless, ADAGER is totally innocent. You must also develop procedures to decrease the probability of this occurrence. You may want to use ADAGER's SCHEMA and XRAY functions to get periodic snapshots of your database and to compare them with your reference documents. Logging ADAGER database changes will help you in this matter. Anyhow, before diving into the pool, your applications program should check whether the pool has any water in it or not! A simple call to DBINFO, for instance, can easily tell your program that a change in a data set's entry definition has taken place. If this is the case and if your program is not qualified to handle the non-standard situation, it should report the discrepancies and quit before it clobbers anything.

Take DBUNLOAD/DBLOAD as still another example. Both Hewlett-Packard programs handle full entries (see reference 6). If you delete or add fields, if you redefine items, or if you delete or add sets, the fact that the unload/load cycle completes successfully does not mean a thing: your database may still be easily clobbered anyway!

One of the best preventive measures you can take is to establish a standard DATA DIRECTORY & DICTIONARY system. Tipton Cole is the Users Group Committee Chairman for this project. He would like to hear from you if you are interested in these ideas (see reference 7).

Logging and recovery are worthy things in themselves. But you can never be protected 100%. For instance, if a system failure happens in the middle of a logging/recovery cycle, your database may be left in an inconsistent state. Rick Bergquist has done extensive work in this area and you may want to contact him for further references (see reference 4). For the time being, HewlettPackard's recovery system allows only roll-forward. Rick's method allows also backout recovery.

The Users Group Contributed Library is certainly full of very valuable software (getting close to two million U.S. Dollars in value). But you must check its programs carefully to be sure you understand them. And you must maintain these programs to keep them in step with the times (and the operating system releases!) For instance, if you installed OVERLORD or SOO a couple of years ago, you will have to change them when you install MPE IV. The Central Contributed Library Office will send you the Contributed Library Tape with the appropriate software but YOU still have to install it yourself. If you have any questions whatsoever, contact Mark Wysoski, the most knowledgeable person regarding the Contributed Library. He will help you and he will guide you, whether you need to use a program or contribute a program. He usually knows whether any bugs are outstanding on a given program that may cause problems in your site. And all the other members of the group would appreciate it if you would report any peculiarities that you observe in the contributed programs. This community-wide network is essential for all of us. Please help. (See reference 8).

Even though Hewlett-Packard equipment is very reliable, it may fail, since it is not exempt from ENTROPY either. The obvious things like printers do not worry me too much. But the subtle things like CPU chips do worry me. My Customer Engineer in Guatemala, Johnny Siemon, taught me how to run the CPU diagnostics. I run them once a day (you can never run too many diagnostics!) If any test whatsoever fails, I do a few things like resetting the CPU and then giving the test another try. If it fails again, I pull the plug and call him immediately, with the exact chip number. He arrives with the right parts and I am up and running very soon. I would much rather do this bit of HARDWARE PREVENTIVE work myself than risk the possibility of "unusual" address calculations on disc for file-write operations, for instance. (Just for the record: I have only found one bad chip in more than a year of operation. Johnny had the part shipped from Guatemala City to Antigua and installed while I had lunch. So, for all practical purposes, our HP3000 has never been down. This is the best motivation we have to keep our budget for investments in preventive measures UP. At any time. All the time. At any cost. It turns out to be dirt-cheap in the long term. And our strategy, as a company, is quite biased towards the long term by choice).

I am amazed at the kinds of cruel environments in which the HP3000 computer survives. Nevertheless, there are a few things that you should watch out for. Controllable by you (at a cost) are things like smoke, dust, and electrical power. I would suggest that any investment you make to control these parameters will pay for itself many times over. Uncontrollable parameters like cosmic rays, earthquakes, floods, wars, etc., will not affect you as much if you have previously established a consistent program of backup and recovery, complete with emergency drills to see how your people behave under pressure. There is a large amount of literature on this subject and I will be happy to mail you some references if you wish. I would also like to hear from you if you have any suggestions or comments regarding the ways that you have implemented in your site.

PERIODIC CHECKUPS

Webster defines DIAGNOSIS as the art of identifying a disease from its signs and symptoms, as an investigation or analysis of the cause or nature of a condition, situation or problem.

I have some friends who can smell a rat in no time at all. I have other friends who could easily spend a year testing a system, full time, without ever finding the errors that do exist in the system. I have decided that the degree of "education", "training", "certification", etc. associated with successful diagnosticians seems to be totally irrelevant. A certain amount of raw gut feeling is in effect here, which defies easy rules and pigeon-holing. Just like music, poetry, painting, photography, architecture and other creative endeavors, "some people have it and some do not". I have simply decided that I prefer to invest my time IDENTIFYING good people and then relying on their judgment rather than to waste my time trying to train untrainable people. This works out to be to everybody's advantage (there is nothing more pitiful than seeing somebody who has NO aptitude whatsoever trying to cope with concepts or actions which are dramatically beyond that somebody's grasp).

You may have noticed that some of the most "naive" users of your computer system can detect, immediately, "when, how, and why" something is wrong. Do NOT underestimate them! Usually they are the very clerks who know "their" data and information quite intimately. Be sure to include such users in your early-warning system. They may be much more valuable to you than some sophisticated programmers who spend all of your computer's resources navigating through masses of data and sailing right past the most obvious problems which, if not corrected immediately, will cause enormous losses. Do not forget that education, salary, title, position, age, sex, etc., are utterly irrelevant. If the Vicepresident in charge of Information Systems and the payroll clerk do not agree on something, I would always place my bet on the clerk's side (Not on matters of policy but on matters of DETECTING ERRORS, of course!)

"Errors" are a hard thing to define. We will use the term here in a somewhat subjective manner related to our "desirability" of certain things. We will say that an ERRONEOUS SYSTEM, given a set of TRIGGERING CONDITIONS, will cause a set of UNDESIRABLE EFFECTS. You certainly do NOT want undesirable effects under any circumstances or conditions whatsoever. Unfortunately, this is an impossible goal, since you could not even dream up ALL possible sets of conditions, much less test and certify them!

A good DIAGNOSIS SYSTEM should analyze the behavior of your system under a meaningful subset of the total CONDITIONS SPACE. This is easier said than done. Defining this meaningful subset is a very difficult task. Even more, the fact that a diagnosis system does not uncover undesirable effects under its subspace of conditions does NOT mean at all that you have a correct system. You may be just lucky. Your set of tested conditions may be such that those portions of your system which produce undesirable effects are just not exercised. But those nasty portions of your system are precisely the ones you want to discover because they lurk there, ready to jump whenever their set of conditions happens (according to Murphy, this will happen at the worst possible time).

A good system of diagnosis goes much beyond just saying "no errors detected". It devises as many sets of linearly-independent conditions as possible to see how the system reacts. If it detects undesirable effects, it issues appropriate messages saying "the system fails with this set of conditions (and describes them)", "the effects are (and describes them)", "the error(s) probably reside(s) in such and such part(s) of the system". Instead of just telling you that something is wrong, it tells you how precisely it is wrong, perhaps why it got that way so you may avoid it next time, and how you can fix it.

Please notice that a good diagnosis system must be nasty and sadistic by nature. It has as its primary objective to FIND ERRORS, not to certify a system as being error-free (there is no such system anyway!) A good diagnosis system must also be extremely patient and humble, since it will fail many times. Please keep in mind that there is a psychological inversion in effect here: A good diagnosis system fails if it does not detect any errors. And most of the time it will not detect any errors, since we hope and assume that the entity being tested is reasonably error-free. Naturally, if you are testing a "lemon", almost any diagnosis system will have a successful run. But that is the trivial case!

It is unrealistic to even attempt to test ALL POSSIBLE sets of conditions. It is unrealistic to even suggest to test the much smaller subset of ALL PROBABLE sets of conditions. The challenge for the diagnosis system then is to find the MAXIMUM number of errors and faults with the MINIMUM investment in diagnosis tools and test runs.

A large number of diagnosis tools and diagnostic runs may be totally worthless if they are poorly designed and implemented (they will simply say "end of step 154, no errors found" or "error 538 found in step 47, loop 10457"). On the other hand, a small number of these tools, intelligently put together, may be very effective in finding elusive errors. Such a system, naturally, is very difficult to design and implement. We are just now in the threshold of a new era in software engineering.

It turns out that Zeno's paradox applies equally well to finding errors as it does to shooting arrows: Finding the first "half" of all errors present in the system requires some effort; finding half of the remaining errors takes about the same amount of effort, and so on ad infinitum, with the last difficult bugs taking more time to find than the first (to quote from Brooks' "The Mythical Man Month"). We have here something akin to a half-life type of situation for bugs!

For specific programs that you can run now to diagnose IMAGE database errors, please write to me and I will mail you an up-to-the-minute list of references. In the meantime, here is a partial list:

- DBCHECK (also called STRUKCHK). In the Contributed Library. Distributed also in the QEDIT and ADAGER tapes. It cruises through either one or all data sets in a database. It does a fairly thorough examination of all the structural parts of the data sets and reports, in great octal detail, those things that it does not like. A few of those things may not be errors at all but it always nice to know that the program detected them.

- DBLOADNG. In the Contributed Library, QEDIT and ADAGER tapes also. It also cruises through a database's data sets and produces a summary report on the general state of the database. It indicates conditions that may require attention (like master capacities which are not prime numbers, or master data sets which are more than 80% full, etc.)

- ADAGER's XRAY and SCHEMA functions, as well as QUERY's FORM command, will give you a fairly good idea of what is really in your database (as opposed to what you THINK is in your database!)

- DBSTORE/DBRESTOR (or ADAGER's BACKUP function) will do some reasonableness checking to make sure that, at least, you have all the MPE privileged files that your database is supposed to have.

TREATMENT

There are good procedures for the detection and correction of database errors and faults. They are still primitive but can be very effective in those cases they can now handle. And they are impressively learning to deal with new cases.

Hewlett-Packard has developed a program which is a Systems-Engineer tool. Please attend Duane Souder's session ("DBTEST: A database structure test and repair facility") and please read his paper in these Orlando-81 proceedings.

Rego Software PTY has developed an ADAGER function called NURSE which is currently undergoing pre-release tests. Many ADAGER modules, as a matter of fact, have already incorporated some of NURSE's functions and have been released since 1980. NURSE has been developed with the help of many ADAGER users all over the world. Whether you are an ADAGER user or not, I would appreciate your help during our ongoing research and development efforts. Please provide us with samples of database errors so we can analyze them and classify them.

ADAGER's NURSE function is developing in an evolutionary way. It is now a recognized expert on those categories of database errors that it has catalogued in its "vocabulary". And it provides a commented report on those database errors that it has not (yet) learned to correct. The ADAGER-formatted file created by NURSE contains enough information to allow us to incorporate a new linearly-independent vector in NURSE's vector-space base. This way, the cycle keeps on going: Our friends (and NURSE) report to us new categories; we study them and incorporate them into NURSE's bag of tricks... and so on!

Wayne Benoit's DBGROOM (in the Contributed library) allows you to directly manipulate structural elements in an IMAGE database. It allows you better access than if you were using DISKEDIT. But you are still fully responsible for all that octal thinking! If you blow even one vital root-file bit, nobody will warn you. This is a great tool if you REALLY know what you are doing (and Wayne Benoit certainly knows what <u>HE</u> is doing, since his knowledge of the internals of IMAGE is very thorough); but this tool can destroy everything if you do not know what you are doing.

Hewlett-Packard's PRIVILEGED DEBUG and DISKEDIT allow you unlimited access to every single bit in memory and on disc. You can fix ANYTHING, in theory, by means of these tools. But you have to think strictly in terms of bits. Zillions of them. Each with a vital meaning. I sincerely do not believe that anybody can use these tools for reasonably complicated surgical operations on databases. But they are certainly available anyway to anybody authorized by your HP3000 System Manager.

FOLLOW-UP

You cannot afford to lower your guard. You must follow up. Always. One way to follow up is to keep up with IMAGE-related research and development. Read the Users Group Journals and Newsletters. Attend your local Users Groups meetings. Call your HP3000 colleagues whenever you have any questions, comments or suggestions. Contact your Hewlett-Packard Systems Engineer and your independent software suppliers.

Hewlett-Packard's IMAGE is an excellent database management system. It is simple. It is reliable. It is robust. Hewlett-Packard is committed to its continued improvement. Rego Software PTY is committed to ADAGER's continued improvement. Other software vendors are committed to the continued improvement of their IMAGE-related products.

If all of us in the world-wide Hewlett-Packard family integrate the use of our combined resources for the attainment of our common goal, we can all look forward to IMAGE's unbounded success. THIS IS OUR COMMON GOAL. Let us all follow up!

Thank you.

REFERENCES

1 - Robert M. Green (604) 943-8021 Telex 04-352848
Robelle Consulting Ltd.
5421 - 10th Avenue, Suite 130
Delta, British Columbia V4M 3T9
Canada

2 - IMAGE LOCKING AND APPLICATION DESIGN
Hewlett-Packard Users Group Journal,
First Quarter 1981 Vol IV No. 1
Gerald W. Davison
Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois 60439
U.S.A.

3 - DESIGN AND MAINTENANCE CRITERIA FOR IMAGE/3000
Hewlett-Packard Users Group Journal,
Fourth Quarter 1980 Vol III No. 4
F. Alfredo Rego
Rego Software PTY
Calle del Arco 24
Antigua
Guatemala

4 - OPTIMIZING IMAGE: AN INTRODUCTION
Hewlett-Packard Users Group Journal,
Second Quarter 1980 Vol III No. 2
Rick Bergquist (415) 573-9481
American Management Systems
561 Pilgrim Drive
San Mateo, California 94404
U.S.A.

5 - Larry Simonsen (801) 489-8611
VALTEK
Mountain Springs Parkway
Springville, Utah 84663
U.S.A.

6 - IMAGE/3000 REFERENCE MANUAL
Hewlett-Packard Company (408) 725-8111
19447 Pruneridge Avenue
Cupertino, California 95014
U.S.A.

7 - Tipton Cole (512) 452-0247
Cole & Van Sickle
7701 Cameron Road
Austin, Texas 78761
U.S.A.

8 - Mark C. Wysoski (509) 527-5360
Manager, Hewlett-Packard Users Group
Contributed Library
Whitman College
Walla Walla, Washington 99362
U.S.A.

ANSI COBOL 198X: A SNEAK PREVIEW

Greg Gloss
Hewlett-Packard Information Systems Division

ABSTRACT

The ANSI (American National Standards Institute) X3J4 technical committee is in the final stages of work on the next version of the COBOL standard. This paper will discuss some of the major new features which are expected to be included such as structured programming constructs together with those items which will no longer be allowed in the next standard. The standardization process will also be covered briefly.

BACKGROUND

The current version of ANSI COBOL was adopted in 1974. Since 1977, the ANSI X3J4 committee has been working on the next version of the COBOL standard. Since it is not clear how long this process will take, I will refer to the next standard as COBOL ′8X.

OVERVIEW

The next standard will have changes in the following categories:

a. New Features
b. Transitional Features
c. Deleted Features
d. Specification Changes
e. New Reserved Words

A significant effort has been put into incorporating structured programming constructs into COBOL. In addition, other new facilities have been added to make programming in COBOL easier. Some current features have been flagged for deletion either in the new standard or in the subsequent standard. Those features which are in the new standard, but which are not expected to be in the subsequent standard are called transitional. There have also been some changes to the rules and additional reserved words included which may affect existing programs.

STRUCTURED PROGRAMMING

The new structured programming constructs which have been defined for COBOL include Scope Terminators, PERFORM statement enhancements, the EVALUATE statement, and the CONTINUE statement.

Scope Terminators

Under COBOL ′74, conditional statements could not be included with the statement group following a conditional phrase such as

AT END or ON SIZE ERROR. New reserved words have been added such that any conditional statement can be turned into an imperative statement and used as part of the conditional statement group. For example,

```
     READ FILE-IN AT END
        ADD A TO B ON SIZE ERROR
           PERFORM OVERFLOW-ROUTINE
        END-ADD
        MOVE SPACES TO REC-IN.
```

Under COBOL '74, it is not legal to specify the ON SIZE ERROR phrase in the above example because it turns the ADD statement into a conditional statement and only imperative statements are allowed following the AT END phrase. However, with the scope terminator, END-ADD, the ADD statement with the SIZE ERROR option becomes an imperative statement and is legal in this situation. The MOVE statement is the second imperative statement to be executed if the AT END branch is taken and the period terminates the READ statement. If the READ itself were nested under a conditional such as an IF, it would be terminated by a END-READ instead of the period.

PERFORM Statement Enhancements

The PERFORM statement has been enhanced to allow a list of imperative statements to be embedded within the statement instead of paragraph names and to allow the programmer to specify whether the UNTIL conditions are to be tested before or after the specified set of statements has been executed.

An example of an in-line PERFORM is shown below:

```
     PERFORM 10 TIMES
        ADD A TO B
        ADD 1 TO A
     END-PERFORM.
```

The two ADD statements will be executed 10 times.

Under COBOL '74, the UNTIL conditions are always tested before executing the specified paragraphs. The new specifications will allow the test to be made afterwards. For example,

```
     PERFORM READ-LOOP
        WITH TEST AFTER
        UNTIL EOF-FLAG.
```

Control will always transfer to READ-LOOP at least once. The test option may also be specified with an in-line PERFORM.

EVALUATE Statement

The EVALUATE statement adds a multi-condition CASE construct to COBOL. The EVALUATE statement causes a set of subjects to be evaluated and compared with a set of objects. If the comparisons are all true, a specified group of statements is executed. For example,

```
EVALUATE HOURS-WORKED EXEMPT
   WHEN 0           ANY  PERFORM NO-PAY
   WHEN 1  THRU 40 ANY  PERFORM REG-PAY
   WHEN 41 THRU 80 "N"  PERFORM OVERTIME-PAY
   WHEN 41 THRU 80 "Y"  PERFORM REG-PAY
   WHEN OTHER           PERFORM PAY-ERROR.
```

The above example evaluates two data items, HOURS-WORKED and EXEMPT. If HOURS-WORKED is 0, any value for EXEMPT will be true and NO-PAY will be performed. If HOURS-WORKED is between 1 and 40, REG-PAY will be performed. If HOURS-WORKED is between 41 and 80 and EXEMPT contains "N", OVERTIME-PAY will be performed. If HOURS-WORKED is between 41 and 80 and EXEMPT contains a "Y", REG-PAY is performed. If none of the above conditions are true, PAY-ERROR is executed.

CONTINUE Statement

The CONTINUE statement is a no operation statement which indicates that no executable statement is present. It may be used anywhere a conditional statement or an imperative statement may be used. For example,

```
IF A < B THEN
   IF A < C THEN
      CONTINUE
   ELSE
      MOVE ZERO TO A
   END-IF
   ADD B TO C.
SUBTRACT C FROM D.
```

The CONTINUE statement allows control to go to the ADD statement following the IF when A is less than C. If the NEXT SENTENCE option had been used, control would have transferred to the SUBTRACT statement instead.

OTHER NEW FEATURES

There is a long list of other new features which should make the job of the COBOL programmer easier. The more significant ones are listed here.

Reference Modification

Reference modification allows you to reference a portion of a data item by specifying a leftmost character position and a length. For example,

```
MOVE A (3:5) TO B.
```

will move the third through seventh characters of A to B.

INITIALIZE Statement

The INITIALIZE statement provides the ability to set selected types of data fields to predetermined values. Assume RECORD-1 was described as follows:

```
01  RECORD-1.
    05  EMP-NO    PIC 9(6).
    05  EMP-NAME  PIC X(20).
    05  EMP-PAY   PIC 9(5)V99.
    05  JOB-TITLE PIC X(20).
```

The following INITIALIZE statements in the Procedure Division could be used to put values into the record:

```
INITIALIZE RECORD-1 REPLACING NUMERIC BY ZERO.
INITIALIZE RECORD-1 REPLACING ALPHANUMERIC BY SPACES.
```

The effect would be the same as:

```
MOVE ZERO TO EMP-NO EMP-PAY.
MOVE SPACES TO EMP-NAME JOB-TITLE.
```

De-editing

Under COBOL '74, it is not legal to move from an edited field to a numeric or numeric edited field. The new specifications will allow moving from a numeric edited item to either a numeric or numeric edited item. The edited item which is the sending item will be converted to its numeric value and moved to the receiving field.

REPLACE Statement

The REPLACE statement function is similar to that of a COPY... REPLACING except that the REPLACE statement operates on all source program text, not just text in libraries. Thus, if one of the new reserved words is used heavily in an existing program, you may want to use a REPLACE statement to change it. For example,

```
     REPLACE ==TEST== BY ==TESTT==
```

will replace all subsequent occurrences of TEST by TESTT in the source program until another REPLACE statement, a REPLACE OFF statement, or the end of the source program.

Optional FILLER

The word FILLER is now optional for data items which need not be named.

```
     01  A.
         05  B  PIC X(5).
         05     PIC X(5) VALUE "NAME:".
```

INITIAL Attribute

The INITIAL clause in the PROGRAM-ID paragraph indicates that every time the program is called, the internal data is initialized. This function is the same as the $CONTROL DYNAMIC option on the HP-3000.

```
     PROGRAM-ID.  SUB-PROG  INITIAL.
```

EXTERNAL Attribute

The EXTERNAL clause specifies that a data item or file is available to every program in the run unit which describes the data item or file.

```
     FD  FILE-1  IS EXTERNAL.
```

SYMBOLIC CHARACTERS Clause

The SYMBOLIC CHARACTERS clause in the SPECIAL-NAMES paragraph of the Environment Division allows the programmer to equate a name to a specific character. This feature can be useful for non-printable characters. For example,

```
     SYMBOLIC CHARACTERS BELL IS 7, CARRIAGE-RETURN IS 13.
```

This clause would allow a MOVE statement such as

```
     MOVE BELL TO A.
```

ADD Statement Enhancement

Under COBOL '74, the ADD statement allows either a TO or a GIVING format, but a statement of the form

```
ADD A TO B GIVING C
```

is not allowed. The new specifications will allow the TO before the last operand when the GIVING option is used.

Alphabetic Tests

Two new alphabetic class tests have been defined:

1. ALPHABETIC-UPPER will be true if the data item being tested contains only A-Z and spaces.

2. ALPHABETIC-LOWER will be true if the data item being tested contains only a-z and spaces.

TRANSITIONAL CATEGORY

There are some features of the current standard which are scheduled for a phased deletion. Implementations must still support these features in the new standard, but not in the subsequent standard. Certain clauses have been moved from the file control entry in the Environment Division to the file description entry in the Data Division and vice versa. The old locations are specified as transitional elements so implementations of the new standard must support programs which contain the clauses in either the old or the new locations. The following Environment Division clauses are included in the transitional category:

```
FILE-STATUS
RECORD KEY
ALTERNATE RECORD KEY
ACCESS MODE
```

The following Data Divison clauses are included in the transitional category:

```
BLOCK CONTAINS
CODE-SET
```

The Identification Division paragraphs are included in the transitional category in favor of the more general comment facility (* in column 7).

The INSPECT...TALLYING...REPLACING format of the INSPECT statement is included in the transitional category since the same function can be accomplished with two separate INSPECT statements.

DELETED FEATURES

The following features are not included in the next standard:

1. The ALTER statement.
2. The ENTER statement.
3. The MEMORY SIZE clause.

OTHER CHANGES

New status code values for file errors are being defined. These codes will cover situations which violate the standard but for which no standard status code was defined. For example, trying to open an indexed file in a program which declares it to be a relative file.

The order of the steps in a multi-conditional PERFORM...VARYING statement has been changed. Under COBOL '74, the statement

```
     PERFORM PAR-1 VARYING I FROM 1 BY 1 UNTIL I>10
                   AFTER   J FROM I BY 1 UNTIL J>10
```

would set I to 1 and vary J from 1 to 10 and then set J to 1, increment I to 2 and vary J until 10. The new specifications will increment I to 2 before setting J to I. Thus, on the second cycle, J will vary from 2 to 10 instead of 1 to 10 as under COBOL '74.

The new reserved word ALPHABET is required in the alphabet clause of the SPECIAL-NAMES paragraph.

```
     ALPHABET ASCII IS STANDARD-1.
```

RESERVED WORDS

The following new reserved words have been added:

ALPHABET
ALPHABETIC-LOWER
ALPHABETIC-UPPER
ALPHANUMERIC
ALPHANUMERIC-EDITED
ANY
CONTENT
CONTINUE
CONVERSION
CONVERTING
DAY-OF-WEEK
END-ADD
END-CALL
END-COMPUTE
END-DELETE
END-DIVIDE
END-EVALUATE
END-IF
END-MULTIPLY
END-PERFORM
END-READ
END-RECEIVE
END-RETURN
END-REWRITE
END-SEARCH
END-START
END-STRING
END-SUBTRACT
END-UNSTRING
END-WRITE
EVALUATE
EXTERNAL
FALSE
INITIALIZE
NUMERIC-EDITED
OTHER
PADDING
REPLACE
STANDARD-2
TEST
TRUE

STANDARDIZATION PROCESS

There are two committees which work on defining COBOL. The CODASYL COBOL Committee has the responsibility of developing the language. The ANSI X3J4 committee has the responsibility of standardizing the language. When working on a new standard, X3J4 can adopt specifications from either the previous standard or from the Journal of Development which reflects the work of the CODASYL COBOL Committee. If there is a problem with the JOD specifications, X3J4 must either subset the specifications from the JOD so that the problem does not appear in the standard or request that CCC resolve the problem. Both committees have representatives from implementors, users, and government. X3J4 currently has 23 members and holds six 4-day meetings each year. The work on the next standard is nearing completion as the committee is currently processing comments on the formal letter ballot of its members. Assuming that a two-thirds vote in favor the proposed draft is achieved, the document is forwaded to X3 who, in turn, votes to send it out for an official public comment period of at least four months. The X3J4 committee will review all comments received during this period. After all negative comments have been processed, the X3 committee votes on sending the draft proposed standard to ANSI to be formally processed as a new standard.

During the standard revision process, X3J4 has published information concerning its work in COBOL Information Bulletins. The latest one was CIB 19 which was published in May, 1980. Comments concerning the draft standard will be officially requested during the public review period; however, comments may be submitted earlier to:

Chairperson, X3J4
CBEMA
1828 L St. N.W.
Washington, D.C. 20036

Success with Manufacturing Applications: Implementation of Materials Management/3000

by
Beth Eikenbary
Manufacturing Systems Operation
Hewlett-Packard Co.

Introduction

1. Objective: The purpose of this paper is to describe and discuss key elements involved in defining the processes for the implementation of Materials Management/3000. This discussion is not intended to be complete - only descriptive, depicting the wide range of activities involed in the process of implementation. Of paramount importance throughout this paper is "user involvement". Any successful implementatin must work with the user community in all phases of the project.

2. Process: The following topics will be discussed in the body of this paper.

   a. Project Team Development: Identification of project team members and structure as an integral part of the implementation process.
   b. Operational Audit: Analysis of the current manufacturing organization, including the interfaces to other control systems.
   c. Develop Implemenation Plan:
      - General Design: Define system objectives; develop conceptual design models; and apply organizational constraints to the general design.
      - Detailed Design: Translate general design into a usable format for customizing Materials Management/3000.
   d. Customizing: Implement the modifications generated from the previous step. Verify these changes as you begin preparations for user training.
   e. User Implementation: Training schedules for all affected users should be established and adhered to.
   f. Installation: Perform all necessary testing of the customized Materials Management/3000 system, test interface programs and complete all required user training. A pilot system should be initiated at this time.
   g. Completed Implementation: Full-scale start up of the system with user reliance upon the system.

h. New System Audit: Between three and six months after implementation of any module, a system audit should be conducted to determine the success of the implmentation process.

3. Structure: The systems implmentation structure should be people-oriented, team-oriented, and participative geared toward the development, implementation and monitoring of the company's management systems. This structure, if properly developed and supported, will not only help avoid the confusion and frustration that usually accompanies automation; it will also provide a good balance between data processing and user organizations.

4. Summary: This paper will describe the implementation of the entire system. However, the actual implementation process is often made up of modularized implementation sequences. Modules of Materials Management/3000 may be installed in a variety of different sequences. The module sequence is is dependent upon the specific needs of the manufacturing organization involved. Determining the implementation sequence is a function of the Project Team and is accomplished during the Design Phase. These guidelines are appropriate for implementation of each module cluster as well as the entire system.

Project Team Development.

1. Objective: To define a project team within the company organization which will oversee the implementation process and ensure its success.

2. Process: The success of a manufacturing system implementation is dependent to a large extent upon the effectiveness of the Project Team designated to monitor the implementation. This team must have the authority and responsibility for all phases of implementation of Materials Management/3000. The group should be composed of working representatives from the affected departments, and should undergo a continuing education program in Materials Planning topics.

3. Structure: The following functions should provide the core for the Project Team.

```
                       ------------------
                       !   project      !
                       !   manager      !
                       !________________!
                               :
           :-------------------:--------------------:
           :                                        :
 ------------------                                 :
 !     user       !                       -----------------
 !     group      !                       !    systems    !
 !________________!                       ! administrator !
                                          !_______________!
```

The Project Manager is responsible for planning, orgranizing, and controlling the project team. A review of some of the tasks involved in successful management of the project team is included here.

* Planning activities, tasks, and end results
* Coordinating tasks and resources
* Interfacing between team and management
* Effective utilization of systems and user personnel
* Monitoring project status against plan
* Identifying problem areas (both functionally and technically)
* Solving problems and dealing effectively with crisis situations

User commitment is essential to the success of any information system implemenation. An individual should be identified as the focal point for user group involvement in the process. The responsibilities of this position are:

* Represent user needs and requirements
* Keep user departments informed of the project status
* Educate users in the concepts of formal manufacturing concepts
* Train users on effective utilization of the system

The System Administrator is responsible for the technical implementation of Materials Management/3000. These responsibilities include:

* Understanding user needs and including those requirements within the system
* System Customization
* Data entry and system conversion
* Interface with related systems
* System performance

4. Summary: Directed by the Project Manager, the Project Team is a key working group in the implementation process. The exact make-up in personnel of the Project Team will vary from company to company, however, it is essential that an individual be designated and held responsible for the functions outlined here.

Operational Audit

1. Objective: To ensure the customer has a complete understanding of their manufacturing organization, as it stands today; and to identify informational requirements of the system. The purpose here is to complete an objective analysis of the existing manufacturing systems (manual and automated). This will familiarize the project team with the current procedures, the people in the user departments and their functions, and the interfaces to the present system.

2. Process:
   a. Identify existing systems flows, i.e. manual and automated -- formal and informal. These systems include master scheduling, bills of materials (including work centers, standard routing & cost accounting information), inventory and order control, order processing, materials planning. Example of tools employed here are system flow diagrams, data base schema's, decision level analysis, information flow analysis and input/output analysis. Decision level analysis breaks a system down into decision points which control resourses (both tangible and intangible). Examples of resources are inventories, employee skills and so forth. These resources are defined and appropriate decision rules are formulated by management. The analysis determines how the information will be produced to meet the requirements of these decision rules. Information flow analysis shows what information is required, by whom, and from where it is obtained. Input/output analysis simply shows the data inputs and information outputs of a system without concern for decisions made from the outputs.
   b. Identify unsolved problems, e.g. production problems, people problems, software/hardware, etc..
   c. Identify policies, e.g. lead times, order policies legal, managerial, accounting, engineering, etc..
   d. Identify existing costs.
   e. Requirements analysis will be done by the systems analyst in the customers organization. There are three general steps involved, first the analyst and the user community need to generate ideas on what information is needed by them to effectively do their jobs. Second, this information must be validated for relevance and then prioritized. Last, the economical and technological feasibility must be assessed.

3. Structure: An established project team.

4. Summary: At this point we have a documented statement of where this organization's current information systems stands, both it's strengths and weaknesses.
Studying the existing system provides the analysis with: one, effectiveness of the present system; two, resource recognition, what is available in terms of hardware and people; three, conversion knowledge, when the new system is implemented the analyst has identified what tasks are necessary to begin operating the new system and to phase out the old system; and fourthly, he has a common starting point with management, to minimize 'resistance' of the change in the organization, the analyst can contrast the new system to the old system and demonstrate that it is not entirely new, and specifically show points of similarity and differences.

General Design.

1. Objective: To define system objectives; to develop conceptual design models; and to apply organizational constraints.

2. Process: Defining system objectives results from evaluating requirements described in the previous phase. The goal is obtained by abstracting certain characteristics from all the information requirements.
Development of conceptual design models entails developing high level flow diagrams which generally describe inputs and outputs of the system, treating the system as a 'black box'.
Once the model is established, the analyst begins to pragmatize it by applying the additional systems requirements (i.e. costs, performance, reliability, maintainability, flexibility, installatio schedule, expected growth potential and anticipated life expectancy and considering the available organizational resources.
Applying organizational constraints to the design process requires the extensive use of organizational resources. Many activities are pursued within the organization which also require use of these resources. Thus, the information system must vie with these other activities to obtain necassary resources. Organizational resources are usually allocated to those activities which will provide the greatest cost/effectiveness to the organization.
A summary of basic design alternatives is demonstrated below.

```
                      -----------------
                      !     user      !
                      !  requirements !
                      !_______________!
                              :
          :-------------------:--------------------:
          :                   :                    :
 -----------------    -----------------    -----------------
 !      do       !    !   design new  !    !  modify old   !
 !    nothing    !    !     system    !    !    system     !
 !_______________!    !_______________!    !_______________!
                              :
 -----------------    -----------------    -----------------
 !  develop new  !    !  develop new  !    !   purchase    !
 !  system with  !    !    system     !    !    system     !
 !  outside help !    !   in-house    !    ! from outside  !
 !_______________!    !_______________!    !_______________!
```

The analyst will have three basic design alternatives when evaluating a set of system and user requirements: (1) do nothing (2) modify existing system and (3) design new system. When the decision is to design a new system, the analyst considers the 'make' or 'buy' alternatives. In our case the decision is to buy.

3. **Structure: Generally the HP3000 Industry Specialist is not actively involved in these steps.**

4. Summary: At this time the customer has identified where they are and and where they want to be. This information is the basis for customizing the Materials Management/3000.

Detailed Design - Preparation for Customizing.

1. Objective: To translate the requirements specified in the general conceptual design into a format which will facilitate customizing Materials Management/3000; to propose an implementation strategy for this system; and to install the standard Materials Management/3000 system. The purpose is to minimize the total elapsed time of implementing a usable system.

2. Process: The industry specialist will provide the customer with aids which will facilitate the initial customizing requirements. These pre-customizing aids reflect the standard system as it leaves the factory, they are contained in the Industry Specialist Consulting Kit. They are as follows:
   a. Schema information - detailed description of all the data items, their data types, field sizes, default values and the data sets they appear on.
   b. Screen definitions and layouts.
   c. Report layouts.
   d. Forms file listings - listings from VIEW of the forms file, this will be done at the customer's if possible.

   The System Administrator's manual describing customizing contains all the information you will need to list other customizing aids. These are not necessary at this stage, but if for some reason they are needed the industry specialist will have access to these aids at their office.
   The proposed implementation strategy is a prototype development plan. That is, to first bring up the system as is, then start the customizing in phases. This piecemeal approach will get the customer up on the system allowing him to become familiar with it and to start their testing. The customizing will be completed in phases, thus minimizing overall complexity of the project, since it is easy to recustomize the system.

3. **Structure: The Industry Specialist is now taking an active role in the implementation design, working with the customer in preparing for system customization.**

4. Summary: This phase contains two main events one, installing the standard system and two, preparing for customizing. Outputs will be the initial required modifications of the standard product. That is, discrepencies in data items and data sets; and screens to be modified, added or deleted. preliminary investigation for the other parts of customizing will be started now and initial customizing will commence.

Customizing Materials Management/3000.

1. Objective: To complete the required customization of Materials Management/3000 to fit the needs of the customer; to identify measures of data base integrity; to identify and implement security requirements; and to initiate user training.

2. Process: The needed processes will be broken down into functions. The timing is implied by the order presented for tasks a, b & c.
   a. Organize the screens by category, i.e. add, change, delete. Complete VIEW screen layout forms (Appendix I), keep them ordered by category. Next either RUN FORMSPEC or enter VIEW via the customizer. In VIEW, the order in which you modify the forms file is not important, but it is best to do all adds at one time and so on for changing and deleting. Remember to compile the forms file before exiting VIEW.
   b. The next step is the data item definition within customizer. This entails the adding, deleting and changing of data items existing in Materials Management/3000. Use the Data Items Work Sheet (Appendix II) to organize your modifications to the data bases. This form allows you to organize these updates by data set. This sheet's information can also help with assigning the new data items to data sets, via the 'ADD FIELDS TO A DATA SET' screen.
   c. Following the definition of your data items you can modify the standard sreens supplied by Materials Management/3000. There are two functions involved in modifing screens in customizer, first is screen definition and secondly defining the screen layout. Note that for MENU Screens you only define the screen, there is no layout function. The Screen Worksheet (Appendix III) contains both functions. The top portion corresponds to the screen defition and the bottom is the screen layout. When entering this information on the system, you should do all your sreen definitions together and then do all your screen layouts. The reason for this separation is two fold, first the screen has to be defined before you can do the layout and secondly these functions are separated in the system. Therefore, it is better to do the screen defintions first then do the layouts.
   Now that we have modified all the necessary screens and data sets, we can develop and enter the processing specifications. There spec.'s are not the same as the ones seen in VIEW, they are however a mechanism which will enable the user to perform some computional operations on data at the time the screen is being processed by Materials Management/3000. Appendix IV contains a Processing Specifications Worksheet, this can be used in organizing and entering the required operations for the screens.
   d. The above processes are the main events of customizing, there are many other tasks which must be performed prior to the final implementation of Materials Management/3000. Here the order is not important. These functions are:
      (1) Terminal Configuration -- Appendix V;
      (2) Report Layout Modifications -- Appendix VI;
      (3) Background Job Schedule -- Appendix VII;
      (4) System Values -- Appendix VIII;
      (5) Customizing Messages;

```
                CUSTOMIZATION --- THE MAIN EVENTS
                ---------------------------------

 ------------------
 !                !                    *Add, change, delete sreens
 !      VIEW      !                     and data items.
 !                !                    *Verify*
 !________________!
         :
         :
- - - - -:- - - - - - - - - - - - - - - - - - - - - - - - -
         :
 ------------------
 !                !                    *Add, change, delete data items
 !   DATA         !                    *Assign data items to data sets
 !   ITEM         !                    *Verify*
 !   DEFITION     !
 !________________!
         :
         :
         :
 ------------------
 !                !                    *Add, change, delete screen
 !   DEFINE       !                     definitions for menu, retrieval
 !   SCREEN       !                     and data entry.
 !                !                    *Verify*
 !________________!
         :
         :
         :
 ------------------
 !                !                    *Adding, changing screen layout
 !   DEFINE       !                    *Sequencing and description of
 !   SCREEN       !                     data items on the screen
 !   LAYOUT       !                     for retreival and entry only.
 !________________!                    *verify*
         :
         :
         :
 ------------------
 !                !
 ! PROCESSING     !                    *Add computational features to
 ! SPECIFICATIONS !                     the screens.
 !                !                    *Verify*
 !________________!
         :
         :
- - - - -:- - - - - - - - - - - - - - - - - - - - - - - - -
         :
         :
 ------------------
 !                !                    *Configuration of terminals
 !    OTHER       !                    *Report Layout Modifications
 !    EVENTS      !                    *Background Jobs
 !                !                    *System Values Modifications
 !________________!                    *Messages Customization
                                       *Help Screens Customization
                                       *Security
                                       *Database Generation
```

-- FIGURE 1 --

(6) Customizing Help Screens;
(7) Security;
(8) Database Generation.

Functions (1) through (4) have associated worksheets provided in the appendices to help in organizing, entering and verifing the results on the system. All these tasks are addressed in the System Administrator Manual. Also security can be done at various levels in this system, at the lowest level -- data item level, at an intermediate level -- VIEW and at the highest level -- terminal security.

3. Structure: A full staff with programmers to design and program the the interfaces between Materials Managment/3000 and the other functions of the company (e.g. accounting).

```
                              ----------------
                              !   Project    !
                              !   Manager    !
                              !______________!
                                     :
                                     :
  :---------------:------------------:---------------:--------------:
------------    ------------    ------------    ------------    ------------
! Systems  !    !          !    ! Systems  !    ! Technical!    ! Hardware !
! Analyst  !    !Programming!   ! Admin.   !    ! Support  !    ! Planning !
!          !    !          !    !          !    !          !    !          !
!          !    !          !    !          !    !          !    !          !
!          !    !          !    !          !    !          !    !          !
!__________!    !__________!    !__________!    !__________!    !__________!
```

Each function shown in the above chart has certain duties to perform, they will vary from company to company. Some responsiblities of the system analyst would be the initial analyst work of the materials and manufacturing departments,analyst work on Materials Management System, training, manual procedures, file conversion, conversion from the old system to the new one and design work on the interface programs. The programmer would be involved with writing interface and conversion programs, file conversion and system testing. The systems administrator would be involved with customizing, user training, and some analyst work. Technical support and hardware planing will be supporting and planning for the hardware. Finally, the project manager will be responsible for the administration of the entire project. This person will also be communicating with top management and other key managers soliciting their support and reporting current status of the project.

4. Summary: Following this phase we would have completed all the necessary tasks in customizer and VIEW. We will have a fully cusomized system. The validity of all the modifications rests on a complete and precise validation of each task after is completed. The forms supplied in the appendices provide an effective means to verify your results by comparing them with the aids supplied by the customizer. These forms are designed such that they are consistent with the screens in customizer and with each other, thereby minimizing the complexity of this verification process. Once the modifications have been validated by the systems administrator (and even in conjunction with the analyst) we are ready for our initial tests.

User Implementation

1. Objective: To establish education and training programs for end-users to ensure achievement of skills necessary for proper use of the system.

2. Process: Education and training must translate the general principles of MRP into the specifics of operations at the company. Education is defined here as the learning of manufacturing concepts and basic operations of the system. Training is the develop of skills in using the particular tools provided by the system. Education and training should be provided at the appropriate level of detail and on the appropriate portion of the system for all personnel that can have an effect on the success of the system. A generalized outline is included here.

   1. Hewlett-Packard training courses on the HP3000
      * Programmer's Introduction
      * System Operator
      * System Manager
      * Image--optional
      * V/3000--optional

   2. Hewlett-Packard training for Materials Management/3000
      * System Administrator I
      * System Administrator II
      * These courses are designed to "train the trainers"

   3. Formal Manufacturing Concepts
      * ASI video courses on Managing Production and Inventories and MRP.
      * These courses should be given to personnel at all levels of operation including top management.

   4. In-house User Training
      * Each department is responsible for training personnel on the specific use of the computer and the application.
      * Procedures and decoumentations for users, operations, and the system administrator should be completed at this time.
      * Development of backup-recovery and disaster plans.

3. Structure: The Project Team is expanded to include department representatives and training teams.

4. Summary: The people part of an MRP system is fully 80% of the system. The system will work only when people uderstand what it is, how it works, and what their responsibilities are. For this reason, education and training are critical steps in the implementation process.

Installation

1. Objective: To install a pilot start-up system and to perform all required system tests which would insure the integrity of the Materials Management/3000 system.

2. Process: Several controlled test must be completed prior to final conversion to the new system. A start-up pilot installation can provide an excellent format for testing the system. There are three main areas to test:
   1. Loading programs--data base loading;
   2. System tests--verification of customized system;
   3. Interface progams--verification of user programs.

   A pilot system also gives the operating departments an opportunity to learn the new system functions while the daily volumes are still at relatively low levels.

3. Structure: The entire project team will be involved in this phase.

4. Summary: The pilot start-up is a necessary phase of the implementation process. The pilot system can help identify problems and solutions before the total system is in place. Solutions at this stage can be more readily implemented than at a later point in system conversion. Firm timetables should be established for completion of the pilot phase, with adequate time for implementation of operating procedure changes as problems are identified.

Completed Implementation

1. Objective: To complete system installation and to establish effective utilization of the system by the users.

2. Process: At this point in the implementation, users have been trained and the system has been tested. The final steps at this stage are:
   1. Load data bases;
   2. Perform test runs of batch processes;
   3. Initialize and test interface programs;
   4. Switch over from existing systems.

   Depending upon the complexity of the conversion, it may be advisable to run modules of Materials Management/3000 in parallel with existing systems until users have gained complete confidence in the system.

3. Structure: The entire project team, as well as affected users, will be involved in this phase.

4. Summary: At this time the system has been installed and tested to user satisfaction. The success of the application is now the responsility of management. It is extermely important that users realize that management plans to run the business based upon the system. In doing this, it will become clear that the new system is designed to be an integral piece in business operations.

New System Audit

1. Objective: To evaluate the success of the implementation and the system.

2. Process: Between three and six months after implementation of any module of Materials Management/3000 there should be a formal system audit. Questions to be answered at this step in the implementation process are:
   * Are users effectively utilizing the system capabilities?
   * Are operating policies being adhered to?
   * What problems exist with the system?
   * What user requirements were overlooked?
   * Can the system accomodate these changes?
   * How are you preforming against established measurement targets?

   Any problems identified in the audit should be documented, investigated, and resolved.

3. Structure: The primary project team should be involved at this time.

4. Summary: A successful implementation of Materials Management/3000 should take between nine and 12 months. There are three keys to the success of any implementation:
   1. Strong project management;
   2. Thorough training of users, systems personnel, and management;
   3. Top management commitment.

   Without these three ingredients, implementation will slow down, frustrations mount, and users doubt the "reality" of the new system. Once momentum has been lost during this process, it becomes doubly hard to restore.

Conclusions

This paper was not intended to be an indepth guide to implementation of Materials Management/3000, but a document which highlights the issues and concerns surrounding the implementation of any manufacturing application. The success of any manufacturing application ultimately resides with the effeciveness of the implementation process. A key point to remember is that manufacturing organizations are not static; they will change. These changes will need to be reflected in the organization's information systems. Implementation is a process rather than a project; when these changes occur, it is again time to reiterate the analysis and customization of the Materials Management/3000 system.

Acknowledgement

I would like to recognize the assistance of Tim Mahoney in the design, development, and writing of this paper. Without the "unpublished paper" this paper would not exist.

```
Form:                                                            [ ] ADD
  Repeat Option: _____                                           [ ] CHANGE
  Next Form Option:___                                           [ ] DELETE
  Next form: ________
                         SCREEN LAYOUT
*****************************************************************************
```

```
*****************************************************************************
Field: __________________
 Num: 1 Len:    Name:____________________ Enh: HI____ Ftype:_____ Dtype:____
 Init Value:__________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:____________________ Enh: HI____ Ftype:_____ Dtype:____
 Init Value:__________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:____________________ Enh: HI____ Ftype:_____ Dtype:____
 Init Value:__________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:____________________ Enh: HI____ Ftype:_____ Dtype:____
 Init Value:__________________________________________________
*** PROCESSING SPECIFICATIONS ***
```

```
Field: __________________
 Num: 1 Len:    Name:___________________ Enh: HI___ Ftype:____ Dtype:____
 Init Value:________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:___________________ Enh: HI___ Ftype:____ Dtype:____
 Init Value:________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:___________________ Enh: HI___ Ftype:____ Dtype:____
 Init Value:________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:___________________ Enh: HI___ Ftype:____ Dtype:____
 Init Value:________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:___________________ Enh: HI___ Ftype:____ Dtype:____
 Init Value:________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:___________________ Enh: HI___ Ftype:____ Dtype:____
 Init Value:________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:___________________ Enh: HI___ Ftype:____ Dtype:____
 Init Value:________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:___________________ Enh: HI___ Ftype:____ Dtype:____
 Init Value:________________________________________________
*** PROCESSING SPECIFICATIONS ***

Field: __________________
 Num: 1 Len:    Name:___________________ Enh: HI___ Ftype:____ Dtype:____
 Init Value:________________________________________________
*** PROCESSING SPECIFICATIONS ***
```

Appendix I (cont.)

DATA ITEMS WORK SHEET

[ ] ADD
[ ] CHANGE
[ ] DELETE

DATA SET [ ]
DATA DEFINITION

Data Item Name	Item Length	Decimal Places	Data Type
[ ]	[ ]	[ ]	[ ]
Default Value [ ]			

Data Item Name	Item Length	Decimal Places	Data Type
[ ]	[ ]	[ ]	[ ]
Default Value [ ]			

Data Item Name	Item Length	Decimal Places	Data Type
[ ]	[ ]	[ ]	[ ]
Default Value [ ]			

Data Item Name	Item Length	Decimal Places	Data Type
[ ]	[ ]	[ ]	[ ]
Default Value [ ]			

Data Item Name	Item Length	Decimal Places	Data Type
[ ]	[ ]	[ ]	[ ]
Default Value [ ]			

Data Item Name	Item Length	Decimal Places	Data Type
[ ]	[ ]	[ ]	[ ]
Default Value [ ]			

```
                            SCREEN WORKSHEET
              [ ] ADD SCREEN                                  [ ] DATA ENTRY
              [ ] CHANGE SCREEN                               [ ] RETRIEVAL
                                                              [ ] MENU

------------------------------------------------------------------------------
!                Screen Name                    Transaction Code             !
!               [                ]          [                  ]             !
!                                                                            !
!          Exit Screen Name       Help Screen Name      Next(Detail) Screen  !
!          [                ]    [                ]    [                  ] !
!                                                                            !
!                         Function Key Definitions                           !
!                                                                            !
!      1[            ] 2[            ]  3[            ] 4[            ]      !
!      5[            ] 6[            ]  7[            ] 8[            ]      !
!                                                                            !
!                                                                            !
!----------------------------------------------------------------------------!

                            SCREEN LAYOUT

------------------------------------------------------------------------------
!                                                                            !
!                            Screen Name                                     !
!                           [               ]                                !
!                                                                            !
! Field                                  Data         Length    Dec.  Required !
!  Seq.  Field Name                      Type       Min. Max.  Places  Field !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
! [   ] [                        ] [          ] [  ] [  ]  [   ]  [   ] !
!----------------------------------------------------------------------------!
```

APPENDIX III

PROCESSING SPECIFICATIONS WORKSHEET

Screen Name	Sequence Number	Processing Specifications
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________
		Operation:__________ Operand 1:______________ Operand 2:______________ Comment:______________________

TERMINAL CONFIGURATION WORKSHEET

Subsystem:________

Terminal Name	Log. Unit#	Scheduled Start	Time Stop	Start Screen	Security Time-out	Aux. LP

APPLICATION REPORT WORKSHEET

Report Name:________________

Field Name	Start Position	Length	End Position	Data Type	Decima Places

BACKGROUND JOB SCHEDULE WORKSHEET

```
Job Name                 Auto   Run Cycle      Run Day/Month        Run Time/Shi
------------------       ----   ---------      -------------        ------------

   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
>
   Description:
  Control File:
--------------------------------------------------------------------------------
```

SYSTEM VALUES WORKSHEET

Value Name	Data Type*	Value

* Data Type can not be modified.

A SUBSYSTEM THAT IMPROVES RESPONSE TIME FOR APPLICATIONS WITH LARGE NUMBERS OF TERMINALS

RALPH HENNEN and BOYD CARLSON

OHIO STATE UNIVERSITY RESEARCH FOUNDATION
and
SYSTEM RESOURCES GROUP, INC.

INTRODUCTION

Time-sharing has always plagued users with poor system response time and relatively high expense, though systems eventually evolved which alleviated both conditions. Chip processors and chip memory added to the attraction of time-sharing and it began to develop as a system rather than an option to a batch system.

As the systems became more sophisticated, however, so did the applications which were run on those systems. Because of this, time-sharing systems frequently frustrated the user.

Distributed systems, too, offered a breakthrough for response times in large systems. They allowed more users to access stored information without excessive response time. Their drawback was that they only applied to a large data volume/low transaction environment.

So it seems that there is a third need for improved service in the system-and-user market: frequent transactions with associated low volume of data and processing. Though both the time-sharing and the distributive systems are overelaborate for this need, their response to this type of application can still be extremely sluggish.

Applications which belong to this class of fast transaction-low data are: plant production control, computer auctioning of goods, security monitoring systems, data entry, data inquiry, and monitor control. One such design is currently in production on the HP3000.

THE NEED

This design is the result of a research project which required a CPU to service, with a response time of two seconds or better, at least 60 terminals. The project budget complicated the order; it would not allow for a large, main frame computer. The salesmen doubted that it was possible to accomplish a task of this scope, and the hardware search increased their doubts. But the HP3000 turned out to have the system tools we needed to develop the required system. Much of the system design work had been done on what was known as a "run-time system."

This system was to be used for another almost unheard-of application: the electronic market. An electronic market facilitates trading between buyers and sellers over the terminal using the CPU as auctioneer, bookkeeper, banker, statistician, and marketing information source. Several markets trade simultaneously. Other users can do other things during trading, such as getting detailed information on the goods to be sold, getting price information for goods already sold, or even looking at the regional weather forecast.

The user audience in this case was to be diverse. There was no guarantee that all terminals which might potentially talk to the system would have identical characteristics. Not only were authorized buyers and sellers to use the system, but several news services were interested in tapping into the marketing information for their customers.

Communicating with the users was not going to be a simple task. We had little control over users who already had a communication network and who simply wanted to connect to our system. Our communications would have to accommodate the Bell System as well as several other vendors. We were to serve both leased line and dial-up users. Some of the users were on point-to-point lines and asynchronous multipoint lines.

As information flowed into our offices regarding what we needed to do to bring the users onto the system, it became obvious that we needed all the tools we could find to make the system work. The puzzle became larger and more difficult daily.

THE RESEARCH

The system design task for the run-time system took about five months to accomplish. During that period, we evaluated five different systems with regard to their potential efficiency, programmability, maintainability, and practicality. In time we developed a design that seemed potentially compatible with the HP3000 hardware and some of MPE III.

We designed the system with structured and modular programming techniques. The modules were separated into two groups: system and application. The application modules solved a particular problem between user and data; the system modules connected the hardware to the user. The communication modules were coded to be tested apart from the rest of the subsystem. This separation gave them adaptability in case we had to change them or add other codes. Many field situations required considerable tuning of the communication modules.

The true application programs were lowest in the hierarchy of this subsystem because of convenience. The sponsor who had the final word on how the application programs would run did not always agree with us about how they should work. Often the sponsor could not foresee future needs to an extent that would allow us to write definitive specifications for the application programs. Fortunately, the application modules had little bearing on the system code.

After we attached the application programs to the completed subsystem, we began testing and tuning. With alteration of some subsystem parameters and changing of some MPE III configuration parameters, the run-time system performed much better than our team or anyone else had expected. For our application, the response time was better than 1.5 seconds. We inserted pauses in the code to slow the execution so the users could expect uniformity of response and to let them get used to the cadence of activities at the terminal.

THE SYSTEM

The system is a collection of processes which are run under MPE III. These processes are organized to communicate data and to synchronize connections between many users and many application programs, as shown in Figure 1. Much of the work which would be done by MPE in the time-sharing environment is managed under the subsystem. The disadvantage of this structure is that users do not have the use of the system hardware that they would have in a time-sharing environment. To the contrary, the hardware resources are transparent to the user on the subsystem; they are available through the applications level. This organization is an attempt to relieve the overhead of managing the system resources to control the I/O to the user.

This subsystem minimizes the movement of data throughout the processes. Data bound for system I/O to the user are stored in a memory buffer. The characteristics of the data are passed through the subsystem and the data are accessed only at the point where the application must use them. In the same way, data produced by the application program are placed in a buffer; they are not moved until they leave the system.

Figure 2 illustrates the design of the subsystem, showing it between the MPE environment and the application environment. Figure 3 illustrates in more detail the core processes and programs of the subsystem. The system is loaded and executed at a terminal which becomes the run-time console. This console is a session under MPE; it receives information from both the run-time system and MPE. The console gives the operator control over the rest of the subsystem. The operator may create initial conditions of the system. These conditions may vary, depending upon the application environment the subsystem is serving. The operator can also monitor certain aspects of the system and intervene when necessary.

The I/O Controller handles all the communication activities and problems. This module works closely with the I/O driver provided by MPE to intercept the I/O from external devices. It handles the communication protocol information that the drivers pass to the subsystem and checks errors on the communication lines. The I/O Controller regards the remote I/O device as the primary initiator of the input. It intercepts input as it comes; when the input is complete, the data are accounted for and ready for processing. The I/O Controller then sends a message to the next process on the system. That message contains all the characteristics of the data and in most cases also indicates which application program should be used to process the data. In much the same way, the application program then returns data through the system and back to the terminal.

The I/O Controller controls several resources within the subsystem. It initializes and maintains tables that track activity within the system and maintains and keeps records of all I/O activities on all devices. All information collected by the I/O Controller can be made available throughout the subsystem.

The I/O Controller distinguishes between clock-dependent activities and those which are not clock-dependent. Clock-dependent activities may require a specified delay between request and response. These activities have the highest priority for response in the system. The attached application programs must then run faster than user-specified response time. Other clock-dependent transactions receive data at the remote device without solicitation. These transactions include countdown information, timer processes, pulse bursts, and monitor requests. Activities which are not clock-dependent pass through the other part of the system and may include application programs with variable execution times between user requests.

These two classes of activities have separate Interrupt Processors. Both processors queue requests that may use the same system resource or application program. They wake up the application program appropriate to the requests and issue errors for bad syntax and unknown requests. All usable applications for each generation on the system are known to the processors. It is necessary to reconstruct the system before changes in the available programs can be attached.

Generally, each terminal or remote device request will stimulate a response from some application level program. This transaction will cause a complete pass through the system (down and back) and one pass through the application program. In some instances, the application program level may be in a conversation-like mode with the user device or terminal: for example, when the user must choose among the items of a menu produced at the terminal. The Interrupt Processor must then remember which terminals are in that mode, and the application program must find the position in the code to which the transaction must return. This position is stored and becomes part of the communication structure between the interrupt process level and the application level. The application program may tell the Interrupt Processors that the transaction is incomplete and indicate the code location to begin execution upon further input from the device. This allows the application program to process another request while the user (or user device) is deciding which option to take. Should the user choose not to continue the chain of activity to the particular application program, the communication between the I/O Controller and the Interrupt Processors will flag that case. All pending activity for the device is then cancelled and the new request becomes a new transaction. This type of communication allows the subsystem to be somewhat transparent to the user and gives the user interactive capability within the program.

Other small system programs do not appear in these generalized diagrams. They manage some of the common resources in the run-time environment. These resources can be thought of as highly dependent or shared resources in the subsystem. Their use dictates the results of some transactions. For example, in the situation of the electronic market, each item offered to the market is posted on a market screen. The transaction that results in an offer is completely separate from those which play the market, though the two activities are tied together by the market screen. The market screen must have a governor to ensure that all offered items are advertised to all marketeers, each of whom is interacting with the market screen uniquely. These management processes tend to change in character and number as the run-time environment dictates; resource management for an electronic market would not be the same as for a production control run-time system, for example, or a security monitoring system.

THE INSTALLATION

This run-time system can be thought of as an application system. Since the application environment is usually tailored to a specific user audience, run-time systems vary.-Each system is designed to be installed for each new application environment. The system is modular, and, though a few core system programs occur in all instances, the remaining peripheral software can be installed by trained personnel to accommodate the application environment. Not all applications can take advantage of this type of subsystem; programs which massage a large volume of data or perform large calculations with few user transactions will receive little value from it.

Existing needs and software should be evaluated regarding their compatibility with the subsystem before installation, but ideally the application programs should be written with the subsystem in mind.

Many of the rules for efficient stand-alone programs apply to this environment. Code written with some consideration of segmentation, memory allocation, and possible thrashing problems will, of course, run more quickly. Code designed in a structured, top-down fashion with one pass through for each transaction state is ideal. Programs which use data bases efficiently by limiting the number of cross datasets searches will run faster. Single programs which do not try to accommodate every user's needs at once will be at an advantage.

The subsystem offers a host of tools which can be used successfully at the application program level. The library includes I/O statements for user device accessing, error diagnostics to test for errors in the operations, error file operations for user-defined errors, status checking for the transaction condition codes and data location, and others.

These subsystem monitors also provide tools for the system environment. Files can be allocated to save the monitor information for inspection at a later time. This log of system activity can be useful for tuning the system.

A special consideration at installation time should be the various devices attached to the subsystem. The communication modules in the I/O Controller will have to be selected and installed based on the communication protocol and hardware interface for the device. This will require different considerations for different communication subsystems.

THE PERFORMANCE

The run-time system is in production; it has been performing satisfactorily for over a year with few modifications of the code.

The largest production encompasses 55 terminals. Under normal operation, response time has shown no noticeable degradation, even with increased activity within the run-time system. The main system currently feeds to five time-sharing terminals which are used for program development and editing large data sets. They run simultaneously with the run-time system and 55 terminals. There is no noticeable degradation in response time at either the run-time terminals or the time-sharing terminals, with one exception. There is a slight delay in the time-sharing terminals when the Clock Interrupt Processor is sending countdown information every 1.5 seconds to 40 or more terminals.

Performance data are being collected from inside the run-time system as well as in the time-sharing environment under MPE. Information is being collected in the run-time system as to the average process time for different types of application programs: those which are clock dependent, those accessing external file data sets, those accessing IMAGE data sets, those broadcasting information to more than one receiving terminal. In the time-sharing environment, data are collected as to execution times for a mix of compiles, preps, edits, data entry and access, and streamed jobs. This is all done for the stand-along, time-sharing environment; stand-along, run-time environment; and the mixed environment, for the purpose of comparing the differences among the environments.

Several system configurations under MPE obviously play a significant part in the response time in the run-time system. The parameters are time quantum, max extra data segment size, and max data. These parameters will take on different values for different configurations of the run-time system. These should be monitored and tuned before the run-time system is put into production.

CONCLUSION

Many hybrid systems are appearing in the manufacturer and OEM market. These systems reduce the programming load for the installations who buy the software. It costs less to purchase an accounting system than it does to construct one in-house.

The run-time system provides more efficient use of a CPU for installations serving large numbers of transactions with data volume in standard CRT screen unite (1920 bytes). The added benefit of this subsystem is the ability to communicate to a large number of terminals attached to a single CPU.

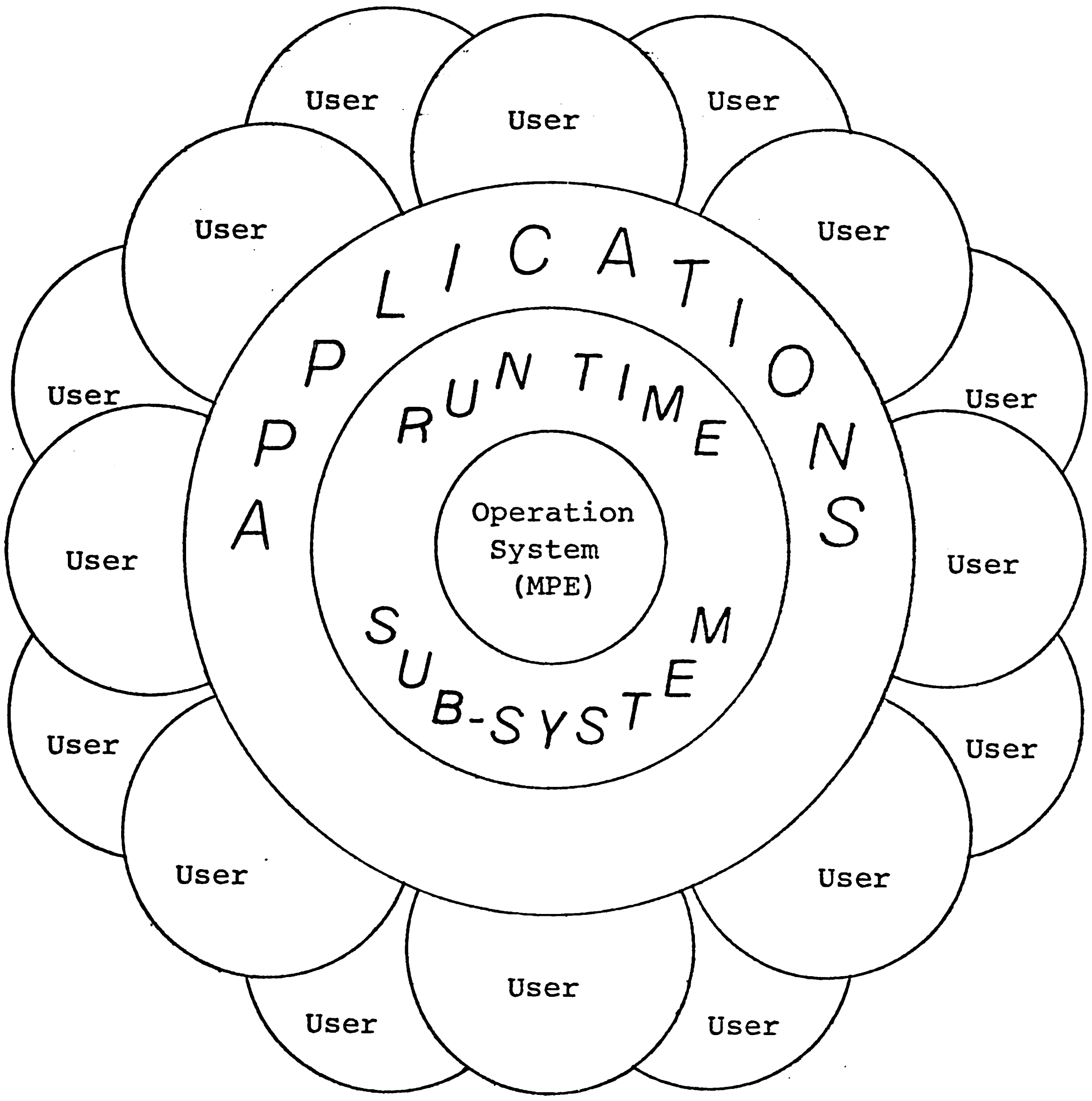

User
User
User
User
User
User
User
User
User
User
User
User
User
User
User
User
User
User
APPLICATIONS
RUNTIME SUB-SYSTEM
Operation System (MPE)

Figure 1

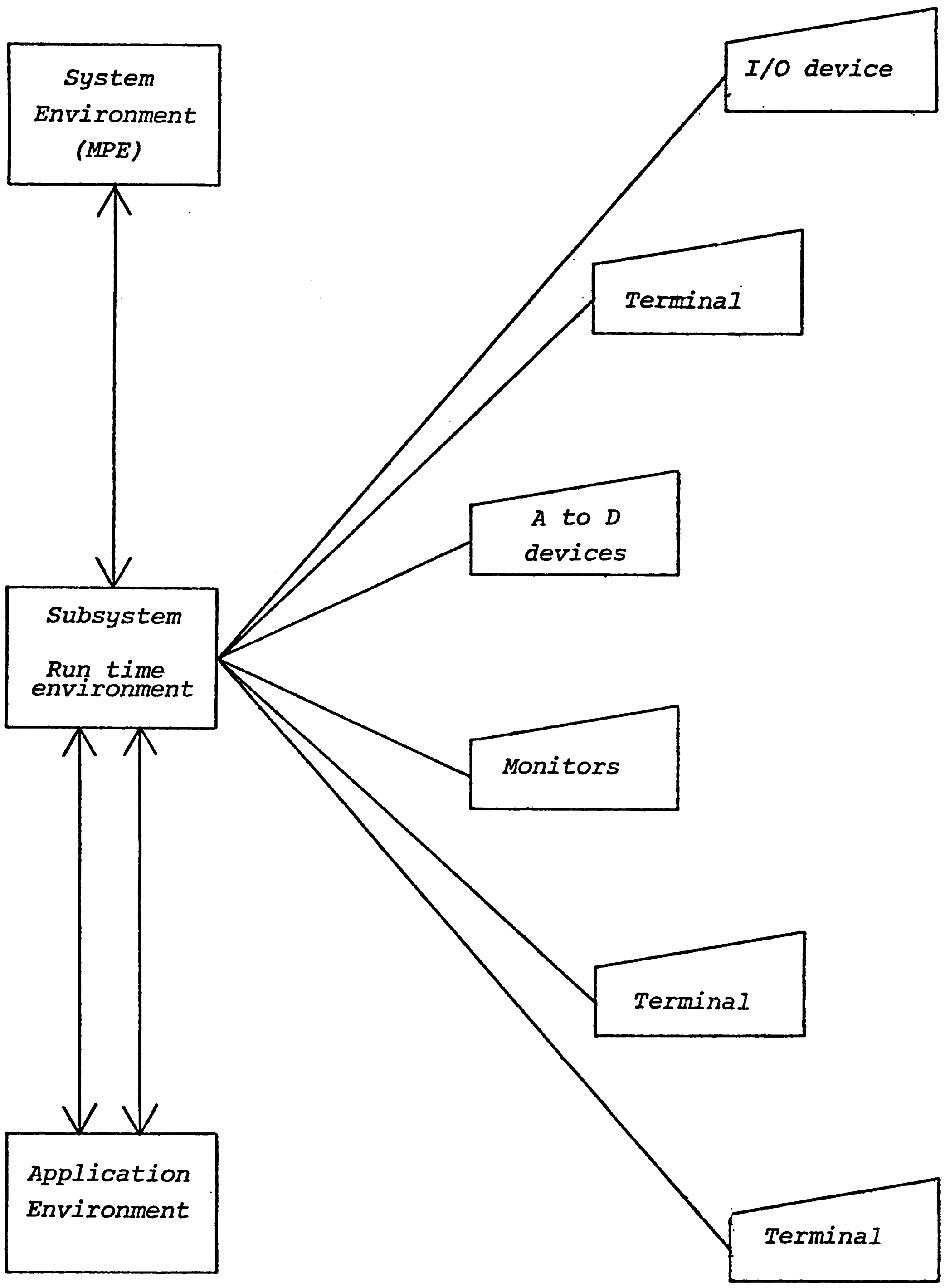

System Environment (MPE)
I/O device
Terminal
A to D devices
Subsystem Run time environment
Monitors
Terminal
Application Environment
Terminal

Figure 2

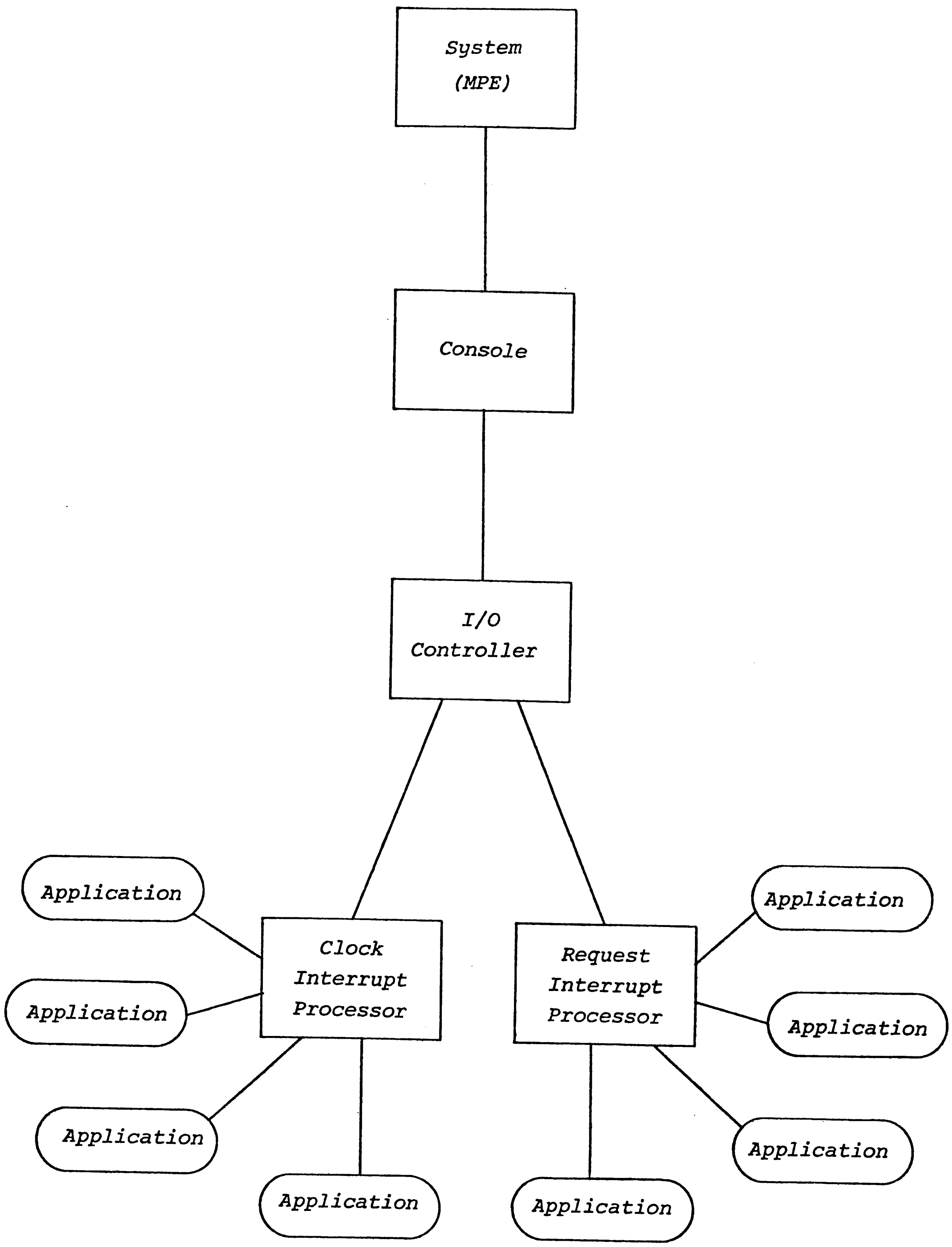

System
(MPE)
Console
I/O
Controller
Clock
Interrupt
Processor
Request
Interrupt
Processor
Application
Application
Application
Application
Application
Application
Application
Application

Figure 3

COMPUTER AIDED LEARNING AT UTC

Dr. Lloyd D. Davis

Introduction

With approximately 8,000 students, 250 plus faculty, and a primarily undergraduate orientation, the University of Tennessee at Chattanooga is very typical, in terms of mission, role, and scope, to other four year state colleges and universities. Somewhat unusual are the computer resources dedicated to instruction and research at UTC. With both a HP2000 and HP3000 allocated solely to instruction and research, with over eighty terminals accessing these computers and with a vigorous acquisition program for selecting quality software, UTC has "infused" computing into many areas of its curriculum including several unusual subjects. Additional computing via the network approach is also done at UTC for both instruction and research. This presentation will detail these efforts in terms of four areas - problem solving, data analyses, simulations, and computer-aided instruction. In addition, the topic of research computing will be discussed in terms of packages, availability, and general utility. The thrust of this presentation will be that of detailing the ingredients for establishing a successful academic computing service and maintaining that service for students and faculty.

General

The program to promote computer literacy at UTC has been in existence for six years. Prior to the establishment of Academic Computer Services student computer programs were run on a local basis on an IBM 360/30 or remotely for a brief period on an IBM 360/65. There was little or no provision for Computer Aided Learning (CAL).

The Office of Academic Computer Services opened on July 1, 1975 with the objective of providing competent and comprehensive computer facilities to faculty and students in the areas of instruction and research. For the past six years, the Academic Computer Center has sought to fulfill this goal. It was originally staffed by a full-time director and secretary; since then, another full-time employee has been added and student-programmers have been hired on a part-time basis. Funded by the University of Chattanooga Foundation, an HP2000 was installed in October 1975 and was linked to thirty-two terminals located in clusters around the campus. In November 1978, a second academic system, an HP3000, was installed along with additional terminals.

The six years since 1975 have been spent in consolidating these systems and acquiring programs suitable for college level usage. During this time, efforts have been made, using newsletters and workshops, to introduce the faculty to the facilities available and to encourage them to undertake authoring projects and research.

Usage

Departments with heavy usage of Academic Computing include Business Administration, Mathematics, English, Engineering, Computer Science, Psychology, Chemistry, Biology and Physics. Moderate usage is experienced in each of the departments of Economics, Education, Criminal Justice, Political Science and Sociology. Other departments, such as Music and Home Economics, make occasional use of the services.

More impressive is the fact that departments not normally strongly associated with computer expertise have asked for and received services. Two examples of this spread of computer knowledge are the Music Department, which has a terminal for drilling students in music rudiments, and the English Department, which has a dedicated cluster of terminals for Computer Aided Instruction (CAI).

Faculty Involvement

At this point, it should be emphasized that the faculty participate not only by using our facilities but go far beyond that by being strong initiators of new programs. Many of the faculty have devoted endless hours to writing new packages, creating CAI materials, reading professional journals for software sources, entreating Academic Computing to purchase materials of interest, and evaluating these materials once installed.

One of the most important software sources is CONDUIT, which is located in Iowa City, Iowa on the campus of the University of Iowa. CONDUIT's purpose is to locate quality software appropriate for undergraduate education (which often includes the high school level), to package it in a form that is widely and easily transportable, and to market these materials nationally. CONDUIT, which has some NSF support as well as other federal funding, has published three CAI packages authored at UTC and is currently reviewing two more. In addition, the next issue of the CONDUIT publication, PIPELINE, will contain a panel discussion authored by UTC faculty and staff, on an English CAI project at UTC.

Although not a CONDUIT package, a statistical package for the HP3000 has been developed by a psychologist on the faculty. This interactive package is being marketed by the university with profits being shared by the university, the Psychology Department, Academic Computing, and the author.

Such national recognition is strong motivation for both faculty and administrators becoming involved in instructional computing and chasing those scarce resources of time and money necessary to make it successful.

Academic Computing Services

Throughout the year Academic Computing Services offers free short courses to faculty and staff. These courses cover topics such as authoring languages, as well as EDITOR, EDIT2, SPSS and the BASIC language. The typical course covers a five week period and consists of a one hour meeting per week. In addition, consulting services and portable terminals are made readily available on a reservation basis for faculty and staff.

In the spring of 1979, we undertook a new procedure. At the request of the Nursing Department we instituted an intensive two and one-half day seminar on Instructional Dialogue Facility (IDF), an HP2000 CAI authoring language. Participants were taught how to use the facility to write and edit their course material. Each person produced a mini-package containing four or five multiple choice questions related to the nursing instructor's area of teaching. This project was well received and successful enough to encourage us to utilize the same format again.

UTC has obtained a program from CONDUIT which enables the user to translate packages written in IDF into BASIC. Thus, it is not necessary for our faculty authors to learn a computer language in order to produce a transportable CAI package.

When a package is needed on the HP3000 that is indigenous to a special computer or authoring language, student programmers convert this package into HP3000 BASIC or FORTRAN.

Documentation is accomplished by producing a quarterly newsletter and manuals appropriate for local use. Manuals in production are Pocket Guide to the HP3000, HP2000 Users Guide, UTC User's Guide to the HP3000, Computer Assisted Instruction at UTC, Users Guide to IBM 370 System at UT-Knoxville, and many short pamphlets on miscellaneous subjects such as EDITOR, BASIC, and COBOL.

Inventory of Courses

Every second year, all departments are contacted and asked to return a one-page questionnaire on each course within their department which made use of the computer. These questionnaires ascertain which computer systems were utilized, how many students were involved and whether the role the computer played in relation to the course was required, enhancement or supplementary. Table I, based on data collected in these inventories of courses, shows the increase in computer usage over the period 1976 to 1980.

TABLE I

Inventory of Computer Usage in Courses by Discipline

Department	Number of Courses Utilizing Computing 1976	1978	1980
Art	0	0	2
Biology	0	2	2
Business Administration	8	8	14
Chemistry	7	7	8
Computer Science	10	13	20
Criminal Justice	0	0	3
Economics	1	3	0
Education	2	9	3
Engineering	16	15	19
English	0	2	3
Foreign Language	0	0	10
General Science	1	2	4
Interdisciplinary Studies	0	0	1
Health and Physical Education	1	1	1
Home Economics	0	2	4
Mathematics	11	14	11
Music	4	4	5
Nursing	0	0	1
Physics	5	7	3
Political Science	1	1	2
Psychology	8	9	5
Social Work	0	0	3
Sociology	1	1	3
Total	76	100	127

These inventories enable Academic Computing to keep, in a centralized location, instructions on how to utilize the computer materials, the end to which they are intended, and other pertinent information the instructor deems appropriate. This is abstracted each year into the form required to update the CAI manual for UTC.

Focus for Academic Computing

Through surveys, needs analyses, departmental self-studies, and literature research, Academic Computing focuses perceived needs into a five-year plan that is updated each year and extended for one year. This is presented to the Chancellor and his staff for review, critique, and approval. When finalized, it generally becomes a part of the projected capital budget plan. Hence, the finance officer is able to plan in a consistent, open fashion for legitimate, required computer expansions.

UTC early opted for the complete separation of academic and administrative computing. Each area has its own equipment and each is headed by its own director. Furthermore, the Director of Administrative Computing reports to the Vice Chancellor for Administration and the Director of Academic Computing reports to the Vice Chancellor for Academic Affairs. These separated organizational lines are important in keeping Academic Computing's interest visible, viable, and paramount to the university community. By tying Academic Computing into the instructional side of the university, its success is ensured in meeting student, faculty and curricular needs. By keeping Academic Computing free of Administrative Computing's control, the "water trickling down hill" analogy does not nurish Administrative Computing by diverting needed resources from the academic side. On any campus, organization and staff are major ingredients in the success of a successful instructional/research computing endeavor.

Problem Solving

Problem solving at UTC is defined as the process wherein the student designs, constructs, tests, and evaluates his/her own program, and is distinguished from other computing by calling it programming. The two dominant programming languages at UTC are BASIC and FORTRAN. These account for well over 90% of programming activity. Surprisingly, BASIC is firmly entrenched in Engineering and more problem solving is done among engineers in BASIC than in FORTRAN. Computer Science, on the other hand, favors FORTRAN over BASIC and, except in personal computing and service type courses, BASIC is not used. PASCAL is a major need at this institution since our campus is about to shift from FORTRAN as the basic Computer Science course to PL/I on a remote computer. UTC believes that the imminent HP announcement of PASCAL's availability on the HP3000 will be attractive and cause UTC to review its decision to use PL/I.

COBOL or COBOL II, coupled with IMAGE Data Base package, is the resource used in teaching our data base course. Some use of KSAM may occur in conjunction with this. APL is utilized by an advanced language course but is a disappointment in terms of system's load, requirements for special terminals, and minimal desirability for either Computer Science or Engineering. Although SPL has been taught once to students in Computer Science, it is neither a popular language nor a necessary language. Students requiring assembler language are required to utilize IBM's Assembler at our remote IBM computer.

The growth in all areas of computing, particularly in problem solving, is indicated by the statistics kept on the three systems available at UTC. In 1979 and 1980, usage on the HP2000 system was 42,757 hours and 42,810 hours, respectively. On the HP3000 system, installed in 1978, connect time (in hours) increased from 9,183 hours in 1979 to 18,607 hours in 1980. The number of jobs submitted via RJE to a remote IBM system increased from 35,924 jobs in 1978 and 39,074 jobs in 1979 to 50,997 jobs in 1980; this is an increase of 42% between 1978 and 1980.

The PLOT10 graphics package of Tektronix is widely utilized on the HP 3000 and is accessed by both Tek4010 and 4025 type terminals. PLOT10 is a set of graphics subroutines callable from FORTRAN. Hence, PLOT10 users write programs calling these routines, which makes this package also problem solving. This package has been available from Tektronix at a one time purchase cost of $500, and is vary satisfactory.

The use of terminals to send programs and data to UT at Knoxville, which has twin IBM 370/3031's, has accelerated from nearly nothing to several hundred per day. Although currently this transfer is performed on the HP2000, it will be shifted to the HP3000 this summer using MRJE, SPOOK, and the Robelle editor QEDIT. All departments are de-emphasizing cards and instead focusing on distributed computing via terminals including the HP2621 and others.

MPE III and apparently MPE IV are excellent operating systems that are both capable and friendly. Problem solving fits well within the system. PASCAL, when available, will be of major utility. It is needed now. On the other hand, the HP3000 needs a new ANSI package, FORTRAN 77, on the software side. On the hardware side, HP limitations on the multiplexor throughput, response time degradation in a loaded system, and relatively slow internal speeds, could be alleviated by going to a 32 bit architecture or providing a floating point hardware box to accelerate such computations.

Data Analysis

Data analysis, a euphemism for statistics, is accomplished largely with canned packages. UTC utilizes a wide variety of these on both the HP 2000 and the HP3000. Sources vary from proprietary to contributed library to home grown.

On the HP2000, UTC maintains IDA developed at the Unviversity of Chicago, GUS from the University of Iowa, SPSSHP from DePaul University, and PSY developed at UTC. IDA and PSY are complimentary packages and each has been modified to accommodate easy transfer from each to the other. Although many institutions are familiar with IDA, they may not know about PSY, which provides inferential tests of statistics in a IDA type environment of commands and data. Questions concerning SPSSHP must now be forwarded to SPSS, Inc. as the authors are no longer with DePaul University.

On the HP3000, UTC utilizes the very excellent versions of SPSS and BMDP (BioMedical Data Package) that are available from McMaster University. SPSS is the primary package for research, but the many specific models available in ANOVA make BMDP a valuable tool within instructional computing. The previously mentioned packages, BMDP and SPSS, are both for research and instruction but are batch oriented. A major instruction system at UTC is QSTAT, which was written locally. It is an extension of the previously mentioned PSY and utilizes an IDA approach to commands and data within the inferential area. It also maintains an extensive HELP command and a very flexible BACK command to allow users to undo mistakes in commands.

Interactive Statistical Economic Analysis, ISEA, is available from Dr. John Eaton, London Graduate School of Business. Our economists are funding this package, which provides the specialized statistical tools required in economics. Instead of depending on remote IBM usage of SAS, our institution is making extensive use of ISEA.

The use of these statistical packages in instruction varies from beginning statistics courses to graduate courses requiring analyses. Although most usage is in basic statistics courses, educationists, sociologists, psychologists, economists, and political scientists routinely require computer analyses of data bases as part of course requirements.

Many old, but still good, data sets on Sociology and Political Science are available from CONDUIT. National Opinion Research (NORC) has a major data base of 9120 cases with 640 variables on sociological issues and trends for the years 1972 to 1977 available from CONDUIT. Four major historical demographic data bases are available from the Laboratory of Political Research at the Univeristy of Iowa. The U.S. government and its agencies, such as the Census Bureau, also have many data bases of interest to educators.

Certainly local faculty are capable of building important data bases to support their research. For example, a faculty member is currently examining thousands of pieces of data collected from German cities circa 1600 to determine principal food grains and how the fluctuation in grain prices related to taxes, inflation, and standard of living.

A somewhat unusual data base and package is a nutrition package, written in COBOL, acquired from Clarke College, Dubuque Iowa. The package was modified from a PDP II system to accept KSAM and EDITOR input files. This program analyzes an individual's diet for a one meal, one day, or several days, in terms of basic nutritional requirements. It is an unusual data base in that individuals may select dietary needs from categories such as lactating mothers, small children, or adults with low sodium diets and have their diets analyzed in terms of calories, fats, minerals, and vitamins for hundreds of food items. Such data bases and associated packages are a good means of introducing computing into departments traditionally not computer oriented.

Simulations

For many students, especially those in the Natural and Social Sciences, simulations via the computer offer possibilities otherwise unavailable. Of course, simulations do not have to be performed on the computer, however, such simulations offer the speed, interaction, and graphics necessary for viable, realistic curricular experiences. For UTC the most important simulations have been those from CONDUIT (see Table II).

TABLE II
CONDUIT SIMULATION TYPE PACKAGES

BIOLOGY
- ANAEROBIC GLYCOLYSIS
- COEXIST - Population Growth
- COMPETE - Plant Ecology
- ECOEXX 1 - Mark Recapture Experiment
- ECOEXX 2 - Population Growth Models
- ECOLOGICAL MODELING
- ENZKIN - Enzyme-Catalyzed Reactions
- ENZYME - SEQUEN:
  - Enzyme Substrate Interactions
  - Amino Acid sequences of proteins
- EVOLUT - Evolution Genetics
- LINKOVER - Genetic Mappings
- TRIBBLES - Scientific Method

CHEMISTRY
- FIRSTLW - First Law Thermodynamics
- HABER - Haber Simulation Ammonia
- IDGAME - Qualitative Organic Identification
- KSIMS - Kinetic Experiments
- NEUTRON - Neutron Activation
- NMR - Nuclear Magnetic Resonance
- RKINET - Chemical Reaction Kinetics
- QUANTUM - Quantum Chemistry
- TITRATION - Titration Ionic Equilibria
- XRAY - Xray Crystallography

SOCIOLOGY
- CHANGE AGENT
- DEMO-GRAPHICS
- PROFIS - Introductory Sociology
- USPOP - US Population Studies

PSYCHOLOGY
- COGNITIVE PSYCHOLOGY
- IMPRINTING
- SCHIZOPHRENIA

PHYSICS
- ENERGY and ENVIRONMENT
- GROUP VELOCITY
- ICBM I - Computer Based Mechanics
- INTERP - Wave Superposition
- MECHANICS - Physical Mechanics
- NEWTON - Satellite Orbits
- QUANTUM MECHANICS
- SCATTER - Nuclear Scattering
- USING COMPUTERS IN PHYSICS

MANAGEMENT
- BUSINESS DECISION SIMULATIONS
- BUSINESS - Management Laboratory
- COMPUTER MODELS IN OPERATIONS MANAGEMENT
- COMPUTER MODELS IN OPERATIONS RESEARCH
- EXECUTIVE GAME
- FINANSIM - Financial Management
- MARKETING IN ACTION
- SIMQUEUE - Queueing Theory

OTHERS
- INS2 - Inter-NationSimulation
- CRITICAL INCIDENTS IN EDUCATION
- COMPUTER SIMULATIONS MACROECONOMICS
- MODSIM - Economic Modeling

Several of these from CONDUIT can be used to illustrate the concept of simulation. Critical Incidents in Education is a set of twenty plus scenarios on classroom management problems such as breakage of science equipment, passing of love notes, and obscene phone calls. This is excellent exposure for the student teacher before the real classroom experience. Another simulation is HABER, which involves the synthesis of ammonia commercially. The actual experiment is lengthy (over six weeks) and so expensive it is seldom done in the classrooms. Hence, HABER is realistic for collegiate education since it can be done quickly and at low cost on the computer.

Another simulation is the NDTRAN (available for the HP3000) simulator from Notre Dame. This package allows one to construct the equations governing a model and its constraints, to model the system over a speeded-up time period, and to summarize the results for easier interpretation.

UTC has acquired also simulations from the chemistry departments at Western Washington State and Illinois Institute Technology, as well as from the Hewlett Packard 2000 Contributed Library (HPGSUG). A number of other sources exist, including Lehigh's Economics and Limits to Growth, University of Wisconsin at Madison's FCHART simulation of a solar dwelling, Hunington II simulations from DEC, and numerous packages detailed both in Creative Computing and BYTE magazines.

Although canned programs to find the solution to a specific class of problems may not be simulations in a literal sense, the inherent model is there for use by the student. More importantly, they can be adapted from textbook exercises. As examples, consider the canned programs to calculate a bond return, worth of a stock portfolio, drawing of random phone numbers for interviewing, electrical values for a circuit, or the size of structural members for a building. These valuable programs are available through the HPGSUG contributed tape library. Good programs can be rented or purchased from software and engineering firms. Through shared national facilities such as EDUNET, an individual anywhere can access over twenty major campuses with extensive software libraries of heterogenous and complimentary holdings (EDUNET may be contacted directly or through EDUCOM).

CAI

The largest CAI project at UTC has been one involving basic English skills. In the spring of 1977 the Administration, the English Department and the Office of Academic Computing jointly endorsed a proposal to use a major English CAI package for at least one year. Initial planning called for this package to be assigned to one third of the freshman class, with each of the 600 student participants being allocated from nine to twelve hours of CAI work during the Fall 1977 semester. The package was developed by the Computer Curriculum Corporation (CCC) of Palo Alto, California. This firm no longer markets Basic English for Remedial Students as a software package, but instead includes it with hardware on a turn-key basis. CCC is a valuable source for other CAI software in the areas of reading and mathematics. At the conclusion of the first semester of using the English CAI, a survey established that, of the students using the CAI, 23% felt that access to these programs had materially altered their grade. At the end of four years of operation, a new English course was established with this package as an integral, operating mode of instruction.

Eleven units of descriptive statistics for use in graduate education courses and nearly twenty units of freshmen chemistry were acquired from the University of Akron. Although originally in IBM's CW III, they were converted to HP's CWF and are currently being translated into BASIC.

Two major systems in remedial algebra, drill and practice, have been authored for our Mathematics Department. In addition, a faculty member in Sociology has written one dealing with basic sociological terms.

A music package covering theory from the Music Department at the University of Iowa has had considerable success. With over twenty programs, it can give students extensive drill and practice in this innovative area.

CONDUIT has two notable CAI packages. The first is DIALOGUE for the area of English grammer and the other is SPANCOM for the teaching of Spanish. Another major source for quality CAI is the Association for Development of Computer Based Instructional materials, (ADCIS), which has many member firms and individuals who sell, swap, and distribute CAI.

The process of developing CAI is in a continual state of evolution and development. However, experience has shown us that the usage of CAI is increasing and has a definite educational value. Our academic departments are increasingly finding that extending their computer literacy is a worthwhile investment of their time.

Summary

In this paper we have described the sources for the academic software available to UTC users. This paper thus may provide fledging computer centers or new HP3000 users with a list of places which can be contacted for academic software. HP users entering the area of instructional computing frequently call UTC to determine where applicable software can be obtained or purchased. Although UTC has a small department of Academic Computing Services, it has been able to provide the faculty with a full range of CAI, simulations problem solving, and data analysis, in part because of the wide range of nationally obtainable software.

UTC's Office of Academic Computing is a major strength for the support and effectiveness of computing at UTC. UTC particularly espouses the enrichment of the curriculum with computer related experiences. To that end, a continual and diligent search is made to secure quality educational software that is relevant to UTC's academic programs.

A GENERALIZED NAME INDEXING METHOD FOR IMAGE DATABASES

Robert B. Garvey

Robert L. Womack IV

Witan Inc.
Kansas City, Missouri

A Generalized Name Indexing Method for Image Databases

Robert B. Garvey
Witan, Inc.
Robert L. Womack, IV
Third Judicial District Court
of Kansas

March 13, 1981

1. Name Searching -- The Problem Part 1

Traditional approaches to searching a list of names usually involve a KSAMish approach to the data file: i.e. positioning the program (or a terminal user) at the beginning of a group of names spelled in like manner, and then performing additional subsetting or processing of the group of names. KSAM (or ISAM) is typically used for such applications because of their ability to position the record pointer using only a partial key. IMAGE, on the other hand, is seldom used for such applications since inquiries must be made against a complete search item. For applications in which inquiry against an existing "people" base is required, the complete entry of a name may be at best redundant and at worst extremely subject to error since variations in spelling of a name at data entry time from that of the master name on file are difficult to handle. Criteria for a first cut name search algorithm should then include at least the following:

- .The ability to search on a partial or incomplete name
- .The ability to operate upon or display to a terminal user a group of similar names.

Name Searching -- Problem Part 2

An additional requirement for many name search applications is the ability to search for names by phonetic equivalent, that is names should be selected for analysis based upon how they sound rather than how they are spelled. Applications which must employ field originated forms are typically in need of such capability. A second-order name search technique would, therefore, require the additional capability of inquiry by the phonetic value of a name rather than (or perhaps, in addition to) the exact spelling or partial spelling of a name. Lastly, in order to increase the "crash-worthiness" and ease of maintenance of a name search system, the person data base should be a part of an organization's IMAGE data base. The ideal name search technique then should have the following attributes:

- .The ability to search on a partial or incomplete name
- .The ability to operate upon or display to a terminal user a group of similar names.
- .The ability to search by a phonetic key
- .Suitability for implementation using IMAGE

The remainder of this paper will etch out a partial data base, compare two phonetic encryption methods, and suggest an approach to a name inquiry system which will satisfy the above requirements.

2. Phonetic Name Searching and IMAGE

Phonetic name encryption gives the system designer a tool which enables the design of name inquiry systems that meet the four criteria set out in part one of this paper. In this section a partial data base schema to support name indexing will be outlined and standards for evaluating name encryption algorithms will be suggested and applied. The development of synonyms is at the heart of any phonetic name encryption module. It has been suggested that two standards should be used to evaluate phonetic name encryption algorithms: reliability and selectivity (1). In discussing these concepts in a study prepared for the New York State Identification and Intelligence System, Robert Taft defines reliability as , "the probability that the technique will retrieve the correct suspect if, in fact, the suspect is in the file." (2) Continuing the analysis, Taft states selectivity is,

> "the average percentage of [a] file that the technique will accept on an average search request. If, for example, a method has a selectivity factor of one percent, then the method will, on the average, accept one percent of the file and reject ninety nine percent ... the selectivity factor is an accurate measure of the amount of work the system will have to perform." (3)

To these criteria this writer will add yet a third,: The algorithm should be fast in execution time and require a minimal code and data segment on the HP 3000. A simple IMAGE data structure implied by the mechanics of phonetic name transformation techniques is shown below in Figure 1.

Figure 1

```
..............
. Phonetic .
 .  Code  .                              ..................
  .(Auto).-------------------------->.   Name Detail   .
   .    .   Primary Key: Phonetic    .               .
    .  .                 Code          .             .
     ..                                 .           .
                                         ..........
```

Since the object of the encryption algorithm is to group similar sounding names into the same group (i.e. assign them the same search item value), a straight-forward representation of this in a data base is a detail set with a primary key of the phonetic name code indexed by an automatic master. The detail set allows for multiple records with the same search item value; making the phonetic key the primary key of the detail set will speed access by insuring that records with the same phonetic key will be stored contiguously on disk after a DBUNLOAD/DBLOAD cycle. This same set may be indexed by other master sets and should contain at a minimum the name and name associated information for the entity being indexed.

SOUNDEX and NYSIIS Name Encryption Techniques

In this section two name encryption techniques will be specified and evaluated using the standards outlined in Section 2. The SOUNDEX algorithm is an especially common name grouping technique in use today. Its implementation is extremely straightforward on the HP 3000 (see appendix A). The algorithm upon which the SPL procedure displayed in Appendix A is based is given below in Table 1.

Table 1 - SOUNDEX Variant 1 Alogrithm (4)

Rule	Example
	Original name ASHCROFT
1. Remove the letters "W" and "H" from the name	ASCROFT
2. Code all letters using the following table and using "0" for vowels:	0226013

B,F,P,V	= 1
C,G,J,K,Q,S,X,Z	= 2
D,T	= 3
L	= 4
M,N	= 5
R	= 6

3. Make all multiple digits single	026013
4. Remove all zeros after the first position of the value	02613
5. Add sufficient zeros on the right hand side to make six digits	026130
6. Replace the first digit with the first character of the name	A26130

Taft evaluates the reliability of this technique at 95.99 percent. Its selectivity factor is .213 percent, i.e., this method will on the average retrieve .213 percent of a file as matches to a name inquiry (5). The procedure shown in Appendix A takes 69 CPU seconds to encode 5,524 names on an HP 3000 Series III. The NYSIIS name encryption technique represents an effort to develop a phonetic code that is both more selective than SOUNDEX procedures and more effective at handling the "orthographic errors which occur in the recording of 'Spanish' and other South European names" (6). The NYSIIS algorithm is outlined in Table 3, below:

Table 3 -- The NYSIIS Algorithm (7)

Rule

1. If the first letters of the name are
   "MAC" then change these letters to "MCC"
   "KN" then change these letters to "NN"
   "K" then change these letters to "C"

```
      "PH"  then change these letters to "FF"
      "PF" then change these letters to "FF"
      "SCH" then change these letters to "SSS"
2.   If the last letters of the name are
      "EE" then change these letters to "Y "
      "IE" then change these letters to "Y "
      "DT","RT","RD","NT","ND" then
         Change these letters to "D "
3.   The first character of the NYSIIS code is the
     first character of the name
4.   In the following rules, a scan is performed on
     the characters of the name.  This is described
     in terms of a program loop.  A pointer is used
     to point to the current position under
     consideration in the name.
     Step 4 is to set the pointer to the second
     character of the name.
5.   Considering the position of the pointer, only
     one of the following statements can be
     executed:
     If blank then go to rule 7
     If the current position is a vowel (AEIOU)
        then
        If equal to "EV" then change to "AF"
        otherwise change current position to "A"
     If the current position is the letter
        "Q" then change the letter to "G"
        "Z" then change the letter to "S"
        "M" then change the letter to "N"
     If the current position is the letter "K" then
        If the next letter is "N" then
           replace the current position by "N"
        otherwise
           replace the current position by "C"
     If the current position points to the letter
        string "SCH" then
         replace the string with "SSS"
     otherwise If the current position points to
        the letter string "PH" then
        replace the string with "FF"
     If the current position is the letter "H" and
        either the proceding or following letter is
        not a vowel (AEIOU) then replace the current
        position with the preceding letter
     If the current position is the letter "W" and
        the preceding letter is a vowel then
           replace the current position with the
              preceding position
     If none of these rules applies, then retain
        the current position letter value
6.  If the current position letter is equal to the
    last letter placed in the code then
       set the pointer to point to the next letter
       go to rule 5
```

```
7.   If the last character of the NYSIIS code is
        the letter "S" then
        remove it
8.   If the last two characters of the NYSIIS code
        are the letters "AY" then
        replace them with the single letter "Y"
9.   If the last character of the NYSIIS code is the
        letter "A" then
        remove this letter
```

Taft evaluates the reliability of this technique at 98.72%. The selectivity factor of the NYSIIS routine is .164% (7). A program using an SPL implementation of the NYSIIS algorithm developed by the Illinois Law Enforcement Commission (8) took 66 CPU seconds to encrypt 5424 names.

Detailed Comparison

The SOUNDEX and NYSIIS algorithms were compared using the following procedure:

1. A file of 5424 names was encrypted using each technique
   A sequential MPE file containing the phonetic code for each name was generated.
2. Timings were derived for both processes
3. The files of phonetic codes were sorted in ascending order
4. Duplicate codes in each file were eliminated. Each code was tagged with the number synonyms for that code that had been generated.
5. Summary statistics were derived using SPSS (9).

The results of the SPSS runs and other summary measures are given in Table 4, below:

Table 4 -- SOUNDEX vs. NYSIIS

Routine Name	CPU Time	Segment Size	# Codes Generated	Avg. No. Synonyms/ Code	STD DEV	Maximum No. Synonyms/ Code
NYSIIS	66	1021	1873	2.895	4.644	77
SOUNDEX	69	434	1478	3.669	5.992	84

The comparison data displayed in Table 4 reveals that the performance characteristics of the NYSIIS routine are on the average about twenty percent more efficient, both in terms of the number of entries generated and the average number of synonyms per entry, than SOUNDEX. It should be noted, however, that applications using one algorithm as opposed to the other will not necessarily show equivalent differences in performance. This is especially true for applications using IMAGE databases which are perodically DBUNLOAD/DBLOADed. Empirical data at the Third District Court indicate that IMAGE will block moderate sized name detail sets (for example, doubly keyed records with 50 byte name field, six byte soundex primary key, ten byte secondary key and eight bytes of miscellaneous data) at from ten to eleven blocks per physical record. Thus one disc I/O will all matching phonetic entries all but five to seven percent of the time, depending upon the algorithm selected. If DBUNLOAD/DBLOADs are not done on a periodic basis, then performance differentials should approach that found in the sample data used in this paper.

Summary

Applications using either of the phonetic encryptions algorithms discussed in this paper will be able to search an appropriatly keyed detail set by phonetic key. Both methods generate fixed length keys. Finally, by doing phonetic searches on the last name and exact or weighted matches on the remaining characters in a name specified at data entry time, an application program may both search on an incomplete name and develop a subset of matching names for further analysis.

E N D N O T E S

(1) Robert L. Taft, "Name Search Techniques," Albany: New York State Identification and Intelligence System, undated, pp 37-38. This paper is an excellent analysis of the merits and demerits of several different name encryption systems.

(2) *Ibid.*, p. 37. (3) *Ibid.*, p. 38.

(4) *Ibid.*, p. 58. (5) *Ibid.*

(6) *Ibid.*, p. 86. (7) *Ibid.*, pp. 88-90.

(8) NYSIIS was implemented on the HP 3000 by J. David Coldren and his staff at the Illinois Law Enforcement Commission in May, 1977.

(9) See Norman H. Nie, *et. al.*, *Statistical Package for the Social Sciences, 2nd Edition.*, New York: Mc Graw Hill and Company, pp 194-202 for an explanation of the FREQUENCIES procedure used to analyze the data referenced in this paper.

Appendix A

```
$CONTROL SUBPROGRAM,LIST,SEGMENT=RUSSELLSOUNDEX
BEGIN
PROCEDURE SOUNDEX(NAMESTRING,SOUNDEX,ERRCODE);
LOGICAL ARRAY NAMESTRING,SOUNDEX;
INTEGER ERRCODE;
BEGIN
   BYTE ARRAY NAMESTRING'B(*)=NAMESTRING;
   BYTE ARRAY SOUNDEX'B(*)=SOUNDEX;
   BYTE ARRAY NAMESTRING'S(0:50),STR(0:50);
   BYTE ARRAY RUSSELL'SOUNDEX(0:255);
   INTEGER I,J;
   INTEGER STRING'LENGTH;
   BYTE FIRST'LETTER;
   INTRINSIC CTRANSLATE;

   MOVE RUSSELL'SOUNDEX:=65("0"),2;
      MOVE *:="0123012002245501262301020200000001230120022455O",2;
      MOVE *:="12623010202",2;
      MOVE *:=133("0");
   MOVE NAMESTRING'S:=NAMESTRING'B,(50); << COPY NAME TO WORK AREA >>
   NAMESTRING'S(50):=" ";      << SET TERMINATOR BYTE >>
   SCAN NAMESTRING'S UNTIL ", ",1;       << SCAN FOR END OF LAST NAME >>
   STRING'LENGTH:=TOS-@NAMESTRING'S+1;  << SET STRINGLENGTH >>
   MOVE STR:=NAMESTRING'S,(STRING'LENGTH); << MOVE LASTNAME INTO STR >>
   FIRST'LETTER:=STR;               << SAVE FIRST LETTER OF LAST NAME >>
   IF INTEGER(FIRST'LETTER)>90 THEN
      FIRST'LETTER:=BYTE(INTEGER(FIRST'LETTER)-32);
   IF FIRST'LETTER <> ALPHA THEN
      BEGIN
      ERRCODE:=1;
      RETURN;
      END;
   CTRANSLATE(0,STR,STR,STRING'LENGTH,RUSSELL'SOUNDEX);
   IF  <  THEN BEGIN                      << TEST FOR SUCCESSFUL >>
      ERRCODE:=1;                         << TRANSLATION AND SET >>
      RETURN;                             << ERRCODE ACCORDINGLY >>
      END;
   ERRCODE:=0;                            << INITIALIZE ERRCODE >>
   FOR I:=0 STEP 1 UNTIL STRING'LENGTH-2 DO
      BEGIN
      J:=I;
      WHILE STR(I)=STR(J+1) AND J<=STRING'LENGTH-1 DO
         BEGIN
         STR(J+1):="0";
         J:=J+1;
         END;
      END;
   J:=1;
   FOR I:=1 STEP 1 UNTIL STRING'LENGTH-1 DO  << SQUEEZE OUT ZEROS >>
      IF STR(I)<>"0" AND J<6 THEN
         BEGIN
         SOUNDEX'B(J):=STR(I);
         J:=J+1;
         END;
   IF J<6 THEN                            << POSTFIX ZEROS TO MAKE >>
                                          << SIX DIGIT CODE       >>
      FOR I:=J STEP 1 UNTIL 5 DO
         SOUNDEX'B(I):="0";
   SOUNDEX'B:=FIRST'LETTER;               << PREFIX SOUNDEX CODE WITH >>
```

```
                                                          << FIRST LETTER OF LAST      >>
                                                          << NAME                      >>
        END;
    END.
```

USING SERIAL AND DEMOUNTABLE DISK VOLUMES

by:
Jim Brand
U.S. Office of Personnel Management

An in-depth discussion of how to create, initialize, use, and change the demountable disk volume feature of the MPE operating system. The paper touches on particular environments where demountable volumes might be applicable, and both the advantages and disadvantages of using demountable disk. It focuses on a detailed technical discussion of creating the account structure for, initializing, and maintaining demountable disk volumes. In particular, this section deals with the "things HP never told you" about demountable disk software, how it works, directory structure, and some of the problems you would be likely to encounter in creating or purging accounts, groups, and files from demountable volumes.

Backup and file storage on serial disk forms the central concern of the last section of the paper. This section suggests some uses, advantages, and flexibility which can be gained by using serial disk backup. It is concluded that demountable and serial disk can be very effective in helping to achieve better utilization of available disk spindles. There are several problems which should be considered, however, before its implementation.

MEASURING TRANSACTION RESPONSE TIMES

Chuck Storla
Hewlett Packard
Rolling Meadows, IL

ABSTRACT: One of the major problems confronting system managers and programmers trying to improve the performance of an application is their lack of knowledge of exactly where the application is spending its time. This paper discusses a method of "instrumenting" the user's application so that precise timings are available to the development staff. One of the major benefits of this method is that it may not require any modification or even recompilation of existing code.

I. Introduction.

An on-line application program typically contains subsystem calls and code which performs the following functions: terminal I/O, file handling and logic/computation. All of the specific functions of a user's program such as data entry, inquiry, and error handling, will fall into these three categories. Many current users of the HP3000 have existing applications which perform these respective functions with V/3000, IMAGE/3000 and a high-level language such as COBOL. Although the method discussed in this paper has some general applicability, it will be specifically aimed at the users in this environment.

Should a user experience performance problems with a particular application, the programming staff needs to determine where the bottlenecks are. This can be accomplished through intelligent guessing, through consultation with a Hewlett Packard Performance Specialist or by adding timing code to the program(s) under suspicion. We will add this extra code to the program to determine which phase is taking an unusually large amount of time to execute. If we can narrow the scope of our investigation to that portion of the application which takes the most time to perform or at least takes longer than we feel it should, then we are much closer to knowing how to solve our performance problem. We might find, for example, that for a transaction which takes 20 seconds, the program uses five seconds to retrieve all of the records we need, 2 seconds to display the data, but thirteen seconds to format it. We might then choose to spend our time improving the performance of this formatting code, rather than on a redesign of the database.

This is not meant to imply that the portion of a program which requires the most time to execute is necessarily the least efficient or even the cause of the problem. However, any area in which a program spends a great deal of time will certainly be of interest to anyone wishing to optimize its execution.

II. DETERMINING WHAT TO TIME.

If we examine a typical on-line application we might find that the program had the following structure:

```
BEGIN
Initialize data, Open files;                    << 1 >>
Display formatted screen;                       << 2 >>
WHILE NOT DONE DO
   BEGIN
   Read screen;  << for input data >>           << 3 >>
   IF end THEN
      done:=TRUE
   ELSE
      BEGIN
      Retrieve data from files;                 << 4 >>
      Format for output to screen;              << 5 >>
      Display data;                             << 6 >>
      END;
   END;
Clean-up, Close files;                          << 7 >>
END.
```

This program is a simplified version of many now in use on HP3000's. This description does not include any error handling and is limited to a single screen, but will serve for the purposes of this discussion. Our interest in the performance of this program would probably center on steps << 3 >> through << 6 >>. The additional steps (1,2 and 7) would only be of interest if this program was run many times and for brief periods. Unless this is true, the start-up and shut-down functions are probably not as crucial to us as the intermediate processing. However, since file opens can be very high overhead operations, we might need to consider these steps in other programs.

To the terminal operator, the function of this program would appear as follows:

```
  (a)   :run program
                  << wait for 1 & 2 to complete >>
  (b)    enter data, hit ENTER
                  << wait for 3 through 6 to complete >>
  (c)    look at data and either EXIT or return to step (b)
                  << if EXIT wait for 3 & 7 to complete >>
```

Obviously, from the terminal user's perspective, the important timing consideration here is the time he or she must sit and wait between hitting ENTER and receiving a full response. Time from the operator's perspective will from now on be referred to as "wall time" and will be distinquished from the amount of time the computer system spends executing instructions on the behalf of this user, which is known as "cpu time". We are interested in cpu time only as it pertains to improving the wall time responses we can provide our terminal user.

It is very important to note that there are many instances in which our program can require significant amounts of wall time for a step that requires little cpu time. This will be especially true when our program must compete for a resource such as a file or data-base lock or must wait for the operating system to grant a buffer, sufficient room in memory or a disc I/O. During the time we are waiting for a particular resource, our process is said to be impeded. While in this state, no cpu time will be used by our process, but many seconds or even minutes of wall time may go by. This being the case, we are interested in timing both wall and cpu times.

Returning to our hypothetical program, we see that we might like to add our timing code just before and just after each of the major steps, especially << 3 >> through << 6 >>. Given the nature of these steps we might assume that step three, reading the screen, will not take much cpu time but due to the operator's typing speed and delays caused by his or her decision making, our measured wall time will be significant. The wall time attributed to thinking about what to enter and actually typing the data is usually referred to as "think time". The wall time from the point at which ENTER is hit until the screen is ready to accept new input will be referred to as "transaction time".

III. HOW TO TIME.

There are several intrinsics available to programmers on the HP3000 to determine both cpu time and wall time. The first of these intrinsics are PROCTIME and TIMER, respectively.* The PROCTIME intrinsic will return the number of milliseconds that have been "charged" to a process for cpu usage. The TIMER intrinsic returns the number of milliseconds from the last time the system was started, or since the last automatic reset to zero. This reset occurs on twenty-four day intervals at midnight.

It should be noted that these intrinsics each have a resolution of one millisecond and therefore should not be used to make single measurements in that range. At the application level in which we are currently interested, most measurements will result in times greater than a tenth of a second and often into seconds. Should an investigation require timings in the millisecond range or below, special methods must be used.

IV. A METHOD OF IMBEDDING TIMING CALLS.

Part of our method then will be to insert into our application program the calls to PROCTIME and TIMER so that we can arrive at elapsed times for cpu and wall time. In many cases however, it may be difficult or undesirable to modify an existing program to add the necessary timing calls. If the calls were imbedded within the source code itself we would as well need a means of disabling the timing during normal operation. In looking at our example program and considering it to be written so that the screen handling is done through calls to V/3000 intrinsics and the file handling is done through calls to IMAGE/3000 intrinsics, we might modify our pseudo-code as follows:

*These intrinsics are documented in the "MPE Intrinsics Reference Manual" (30000-90010).

```
BEGIN
Initialize data, Open files;                    << 1 >>
VSHOWFORM; << Display screen >>                 << 2 >>
WHILE NOT DONE DO
   BEGIN
   VREADFIELDS; << Read screen >>               << 3 >>
   VFIELDEDITS;
   IF end THEN
      done:=TRUE
   ELSE
      BEGIN
      VGETBUFFER;
      << Retrieve data from files >>            << 4 >>
      DBFIND's & DBGET's;
      Format for output to screen;              << 5 >>
      VPUTBUFFER;
      VSHOWFORM; << Display data >>             << 6 >>
      END;
   END;
Clean-up, Close files;                          << 7 >>
END.
```

We can now see that several of the major steps which we wish to time become one or more intrinsic calls. If we could time each of the indivdual intrinsic calls and as well capture the time between several of the calls we could get the bulk of the data we need.

The method used by MPE to link a program to external procedures provides the mechanism we need. At the time a program is executed with the :RUN command** the MPE LOADER will try to resolve any unresolved externals by searching one or more Segmented Libraries. The default is to search only the SL.PUB.SYS file, while at most three SLs will be searched. Assuming a program file resides in a group other than PUB of an account other than SYS, the command ":RUN program;LIB=G" will cause the search to proceed as follows:

**or is allocated with the :ALLOCATE command or has a process created with the CREATE intrinsic.

SEGMENTED LIBRARY SEARCH ORDER

1) SL.group.account is searched

any externals not resolved or any "new" externals will cause a search of

2) SL.PUB.account

any externals not resolved or any "new" externals will cause a search of

3) SL.PUB.SYS

any unresolved externals at this point will cause the load to abort

The reference to any "new" externals after 1) and 2) refers to the following situation: Assume a program which has two unresolved externals, procedures "A" and "B". If we run the program and specify LIB=G, then SL.group.account will be searched. Assuming that this SL contains procedure "A", it will be resolved. When procedure "A", in turn calls "X" which is not contained in SL.group.account, then "X" is a new external and it must be resolved at the account or system SL level.

If the program we wish to time exists in a non-PUB group of a non-SYS account then it might search for the V/3000 and IMAGE/3000 intrinsics in the first two groups and, not finding them, will finally link to those routines in SL.PUB.SYS. If we wish to "intercept" the calls to a particular intrinsic, we can accomplish this at the group SL level. To intercept each call to DBGET, for example, to determine how much time our data-base reads were taking we could write our own version of DBGET and place it in SL.group.account. Now all calls to DBGET would execute our own DBGET routine. This solves the problem of capturing the calls to DBGET, but to allow the program to operate correctly we need to somehow get from our routine to the "real" DBGET in SL.PUB.SYS. If our local DBGET makes a call to DBGET it will simply be recursive on itself.

The solution involves calling another user written routine which we will call DBGET'. This routine will simply accept all of the standard DBGET parameters and use them to call DBGET. It will be placed into the account SL so that its call to DBGET will be resolved to the "real" DBGET in the system SL. Thus, when our program calls DBGET, it executes our group SL version of DBGET which logs some timing information and then calls DBGET'. This routine in SL.PUB.account in turn, calls DBGET in SL.PUB.SYS which executes the actual function.

This solution solves several of the problems we discussed previously. SInce it requires no modification of user or system code, it is easy to implement and can be used in some situations where original source code is not available for modification and recompilation. It also allows us to enable and disable the timing code easily. To enable the timing simply run the program(s) with LIB=G. The default case is disabled (LIB=S).

V. CONSIDERATIONS.

The following must be considered when evaluating this technique:

- Programs must reside in non-PUB group of non-SYS account.

- If the programs currently use the group SL, then two versions must be created. One including timing calls, one without.

- Performance may be minimally affected by the additional procedure calls (two per call to a timed intrinsic) and the additional timing and logging code.

- If timing information is being written to a file, provisions must be made to allow it to be shared between multiple users.

- Timing of some portions of the program must be inferred, since only certain procedure calls are captured.

- No indication is given as to the cause of poor performance in the case of impedes.

The method discussed here provides a tool for application programmers to use in learning more about the behavior of their systems. The use of dummy intrinsics in group and account SLs has been used successfully by various Hewlett Packard personnel as a method of timing the execution of some routines and as a method of logging all terminal interaction to a disc file for later manipulation. Within the limitations of the data gathered and depending upon the analysis of that data it provides a mechanism that is both easy to implement and is specific to the area in which data is desired.

DATA BASE NORMALIZATION

By:

Gurdo VanBempt
Jaak VanDamme

Syvas, Inc.

Paper to be presented
at Conference

THIS PAPER ON THE HEWLETT PACKARD 2680A LASER PRINTING SYSTEM
WAS PREPARED FOR THE HP 3000 USERS GROUP MEETING
APRIL 29 IN ORLANDO FLORIDA

by Jim Langley
HP 2680A R&D Project Manager

Abstract

This paper describes the HP 2680A Laser Printing System from the perspective of the HP 3000 programmer. The printer hardware is first described, then its features are explained. Concepts of downloadable character sets, electronic forms and logical pages are discussed. The implementation and use of these printer features via the system software is also covered. The impact of the laser printer in a distributed computing environment is briefly explored.

Overview

The HP 2680A is Hewlett Packards first page printer. It is based on an electrophotographic process which was licensed from Canon, a Japanese firm. The printer was designed and is manufactured by the Boise division in Boise Idaho.

Several key objectives were established at the start of the program. Reliability, flexibility, features matched to 3000 user needs, simple powerfail and paper jam recovery, very low CPU overhead, and the ability to access the printer and its features from existing programs without modification became the primary objectives of the development effort.

From the beginning the printer was designed as an extension of MPE, not an added on peripheral. This tight coupling yielded a fully integrated printing system that is fully supported by the MPE file system and spooler. In addition a powerful subsystem exists which allows complete character set and forms design. Flexible page formatting and a full complement of intrinsics provide access to all printer features.

By fully integrating the printer into the 3000 simple and reliable power fail and paper jam recovery is realized. All these benefits were achieved while the CPU overhead to drive the printer was reduced an order of magnitude from that required to drive conventional printers at comparable data rates.

Hardware

The 2680A is approximately 5.5 feet long, 2.5 feet deep and 4 feet high. It weights about 875 pounds. Power requirements are 4500 watts when printing. Throughput is 45 8.5 by 11 inch pages per minute. The equivalent lines per minute speed is 2900 lpm ranging up to 12000 lpm in reduction mode.

The paper path is short and readily accessible to the operator. It features a powered paper stacker. The fusing system is radiant, eliminating any pressure or high temperature rollers. Nothing contacts the upper side of the paper once the image is transferred from the drum to the paper, contributing to excellent reliability. The web is pulled by a programmable torque motor on the output tractors, and paper motion is gated by stepper motor driven input tractors. A solenoid powered retraction mechanism pulls the paper away from the drum when the seam on the drum comes around. The input paper platform acts as a splice table; a vacuum is used to hold the paper onto the table when splicing a new box of paper onto the end of the previous box. The paper path can accomodate various widths up to 12.5 inches and lengths up to 17 inches. A width sensor on the input tractors allows the printer to energize only the correct width in the preheater section of the fuser. Paper which is heated produces odors, which are trapped and oxidized in replacable filter cartridges.

The image forming process is electrophotographic. The heart of the system is a photoconductive drum about 19 inches in circumference and 14 inches long. The drum is coated with cadmium sulfide and wrapped in mylar for protection. The drum is uniformly erased and then charged to several hundred volts at the first station. The laser is then scanned across the drum perpendicular to the direction of rotation. The beam is modulated to give 180 dots per inch resolution. There are 2048 dots in one scan line, giving a printable area 11.38 inches across. The drum rotation allows the sweeping laser beam to cover an area 17 inches long. The circular dot is about .008 inches in diameter on a grid .0055 inches square. When the laser beam hits the drum the voltage is depleted. Next the drum rotates past a cloud of fine flour-like black plastic. The plastic toner is attracted to areas of no voltage by electrostatic forces. The pattern traced by the scanning laser beam is now visable as a sharp black image on the drum. Finally the paper and the drum are brought together for about 1 inch of tangential contact. The paper is correctly charged to firmly attract the toner off the drum. The small amount of residual toner not deposited on the paper is then scraped off of the drum by a urethane wiper blade and collected by a vacuum system. As the drum turns these processes are executed simultaneously at different stations around the drum.

In order to achieve high print quality over a wide range of ambient conditions the HP 2680A has two closed loop control systems. The electrostatic control system monitors the potentials on the drum just after the laser station. The voltage is measured both where the laser exposed the drum and where it did not. The microprocessor taking the measurements then controls several programmable power supplys to maintain the correct drum potentials. The readings and ajustments are made once per drum rotation. The electrostatic closed loop compensates for variations between replacement drums, drum degradation over time, humidity, temperature and altitude variations, and toner mixture fatigue.

A second closed loop system monitors the developed image on the drum to control print density on the page. By varying the amount of toner in the developer assembly which brushes the toner mixture across the drum the amount of toner on the drum and hence the final print darkness on the paper can be controlled.

The mechanical features of the printer were designed to be simple and reliable, and the operator functions are easy to learn and execute. A vacuum system in the printer contributes to cleanliness. It is used to recover toner wiped off the drum. It also is used for the splice table and to maintain good contact between the preheater pad in the fuser and the paper. The operator loads a fresh kilogram of toner into the machine about once every eight hours of printing. Unused toner is collected with the vacuum system, trapped centrifigally and deposited in a disposable bottle which is replaced every couple of days. A new box of paper is loaded every hour. The new box can be conveniently spliced onto the end of the previous box or the new box can be easily loaded with the THREAD button.

There are two microprocessors in the HP 2680A. One is a 16 bit HP proprietary SOS device which controls all machine functions such as the operator keyboard and alphanumeric display, the paper path, the closed loop systems and internal diagnostics. The second processor is a high speed bi-polar bit slice processor which communicates with the 3000 and performs all processing on the data stream and ultimately modulating the laser beam to form the correct images at the proper place on the drum. This processor uses 256k bytes of RAM, with a second 256K available as an option. Approximately 40K bytes of this memory is used for tables and buffers, the remaining memory is partioned dynamically during each job for character sets, forms, and page buffering.

Extensive internal diagnostics constantly monitor the state of the machine, alerting the operator if a service engineer should be called. When arriving on site the service engineer can use additional internally contained diagnostics to troubleshoot any problems. A very complete self test program is available which prints many important parameters on the machine itself. Data such as serial number, drum rotations since last PM, firmware datecodes, and all operating values are labeled and printed. The printer contains a limited amount of nonvolatile memory.

Programming Features

Page printers, even with their inherent benefits of high thruput, low noise and exceptional print quality are rarely viable as simple print and space devices because of their higher cost. However the HP 2680A is a cost effective replacement for many line printers. This is because of the flexibility and features of the printer. Electronic forms allows the inventory of costly specialty forms to be eliminated. Long lead times and form modifiation costs are reduced to a few hours on a terminal. Definable character sets allows the printer to be used in a wide variety of industries and applications where conventional printers are useless. In addition the print quality and crispness in conjunction with the 8.5 by 11 inch paper size means HP2680A output never needs to be copied or reduced before general distribution.

The HP2680A implements a concept called logical pages. A physical page is a sheet of paper bounded by perforations. A physical page can be divided into up to 32 rectangular areas called logical pages. Logical pages can overlap. A programmatic command to page eject moves the print to the next logical page. If all logical pages have been used, the printer goes to the first logical page on the next sheet of paper. Each logical page has several attributes such as an associated vertical format control (VFC) table, a default line spacing, and one of four orientations. In addition each logical page can have up to two forms associated with it. When the logical page is printed the forms are automatically overlaid by the printer. Several logical pages can share the same form and VFC, the printer will automatically relocate it to the correct origin for each logical page. Logical pages are a powerful concept which particularly supports existing programs. By defining the logical page format an existing job can have its output reduced two to 1 or four to 1 or rotated without even recompiling the job. Additionally a job which currently uses preprinted forms can be switched to run on the laser printer without modification. The existing form is converted to electronic format and then the corrsponding logical page is defined to use the form. The job is then printed on the laser printer and the data is merged with the form and printed.

The electronic forms capability is designed for maximum flexibility. Each form can contain horizontal and vertical lines of varying thicknesses, text written with any number of fonts in any of the four basic directions, plus areas or boxes of variable shading. Form elements can be positioned anywhere and are not restricted to certain character positions on the page as a "draw set" is. The printer can support 32 different forms simultaneously. Each logical page can use up to 2 forms as long as the total does not exceed 30. Additionally each physical page can be overlaid with up to 2 forms. Enough memory and processing power exists to create a form which is a dot per bit image of an 8.5 by 11 inch sheet of paper. Forms are easily created for the printer using an interactive program called IDSFORM.

The HP2680A printer accepts user defined character sets. Each character set contains from 1 to 128 characters. Each character has an associated cell of a specified size which contains any dot per bit representation desired. The spacing between characters and between lines can be set to any value. A character set can print in any of the four directions. Proportional character sets are supported. In this case each character has a parameter describing how far to move over after printing each character. The printer also allows the cells to be printed in any relationship to the current "pen" position. This allows centered symbols, or common base lines so different character sets can be mixed properly on a single line. When using more than one character set a primary and secondary set are defined and then selected with either shift in, shift out control codes or by setting the eigth bit of the ASCII code. HP supplies a large number of character sets of various fonts and sizes. In addition character sets and logos can be created interactively by terminal users via IDSCHAR.

Thirty two user definable VFC's are supported by the printer simultaneously. They are easily created with the IFS2680 program.

One additional feature was implemented to allow easy emulation of multi-part forms. When activated each physical page of data will be repeated up to eight times by the printer. As each copy of the page is printed, the printer will automatically overlay any two forms on the page. In this manner the same data can be repeated up to eight times, but each copy can be individual addressed to shipping, purchasing, order processing, etc.

These basic data structures provide a wide range of user features. When combined with the ability to place cells anywhere on the page and overlap at will plus the processing power to handle over 20,000 characters on an 8.5 by 11 inch sheet a truly unique printer results. The maximum number of cells on any raster scan is 255. As the cells get larger, fewer can be printed simultaneously. Character set switching, forms overlay and other features all occur at speed.

The printer's memory is allocated by a memory manager on a job by job basis. Approximately 40K bytes are used by the printer, the remaining memory is allocated to character sets, forms, VFC's and page buffering. As much memory as required is allocated to the user's character sets, forms and VFC's. All remaining memory is used to buffer pages in an intermediate linked list structure. More page buffering insures that pages are printed at speed. Insufficient page buffering causes a lower thruput rate. The programmer can add or delete character sets, forms and VFC's during the job.

Environment Files

All character sets, forms, VFC's and the logical page table and the multicopy forms table are placed in an environment file by a terminal user running IFS2680. This file is then sent to the printer at the start of a job automatically. This allows the output of a job to change appearance by changing the environment file or portions of the environment file. For example if the character set in an environment file is changed from elite to pica the next job to use the file will have output printed in pica. By simply changing the logical page description and substituting a smaller character set a job can be made to print in a 2 to 1 or 4 to 1 reduction mode. HP supplies several standard environment files to cover portrait mode pica and elite, landscape 132 column printer emulation, two to one and four to one reduction. The user can easily create additional environment files.

For new application programs the full power of the printer is available through HP supplied intrinsics. The intrinsics allow features such as writing a string to a named field on an electronic form. The form can be redesigned and rearranged without modification of any programs using the form. The data will automatically be placed in the correct field wherever it is on the page. Intrinsics also allow the pen to be moved, new primary and secondary character sets to be selected, any logical page to be turned on or off and other similar features.

System Software

Extensive system application software allows creation of character sets, forms, VFC's as well as the definition of logical pages and multicopy forms tables.

IDSCHAR provides menu driven interactive creation of character sets on graphic terminals. The program can emulate various shaped dots and grid spacings. The laser printer has round dots about 8 mils in diameter on a 5.5 mil grid. IDSCHAR also supports a digitizer to allow easy input of character or logo outlines. The outline can then be scaled and presented superimposed on the cells grid for easy filling in. IDSCHAR supports lines, arcs, rectangular area fills plus scaling and rotation. Special logo files are supported for use on forms. An experienced graphics designer can create a complex logo in 1 to 4 hours. Generating a high quality character set takes about 40 hours.

IDSFORM provides menu driven interactive forms creation of forms on graphics terminals. It supports horizontal and vertial lines of 3 different thicknesses. Boxes can be shaded from clear to black. IDSFORM supports subforms which can be defined and then easily moved around both on the page and between different forms. Windows describe boxes consisting of headers and data fields. Data fields can be labelled to allow symbolic access allowing the form to be changed around without modifying the program. The 1040 tax form was perfectly emulated in 14 hours by an experienced user of IDSFORM.

IFS2680 is the formatting program which bundles up different character sets, forms, VFC's and a logical page table into an environment file. IFS2680 also is the program which constructs VFC's and the logical page table for the user. Overall job parameters such as the number of copies of each page desired and the multicopy forms table are specified via IFS2680. HP supplied standard environments are available from IFS2680 either as they stand or as a base to begin creating a unique environment for a special job.

A contributed program called TR2680 which interprets commands imbedded in ASCII files is available. Text editors can be used to prepare memos and reports with the imbedded commands to utilize HP2680A features such as multiple character sets, forms overlay, pen moves etc.

Once an environment file is created it is specified with a new option in the file equation :FILE PRINT;DEV=PP;ENV=FOURT01. The environment file is automatically placed in the spool file before the data. This allows existing programs to use all of the features accessible via environment files without modification.

Power fail and jam recovery are very simple and reliable. Non volatile memory exists in the printer. When power resumes the 3000 retransmits the job from the beginning at high speed. The printer processes the data and resumes printing at the correct point in the job. The only operator invention required is to insure top of form is correctly aligned and push run. Paper jams are similar. If no paper was damaged the job can be resumed without system intervention. If the operator wishes to backup several pages the spooler is suspended, the jam cleared, and the command :RESUMESPOOL LDEV#; BACK 5 PAGES is used. This allows backing up or skipping forward an arbitrary number of pages.

Another unique concept introduced with the laser printer is the error trailer. When a program executes an illegal function such as selecting a missing character set, moving the pen off of the logical page or trying to print a character off of the logical page the printer relays this information to the 3000. This information is then printed out at the end of the job before the trailer is printed. The error trailer describes the error in english, along with the record number and actual page number where the error occured.

Distributed Printing

MRJE has been modified to support HP2680 environment files. If the device class is PP for page printer and the forms field is not empty then the forms identifier is used to locate an environment file.

RJE has an option which allows the translator procedure to process each record when received by the 3000. This allows complete access to the printers features from a mainframe.

One internal test site is running a Series 30 to front end the laser printer. They are printing over 130,000 pages per month. One half of the output is generated by an Amdahl 470 and sent at 9600 baud via MRJE.

At 2900 lpm the printer taxes the performance of most data communication systems. System configuration, CPU overhead and data format determine the printer utilization. The range can be from 10% to 100%. We are currently quantifying printer performance in these areas and welcome user inputs and insights.

Summary

The HP2680A laser printing system provides a cost effective solution to many computer output problems for HP3000 users. The reliability and servicability contribute to its low cost of less than 4 cents per page at 200K pages per month. The unmatched features provide capabilities unique in the industry. The complete software appliation package allows immediate turnkey solutions with no programming. The impact of the laser printer in the distributed network is significant and allows non HP systems to utilize the printer as well as enhancing distributed HP systems.

```
** END OF FORMATTING, NO ERRORS
   IN=EDITOR WORKFILE, TEXT FROM PAPER
   OUT=*LP
```

BUSINESS COMPUTER GRAPHICS/DECISION MAKING

by: Jutta Kernke
Product Manager
Hewlett-Packard Company
Information Systems Div.
19420 Homestead Road
Cupertino, CA 95014

Business Computer Graphics software offers visual display of information which is critical to effective business management decision making.

The Human Graphics Processor

The power of visual information has been recognized since cavemen began drawing pictures on cave walls. Confucius' famous quote about a picture being worth a thousand words is taking on dramatic significance today as modern research on information processing in the human brain shows that a picture is probably worth more than a thousand words. While computers may be ideally suited for an alpanumeric interface, a study of neuro-psychology indicates that the human brain may more effectively utilize a graphical format.

Researchers have found that once a mental visualization of a spatial object has been formed in our mind, reading a written description of that object causes the visualization to be "erased." This may explain why we find it so difficult to visualize trends, patterns and interrelationships when reading tabulated numeric data. Reviewing the tabulated numbers to verify a possible pattern tends to further suppress any visualization that may have been formed.

Presenting the same data in a line graph format, however, makes trend and pattern information instantly understandable and it completely avoids the "interference" problem.

Improving the Human-Computer Interface

It is the human capability to rapidly process visual information that makes graphics such an important interface to computers. The use of pictures and graphs to present data provides concise visual information on trends and relationships which are not immediately evident from the numerical data. Interactive graphics allow computers to be used for design tasks by making it possible for the designer to visualize the object and then modify it--forming a close user-computer interface.

The power of graphics to improve the human-computer interface has lead to a great deal of interest in the computer industry.

Advancing technology is bringing more and more powerful computational tools to applications ranging from inventory control to automated drafting and beyond. And as these tools become less and less expensive, they are becoming generally available to a wider variety of users, many without a background or training in computers. As this process continues, a simple and efficient interface between the user and the computer becomes increasingly important.

Getting Started With Business Graphics

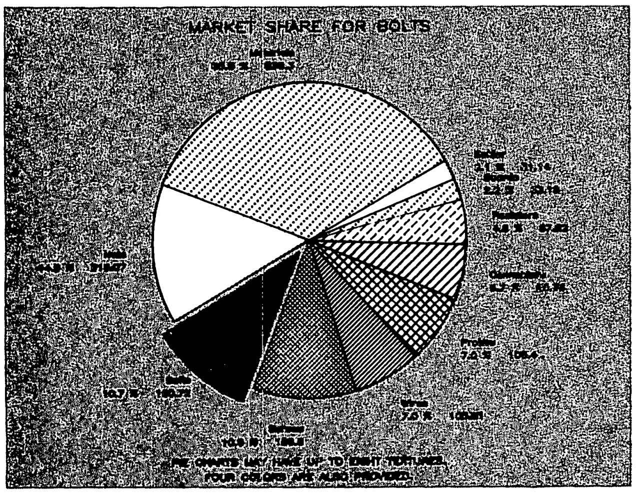
Hewlett-Packard's new interactive business graphics system for the HP 3000 computer family sets high standards for graphics productivity, flexibility and ease of use. It enables the user to take full advantage of computer-stored information resources which are already available to the organization for better understanding and faster interpretation of data.

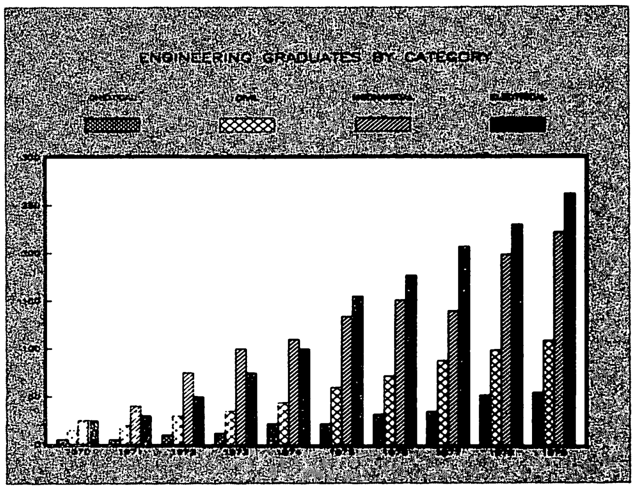
HP Decision Support Graphics/3000 (HP DSG/3000) is a data display system that helps you design, produce, and save business graphs drawn from numerical information kept in any data file on the computer system. The graphs can be displayed on any HP graphics terminal or a printed copy (paper or overhead transparency)can be made on one of the HP multi-color plotters.

HP DSG/3000 is highly interactive and was designed for the periodic and one-time user. Chart design is guided by simple "fill in the blanks" menus presented in logical sequence. A "Help" facility which is built into the system, assists the user upon request. It requires little knowledge of graphics design and the non-computer professional can obtain business graphics without waiting for the development of application programs.

HP DSG/3000 also offers a powerful capability in program development. Both the interactive user and the programmatic user have access to the full capabilities of the product.

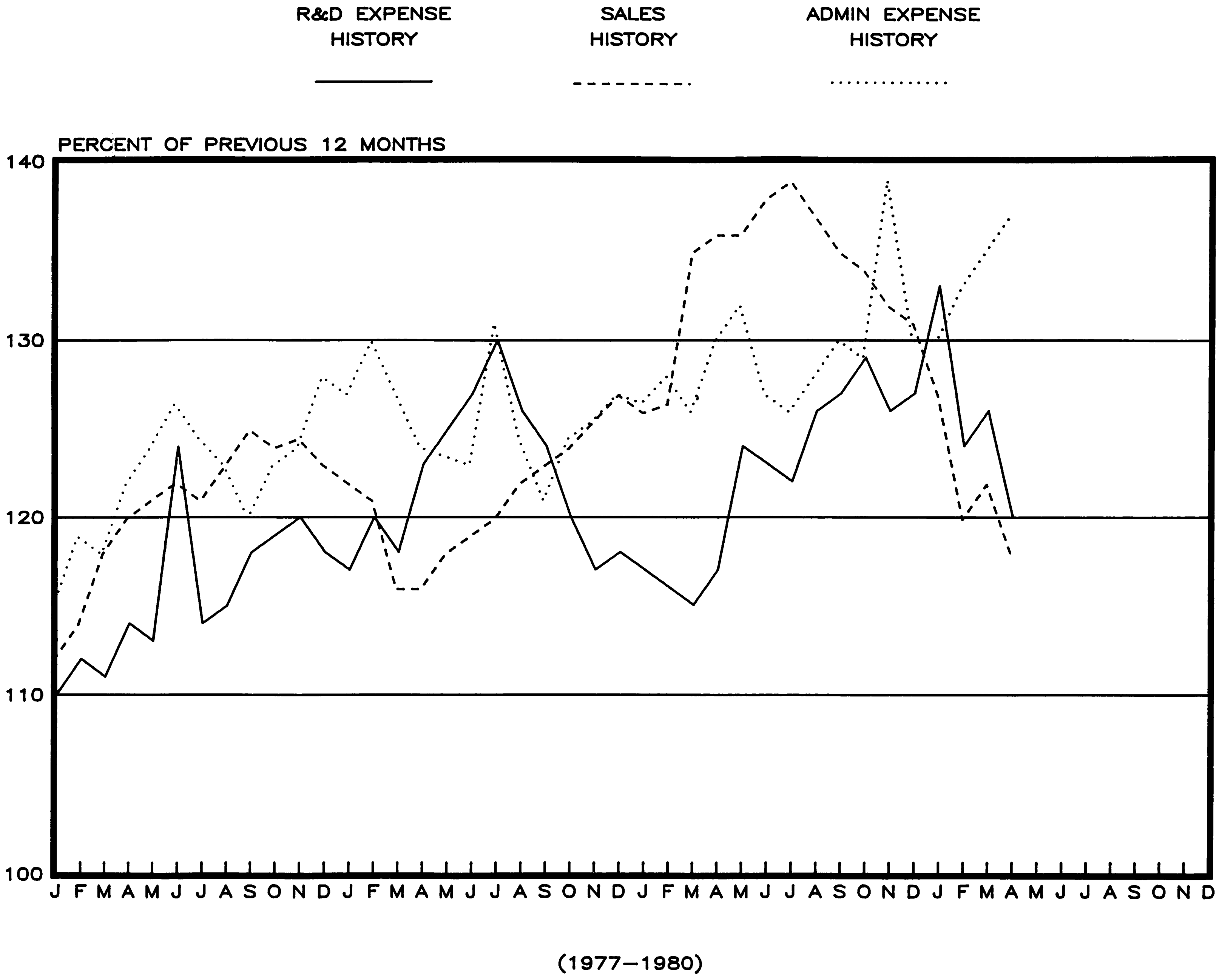
HP DSG/3000 facilitates the creation of line graphs, horizontal and vertical bar charts, pie charts, and scattergrams. All charts can be annotated with symbols and text.





PERIOD EXPENSE CONTROL

12-MONTH MOVING AVERAGE

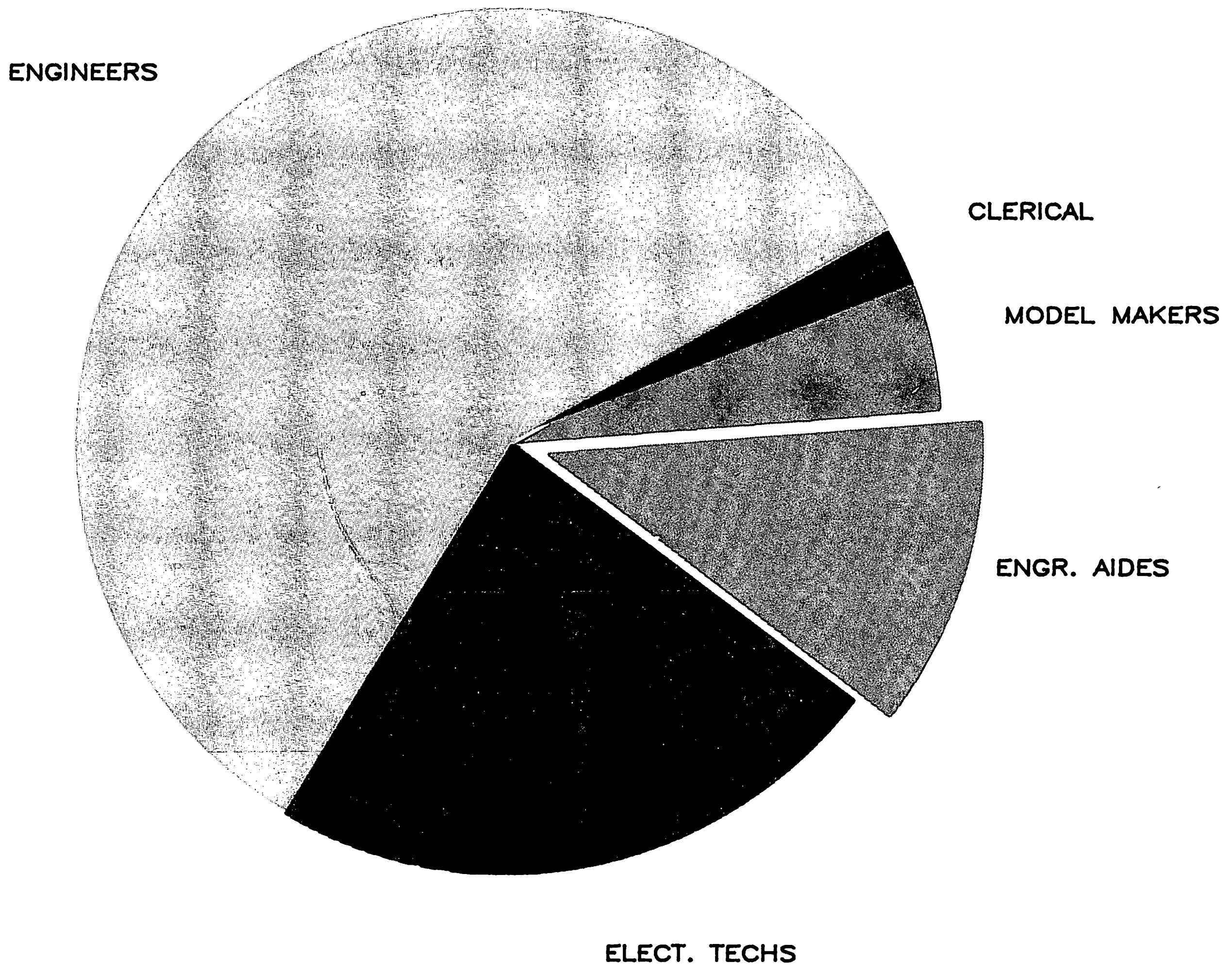


THIS IS A REPRODUCTION OF AN ORIGINAL PLOT CREATED ON AN HP7221 PLOTTER BY HP-DSG/3000 SOFTWARE

P/N: 5953-4075

C-11 - 04

EMPLOYEES BY CATEGORY IN ENGINEERING LAB #5 FOR 1979



C-11 - 05

5953—4074

PSEUDO-DEVICES

by

Paul Primmer - Hewlett-Packard

ABSTRACT

This paper will explain what a pseudo-device is and how it is used in Hewlett-Packard data communication products. Since there are several layers of software on either side of a pseudo-device, it will be necessary to also explain briefly CS, DITs, Monitors, and Attachio. There are differences in the various data communication subsytems and their use of pseudo-devices so this paper will use DS/3000 for all examples.

PSEUDO-DEVICES

SLIDE 1

The objective of this presentation is to give an understanding of pseudo-devices. Pseudo-devices can not be explained in a standalone fashion. That is, the software they link to must also be explained as well as some general I/O concepts. DS/3000 was chosen as an example product to help explain pseudo-devices but it is important to note that there are differences between the different data comm. products.

PSEUDO-DEVICE

AN EFFICIENT MEANS FOR MULTIPLE PROCESSES TO SHARE A NON-SHARABLE DEVICE.

SLIDE 2

SLIDE 2

Pseudo-devices are an efficient means for multiple processes to share a non-sharable device. To properly explain this definition of pseudo-devices requires an understanding of some lower level software and hardware that make up HP's data comm. products.

NON-SHARABLE DEVICES

Intelligent Network Processor - INP

Synchronous Single Line Controller - SSLC

Hardwired Serial Interface - HSI

These three communication devices are accessed through calls to the Communication System Intrinsics (CS). CS does an exclusive open on the above devices similar to an exclusive file open.

SLIDE 3

Currently there are 3 hardware devices available for synchronous data communication: Intelligent Network Processor - INP, Synchronous Single Line Controller - SSLC, and the Hardwired Serial Interface - HSI. All three are accessed by a set of system intrinsics called CS which stands for Communication System. Like the file system, which provides a common set of high level intrinsics for access to dis-similar hardware (i.e tape, disc, and line printers), CS provides a common set of calls for accessing these 3 synchronous devices. In order to use one of these devices a COPEN procedure is called which does an exclusive open on the device making it non-sharable.

CS

The Communication System intrinsics are used by all current HP data comm. products to provide the Binary Synchronous protocol.

BISYNC:

```
                 ENQ--->
                          <---ACK0
 BCC ETX TEXT STX--->
                          <---ACK1
 BCC ETX TEXT STX--->
                          <---NAK
 BCC ETX TEXT STX--->
                          <---ACK0
                 EOT--->
```

SLIDE 4

All current HP data comm. products are based on IBM's Binary Synchronous Communication protocol which was first introduced in 1966 and has since become the industry de facto standard for medium and high speed data communication. What BISYNC offers is an effective protocol for sending blocks of text and a means of recovering from line errors. This handshaking between stations is accomplished by using special control characters and a predefined line protocol.

DS CONVERSATIONAL BISYNC

```
                    ENQ--->
                              <---ACK0
BCC ETX TEXT STX--->
                              <---STX TEXT ETX BCC
BCC ETX TEXT STX--->
                              <---NAK
BCC ETX TEXT STX--->
                              <---STX TEXT ETX BCC
                    EOT--->
```

TEXT FOR DS:

```
HEADER (16 BYTES)
    +
APPENDAGE (VARIABLE)
    +
DATA (USER DEFINED)
--------------------
TEXT
```

SLIDE 5

DS/3000 uses a modified form of BISYNC called conversational. This conversational form improves efficiency by answering a correctly sent block of text with text. BISYNC can be thought of as the transport mechanism for this text block. For DS/3000 this text block always contains a 16 byte header which contains information such as: the from PIN, the to PIN, what type of message, how long a message, and is the data compressed. In addition to the header, which is always present, there is an optional appendage section which can be thought of as a header extension. Finally, there is the users data. These 3 parts are combined by the upper level DS intrinsics and passed to the lower level CS for transport over the data link.

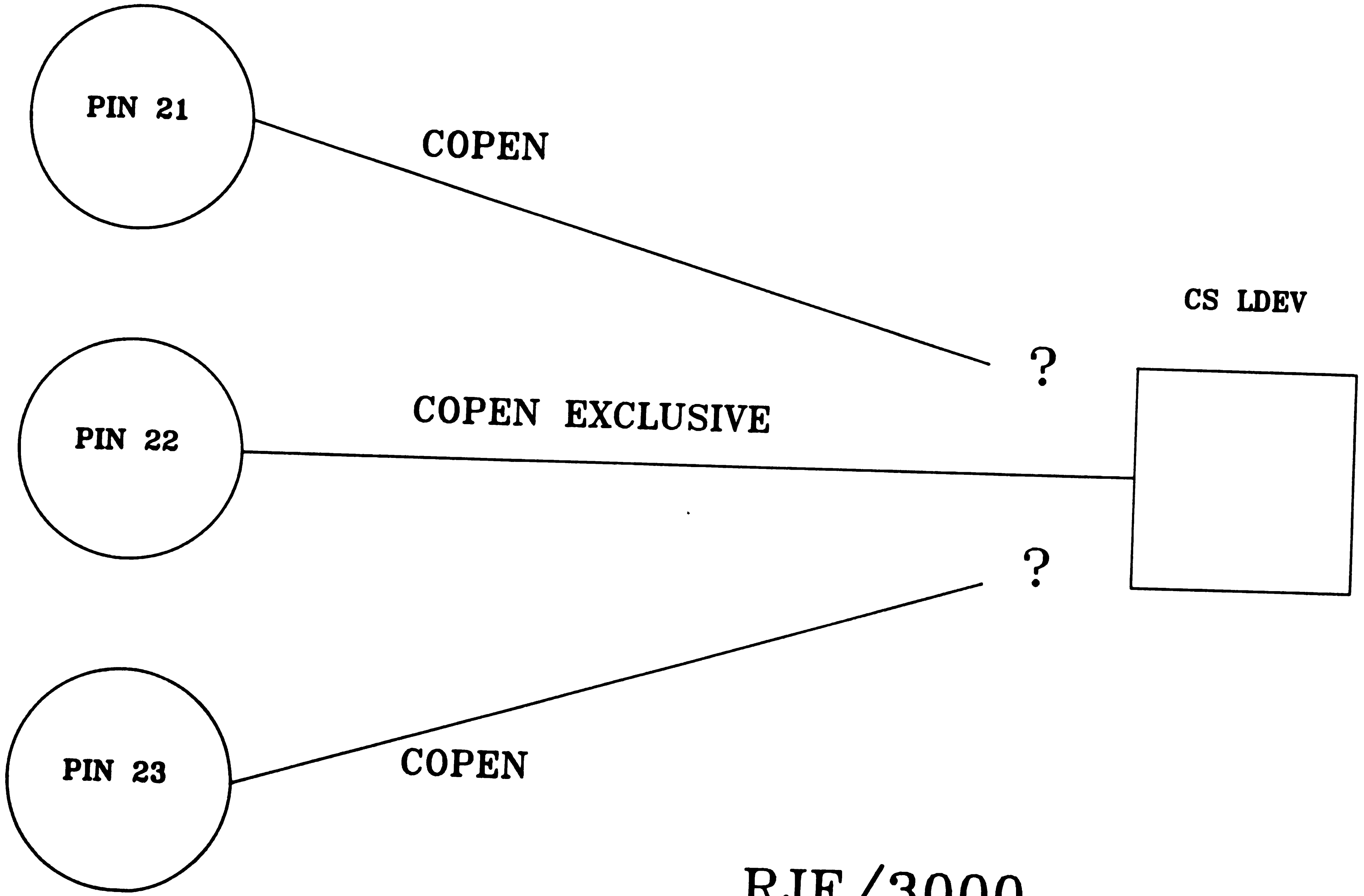
CS INTRINSICS

COPEN	EXCLUSIVE OPEN OF LINE
CCLOSE	CLOSES LINE
CWRITE	WRITES DATA TO LINE
CREAD	READS DATA FROM LINE
CCONTROL	ALLOWS VARIOUS CONTROLS OF THE LINE TO BE PERFORMED

SLIDE 6

Rather then writing a special communication protocol for each data comm. product, a common set of intrinsics were developed. This is analogous to the common set of intrinsics for the file system. Of particular note is that the COPEN intrinsic does an exclusive open. Exclusive access is required due to the nature of the communication link which requires specific responses at specific times. If this device were shared among several processes, the result would be total confusion.

USER PROCESSES

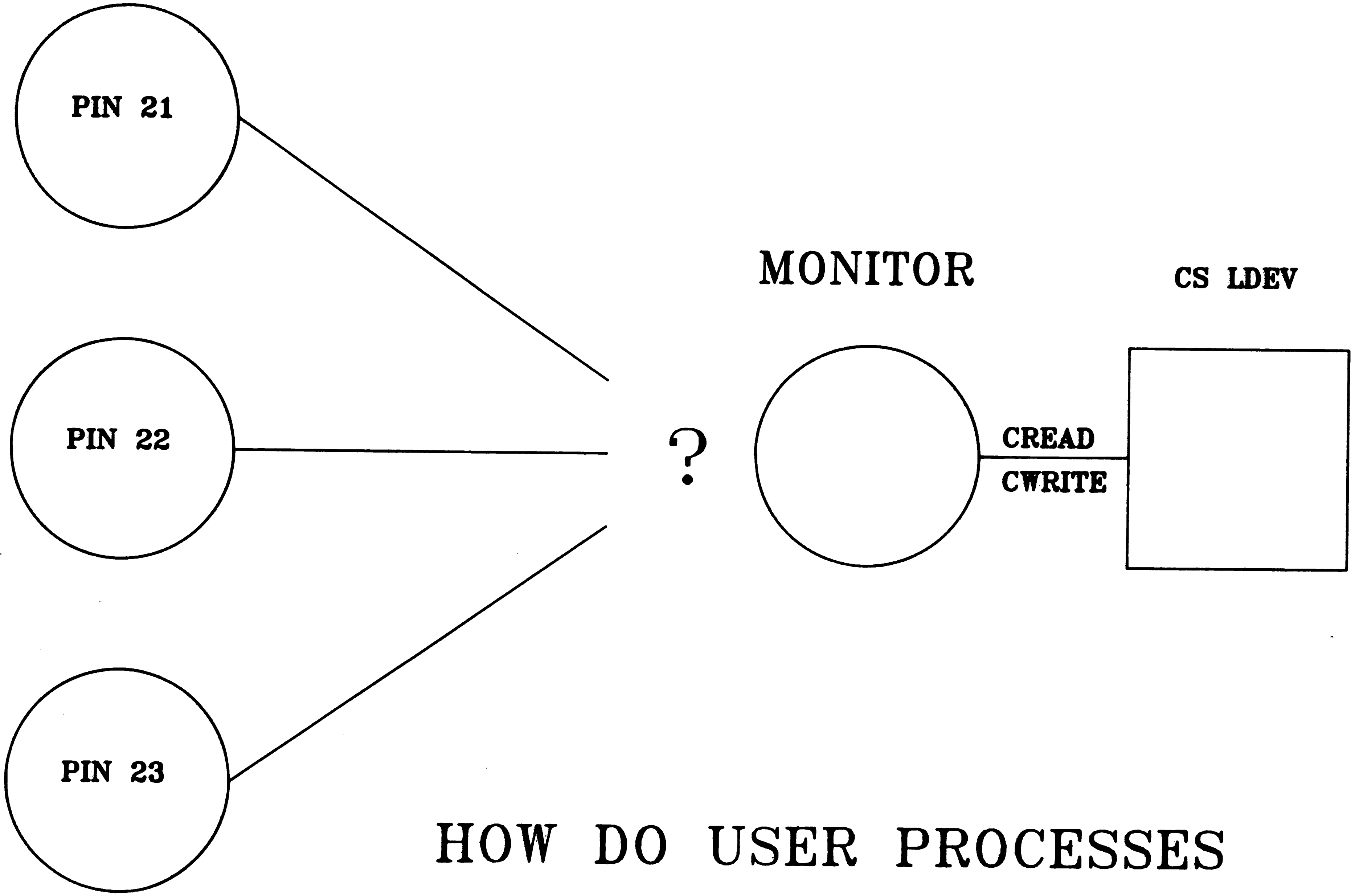


RJE/3000

SLIDE 7

This exclusive open is evident in the product RJE/3000 which is nothing more than a large program issuing CS intrinsic calls. But this scheme would be unacceptable for DS/3000 which allows multiple users at both ends to be using the same communication link.

USER PROCESSES



HOW DO USER PROCESSES TALK TO MONITOR?????

SLIDE 8

A way to overcome this problem is to allow one process to exclusively open the communication link and do all the reading and writing to the line. This process is called the monitor. More specifically in DS/3000 it is called DSMON and is created when the operator types "DSCONTROL ldev;OPEN". Now there is an exclusive owner of the communication device which the other processes can direct their requests. But there are two problems with processes talking to this monitor process.

INTER-PROCESS COMMUNICATION

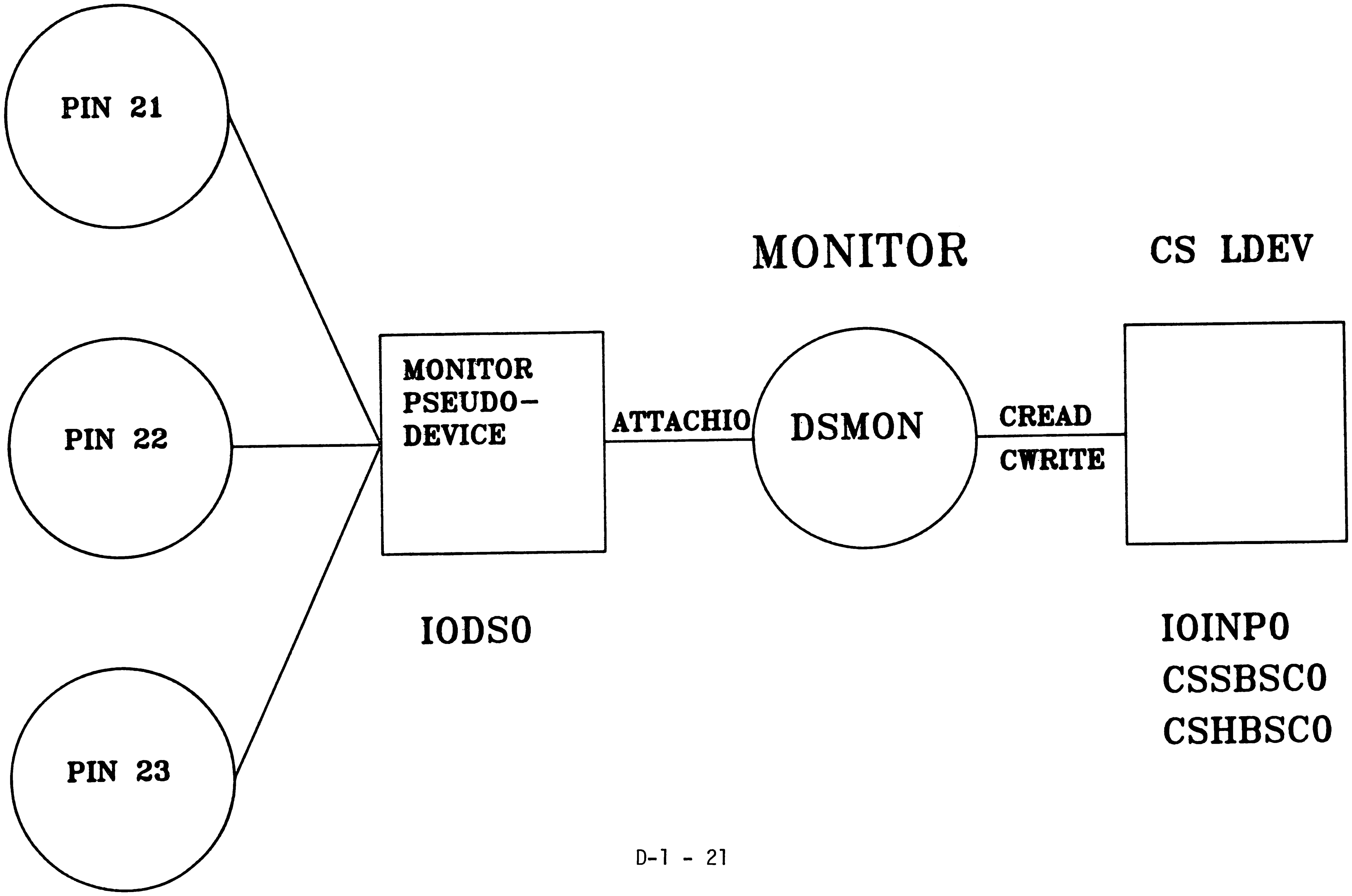
PROBLEM #1: INTER-PROCESS COMMUNICATION ONLY BETWEEN FATHER AND SON PROCESSES.

PROBLEM #2: MONITOR NEEDS TO BE AWAKENED WHEN EITHER CS LINE I/O COMPLETES OR WHEN A USER PROCESS HAS SOMETHING FOR THE MONITOR TO DO.
UNDER MPE III, WE CAN WAIT FOR EITHER I/O COMPLETION OR FOR PROCESS ACTIVATION BUT NOT BOTH AT THE SAME TIME!

SLIDE 9

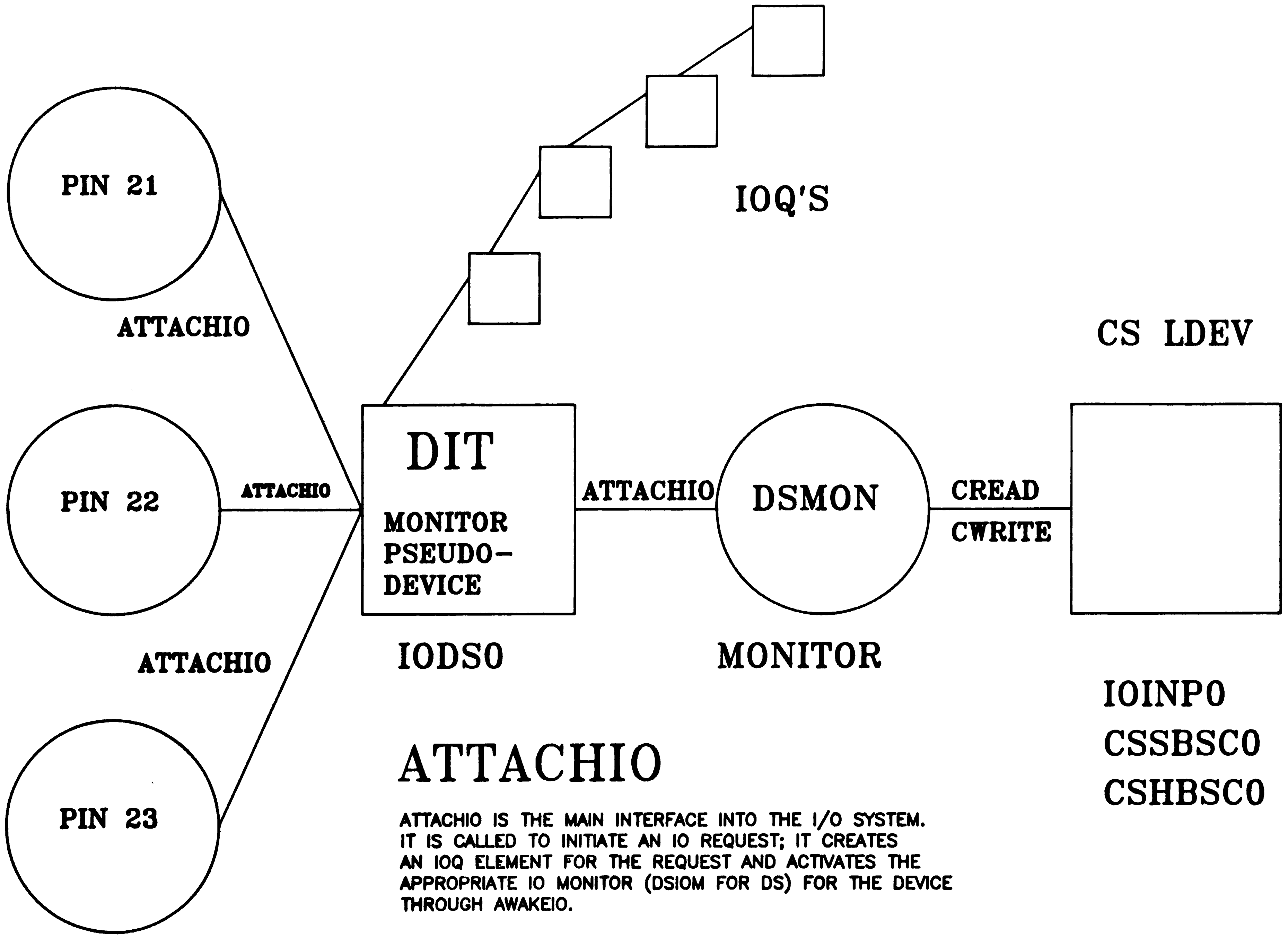
In MPE-III only processes in the same family (father/son) can communicate. Secondly, the monitor process is controlling the communication line and therefore is in an I/O wait. In MPE-III, a process can wait for I/O completion or for process activation but not both at the same time. How then will processes get the monitors attention?

USER PROCESSES



SLIDE 10

By creating a pseudo-device the monitor can issue no-wait I/O requests to both the CS device and the pseudo-device, and service the first request to complete. Now user process get the attention of the monitor by writing their request to the pseudo-device via calls to ATTACHIO. The data in the write request becomes the data for the monitor's read request. The monitor is awakened by completion of I/O and discovers that it is from the pseudo-device so it cancels its' current CS request and performs the desired I/O. In a normal idle state the monitor is asleep with a CREAD to the CS device and an ATTACHIO read to the pseudo device. When either request completes the monitor awakens for service. But who is ATTACHIO?



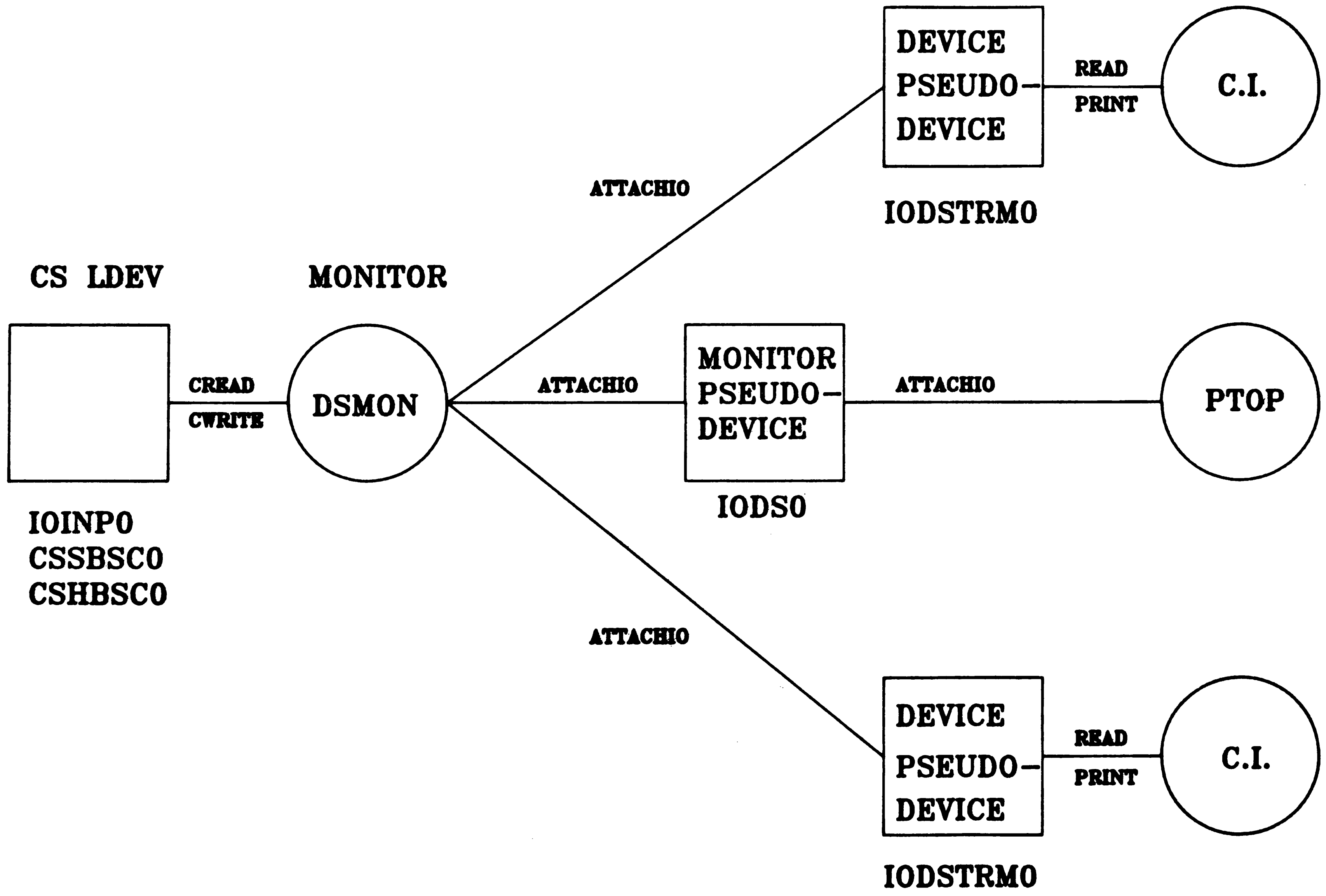
ATTACHIO

ATTACHIO IS THE MAIN INTERFACE INTO THE I/O SYSTEM. IT IS CALLED TO INITIATE AN IO REQUEST; IT CREATES AN IOQ ELEMENT FOR THE REQUEST AND ACTIVATES THE APPROPRIATE IO MONITOR (DSIOM FOR DS) FOR THE DEVICE THROUGH AWAKEIO.

SLIDE 11

ATTACHIO is the main interface into the I/O system. For example, when a user issues a file system intrinsic, that intrinsic will eventually call ATTACHIO. ATTACHIO will create an eleven word IOQ element which describes the nature of the request (i.e read, write, or control), the location of the buffer, the PIN that made the request, and any miscellaneous parameters that are device dependent. This IOQ element is in a common table for all devices, but all requests for the same device are linked together by one of the eleven words. Every device configured in the system has a Device Information Table (DIT) to keep track of what the device is currently doing and a pointer back to the IOQ table for all the requests pending. After ATTACHIO creates the IOQ element, it requests and activates the appropriate I/O monitor. In the case of DS/3000 the I/O monitor is named DSIOM. A real device monitor at this point would call the appropriate drivers to do the real physical I/O. DSIOM is the 'nerve center' for DS I/O and takes care of routing the request to either IODS0 or IODSTRM0. These drivers move the data from the users buffer (pointed to by the IOQ) into DSMON's stack (pointed to by the IOQ request it made earlier). Both IOQs are flagged completed; DSMON is awakened and performs the I/O to the CS device.

SLAVE SIDE


DEVICE PSEUDO-DEVICE
READ
PRINT
C.I.
ATTACHIO
IODSTRM0
CS LDEV
MONITOR
CREAD
CWRITE
DSMON
ATTACHIO
MONITOR PSEUDO-DEVICE
ATTACHIO
PTOP
IODS0
IOINP0
CSSBSC0
CSHBSC0
ATTACHIO
DEVICE PSEUDO-DEVICE
READ
PRINT
C.I.
IODSTRM0

SLIDE 12

At the slave side the monitor is in a I/O wait state when its' CREAD completes, and it reads the data into its' stack. The slave side needs some process to perform the DS request and rather than writing special code to do this, a C.I. is created at the slave end. But the C.I. is designed to talk to an interactive terminal not a monitor, so a pseudo-terminal device was created which looks and acts just like a real terminal to the C.I.. When the slave C.I. responds with an answer it calls the PRINT intrinsic which calls ATTACHIO which calls DSIOM which invokes IODSTRMO. IODSTRMO moves the buffer to DSMON's stack and DSMON CWRITES the request to the CS line.

SUMMARY

A pseudo-device becomes an efficient means for multiple processes to share a non-sharable device. By using a pseudo-device a monitor process can use no-wait I/O to act as a traffic cop to the CS device. Upper level software can use standard I/O routines (ATTACHIO) to make requests to the lower level I/O system. By isolating the different functions a layered software approach can be realized which allows multiple products to share common routines (CS).

??????????????? QUESTIONS ???????????????

ROBELLE CONSULTING LTD.
#130-5421 10th Avenue
Delta, B.C. V4M 3T9
CANADA
(604) 943-8021
Telex 04-352848

HP 3000/OPTIMIZING BATCH JOBS

Technical Report, February 1981, By

ROBERT M. GREEN

Robelle Consulting Ltd.

Summary

The elapsed time of high-volume, stand-alone tasks can sometimes be reduced dramatically. This report is a follow-up to my 1978 paper on optimizing on-line programs. A large number of techniques for batch optimizing are explained, evaluated by empirical testing, and, finally, ranked according to their relative "power". The tests show than some highly-recommended techniques are of little benefit, while other techniques, often overlooked, will make some tasks execute 4 to 12 times faster.

Contents

I. Why Bother With Batch?

II. What is "Batch" and How Fast Should it be?

III. How to Verify Proposals for Improving Batch

IV. Possible Techniques for Batch Optimizing

A. Changes to Data Storage
B. Simple Coding Changes
C. Changes to Application Logic

V. Results: What "Actually" Makes Batch Faster?

VI. Consequences for Different People

Appendix: References (Sorted by Author)

Section I — WHY BOTHER WITH BATCH?

In December of 1978, when I gave my paper in Denver on optimizing on-line programs [15], the techniques that I presented were not common knowledge among users of the HP 3000 (or even among Hewlett-Packard Systems Engineers). The ideas discussed at the meeting ("30 disc I/Os a second" [41]), have spread throughout the user community as a result of numerous papers and articles [39] [48] [09] [19]. With hundreds of intelligent professionals pushing the HP 3000 to support the maximum number of terminals, is it any surprise that many have succeeded?

In less than ten years, the HP 3000 has grown from 128,000 bytes of main memory to four megabytes, and the power of the central processor has increased several times. Software has been rewritten and refined (by Hewlett-Packard and independent vendors) to incorporate the principles for optimizing on-line programs. Today, HP 3000 users develop systems which support 20 to 40 terminals without serious difficulty. In 1973, users had trouble supporting 8 terminals on an HP 3000. The speed of batch jobs, however, is not much better than it was in 1973.

But what about batch processing? One of the most widely recommended techniques for improving on-line response time has been to "dump" big tasks into the batch queue [15] [45]. This is a natural and useful idea; the batch queue is where "batch-type" operations belong. Unfortunately, as applications mature, databases grow in size, batch jobs take longer to complete, and new batch programs are put into production. Eventually, the batch queue is clogged. For many HP 3000 sites, completion of month-end and year-end batch jobs is a major irritant.

Slow batch jobs can be very costly. Operator overtime may be required for a graveyard shift. Extra computers may be needed to handle the batch load. The batch capability of the HP 3000 is not an infinite sink into which "problem" tasks can always be dumped. It is a finite resource that can easily be exhausted.

This report attempts to discover how to complete those high-volume, stand-alone, batch jobs in less time (i.e., 8 hours instead of 16). Although I will note the impact on other users of each optimizing technique, interference with concurrent users will not be a conclusive argument against any proposal. In addition, my focus will be on empirical verification of gains in throughput, with a goal of ranking all optimizing techniques by their relative "power".

This report is very specialized. You will not learn what to do about a flood of small batch jobs that are swamping your HP 3000. (The techniques needed to solve that problem are the same ones that you would apply to on-line sessions: reduce number of logons, eliminate UDCs for batch logons, :ALLOCATE program files, etc.) You will not learn how many batch jobs you should run concurrently. (The answer to that question may change drastically when MPE IV is released.) Nor will you discover how to reduce the impact of batch jobs upon on-line response time. (MPE IV should give you the tools you need to solve that problem, if you are willing to degrade your batch throughput [48].) What you will receive are practical suggestions for improving batch throughput.

This document is the result of an investigation into reducing elapsed time. Many of the conclusions that you will read here were surprises to me, only uncovered when theory was put to the test under controlled conditions. The first three sections of this document are introductory: purpose, theory and method. The fourth section contains the bulk of the investigative results and explanation: thirty proposals for optimizinng. Every technique is given a thorough treatment (although some techniques were found to be much more effective than others). Negative knowledge is as important as positive, if is saves you from wasting time on methods with little chance of success. The last two sections summarize the results: ranking the techniques and showing the implications of the investigation for different readers. Finally, in an appendix, I have listed alphabetically all of the sources that I referenced, plus a suggested reading program for those who would like to become proficient in this subject. The reading list starts with general survey papers and progresses to the most technical reports.

Section II | WHAT IS "BATCH" AND HOW FAST SHOULD IT BE?

The distinguishing characteristic of a batch job is that it has an extremely "stupid" controlling terminal. An interactive session is controlled by a terminal with a human operator; the controlling terminal can tell the session what to do if an unusual situation arises. Given the primitive job control language of MPE, a batch job cannot handle many unusual events. Also, the particular batch jobs that this report addresses are the ones that process a high volume of transactions.

BATCH VERSUS ON-LINE

How does high-volume, stand-alone processing differ from on-line, interactive processing? One big difference is that a batch job does not have a user watching its "response time", waiting, hitting the RETURN key, losing patience, and telephone the DP department. If a batch job is slow, no one is likely to complain, and no one is likely to improve it.

We should be able to use the same general strategy to make all programs run faster (batch and on-line), but we may have to vary the specific techniques we use and the areas where we focus our attention. As a start, I would like to review the five "prinicples for optimizing on-line programs" from my previous paper [15]:

"Make each disc access count" is important to both batch and on-line. Efficiency of disc usage may be the determining factor in the speed of batch jobs.

"Maximize the value of each terminal read" is obviously irrelevant; it was designed to "spread out the load" of many independent, unpredictable, terminal users. A batch job is one, continuously heavy demand for resources. All we can do is make certain that we do not run many batch jobs at the same time.

"Minimize the run-time program size" sounds like a good idea; but, if a job is run stand-alone, the program size is not likely to slow it down. In fact, the "tricks" used to reduce the size of a program may actually increase clock time by a small amount (a loss that cannot be measured in an on-line program may be quite noticeable in a batch program).

"Avoid constant demands for execution" is clearly impossible; that is the definition of a batch job. On-line programs, on the other hand, should consist of "short" bursts of activity, divided by long periods of inactivity. If all of the on-line users demanded the resources of the HP 3000 at the same time, and did not let them go, the system would be swamped.

"Optimize for the common events" should apply even more to batch jobs than it does to on-line programs. The events that are repeated thousands or millions of times offer the most potential for reducing overall elapsed time, even if the improvement per "event" is only a small reduction in CPU time.

GENERAL STRATEGY

In theory, there are four strategies that "should" reduce the elapsed time of batch jobs [48] [47]:

1. Eliminate some disc transfers completely.
2. Do the same work with fewer disc transfers.
3. Use less CPU time.
4. Overlap a disc transfer with useful CPU work.

One strategy deliberately missing from this list is: "Minimize the amount of main memory used". Most documents on performance place a high priority on reducing code/data segment size [15] [41] [39] [26] [47] [18]. It is implicitly assumed that you are memory-bound. However, a Series III or Series 44 model of the HP 3000 with full memory is a very large computer. Without using "extra data segments" or other advanced programming techniques, a single program can use only 64,000 bytes of the main memory for data (that may be a small fraction of the main memory that is actually available). Techniques will be examined that trade-off increased code and/or data space for decreased disc transfers or CPU time.

THEORETICAL SPEED LIMIT

The theoretical limits on the speed of a batch job are the physical speed of the disc drives [41] and the power of the CPU. If zero CPU time were needed to process the data (or, if processing could be exactly overlapped with disc transfers), a batch job could progress at pure disc speed.

Hewlett-Packard disc drives have the following speed characteristics [48] [26]:

1. 400 microseconds to transfer a sector of information (256 bytes); 22.2 milliseconds to transfer a full track on the 7925 (16.6 ms. on 7920, 7906, 7905).

2. 11.1 milliseconds average latency on the 7925 (8.3 ms. on the 7920, 7906, 7905); latency is the time it takes for the disc to revolve so that the "head" will be located over the sector where you want to start the next transfer.

3. 5 ms. track-to-track seek time on all drives (25 ms. average seek, 45 ms. maximum seek from edge to edge).

4. 64 sectors per track on 7925 drive (48 sectors on 7920, 7906, 7905).

Since batch processing often involves large sequential scans, we need a target "best" rate for transferring large chunks of contiguous disc data. Assuming a 7925 disc drive and an average latency of 11.1 ms, what is the time needed to transfer 1024 sectors (16 tracks)? The answer depends upon the size of each individual transfer. To understand this point, you need to calculate the effective data rates, using three different transfer sizes: 256 bytes (one sector), 8,192 bytes (1/2 track) or 16,384 bytes (a full track). In all three cases, overall seek time will be about 80 ms., if the data is located in adjacent tracks (5 ms. times 16 tracks).

The time-per-transfer equals the latency time (spin to proper sector), plus the actual transfer time (.4 ms. per sector). The total time to transfer the 1024 sectors of data equals the time-per-transfer multiplied by the number of transfers, plus the seek time (calculated above to be 80 ms.).

Size-of-Transfer	Time-per-Transfer		Time-for-1024-Sectors
256 bytes	11.5 ms.	(x1024+80=)	11.856 seconds
8,192 bytes	22.2 ms.	(x32+80=)	0.790 seconds
16,384 bytes	33.3 ms.	(x16+80=)	0.613 seconds

Why do larger transfers improve the overall data rate by so much (15 to 19 times faster)? Because the disc revolves at a constant rate! If you read one sector at a time, you must still wait for a full revolution of the disc (22.2 ms.) before you can read the next sector.

In practice, full-track transfers are not quite as efficient as they appear above. An application program cannot make use of every disc revolution; it wastes some of them. After reading the first track, your program cannot generate a new read request for the next track before the disc itself has revolved past the starting sector of the next track. Therefore, the best possible data rate is closer to that for half-track transfers than full-track transfers (i.e., .79 seconds per 1024 sectors, not .61 seconds). To this time must be added the unavoidable CPU overhead of MPE (about 8 ms. per request).

CONCLUSION: A rough estimate of the fastest possible speed for batch processing is ONE SECOND PER 1000 SECTORS (see :LISTF,2 for the number of sectors in a file).

Section III | HOW TO VERIFY PROPOSALS FOR IMPROVING BATCH

MPE and the HP 3000 constitute a very complicated mechanism. Sometimes, so many factors are at work that an "obvious" truth turns out to be totally false (or so hedged with qualifications as to be useless). Wherever possible, I have tested each optimizing proposal by performing a reproducible experiment, rather than depending upon intuition, common sense or theoretical "proofs". Fortunately, stand-alone batch jobs are the easiest computer tasks to measure.

The key to a good experiment in optimizing is to vary only one factor each time a test run is made. For example, to test the effect of different blocking factors, I have attempted to run exactly the same program on exactly the same data, changing only the blocking factor. This sometimes conflicts with another goal, that of using "real" programs in tests. Real programs may perform other operations that vary from run to run and are not exactly reproducible (i.e., spooled output may go to different positions on different discs).

I have focused on reducing the elapsed time of fixed batch jobs. Normally, the CPU time increases or decreases in tandem with the elapsed time; so I have only shown the CPU time when it is exceptional. Unless otherwise noted, all tests were run on a Series III, under MPE release 2011 ("Athena"), using 7925 discs.

Section IV — POSSIBLE TECHNIQUES FOR BATCH OPTIMIZING

In order to compile a list of suggestions for optimizing batch jobs, I searched through past literature on the HP 3000. Although there are no papers on this exact topic, many papers described specific techniques that can be applied to batch jobs. I reviewed all of the HPGSUG publications (journals, newsletters, conference proceeedings), a large body of Hewlett-Packard documentation (S.E. notes, support newsletters, manuals, etc.), and many private reports that I have accumulated in ten years of working on the HP 3000. I reduced the results to a list of about 30 different ideas.

The optimizing techniques were then grouped into three classes, based upon the degree to which existing applications must be modified:

A) CHANGES TO DATA STORAGE (easiest),
B) SIMPLE CODING CHANGES, and
C) CHANGES TO APPLICATION DESIGN (hardest).

For each proposed optimizing technique, I attempted to perform a verifying experiment. When that was not possible, I reported experiments done by other users. In a few cases, I mention ideas that have not been verified to my satisfaction. These ideas are carefully noted; you, the reader, are invited to send me the results of any tests you may do to prove (or disprove) these suggestions.

A. CHANGES TO DATA STORAGE

If throughput can be increased by changes in the way the data is stored, this will often be the most economical optimizing alternative. Since a data storage change does not usually require any program changes, expensive programmer time is not needed.

A1. INCREASE BUFFERS FOR MPE FILES

MPE disc and tape files are "buffered"; several logical records are packed into each physical record ("block"). The file system reads these blocks into buffers that are kept in extra data segments, then passes individual logical records to/from the program as requested. When you access a file sequentially, the file system "pre-reads" the next physical block. Theoretically, buffering should increase batch throughput, allowing overlap of useful CPU work with disc transfers. However, tests have shown that this is not the case.

In a test of file-copies with FORTRAN [09], two buffers (the default) was 1.08 to 1.16 times faster than one buffer, but only if you called FREAD/FWRITE directly (instead of using the FORTRAN I/O statements). Increasing the buffers beyond two did not improve the speed at all (:FILE XXX;BUF=4).

The time to copy 1000 records (333 sectors) varied from 10 to 28 seconds. This works out to an effective "data rate" of 15 to 42 seconds per 1000 sectors (666 sectors must be processed in total, since each sector must be read in and written out). The "ideal" speed for file copies, as deduced in Section II from the speed of of the disc hardware, is less than 2 seconds per 1000 sectors. (Many times faster; certainly room for improvement!)

In another test [38], sort times were 1.04 to 1.07 times faster with two buffers, instead of one, but only if the blocking factor was small (the test was done using the pre-1918 sort, without the benefit of NOBUF). More than two buffers did not improve sort times. In fact, it sometimes caused a slight increase in times. I ran my own tests with FCOPY, and verified these results (BUF=2 is 1.085 times faster than BUF=1, but BUF=4 is not faster than BUF=2).

Why doesn't buffering work? Another paper [48] explains it very well. MPE III uses 2.9 milliseconds of CPU time for each logical transfer (buffer to stack), plus another 8 ms. for each physical transfer (disc to buffer). For an 80-byte record, blocked 16, the CPU time to read a block is 52.5 ms., but a disc access only takes 33 ms. MPE III IS CPU-BOUND! The author of [48] suggests that buffering can help random access files by creating a "cache" of active logical records! But, the only time I tried this technique, the program ran slower. These conclusions may change on the Series 44 (MPE IV), as tests indicate that FCOPY is 2.25 times faster than on a Series III with MPE III [36]. MPE IV may not be CPU-bound.

A2. INCREASE KSAM KEY BLOCK BUFFERS

KSAM, an acronym for Keyed Sequential Access Method, is the Hewlett-Packard equivalent of ISAM. KSAM uses buffers located in an extra data segment, but differently from the file system. KSAM always allocates one buffer for the data block and a number of buffers for the key blocks. It appears that KSAM now allocates enough key block buffers (in most cases) by taking into account the number of levels in the B-Trees. However, if the KSAM file is empty, KSAM may not allocate enough buffers. In one experiment [10], the user improved the time to load an empty KSAM file by 2 to 10 times through an increase in buffers. (The key block buffers are controlled by the 'numcopies' field; :FILE XXX; DEV=,,13 for 13 buffers.)

A3. INCREASE IMAGE BUFFERS IN DBCB

The IMAGE/3000 database system also uses buffers external to the user data stack to provide blocking of logical entries into physical disc blocks. There is a major complication with IMAGE. Unlike KSAM and the file system, IMAGE uses a shared, "global" buffer pool for each database, with the users of the database competing for the available buffers. IMAGE allocates an extra data segment called a DBCB, large enough to hold the number of buffers that IMAGE thinks will be needed. This number is based upon the highest number of paths into a detail dataset, and the number of users who have the base open (i.e., the number of buffers may vary dynamically during the day). This default number of buffers can be overridden with the BUFFSPECS command of DBUTIL. What a tantalizing prospect!

One published test varied the number of buffers while doing a DBLOAD [49]. When DBLOAD reloads a database from tape, it is acting like a long batch program that does many DBPUT operations. The reload time was reduced 1.26 to 2.53 times by progressively increasing the buffers for one accessor from the default (9) to the maximum (255). The fastest time occurred between 30 and 100 buffers. That is not surprising, since the actual number of buffers that can be allocated is limited by the size of the largest extra data segment (32,000 words). With a blocksize of 512 words, the maximum is about 50 buffers (100 buffers for 256 words, 25 for 1024 and only 12 for 2048 words). DBUTIL does NOT give you a warning if you request more buffers than is possible [07]. I have contributed a program called LISTDBCB that shows the current size of the DBCB of any database (some versions of SOO show the DBCB also).

The one experiment that I did to verify this technique gave disappointing results. By doubling the number of buffers, I only improved the elapsed time (to do 2551 DBPUTs) by 1.03 times. Perhaps my DBPUT operation was not complex enough to require the extra buffers that were available; or, doubling the buffers may not have been enough.

During on-line access to the database, I recommend the default BUFFSPECS. Due to the algorithm that IMAGE uses to allocate buffers to users, it is very easy for the entire buffer pool to be "flushed" [07]. Thus, an increase in the total number of buffers can actually make your on-line response time worse by increasing the size of the DBCB that must be swapped. For stand-alone batch access, it is worth experimenting with increased buffers, especially if you are doing complex transactions involving puts, deletes and updates to several datasets, or to datasets with many paths. I suggest that, for one database accessor, you ask for the maximum buffers (i.e., set BUFFSPECS to 255 and let IMAGE allocate as many buffers as it can); there is no point in choosing a compromise number. Also, both of the tests reported above were run in "output-deferred" mode (see technique B4 below); the results may not be as good with the default "output-complete" mode.

A4. INCREASE BLOCK SIZE OF MPE FILES

The most well-known maxim of optimizing is: "increase the block size for batch (serial) access and decrease the block size for on-line (random) access" [33]. How well does this maxim actually work on the HP 3000? The block size is determined by the blocking factor, which is the number of logical records stored in a single physical block of disc space. A block always starts on a sector boundary and is the smallest unit that can be transferred between disc and memory. Since lack of disc transfers is a primary limitation on throughput, increasing the number of records retrieved in each physical transfer should increase the speed of batch programs.

For MPE files, the blocking factor is determined when the file is created (:BUILD XXX;REC=-80,16,F,ASCII specifies a blocking factor of 16 and a block size of 1280 bytes). If you do not specify the blocking factor explicitly, MPE selects a default value by dividing the record size into 128 words (a sector). If the record size is over 128 words, MPE uses a value of one. Thus, MPE chooses the smallest possible block size. Several programs are available in the contributed library that select an alternate blocking factor to minimize the disc space allocated. Another way to choose the blocking factor is to minimize processing time (but, what block size does optimize for speed?).

A number of published tests have measured the effect of varying the blocking factor. A test of file copies with FORTRAN [09] found that increasing the block size above the default improved speed by 1.25 to 1.53 times, with benefits diminishing when the block size exceeded 512 words. The improvement was more marked when I/O was done using the FORTRAN read/write statements than when FREAD/FWRITE were called directly (perhaps because the base time was much slower). The test was for 80 byte records, blocked from 3 (default) to 32. The test also showed that the default blocking (3 r/b) was 1.8 to 2.22 times faster than one record per block. I reproduced this test using FCOPY instead of FORTRAN and obtained similar results, except that the improvement was not as great.

A test of sorts on disc files (before the 1918 release of NOBUF sorts) showed speed increases of 1.26 to 1.42 times, with no improvement above a block size of 1000 words [38]. In order to investigate the impact of actual block size (as opposed to blocking factor) on performance, I ran some FCOPY comparisons with 150-word records. I varied the blocking factor from 1 (default) to 10 and recorded speed increases of 1.17 times (blocked 2) to 1.28 times (blocked 10), with only a small improvement above 600 words. In no case did I find a significant increase in buffered access speeds by increasing the block size from 512 to 1024 words.

I conclude that there is so much CPU overhead in the file system (and the other general interfaces that sit on top of the file system), that the best you can expect is an improvement of

1.25 times (if your blocks are small currently). Since very large blocks can have a detrimental impact on terminal users and show no performance benefit, you should pick the block size between 512 and 1024 words that minimizes disc space. However, it is also possible to access files in a non-buffered mode (see ideas B2 and B3) for better performance. I will show that it is possible, with NOBUF access, to have "the best of both worlds" (big block for batch and small block for on-line), if you choose your block size correctly.

A5. INCREASE BLOCK SIZES FOR KSAM

In KSAM, there is both a data file blocking factor and a key file blocking factor [10]. The blocking factor of the data file should be chosen using the same guidelines as for an MPE file. The blocking factor for the key file "should" impact performance (according to the KSAM manual). I haven't used KSAM enough to deduce a better method for selecting this number than that used by KSAM itself.

Each KSAM user has a separate, extra data segment for each KSAM file opened. Increasing the block size (or the number of key block buffers) will also increase the size of these extra data segments, and may have a dramatically bad effect on terminal response time. (The same thing is true of regular MPE files as well, but not of IMAGE.)

A6. INCREASE BLOCK SIZE OF IMAGE DATABASE

The dataset blocking factor of an IMAGE dataset is comparable to the blocking factor for MPE files and KSAM data files [48] [33] [07]. The largest block size allowed in a database is determined by the $CONTROL BLOCKMAX command in the schema file, with the default value being 512 words. Given a "target", IMAGE chooses the smallest block size within the target that mimimizes the amount of disc space for the dataset. Some datasets may be assigned a block size just below 512 words and others a block size just below 384 or 256 words. The buffers that are allocated in the DBCB, however, are all the size of the largest block in the database. Therefore, if you have a single dataset with a block size of 2048 words, you are limited to 12 or 13 DBCB buffers, even if your other blocks are only 1024 words long. This will not only decimate your terminal users, but may even degrade batch jobs which perform complex database transactions.

I did a large number of sequential extract tests on a dataset with 105,000 records, using a block size of 256 words, 512 words and 640 words (I wanted 1024, but IMAGE chose 640 instead). The 512-word blocks could be scanned 1.05 to 1.24 times faster than the 256-word blocks. Blocks of 640 words were slightly faster than 512. Given the problems for on-line users, it appears that the IMAGE default BLOCKMAX is optimum.

In these database extract tests, the "data rate" varied from a low of 56 seconds per 1000 sectors (using QUERY on 256-word blocks) to a high of 29 seconds per 1000 sectors (using * field list, dataset number and 512 word blocks). For a database extract task, the ideal data rate is 1 second per 1000 sectors. Why was our rate at least 29 times slower? There are two reasons: IMAGE needs CPU time to check and execute each intrinsic call, and IMAGE reads only one data block per revolution of the disc. See topic B2 below for a method of bypassing the CPU time of IMAGE, and topic B3 for a method of retrieving several IMAGE disc blocks per disc read (up to a full track of information).

A7. IMPROVE HASHING OF IMAGE MASTER DATASETS

The topic of "hashing" in IMAGE master datasets is well covered in other documents [15] [45] [07] [30] [b10] [34] [33]. I suggest that you run the contributed program DBLOADNG [07] at least once a month. Look for a high percentage of "inefficient pointers" in your master datasets (over 20% means you are often using more than one disc read to locate a record), and a high "elongation" (over 1.5 indicates a problem). Any improvement in the hashing of master datasets will benefit both batch jobs and on-line programs.

What can you do to improve hashing? Either increase the block size, reduce the record size, or expand the capacity to a higher prime number. How much will this improve the speed of your batch jobs? I have not yet run a controlled experiment. However, if you have a "regular" problem (dataset too full or block size too small), elimination of this problem "should" improve batch programs (that do DBFINDs or Mode-7 DBGETs) by about 1.25 times. If you have a "serious" problem (clustering of records) caused by a bad choice of key field type (and/or values), solving the problem (by converting to a different data type and/or restructuring the key values), will bring a dramatic improvement. I have seen cases where the elapsed time to do a single DBPUT was between 30 and 60 seconds (due to extensive clustering), when it should have been less than a second.

A8. DBLOAD DATABASE TO OPTIMIZE PRIMARY PATH

If you have an IMAGE detail dataset with 11 records per block, and one of the paths has an average of 11 records per chain, how many disc reads will you need to retrieve the entries on a given chain (not counting the hash to the master dataset)? Did you say "one read"? That would only be true if the dataset were perfectly "organized". In the real world, entries are added to and deleted from chains in a "random" fashion, creating holes that are reused for other chains. Over a period of time, the chains in a dataset tend to become disorganized (assuming puts/deletes). Therefore, it may take as many as 11 disc reads to retrieve all of the records on your "average" chain, or as few as one.

The contributed program DBLOADNG reports on the randomness in the chains of each master-detail path [07]. The last column in the DBLOADNG report is called "elongation" and is the critical value. If elongation equals 1.0, the chains for that path are as well-organized as is possible (given the blocking factor). If elongation equals 5.0, then those chains are unorganized; they are spread among five times more disc blocks than is theoretically necessary.

What can you do to reduce "elongation"? Very little, unless the disorganized path happens to be the PRIMARY PATH for the dataset. In that case, reloading the database will repack the entries so that elongation for that path is close to 1.0 [45] [34]. Of course, this only fixes the primary path, not the other paths into the dataset. Therefore, the primary path for a dataset should be the actively-used path with the longest average chain length (you cannot optimize a chain with only one entry!).

In order to "reload" a database, you must DBUNLOAD it to magnetic tape, erase the database with DBUTIL, and DBLOAD the data from the tape. I reloaded a database of 75,000 sectors (19.2 megabytes) which had undergone many DBPUTs and DBDELETEs without a reorganization. DBLOADNG showed that elongation for the largest dataset (capacity 200,000 entries) was 5.85 (very bad)! The DBUNLOAD/DBLOAD took two hours and, as predicted, elongation dropped to 1.17. The average blocks per chain was reduced from 8.3 to 2.3, over 5 times better.

I reloaded the database again to double the block size; this further reduced the blocks per chain to 1.6. If chained access is very frequent, the larger block size, combined with regular reloads, should improve batch speeds (as well as on-line response). I have not had time, however, to verify this hypothesis on an actual application.

A9. RELOAD SYSTEM TO REDUCE DISC FRAGMENTATION

Many sources suggest that you keep at least 20% free disc space (15,000 sectors minimum), and that you not let the number of entries in the free space tables get too large [23] [39] [41]. This is accomplished through a RELOAD from magnetic tape (a full backup). I am not convinced that this will make any difference in the speed of batch tasks. However, my personal experience does verify that disc fragmentation and lack of free space lead directly to increased System Failures. Since System Failures obviously reduce system throughput (on-line and batch) [06], I have no hesitation in suggesting that you do frequent RELOADs.

A10. CONTROL FILE PLACEMENT FOR "HEAD LOCALITY"

The favorite strategy in the optimizing literature is called "head locality" [48] [23] [15] [47] [41] [42] [07]. By placing files carefully on selected spindles, we ought to be able to keep the arm from moving back and forth so much. According to this theory, "head locality" plays a big factor in performance by reducing the average disc access time. It is frequently suggested, for example, that master datasets should be placed on different drives from their detail datasets, because the two are often accessed at the "same time".

The only tests of this technique that I have uncovered [38] [04] do not demonstrate a universal benefit from head locality. The first test attempted to improve sort performance by placing SORTSCR on specific drives. Unfortunately, sort times were actually slightly faster when all three files (input, output, scratch) were ON THE SAME DRIVE. In the second test, the user achieved a 6% improvement in his batch jobs after moving datasets to separate drives. This is not much improvement, considering the inconvenience and time required to obtain "head locality" (:STORE, :RESTORE, or copy from disc to disc). In the final test, the user improved file copies by 1.14 to 2.3 times when he copied from one drive to another [13]. However, a straight file copy is much simpler than most actual application programs.

I was concerned that these inconclusive findings might be "flukes", caused by faults in the experiments. Therefore, I devised several experiments to prove that "head locality" would pay big dividends. First, I reproduced the sort experiments referred to above; the sort times were essentially EQUIVALENT, regardless of where the input, output and scratch files were located. I tested file copies, extracting 10,000 records from a file with 100,000 records. When the files were on separate drives, the copy was 1.08 times faster than when they were on the same drive. This was not very encouraging. Finally, I tested test the effect of separating masters and details. I read 2000 records from a disc file and DBPUT them to a detail dataset that was indexed by one master dataset. When all three files involved were carefully isolated on three separate drives, the extract task ran slightly SLOWER than when all three files were moved to the same drive. This was a most suprising result.

The situation with masters and details may be complicated by the fact that DBPUT and DBDELETE must update the "userlabel" on each dataset (to record the number of available entries), and this label is always located at the beginning of the dataset. Perhaps the steps involved in achieving "head locality" (:STORE, :RESTORE) reduce the overall efficiency of the system by fragmenting the free space. These results show that MPE is seldom as simple as it seems. "Head locality" deserves more research, especially on MPE IV. Until that research is completed, I suggest that users discontinue efforts to obtain head locality.

B. SIMPLE CODING CHANGES

A "simple coding change" is one that changes the method of a program (the "how") without changing the purpose of the program (the "what"). The techniques proposed here can be implemented by mechanical changes to programs (i.e., code substitution). In fact, most of these ideas could easily be provided by the language subsystems automatically (e.g., by the COBOL compiler). Since the objectives of the application program are not modified, only programmer time, not system analyst time, is needed.

B1. USE MORE EFFICIENT PARAMETERS ON IMAGE CALLS

Another technique that is universally recommended is to code your IMAGE database calls with the "best" parameters [01] [45] [07] [30] [34] [etc.]. For the field list parameter (which fields within the dataset to process), the source with the most detailed testing [01] concluded that write access, plus the "@" list (all fields) is always the fastest. However, write access, plus a field list that is subsequently replaced by an "*", is almost as fast. Tests that I performed (doing DBGET extracts serially from a large dataset, varying the field list parameter) verified these results. When there were only four fields in the dataset, @ improved DBGET speed by 1.34 times, and "field-names-plus-*" improved DBGET by 1.33 times. Since the @ option can lead to program maintenance problems, I suggest the use of field names (only the fields you need) followed by an *. I also checked use of the dataset number in place of the dataset name, and found the improvement to be too small to be worth the trouble (861 seconds instead of 869).

Using the optimum parameters in IMAGE intrinsic calls improved the "data rate" for database extracts from 45 seconds per 1000 sectors of data (using field names) to 29 seconds per 1000 sectors (using *). In order to get closer to the "ideal" data rate of 1 second per 1000 sectors, we must use another approach which is far faster than any DBGET call: bypass the normal database overhead completely for high-volume serial access by reading the data directly using NOBUF. This technique will be explored next.

B2. USE NOBUF ACCESS TO REDUCE CPU TIME

The most powerful technique for making batch jobs faster is NOBUF access. Normally, files are accessed through intermediate buffers provided by MPE, KSAM or IMAGE (as described above under A1-A6). However, in my 1978 paper on optimizing [15], I described how you can disable buffering using the NOBUF bit (in FOPEN) and read physical blocks directly into your data stack. When you use NOBUF, it is your responsibility to "deblock" the logical records from the physical blocks. At the time, I was discussing the application of NOBUF to on-line problems; specifically, how to write a program development editor that would not be a drain on the system. The resulting program, called QEDIT, used NOBUF (and other techniques) to cut the load of editing at least in half, while still providing editing speeds that were 2 to 12 times faster than EDIT/3000.

Encouraged by the success of NOBUF in QEDIT, I wrote another program in 1978, called SUPRSORT, that used NOBUF access for file copies and file sorts. This program proved to be 2 to 10 times faster than FCOPY and 2 to 5 times faster than SORT/3000, primarily due to use of NOBUF (see topic B3 below for more information). NOBUF works so well because it REDUCES CPU TIME DRAMATICALLY, by eliminating calls to the file system (or KSAM or IMAGE) and by replacing the general-purpose "deblocking" code of those systems with specialized deblocking code written by the user.

Since 1978, the "secret" of NOBUF has spread from user to user and has been explained in many papers [09] [08] [29] [11]. One of Hewlett-Packard's System Engineers wrote (and contributed) FASTIO, a set of general-purpose deblocking routines that allow the COBOL programmer to benefit from NOBUF easily [44]. Users then reported results such as a reduction in CPU time from 76 seconds to 10 seconds, a reduction in elapsed time from 7.5 minutes to 1.3 minutes, and scan rates of 60,000 records per second instead of 8500 [03] [02].

FASTIO was designed to accelerate sequential access to files; but, an enterprising vendor wrote another set of deblocking routines (called BREAD [26]), that provides both random and sequential NOBUF access. BREAD is callable from BASIC, as well as COBOL, SPL, and FORTRAN and it has better documentation than you can expect from a contributed routine like FASTIO. Using either FASTIO or BREAD, a batch application file can access disc files 3 to 20 times faster than by using the standard READ/WRITE statements provided by the programming language.

With the 1918 release of MPE in 1980, Hewlett-Packard upgraded SORT/3000 to use NOBUF. (There can be problems with NOBUF if you are not careful. Witness the failure of the 1918 sort release to work with card input files or line printer output files.) With these enhancements, SORT/3000 is now slightly faster than SUPRSORT at sorting MPE disc files (up to 10%).

A user who ran timing tests on a file of 50,000 records (64 words long, 20 records per block), found that a NOBUF sort (using either SUPRSORT or the 1918 version of SORT/3000) ran slightly faster than a simple FCOPY of the same file (he also found that SUPRSORT copied this file four times faster than FCOPY) [36]. In other words, the MPE deblocking of 50,000 records took the same amount of time as a sort of the same records. At another site, a junior programmer wrote a set of deblocking routines for magnetic tape blocks (slightly more complicated than disc blocks). After modifying several report programs that scanned tapes, he was able to achieve a ten-fold reduction in CPU time and two-times-faster elapsed times.

Since NOBUF works by reducing CPU time, any changes in MPE to make the file system faster will reduce the "perceived" benefit of NOBUF. This has happened with MPE IV and the Series 44. One of the test sites for this new Hewlett-Packard product found that FCOPY runs twice as fast as it did on the Series III with MPE III [36]. The NOBUF copy was now only 2 times faster than FCOPY, not 4 times.

NOBUF can also be applied to KSAM files, but with some difficulty (there are problems with deleted records and the end-of-file). SUPRSORT supports NOBUF access to KSAM files, while SORT/3000 does not. In a sort of 12,444 records (-80, 16, F, ASCII; KEY=B,40,20), SUPRSORT took 1.7 minutes and SORT/3000 took 5.6 minutes, an improvement of 3.3 times. Since SORT/3000 uses the default access to KSAM files, KSAM must use the key file to put the data records into primary key sequence, after which SORT/3000 sorts them again, into the final desired order. SUPRSORT reads the records in chronological sequence, instead of in primary key sequence. For a KSAM file that has been loaded randomly over time, primary key sequence can require a great deal of disc head movement to retrieve the records.

NOBUF access can also be applied to IMAGE datasets, but SUPRSORT and ADAGER (Adapter/Manager for IMAGE [33]) are the only software tools that I know of which do this. SUPRSORT, for example, retrieves records from an IMAGE dataset by reading the data blocks directly from the disc, rather than calling the DBGET intrinsic for each record. This results in a dramatic savings of CPU time (and elapsed time).

In 1980, I added a generalized selection capability to SUPRSORT (i.e., >IF AMOUNT>100000 AND ORDSTAT<>"X" OR DATE>801030) and changed the name of the software product to SUPRTOOL. In one comparison test, I used SUPRTOOL to select 1033 records from a dataset with 105,504 current entries. When the dataset had 256-word blocks, SUPRTOOL took 222 seconds to do this task, 2.9 times faster than the best SPL program (using DBGET) and 4.8 times faster than QUERY/3000. The effective data rate in this experiment was 11.62 seconds per 1000 sectors of data scanned, versus 34 seconds with the best SPL program. This gain is strictly due to savings in CPU time, since both cases read a single data block per disc revolution.

When the database was reloaded with 512-word blocks, SUPRTOOL took only 110 seconds (twice as fast) and was 4.8 times faster than the SPL program (7.4 times faster than QUERY/3000). The larger block size improved SUPRTOOL's data rate from 11 seconds per 1000 sectors to 6 seconds. (Warning: before you reload your databases with larger blocksizes, read the next section! With the IMAGE database, you can enjoy the benefits of large disc blocks, while still retaining small blocks for your on-line users.)

B3. USE NOBUF ACCESS AND TRANSFER SEVERAL BLOCKS AT ONCE

When you open a file with NOBUF, MPE allows you to transfer one OR MORE blocks on each call. If NOBUF with one block per transfer is good (see previous section), would several contiguous blocks per transfer be even better? Sometimes it is, and sometimes it isn't. I ran a test with SUPRTOOL to demonstrate this phenomenon. From an IMAGE dataset with 251-word blocks (117 words per record, two records per block, 15,255 entries in the dataset), I extracted all of the entries that contained a customer number starting with "X". SUPRTOOL read one disc block at a time and the task took 176 seconds (65 CPU seconds). When I forced SUPRTOOL to read 16 blocks at a time (using DEBUG), the task took 1.44 times LONGER (189 seconds), but used less CPU time (only 45 seconds)!

How can 16 blocks per transfer be worse than one block per transfer? The CPU time was reduced, as we would expect, because there were 16 times fewer calls to the FREAD intrinsic; but the elapsed time increased by 1.44 times instead of decreasing. The answer lies in the BLOCK SIZE: 251 words. Since each block MUST START ON A SECTOR BOUNDARY, a 251-word block must have 128 data words in the first sector and 123 data words in the second sector. That leaves 5 WASTED WORDS at the end of each block. When SUPRTOOL asked MPE to read 16 contiguous blocks from the disc, MPE had to remove those 5 extra words at the end of each block. Since the disc controller is not smart (or fast) enough to perform this chore, the only way that MPE can remove the words is to issue a separate read request for every. Thus, a 16-block read takes at least 15 full revolutions of the disc to complete. When SUPRTOOL asked for one block at a time, each read request took about one-half a revolution to complete (depending upon where the head was when the read occurred). When there are unused words at the end of a block, reading one block at a time is the fastest method.

However, when there are no wasted words at the end of each block, multi-block transfers allow serial processing at rates which reach the theoretical limits of the HP 3000. For example, in copying a file of 32 word records, with 4 records per block, FCOPY has a "copy" speed of 57 seconds per 1000 sectors. SUPRTOOL, using transfers of 4096 words, has a measured copy speed of 1.6 seconds per 1000 sectors, faster than the hypothetical limit of 2 seconds. Of the total 1.6 seconds, 1.46 seconds are spent in either the FREAD or FWRITE intrinsics. The only way that SUPRTOOL can be faster than the "speed limit", is if the actual latency per transfer is less than the average (11.1 ms., or 1/2 revolution). See Section II for the detailed calculations.

Prior to the Athena release of IMAGE (2011), databases were always created with "odd" block sizes, such as 251 words. But, if you build (or rebuild and reload) a database under the 2011 release (or later), IMAGE will round up the block size of each dataset to the next sector boundary (128 words). This was done to allow the DBUTIL program to use multi-block writes to erase a database.

What are the results if I reload my database under 2011 IMAGE and ask SUPRTOOL to scan the database using varying numbers of blocks at a time? In the previous section, I reported an extract job that took 222 seconds to perform, reading one 256-word block at a time from the dataset. Here are the times for the same job, reading 1 to 32 blocks at a time (block size is exactly 256 words):

Number of Blocks:	1	2	4	8	16	32
Words per Read:	256	512	1024	2048	4096	8192
Elapsed Time (s):	223	116	115	89	75	69
Times Faster:	---	1.9	1.9	2.5	3.0	3.2

These times are more like it -- up to 3 times faster. The conclusion that you should reach is: ALWAYS PICK A COMBINATION OF RECORDS IZE AND BLOCKING FACTOR WHICH PRODUCES A BLOCK SIZE EVENLY DIVISIBLE BY 128 WORDS [11], even if you must "waste" a few words per record.

If you follow this rule, you can achieve data processing speeds that are close to the theoretical hardware limits (1 second per 1000 sectors). SUPRTOOL, for example, performs selective data extracts on "even" block sizes at rates up to 3.6 seconds per 1000 sectors. Straight copy operations, without selection, are even faster: 1.6 seconds per 1000 sectors. When doing selective extracts, SUPRTOOL spends .75 seconds per 1000 sectors in the FREAD intrinsic. That is faster than our target speed limit. It shows, once again, that CPU time has the power to stretch out the elapsed time for any task. SUPRTOOL uses 2.85 seconds per 1000 sectors to evaluate which records to select (and to move them to output buffers) and only .75 seconds per 1000 sectors to read and write the data blocks.

Another advantage of using NOBUF reads of the database (as SUPRTOOL does), instead of calling DBGET, is that a serial search does not disturb the shared "buffer pool" in the global DBCB extra data segment (see topic A3 above). If there are other users accessing the database, a batch program that makes 100,000 calls to DBGET in a short time will "flush" the buffers that are allocated to other users. NOBUF access bypasses these buffers completely.

With an "even" block size, the speed of multi-block NOBUF is exactly as fast as single-block NOBUF using very large blocks. Therefore, you can keep your actual block sizes small (but they must be divisible by 128 words with no remainder) to optimize on-line access, but not too small (I suggest 512 to 1024 words) to optimize cases where you choose to use buffered access instead of NOBUF access. Since IMAGE now rounds up blocks for you automatically, I suggest that you use the default IMAGE block size (512 words), except in cases where a record size does not block well into 512 words (see A3 and A6 above). If you have an existing database with "odd" block sizes, you can convert to "even" blocks either by building a new database and reloading from tape (DBLOAD), or by running the new REBLOCK function of ADAGER/3000. For magnetic tape files, I suggest a very large block size (4000 to

8000 words) and NOBUF access, one block at a time. Multi-block tape transfers seem to crash some releases of MPE.

B4. USE DBCONTROL TO DEFER DISC WRITES BY IMAGE

Normally, after the completion of each IMAGE intrinsic call, the IMAGE database system flushes all "dirty" blocks in the buffer pool back to the disc. This is one of the reasons IMAGE data structures are so reliable: the time window during which they are inconsistent is very small.

The DBCONTROL intrinsic provides a way of disabling the automatic posting of dirty blocks. It has been suggested that DBCONTROL be used to improve large batch jobs [34]. The DBLOAD utility, for example, uses this feature to accelerate the loading of databases from tape. However, if the system should crash while you are in "deferred-output" mode, you must RESTORE your database from a backup tape. Otherwise, there could be serious logical and structural inconsistencies in your database.

One user test of DBCONTROL found a reduction in elapsed time from 21 minutes to 16 minutes for one application, and from 22 minutes to 8 minutes for another [04]. These are improvements of 1.3 times, and 2.75 times, respectively. It is likely that tasks requiring a large number of puts and deletes (relative to gets and updates) will show the biggest improvement. Designers are often encouraged to avoid doing on-line deletes from the database by flagging "dead" records and deleting them with a batch program [45]. Such a batch maintenance program is a good candidate for DBCONTROL, since it can easily be scheduled to run after a database backup (essential to avoid losing your database without any way to recover).

When I upgraded my SUPRSORT utility into SUPRTOOL, I added the ability to write the output records to an IMAGE dataset (i.e., do a DBPUT). I also included a command to enable "deferred-output" mode through DBCONTROL. Using these features, I loaded 2551 records into a detail dataset that had a single key field. In normal mode ("complete-output"), this task took 333 seconds; in deferred mode, it took 156 seconds, an improvement of 2.13 times. (See topic A3 above for related ideas.)

The dramatic difference between a DBPUT and an FWRITE to a sequential file can be seen by calculating a "data rate" for DBPUT. In the test mentioned above, with a single path to update, DBPUT had a rate of 203 seconds per 1000 sectors of data in "complete-output" mode (default) and 94 seconds per 1000 sectors in "deferred-output" mode. This compares with a rate of 1 to 2 seconds per 1000 sectors for NOBUF, multi-block transfers using FREAD and FWRITE. The reason DBPUT takes so much longer is that it must do random (i.e., "hashed") reads from the master dataset (about 1570 reads per 1000 sectors, plus 1570 writes, times .03 seconds per transfer, equals 94 seconds).

B5. RECOMPILE COBOL PROGRAMS WITH COBOL II COMPILER

Hewlett-Packard has run a large number of tests to measure the performance of COBOL II, relative to COBOL/3000 (also known as COBOL I or COBOL 68). In analyzing their results [22], I have noticed a number of interesting points. Batch jobs should run a bit faster (1.1 to 3.0 times faster), depending upon how much I/O they do (compute-bound programs are improved the most). On-line programs use less CPU time, but users do not see any improvement in response time (since on-line programs spend most of their time in IMAGE and V/3000).

The big surprise is that data stack sizes are often reduced by 25-60% due to the elimination of RUNNING-PICTURE and PARAGRAPH-RETURN tables. For on-line users, this "should" mean you can run more terminals with the same memory (unless you are already disc-bound). For batch programs, this savings in stack size can be used to eliminate disc transfers by copying tables (i.e., small master datasets) from the disc into the stack. The only problem with COBOL II is getting a version that works right. The "Cheetah" version appears to have most of the bugs worked out (HP32233X.00.02C?).

B6. REWRITE SOME CODE IN SPL/3000

Rewriting COBOL programs in SPL is a technique that I have recommended for several years [17]. However, this does not apply to batch programs. Without proper training materials and programming guidelines [16], you could actually make things slower by rewriting COBOL code in SPL. I can give two examples that I have seen in actual user sites.

First, by converting to SPL, you lose the built-in data conversion power of the COBOL language. You you must code explicit conversions between ASCII and BINARY formats. If you use the intrinsics provided with MPE to perform this task, your batch tasks will take longer in SPL than in COBOL. The MPE intrinsics take 3 times longer to execute than substitute routines that I wrote in SPL. This is probably due to the enabling and disabling of the error trap mechainisms which must be done in every MPE intrinsic.

Second, since SPL has no "packed decimal" data type, you may be forced to write (or acquire) SPL subroutines to do packed-addition, packed-subtraction, etc. Because each addition (subtraction, etc.) now requires a procedure call, these tasks will take significantly longer than they do in COBOL (which generates in-line code). These considerations are even more applicable to COBOL II, because it consumes less CPU time than COBOL 68.

In summary, while it makes sense to rewrite MPE intrinsics (such as ASCII) in SPL and it may make sense to rewrite high-usage COBOL subroutines in SPL (i.e., checkdigit, edit validations), it does not make sense to rewrite batch reports in SPL. SPL should be reserved for on-line optimization, for specialized support routines (access to MPE capabilities and hardware features), compute-bound subroutines which are called frequently, and utility software. Many of the optimizing techniques mentioned in this paper be used most easily in SPL, but you do not need to hire SPL programmers to take advantage of them. You can buy the optimizing tools, already coded, debugged and documented, from a software vendor. (If you want to develop SPL expertise, see [16] for suggestions.) Production batch jobs should be written in COBOL II (or RPG or FORTRAN), unless there is some compelling reason to write them in SPL.

B7. BYPASS THE FORTRAN FORMATTER (ETC.)

Each implementation of a programming language has problems. In FORTRAN/3000, the notorious trouble spot is the "formatter" (processing of FORMAT statements) [43] [40]. In a bizarre example sent to me by a customer [43], performance of a particular program varied by a factor of 58 times, depending upon how the READ statements were coded. The differences were totally a matter of CPU time spent in the formatter (the program read 4600 records, 2040 bytes each, blocked 4, using default buffering).

```
Case 1.  4632.8 CPU seconds (over 1 hour to read 4600 records!)
              3  FORMAT ( 2040 ( A1 ) )
                 CHARACTER A2040*1(2040)
                 READ(10,3,END=2) A2040
Case 2.  108.91 CPU seconds (42.5 times faster)
              3  FORMAT ( 10 ( A204 ) )
                 CHARACTER A2040*204(10)
                 READ(10,3,END=2) A2040
Case 3.  79.16 CPU seconds (58.5 times faster, without format)
                 CHARACTER A2040*1(2040)
                 READ(10,END=2) A2040
```

Apparently, the compiler generates 2040 calls to the formatter for each record in Case 1. Horror stories like this prove that CPU time matters. In batch processing, a small inefficiency in the innermost loop can turn into a big increase in elapsed time, when it is repeated a million times (as above). The programmer should learn the constructs to avoid in his language, whether it is FORTRAN, COBOL [22] or RPG [46].

B8. ELIMINATE UNNEEDED DATA CONVERSIONS

One area where standard programming languages usually have performance pitfalls is in data types. Standard languages must "map" their logical data types into the available hardware data types of the HP 3000. When the types match closely, performance is good. When they do not match, the compilers must generate "fixup" code and performance can be bad. In COBOL, for example, numeric data fields can be either signed (S9(4)) or unsigned (9(4)), but most of the HP 3000 hardware data types are only signed. Therefore, according to Hewlett-Packard [22], "using signed instead of unsigned data avoids the need for computing the absolute value of a result after it is obtained. This affects COMP-3 and DISPLAY items more than COMP, and can result in a moderate savings in execution time."

Many implicit data conversions occur in standard programming languages, especially (but not exclusively) when you mix data types in expressions. In COBOL 68, it is rumored that COBOL, all COMPUTE statements are done using packed-decimal arithmetic, regardless of the original data types. In a benchmark that I read about, a COBOL 68 program was reduced from 10 minutes to 12 seconds by changing "ADD 5 TO SUM" to "ADD FIVE TO SUM", with "FIVE" defined in the data division as a numeric field with the same data type as SUM.

In FORTRAN, mixed-mode expressions are discouraged by S.E.'s [31] [40], because conversions occur. In RPG, users are encouraged to avoid using numeric display or binary field types, since RPG must convert them to packed-decimal to perform arithmetic [46].

Saving a few milliseconds may seem like a minor matter. For ON-LINE programs, it IS minor, because on-line programs seldom use enough CPU time in an entire day to be justify optimizing them. But, if something is repeated often enough, the inefficiencies in CPU time start to add up to a significant amount of elapsed time. That is why NOBUF (in place of MPE buffering or the interface of DBGET) makes batch jobs run faster.

In summary, the potential exists in many applications for major improvements through more detailed knowledge of the machine-dependent features of programming languages. However, I strongly suggest that you start the habit of verifying each suggestion before implementing it. This is not difficult to do. Start writing test-case programs, with subroutines to measure the CPU time consumed. In one example that I saw recently [04], the user had followed advice to convert data items to "binary" (from what?). Instead of running faster, the elapsed time increased from 21 minutes to 31 minutes. You should not put total faith in everything you read. Compilers change. MPE changes. Hardware changes. Keeping up with these changes is a continuing process.

B9. REDUCE NUMBER OF "OPENS"

I once optimized a program that took an entire weekend to prepare and print monthly bills for 18,000 customers. The user client needed the program fast enough to produce 400,000 bills per month. One of the most blatant problems in the program was the result of last minute specification changes. An "audit" database was added to the application. The transaction subprogram was modified to create an audit copy of each transaction. To do so, the programmer opened the audit database, did a DBPUT, then closed the database. This meant a minimum of 18,000 extra calls to DBOPEN and DBCLOSE. I changed the program to open the audit database once in the mainline, and to pass in the database name to the transaction subprogram. This one change cut the elapsed time by 15 hours.

Sloppiness of this kind often creeps into batch programs when changes are made at the last minute, and schedule dates are not extended. Testing seldom catches programming errors of this kind, because the the volume of test transactions is usually too small. Other operations that should not be performed 18,000 times are FOPEN, RENAME and SORT.

As I said in my earlier paper [15], one method of optimizing is to focus on the common events. Start with the tasks that are repeated the most times. Ascertain that they are efficient, before worrying about tasks that only occur one tenth as often. You can use the MPE log statistics to detect files which are heavily accessed (or opened too often) [42], and to select programs for review which run often (or for too long).

B10. INCREASE SIZE OF CODE/DATA SEGMENTS

In reading about optimizing, you will see suggestions on segmenting programs into small (4K word) code segments. Code segmentation reduces the impact of a program on the rest of the system, but it DOES NOT MAKE THE PROGRAM RUN FASTER. Since a "local PCAL" (calling a routine in the same code segment) takes 6.1 microseconds, and an "external PCAL" takes at least 14.9 microseconds (34 milliseconds, or more, if the code segment desired is not present in memory), segmenting your program actually makes it run slower. Stand-alone batch programs may run slightly faster with fewer, but larger, code segments. (You should still put routines that call each other in the same segment). I tested this hypothesis by calling data conversion routines 64,000 times. When the routines resided in the same code segment as the calls, the execution time was 1.06 times faster than when the routines were in a separate code segment.

My general point is this: avoid being trapped into inflexible thinking by the constant emphasis upon on-line programming in most manuals and optimizing documents.

Another common prescription for bad performance is to shrink your data stack using the ZSIZE intrinsic [15] [39] [22] [18] [32]. If your stack is larger than necessary for long periods of time, it impacts the response time of other on-line users. It takes only a few milliseconds to contract your stack to give back the unneeded space. But, it takes considerable time to expand your stack again (which is done automatically when you need the space). Stack expansion requires a disc write and a disc read [23]. In a batch program, if you contract your stack after each transaction, you may have a very slow program, if MPE expands your stack again for the next transaction.

When an on-line application program is coded to shrink the stack dynamically, the savings are multiplied by the number of users running the program (1, 10, or 30). Big improvements in response time are possible through this technique because large amounts of main memory are freed for other users. Now many copies of a batch job run at once? Usually only one copy. Reducing the size of your batch stack does not increase the speed of your batch program; it only contributes marginally to the response time of other jobs. Given the rapid increase in main memory capacity of the HP 3000, I suggest that batch stacks be increased whenever that will lead to a decrease in CPU time or disc accesses.

For example, consider the practice followed by many HP 3000 programmers of eliminating global data storage by making it local. Instead of making data "dynamic", consider making more of it "static" (global, common, own, subprogram). Allocating local storage "dynamically" each time you enter a subprogram conserves stack space, but it takes CPU time to do the allocation.

The ideas in this section are controversial, and, until sufficient tests have been performed to verify their impact on throughput, they should be treated as "promising", but unverified, proposals. Users are encouraged to do their own performance measurements in this area.

C. CHANGES TO APPLICATION DESIGN

When all possible improvements have been made to data storage methods and programing efficiency, the only area left for attention is the logic of the system. Although design changes are the most expensive to implement, they also provide the greatest potential for gain. If you can eliminate the need to run a certain report, your percentage gain in throughput is infinite.

C1. USE DATA STACK SPACE INSTEAD OF DISC ACCESS

In the previous discussion (topic B10), I suggested that you look for ways to trade off a larger data stack for fewer disc accesses. For example, if a program uses a small temporary file as a buffer space, the entire "file" could be kept in the stack as a large array. (This will only improve performance if the file is accessed many times per transaction.) One of my clients has an invoicing program that uses a temporary disc file to hold the line items of each order (after they are copied in from the database). The number of line items per order varies from 1 to 50, with an average of 3. According to the contributed program FILERPT [42], this file is the second most heavily accessed file on the system. If I add subroutines to the program to simulate the temporary file using an array in the stack, I should be detect a significant decrease in elapsed time for a given invoice run.

Another candidate for a place in the data stack is a "lookup table" for validating data fields. In many batch processing tasks, each transaction record has one or more fields that must be checked against a list of valid values in an IMAGE master dataset. If the number of valid values is small enough, you can move the entire master dataset into your stack (4000 to 16,000 words). This strategy would not make sense for an on-line program, because there are too many copies of the stack that must reside in memory at the same time. Also, on-line programs seldom do enough table lookups per minute to justify each user having a copy of the entire dataset in memory. For a batch task, however, there is only one copy of the stack, and there may easily be 100,000 table lookups in one run.

How much time could the in-stack table save? A computed DBGET takes about 75 milliseconds (including one or two disc reads). For a batch job to do 100,000 DBGETs, it takes about 7500 seconds, or 125 minutes, or 2 hours. Most of this two hours would probably be eliminated by an in-stack table (how much depends upon how long it takes to search the table in the stack).

C2. USE EXTRA DATA SEGMENTS INSTEAD OF DISC ACCESS

When you have exhausted the 64,000 bytes that are possible in your data stack (copying tables and temporary files into the stack, as proposed above in topic C1), there is one more resource that you can tap before you use a disc file: Extra Data Segments (XDS). An XDS is a "chunk" of data space that belongs to your program, but is swapped in and out of memory by MPE, independently of your data stack. Your program accesses data within the XDS via the DMOVIN and DMOVOUT intrinsics. The size of an XDS can vary from a few words (why bother?) to 64,000 bytes (the maximum size is a system configuration value). Each program can create and access up to 255 XDS (also limited by system configuration).

The HP 3000 has a lot of main memory to work with, so why not use it all for a stand-alone batch job? If you have one megabyte of real memory, of which MPE uses 128k and your program needs 256k (for stack, IMAGE DBCB segments, code segments, etc.), you still have 655,360 bytes left. If you limit your XDS size to 16,000 bytes (8000 words, an easy size for MPE to swap), you can have 40 XDS and still not be swapped (this is all theory, subject to experimental verification).

According to one article, an XDS is 25% to 900% faster than a disc file [24]. Of course, it is not as fast as data in your stack, so use the stack for the most frequently accessed data. Access to a subscripted variable in COBOL takes about 60 microseconds, while access to an extra data segment takes 1.3 milliseconds [25]. Access to a buffered MPE file takes 2.9 ms. if the record you want is in the buffers, and 40 ms. (or more) if a disc transfer must be done.

In 1980, David Greer of my staff did a research project to investigate the substitution of main memory resources for disc access resources [19]: "One answer is a memory file. This is a file that looks, and is accessed, as if it were on the disc, but which actually exists in memory. The savings could be great if the number of disc accesses is very high, but the cost should be low, since systems are all tending towards more memory as memory costs continue to drop. The memory file should be implemented with little or no knowledge of the applications programmer. One way to do this would be with a special MPE device class (such as MEMORY). The first time a file with this class is opened, it would be read into main memory. All accesses to the file from then on would be memory to memory, rather than disc to memory. When the file is closed, it would be copied back to the disc. A prototype memory file system was implemented (for exclusive-access files only), using a number of extra data segments, including one to hold control information. All file system calls from the application program were intercepted by interface routines that emulated the MPE file system. Test programs showed that the MPE file system was faster than this double-XDS system UNTIL the MPE file buffers were exhausted (default = two buffers). After that point, the average times for MPE files increased 4 times, while the memory file system times remained steady. The results show that a memory file system could provide a major improvement over the regular file system, for small files that are heavily accessed."

The contributed library contains TBPROC [37], a set of SPL routines that allows a COBOL program to maintain a table in an XDS (routines are provided to define a table, add entries to a table, update entries, sort the table and retrieve entries from the table by key value or relative index). TBPROC allows you to define the maximum number of table entries, the byte length of each entry, the offset of the key value and the byte length of the key value. The author of TBPROC wrote it to hold small IMAGE master datasets, as proposed above. He feels that this change reduced the elapsed time of a large production job by one-third [36], but cannot be certain (since other factors were changed at the same time). I would like to enhance TBPROC to allow for an optional buffer in the stack. The user-supplied buffer would be used instead of the XDS, when the table was small enough. If the table overflowed this space, it would be stored in an XDS and the in-stack srace would be used to optimize access to the XDS.

Another contribution to the library, ARHND, uses up to 64 XDS of 16,000 bytes each to provide one megabyte of virtual array space for a program. Where TBPROC is COBOL-oriented and provides table lookup by key-value, ARHND is FORTRAN-oriented and provides up to 13 virtual arrays (of single or double integers). The size of the arrays can be varied dynamically, and they can be one or two dimensions. Routines are provided to allow the program to emulate sequential files using a virtual array. Using ARHND, it appears that a single FORTRAN program could consume the entire resources of an HP 3000.

There is a certain unavoidable overhead in the DMOVIN and DMOVOUT intrinsics, because they are part of the MPE operating system (if MPE had a NOOP intrinsic, it would take at least ONE millisecond). For maximum speed of access to an XDS, you can use the hardware instructions that move between data segments [14]. Of course, this requires privileged mode and careful programming on your part. Another option is to use EXCHANGEDB and operate in "split-stack" mode. This also requires privileged mode. The

BASIC-callable version of the "BREAD" NOBUF routines uses "split-stack" [13], because the BASIC Interpreter claims total control over data space allocated in the stack. I mention these options only for completeness; I have never seen them put to use in an actual, commercial batch program. They might be very useful, however, in software products (such as [13]) which are designed to support and optimize production batch programs.

One thing about XDS usage worries me. An XDS can be swapped out by MPE if the memory space is required for a higher priority process. When the program next references the XDS, MPE must swap it back into memory again. It might be faster to use NOBUF disc files and manage buffers in your stack (assigning buffers dynamically to the "active" blocks; that way, you control the level of swapping that occurs.

When Hewlett-Packard implemented APL on the HP 3000, they needed a large virtual data area for the APL workspace. Rather than use XDS, they created a new data structure called a "Virtual Array". A VIRTUAL ARRAY can be declared in SPL, but requires the APL firmware at run-time. The APL virtual array is stored on the disc as a series of fixed-length "pages", and the APL interpreter allocates a certain amount of in-stack space as a buffer pool (to hold some of the pages). When the interpeter references a virtual array, the special firmware instructions check to see if the required page is in the buffer pool. If it is, the virtual data is accessed immediately. If it is not, the "least recently used" page is swapped out and the page containing the desired data is swapped in.

I would like to see a controlled experiment matching the ARHND-type of virtual array against a stack-NOBUF-file type of virtual array. This entire area has tremendous potential for optimization of batch programs, but needs a great deal more research.

C3. SAVE DATA OR POINTER FOR RE-USE ONCE RETRIEVED

Having retrieved a specific record(s) from a detail dataset (using 5-10 IMAGE calls), you should save the record number (found in the STATUS array) [45]. It can be used for a DIRECTED-GET later, when you are ready to update or delete the record. Does each subprogram of your COBOL program re-retrieve the customer-master record from the database when it needs it? You should only retrieve each customer-master record once, store it in a global data division, and pass it to the subprograms as a parameter? Does your program validate transaction fields by doing lookups in the database? You should save the key value that was tested for the last transaction in a global place, and check there before checking the database (or keep a working set). IMAGE adds entries to unsorted detail-dataset chains in chronological order. Therefore, if the entries that you need are more likely to be recent than old, you should read up the chain (mode 6), instead of down the chain (mode 5).

A program spends most of its time doing "work" to convert dispersed, raw data into accessible, organized information. Once the program has converted the data into accessible and/or organized information, the program should save that information, if there is any chance that the same information will be needed again soon. Otherwise, it must repeat the work later.

C4. REPLACE MULTIPLE PUTS/DELETES WITH UPDATES

DBPUT and DBDELETE take 5 to 20 times longer per call than DBUPDATE, because DBUPDATE is not allowed to change "critical" fields (search items or sort items). That is, it cannot make "structural" changes to the data. For each path path into or out of a dataset, DBPUT and DBDELETE must update the chains that link entries with the same key value (in a detail dataset) and the chain heads (in the master datasets). This task requires disc accesses and CPU time. The more paths there are into an entry, the longer it takes. That is why optimizing papers often recommend that on-line programs merely flag "dead" records and let them be deleted in batch [45]. Similarly, if you are not changing any critical fields, you should always use DBUPDATE instead of deleting the entry and adding it again [34].

One way to replace PUT/DELETE with UPDATE is to eliminate paths (see topic C5 below). Another way is to merge several independent entries (that must be PUT and DELETEd) into a single entry, with a "column" for each individual piece of data. For example, instead of creating an entry for each month of the year (12 PUTs and 12 DELETEs per year), use one entry for the entire year (1 PUT, 1 DELETE), and UPDATE the entry when you need to record the value for a particular month (12 UPDATEs).

I tried this approach with one client's application. The previous consultant had designed a detail dataset to hold billing transactions, indexed by customer number and sorted by date. Customers were normally billed once a month, for a recurring fixed amount (i.e., $5.00 per month). When they paid, another detail entry was created to show the payment. In order to examine the status of an account, it was necessary to retrieve all of the entries on the chain (24 entries minimum for a year). But, the transactions were predictable (billed $5, paid $5). We replaced the individual transactions with a single entry that showed the amount to be billed per month, and had places to be "checked off" when each month was billed and then paid. This replaced 24 PUTs and 24 DELETEs (you have to get rid of those transactions someday), with 1 PUT and 24 UPDATEs. The impact on batch throughput (as well as on-line response) was impressive; and, this approach was actually closer to the way the company had kept track of its customers before the computer was installed.

C5. CONSIDER SERIAL SEARCH INSTEAD OF CHAINED ACCESS

On-line functions should only do chained (or keyed) reads from the database. They should never do serial scans of an entire dataset (unless the dataset is very small). This is what search items are for: to provide quick response to inquiries.

For a batch program, "response time" (i.e., the time it takes to retrieve a subset of entries from a dataset) does not have to be immediate, it only has to be reasonable. An application may actually run faster (overall) if you eliminate a search item that is accessed exclusively in batch, and use a serial scan instead [45]. The time to PUT and DELETE those records will be reduced, and you may be able to do an UPDATE instead of a DELETE/PUT (if the field to be updated is the deleted path or the sort item). Some of the search items that I have eliminated from datasets are "division number" (where there are only four divisions), "transaction date", "transaction month", and "salesman" (with only 10 salesmen).

In fact, if a given path has only a few unique chains and each chain is very long (i.e., only a few values for that field), a serial scan MAY ACTUALLY BE FASTER than a chained read. This inversion of logic is most likely to occur in datasets that have had many DELETEs and PUTs since the last DBLOAD reorganization. Such activity tends to spread the entries that are on the same "logical" chain into different physical disc blocks (because IMAGE reuses deleted space for new entries). Each chained read can, therefore, require a separate disc read. Serial reads, on the other hand, only do a disc read once for each N entries (N=blockfactor).

In one test that I ran, chained reads took 16 milliseconds each, and serial reads (using DBGET with *) took only 5 milliseconds each. In this case, the chained retrieval will only be faster if there are four or more unique key values (each chain has less than one quarter of the entries). With SUPRTOOL, the time per serial read is much faster, only .63 milliseconds. As a result, SUPRTOOL will be faster than chained access for any path that has less than 25 unique chains.

C6. ISOLATE DATA BY FREQUENCY OF ACCESS

Do you plow through the transactions for an entire year every night, in order to produce an audit report of the activity for the day? Why not put the day's work into a separate dataset? After producing the audit report for the day (which should finish 30 to 400 times faster) and re-validating the transactions (to catch bugs in the on-line programs), move the entries into a month-to-date dataset and delete them from the daily dataset. After month-end closeoff, move the month-to-date entries to the year-to-date dataset. Another benefit of this approach is that each dataset can have a different type of access (different number of search items and sort fields). Finally, after the end of the fiscal year, you can copy the year-to-date dataset to a disc file (never throw away good transactions) and clear out the dataset for the next year. The archive disc file can easily be reported from, since it has the same format as the year-to-date dataset. It can also be stored to tape if there is no immediate call for it. (SUPRTOOL has commands to perform most of these extract/copy operations, and with excellent speed.)

In the previous discussion (topic C5), I suggested eliminating search items that have less than 4 unique values (less than 25, if you have SUPRTOOL). Now I am suggesting that you create separate datasets to isolate entries, when the distinguishing field has only 3 to 5 active values (such as range of date = current day, current month, current year, or other year). "Isolation" reduces the number of physical entries that your serial search programs must read. They need only look in the relevant datasets.

C7. KEEP RUNNING TOTALS IN DATABASE - ELIMINATE SEARCH

I have seen many batch applications which must re-scan all of the transactions for the year (or two years), in order to compute sub-totals by account number, month, division, etc. Once each month, they sort all of these transactions in two or three different ways. Several users have reported to me job times of 20 hours or more. With the availability of the IMAGE database, there should be few designs of this kind. Once a total has been calculated, and you know that you will need it again next month, it should be stored in the database for quick retrieval. The basic principle that I try to follow in disposing of information is: reduce transactions to summary totals as soon as possible, while saving the transactions (with their wealth of detail), in case a question comes up that cannot be answered from the summary information.

General ledger packages seem to be the worst offenders of this rule -- especially the ones that were converted to the HP 3000 from card-oriented IBM systems. By far the most elegant use of IMAGE for a general ledger that I have seen is the hierarchical structure (trees of accounts with sub-totals at each node) that is described in [32]. Anyone designing a new general ledger system should read that paper for ideas.

By saving summary totals, you are "depositing" into the database the CPU time and disc accesses that were used to calculate them. When you need those totals the next time, you have only to look in your safe deposit box (instead of taking out a loan to "buy" the information for a second time).

C8. SORT DATA BEFORE WRITING TO KSAM FILE

Within IMAGE and KSAM, there are "sorted" data structures -- sorted chains for IMAGE and sorted keys for KSAM. The time needed to add entries to such a structure may be reduced, if the entries are in sorted sequence initially [45]. (Of course, if there are multiple sorted keys, only one key can be optimized.) I used

SUPRTOOL to test this hypothesis on KSAM. When SUPRTOOL copied 2551 records to an empty KSAM file (with one key, duplicates allowed), the elapsed time was 92 seconds. KSAMUTIL reported that 2224 key block I/O transfers had been required. SUPRTOOL then repeated the operation, but sorted the 2551 records before writing them. The total time was reduced to 46 seconds, including the sort time (2 times faster), and the number of key block I/O transfers was reduced to 134 (17 times less).

In the tests that I performed, KSAM obtained a load "data rate" of 111 seconds per 1000 sectors with unsorted data and 53 seconds per 1000 sectors with sorted data. Compare this with the rates for DBPUT (203 seconds in default mode and 94 seconds in "deferred-output" mode, which is the way KSAM operates) and the rates for NOBUF calls to the file system (1 to 2 seconds per 1000 sectors). The more organization you demand of your data, the lower the rate at which it can be updated.

C9. SORT ENTRIES BEFORE PUT TO SORTED CHAIN

Since sorting data before writing it to a KSAM file cut the elapsed time in half, I hoped that the same thing would be true for IMAGE. There are two major differences, however, between KSAM and IMAGE. First, the entire KSAM file is sorted by the key field, while only a single chain is sorted in IMAGE. If the IMAGE chains are short (average length less than the blocking factor), it may not take IMAGE very long to put them in order. Second, KSAM always operates in output-deferred mode (it does not post buffers at the end of each FWRITE). IMAGE operates in output-complete mode. Therefore, the disc transfers needed to update the sort sequence will probably be a larger percentage of the total disc transfers in KSAM than they are under IMAGE. I have not seen a satisfactory experiment with these ideas yet.

If you do try this technique, I suggest that you either limit it to large batch tasks (where you sort an entire dataset before copying it to another dataset), or that you write your own internal sort routines. There could be nothing more catastrophic for your throughput than to do 10,000 seperate sorts, one for each sorted chain. Each time that you initiate the Hewlett-Packard sort sub-system (whether by running SORT.PUB.SYS, or using the SORT VERB in COBOL, or calling SORTINIT in SPL and FORTRAN), you are causing a temporary file with 10,000 records to be allocated on the disc. If you are only sorting 15 to 100 records, this is a tremendous waste of resources (both disc accesses and CPU time).

C10. COMBINE KSAM WITH IMAGE IF YOU NEED SORTED ACCESS

If sorted chains in IMAGE are bad [45] [34], and invoking the Hewlett-Packard sort package for each chain of 50 records is also bad (too much overhead to initiate and terminate the sort), what else can be done? Sometimes you need sorted access to IMAGE entries. Sorted access is the easiest way to provide generic search capability. IMAGE is not always the best answer. Sometimes you should use KSAM.

Here is a suggestion: copy your key values from an IMAGE master dataset and write them to a KSAM file (single key, no duplicates) in sorted order. One of the references that I read [38] found that sorting the key only (instead of the full record) could save up to 41% of the sort time. My tests (reported above, C8), found that sorting records before writing them to a KSAM file would cut the load time in half. Therefore, this IMAGE-to-KSAM transfer should be very quick. This is one of the tasks that SUPRTOOL can accomplish (extract and sort keys, write them to a KSAM file, clearing it first).

If you follow this prescription, you will have an "updatable" mechanism for sorted access to an IMAGE dataset. If the key field is "date", and you want to find all entries with dates in the month of June, you do a FREADBYKEY into the KSAM file and use the key values retrieved to get the actual records from the IMAGE dataset. If you want your index updated during the day, you must modify your on-line programs to FWRITE new key values to the KSAM file. Or, you may be willing to have the sorted access updated only once a day (after new transactions are validated in batch). In this case, just run the SUPRTOOL copy operation once every night.

Section V | WHAT "ACTUALLY" MAKES BATCH FASTER?

The last section evaluated many possible techniques to make batch jobs faster; the results are summarized below:

THESE TECHNIQUES WORK (sorted by their "power" to improve)

3x to 300x faster: Isolate data, at design time, by frequency and type of access.
5x to 50x: Save summary totals in a database.
5x-58x: Bypass the FORTRAN Formatter.
5x-20x: Eliminate unneeded data conversions.
10x: Convert DBDELETE/DBPUT to DBUPDATE.
2x-10x: Use NOBUF with MPE files (see FASTIO, BREAD).
2x-10x: Use NOBUF with KSAM/IMAGE (see SUPRTOOL).
3x: Rewrite some MPE routines in SPL (i.e., ASCII/BINARY).
2x-5x: Use more key block buffers, if a KSAM file is empty.
2x: Sort records before loading them into a KSAM file.
1.5x-2.5x: Use DBCONTROL to defer IMAGE writes, if the database is backed up to magnetic tape first.
1.3x-3x: Convert from COBOL 68 to COBOL 74 (COBOL II).
1.1x-3x: Increase IMAGE buffers for a single batch job to the maximum allowed.
1.5x: Keep IMAGE master datasets "clean" (see DBLOADNG).
1.5x: Use "*" field list, with write access to the dataset.
1.2x-1.5x: Increase the block size of MPE files to 1024 words, and that of IMAGE datasets to 512 words.

THESE TECHNIQUES LOOK GOOD, BUT NEED MORE TESTING

Use DBLOAD occasionally to reorganize your database. Reload the entire system to reduce disc fragmentation. Rewrite COBOL subroutines in SPL, if they are called frequently. Increase the size of code and data segments (the opposite of on-line optimizing). Use in-stack tables and extra data segments to eliminate disc accesses. Save pointers and data to eliminate some disc accesses. Consider serial search in place of chained and drop the search item. Sort data before DBPUT to a long sorted chain. Use KSAM with IMAGE to provide sorted access to IMAGE data.

THESE TECHNIQUES DO NOT WORK AS ADVERTISED

Using more than the default MPE file buffers. Dividing program code into small code segments (on-line programs only). Repeatedly contracting the data stack (on-line only). Doing extensive work to optimize head locality through explicit placement of files (except for straight file copies). Increasing IMAGE block sizes above the default (512 words). Increasing MPE file block sizes above 1024 words.

Section VI | CONSEQUENCES FOR DIFFERENT PEOPLE

The findings of this report have different implications for the different groups of people associated with an HP 3000 installation.

END-USERS: Ask for summary information and exceptions instead of mountains of paper; you will get your results faster.

DP MANAGERS: The "stock" HP 3000 may have to be "souped up" to handle your batch processing load. Try to acquire optimizing tools such as SUPRTOOL, BREAD and FASTIO, and give your staff the time needed to investigate the contributed library. Establish procedures to measure the resources consumed by each batch program, and review the results quarterly. Question changes to specifications at the "last minute" and insist on adequate time to implement them properly. Invest in staff development (through training and the users group) to keep up with the changes in hardware and software. Don't skimp on disc space. Ask systems analysts to estimate elapsed time for batch jobs, not just response time of on-line programs, when they design an application.

OPERATIONS STAFF: Don't spend hours moving files to specific disc drives. Run DBLOADNG once a month to check master dataset hashing and detail dataset "randomness". Consider DBLOAD occasionally.

APPLICATIONS PROGRAMMERS: Eliminate disc accesses wherever possible, by using DBCONTROL to defer writes in IMAGE, by copying small master datasets into the data stack, by doing multi-block NOBUF transfers, and by using extra data segments to access the main memory that is outside of your data stack. Save CPU time, especially in modules that are invoked in a loop, by avoiding inefficient constructs in your programming language, by switching to COBOL II, by using NOBUF access, by writing subroutines in SPL, by using the "*" field list with write access, and by hand-coding some operations, instead of using the general-purpose software provided by Hewlett-Packard (e.g., small sort operations). Measure the throughput of your programs using alternative methods. Avoid doing the same "work" twice when you could save the results for re-use later. Concentrate your efforts on "common events", such as the functions of a program that are executed the most often.

APPLICATIONS DESIGNERS: Keep record sizes below 256 words and block sizes between 512 and 1024 words, with the block size always a mutliple of 128 words. Design the database to hold summary totals, rather than recalculate them from the original data every time a batch report must be run. Add datasets to isolate records that may have the same fields, but are used with a different frequency and type of access (serial versus chained). Eliminate some search items with few values and use serial scan intead (use SUPRTOOL for maximum speed). Save search items for on-line access. Combine several records into one and use DBUPDATE instead of DBDELETE/DBPUT. Select search items that will hash well (X8 instead of J2). Consider KSAM for sorted access to IMAGE key values. Identify batch processing tasks at design time and estimate their execution times; use these estimates when designing the database. Develop guidelines, goals, and strategies for setting up job control commands, just as you would with any other high-level programming language.

SYSTEMS PROGRAMMERS: For COBOL (and other standard languages), add a built-in deblocking capability and make use of extra data segments where applicable. For IMAGE, do multi-block reads when a serial DBGET is requested and the buffers are not "busy". Improve the job control language of MPE so that jobs can handle more situations without needing operator intervention.

SYSTEMS SOFTWARE DESIGNERS: Provide special "paths" for batch programs so they can avoid slow, general-purpose systems software (IMAGE, Formatter, file system). Concentrate on making full use of the existing disc hardware speed. Merge KSAM capabilities into the IMAGE database.

HARDWARE DESIGNERS: Build a smarter disc controller (on a full track transfer, begin wherever the heads are located, and adjust the memory address, instead of waiting for the disc to rotate to the correct sector). Add CPU instructions to perform deblocking and database processing. Expand the data stack size to at least one megabyte. Provide multiple disc channels. Build a "black box" to do sorts.

Appendix | REFERENCES (SORTED BY AUTHOR)

For HP 3000 users who would like a "self-teaching course" on optimizing, I have listed below the references that I suggest, in a logical reading sequence. If you belong to the Users Group, you may already have all but two of these documents: [06] [20] [27] [15] [25] [33] [34] [07] [09] [44] [29] [18] [41] [26] [21].

[01] author unknown, "Data Base Retrieval Optimization", SCRUGLETTER, Vol. III, No. 5, 1979.

[02] author unknown, "FASTIO", S.E. newsletter, date unknown.

[03] author unknown, "FASTIO Benchmark Benefits", SCRUGLETTER, Vol. III, No. 5, 1979.

[04] author unknown, "Slides on optimizing" [??], CCRUG Meeting minutes, distributed January 1981.

[05] Keith Baer, "ARHND, Virtual Array Handler", HPGSUG 1980 San Jose Swap Tape.

[06] John Beckett, "Managers, You Can Control Response Time", HPGSUG 1980 San Jose Proceedings.

[07] Rick Bergquist, "Optimizing IMAGE: an Introduction", HPGSUG Journal, Vol. III, No. 2, 1980 (reprint from San Jose meeting).

[08] David Brown, "Disc File Access Optimization Using Multiple-Record, Non-buffered Data Transfer, SCRUG80 Proceedings.

[09] William F. Burggrabe, Jr., "Disc I/O Comparision Chart", HPGSUG Journal, Vol. III, No. 2, 1980.

[10] Stephen M. Butler, "Faster with Fast KSAM", HPGSUG 1978 Proceedings.

[11] Mike Casteel, "Programming for Multi-terminal Applications", HPGSUG 1980 San Jose Proceedings.

[12] Jim Dowling, "Performance Management Techniques for the HP 3000 Series III", paper presented to HPGSUG 1980 San Jose Meeting, not published in proceedings.

[13] EASY Software Company, "Blocked IO Reference Manual", 1980.

[14] Rick Ehrhart, "Using Extra Data Segments - Safe and Efficient", HPGSUG 1978 Proceedings.

[15] Robert M. Green, "Principles for Optimizing Performance of On-line Programs", HPGSUG Vol. II, No. 2, 1978 (also printed in Denver meeting proceedings).

[16] Robert M. Green, "SPL/3000 in a Commercial Installation", training guide published by Robelle Consulting Ltd, 1980.

[17] Robert M. Green, "SPL/3000: Overview and Common Errors", HPGSUG Journal, Fall 1979.

[18] David Greer, "Checkstack and Controlling COBOL Stacks", HPGSUG 1980 San Jose Proceedings.

[19] David J. Greer, "Memory Files", unpublished paper of Robelle Consulting Ltd.

[20] Dick Hamilton, "Tips for News Users", HPGSUG 1980 San Jose Proceedings.

[21] Hewlett-Packard Co., "Application Design and Optimization for the HP 3000", internal training manual [see your S.E.].

[22] Hewlett-Packard Co., "COBOL/3000 vs. COBOL II Performance", 1980.

[23] Hewlett-Packard Co., "Performance and Optimization Seminar for NOWRUG", 1979.

[24] Marc Hoff, "Using Extra Data Segments", HPGSUG Journal, Vol. I, No. 4, 1977.

[25] Jack Howard, "Extra Data Segments and Process-handling with COBOL", HPGSUG Journal, Vol. II, No. 1, 1978 (reprint from SCRUG78).

[26] Jack Howard, "System Design and Optimization Techniques and Tools", HPGSUG Journal, Vol. III, No. 3, 1980 (reprint from SCRUG79).

[27] John E. Hulme, "System Performance and Optimization Techniques for the HP/3000", Applied Cybernetics Inc., 224 Camino del Cerro, Los Gatos, CA, 95030, 1980.

[28] Steve Kaminsky, "KSAM vs. IMAGE", HPGSUG Journal, Vol. I, No. 6, 1978 (reprint from SCRUG78).

[29] Madeline Lombaerde, "NOBUF/NO-WAIT I/O", HPGSUG 1980 San Jose Proceedings.

[30] Eugene H. Mitchell, "IMAGE Access Timing Test", HPGSUG Journal, Vol. III, No. 2, 1980.

[31] Christine Morris, "FORTRAN Optimization", HPGSUG Journal, Vol. I, No. 6, 1978.

[32] Gary B. Nordman, "Using a Hierarchical Data Structure", HPGSUG 1980 San Jose Proceedings.

[33] Alfredo Rego, "Design & Maintenance Criteria for IMAGE/3000", HPGSUG Journal, Vol. III, No. 4, 1980 (reprint from SCRUG80).

[34] Bernadette Reiter, "Performance Optimization for IMAGE", HPGSUG 1980 San Jose Proceedings.

[35] Robelle Consulting Ltd., "SUPRTOOL User Manual", 1981.

[36] Joe Schneider, conversation with the author regarding MPE IV/Series 44 experiences and other topics, January 1981.

[37] Joe Schneider, "TBPROC, Table-handling with Extra Data Segments", HPGSUG Contributed Library Volume 7.

[38] Anil Shenoy, "Sort Performance Guidelines", S.E. Note 170, October 16, 1979.

[39] Rodney Smith, "Application Design for HP 3000", SCRUG80 Proceedings.

[40] Ed Splinter, "Optimizing FORTRAN", HPGSUG 1977 Proceedings.

[41] Jim Squires and Ed Splinter, "System Performance Measurement and Optimization", HPGSUG 1978 Proceedings.

[42] Chuck Storla, "Determining File Usage", paper presented to HPGSUG 1980 San Jose Meeting, not published in proceedings.

[43] Mark Terribile, correspondence with the author, January 1981.

[44] Mike Vislosky, "FASTIO", HPGSUG 1980 San Jose Proceedings.

[45] Geoff Walker, "IMAGE Optimization Checklist", HPGSUG Journal, Vol. II, No. 2, 1978.

[46] Dave Walmsley, "RPG/3000 Program Optimization", HPGSUG 1977 Proceedings.

[47] Frederick White, "Improving Performance of IMAGE Applications", HPGSUG Journal, Vol. II, No. 4, 1979 (reprint from SCRUG79).

[48] Ted Workman and Mats Jonsson, "The Effect of Disc I/O on System Performance", unpublished paper presented to HPGSUG International Meeting in Switzerland, September 1980.

[49] Lu Yamada, "DBLOAD times with varying BUFFSPECS", S.E. newsletter, date unknown.

DATA BASE DESIGN

Polishing Your IMAGE

Presented to:

HP General Systems Users Group
1981 International Meeting
Orlando, Florida
April 27 - May 1

By:

Karl H. Kiefer
Systems Engineer
HP - Englewood, Colorado

Data Base Design - Polishing Your Image

I. Introduction: Context for Data Base Design

The motivation for this essay stems from a perceived lack of understanding among professional programmers and analysts, including Image/3000 users, concerning strategies to adopt, and consequences of choices made, designing and implementing data base systems. From a theoretical view, data base technology is intended to overcome inherent limitations and unnecessary costs associated with the use of historically prior file structures and access methods. Namely, the use of indexed files and flat files resulted in systems characterized by physical redundancy of stored data, dependence between programs and data, update and integrity problems, security problems, and inaccessibility to data for unanticipated requirements. Data Base technology promised to overcome these maladies by providing a means by which users would be able to pool their organizations' information into a centralized, independent structure. Applications would be implemented through a common interface to this structure: the data base management system (DBMS). Actual implementation of data base systems in many, if not most, production shops has fallen far short of the promise of the technology. There are two generally related reasons for these developments.

Systems designers have not yet appreciated the proposition that an institution's management of it's information often determines its ability to react to changes in its environment. Biological evolution can, in one important aspect, be understood as a progression from simple to complex forms, and the complexity of these forms can be explained by the notion of information processing. More complex organisms are typically characterized by more sophisticated mechanisms involved in the processing of information. The sophistication of these processes is typically implemented via complex brains and sense organs. The survival success of these organisms is, in large part, made possible by an efficient means of sensing environmental changes and acting accordingly; of processing information. Similarly, the evolution of social organizations, business enterprises and governmental institutions can be understood in terms of these organizations' ability to process information. Simply stated, businesses which fail to manage information efficiently and wisely will, at best, be less profitable or, at worst, become extinct. Governmental organizations will be needlessly wasteful and, perhaps, fail to provide the service which justifies their reasons for being. This is necessarily so because they lack the capability to act readily according to pertinent changes in their respective environments.

An appreciation of the potential of data base technology to help provide this capability is a necessary, but not sufficient, cause for the success of the technology in realizing its promise. The second reason for its perceived failure is the costly lack of information and guidance from academia and, especially, vendors, with respect to criteria for good data base design and factual information with which designers might more ably evaluate conseauences of their choices.

As a result of this lack of appreciation and/or information, implementors of data base systems have historically viewed their DBMS as just another file structure and access method. Typically, an implementor is charged with the responsiblity of getting an application up and running and, perhaps, a choice is made for IMAGE over, say, KSAM according to some vague notion of "fitness" or performance, but from that point on, the DBMS is just another tool in the application implementation.

With respect to IMAGE/3000, some information regarding design and programming choices affecting systems throughout performance has begun to be disseminated to most users. Very little information however, exists elucidating either the criteria for design strategies or the impacts of design decisions when made. This essay, then, is an attempt to provide an outline of design considerations without claiming to know or state all of the costs or impacts of IMAGE/3000 design decisions. Indeed, it is hoped that if users know first what questions to ask, more complete answers may be obtained from actual implementation experiences which, through forums such as this, can supply valuable, shared information. Moreover, it is hoped that the impact of this essay will be applicable not just to IMAGE, but to data base design generically. This, it seems, is particularly desirable since, as we all know, full comprehension of any single product or aspect of our profession is almost certainly an indication of its obsolescence. By the time any of us knows all that can be usefully known about IMAGE data bases, we will surely be rewarded with a completely new DBMS about which we will know next to nothing, and we can start all over again.

II. Data Design and Relationships

Recent literature on desirable analysis techniques (read: structured analysis) invariably teaches that the first step is a global statement of the functions or objectives of the system. The same holds true for data base design. No technical methodology or checklist of analysis considerations can compensate for less than a thorough conceptual understanding of the functions which are to be implemented using the projected data base. Once these functions are stated and understood, the designer can proceed with the initial phases of design. These consist of the identification of the data items required to process the functions, and an analysis of the relationships which adhere between them.

Systematically associating data items with their functions can be accomplished using a function matrix (see the Data Base Design Kit for the HP 9845C, Hewlett-Packard part #09845-91057) and is a relatively straight-forward task.

Next, it is useful to characterize each item according to two important attributes. An item's VOLATILITY depends on the relative rate at which its contents change value. A part number in an inventory system is not volatile while its on-hand-quantity might be highly volatile. This characterization will impact decisions on performance, integrity of data and storage consideration. An items STRUCTURAL STABILITY refers to the physical way in which it is stored in the data base; i.e., its data type, and length. Ignoring or understating the possibility of structural change can have costly consequences.

The next design decision is fundamental to the success of the data base system. Grouping of items into entries or records obtains a definite relationship between the items which physically binds them together for storage and transfer. Ideally, IMAGE entries should reflect naturally some component of the function with which the related items are associated. An entry describing a part in an inventory system, for example, might naturally relate the following items: part no., description, bin no., quantity on hand, and quantity on order If we define well-organized data as easily accessed, storage space efficient, and easily restructured, then further evaluations are required when grouping items into entries.

RELATION MATRIX

Data Item	Item Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	1	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	2		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	3			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	4				▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	5					▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	6						▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	7							▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	8								▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	9									▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	10										▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	11											▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	12												▨	▨	▨	▨	▨	▨	▨	▨	▨
	13													▨	▨	▨	▨	▨	▨	▨	▨
	14														▨	▨	▨	▨	▨	▨	▨
	15															▨	▨	▨	▨	▨	▨
	16																▨	▨	▨	▨	▨
	17																	▨	▨	▨	▨
	18																		▨	▨	▨
	19																			▨	▨
	20																				▨

FUNCTION MATRIX

Function Number:		1	2	3	4	5	6	7	8
Function Description:									
Data Item	Item Number	DATA ITEM FUNCTIONS							
	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								
	9								
	10								
	11								
	12								
	13								
	14								
	15								
	16								
	17								
	18								
	19								
	20								

figure 2

The following grouping criteria are not necessarily harmonious; that is, increasing the priority of one may decrease the priority of others. Evaluating the grouping choices made according to these criteria will, however, minimize the possibility of costly surprises later on.

1. Group naturally. As indicated above, a natural grouping will simplify the implementation of the associated function.

2. Minimize redundancy. Saves space but may result in more difficult access, complex programming, and lower performance.

3. Group according to volatility. Usually a boon to performance.

4. Group according to entry size. Large entries may require less system I/O but more memory for buffers. Larger entries may obscure the functions to be performed with them. Large entries may obtain more on-line contention for shared data bases and, hence, lower performance.

5. Group for variable iterations. A table or other iterated values such as transactions are perfectly suited to chains. A table might be grouped into an entry; the trade-off is disc storage and memory space requirements versus I/O's.

6. Group according to structural stability. If items which have a good chance of changing structurally are segregated, the impact of change tends to be less severe.

7. Group for security. Security checks by IMAGE at the set level are less costly for performance than item-level checks.

8. Group for multiple views of same data: this may add to redundancy but is usually consistent with natural grouping.

At this point in the design phase, the designer has a preliminary scheme with entries defined and manual and detail sets related as to functional requirements. He may also have discovered that, since IMAGE obtains a network structure, he may need to implement a three or more level hierarchy through implicit programmatic relationships. For example, suppose that a typical manufacturing application needs to keep track of a final product's subassemblies, which could themselves have subassemblies and so on for several hierarchial levels. This can quite readily be represented in IMAGE through a recursive structure implemented programmatically. The master set contains entries for each assembly. The unique assembly number contains each related subassembly in a single detail set. The detail entry contains, besides the search item, only one other item, namely, the assembly number for that subassembly the entry for which is to be found back in the same parent master! (See figure 3)

Implementing multilevel hierarchical structures not recursive in nature consists of redundantly adding (typically) an automatic master as the intermediate link. In a retail accounting system, for example, several stores can be represented in a master chaining to each department for that store. The concatenation of the store and department numbers then obtain an implicit, symbolic pointer to an intermediate, automatic master which holds the chain heads for individual item entries for that particular department in that particular store. (See figure 4)

IMAGE is no different from any other DBMS in the sense that it is limited in the variety of structural relationships which it c¬n faithfully represent through its own internal pointer mechanisms. And since we can, if we are clever enough, represent virtually any desired relationship if we implicitly design and implement such relationships in our application programs, the designer must ask and evaluate the response to the question, what are the trade offs associated with implicit relationships?

It is useful to formally distinguish between explicit relationships and implicit relationships, the former being those available through the DBMS while the latter are those maintained solely by application programs. Further, it is useful to distinguish between implicit relationships which use symbolic pointers (as in both figures 3 and 4) and implicit relationships which make use of direct pointers. The status array is used by IMAGE to communicate with the user, data descriptive of, among other things, structural information. Through this mechanism, the user can not only access IMAGE entries directly, but also use this data in implementing his own implicit relationships.

The trade-offs associated with IMAGE supported, explicit relationships

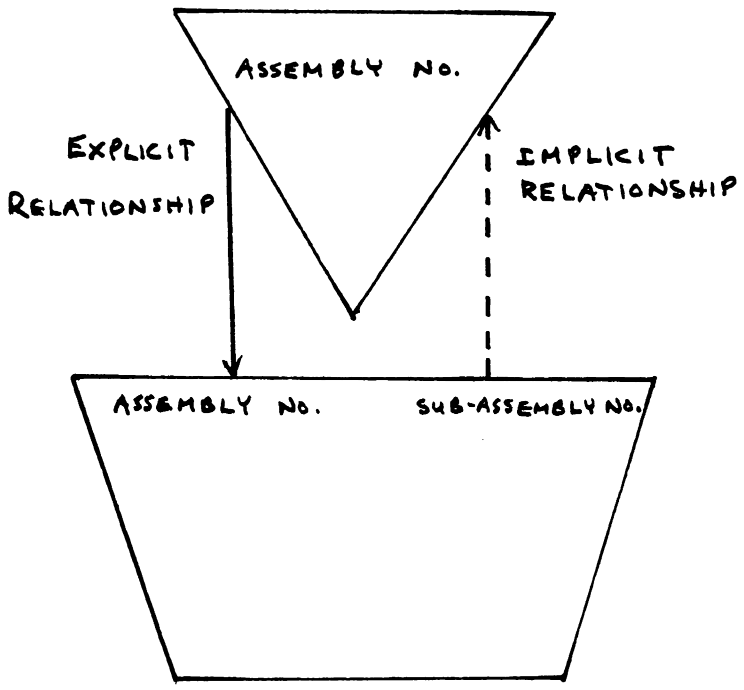


figure 3

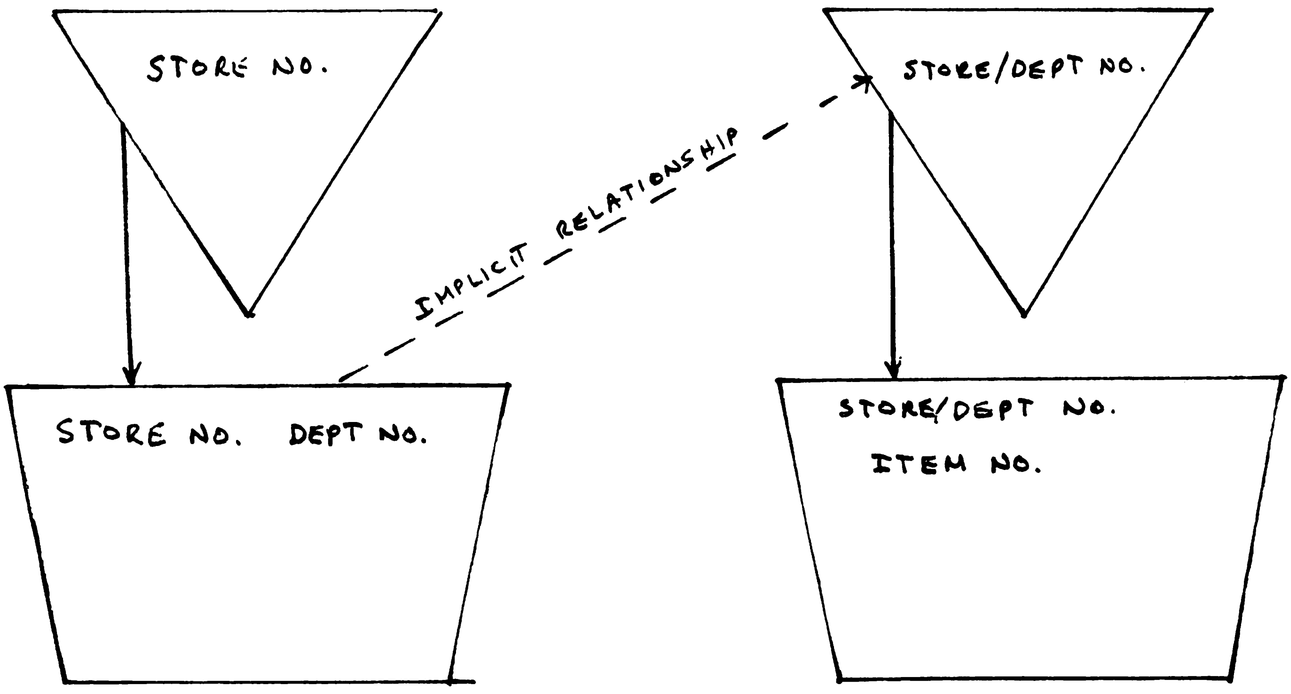


figure 4

include:

1. Limited design flexibility. Faithfully representing all of the organizations' functional relationships may be difficult, if not impossible.

2. IMAGE overhead. Any software tool designed to be general in scope and function has to be intelligent to provide that generality. This translates into overhead and may impact performance (E.G. sorted chains).

3. Low knowledge requirements. IMAGE users are not required to have in-depth structural knowledge. Knowledge is costly.

4. Support utilities. Maintenance of IMAGE data bases is provided by utilities which act consistently with internal structures.

5. Protection from changes. If IMAGE is modified, the DBMS calls are modified accordingly.

The trade-offs associated with implicit relationships include:

1. Unlimited flexibility in representing relationships.

2. Performance may be optimized (Or minimized!)

3. All affected users need to know the structure. High level of knowledge may be required.

4. Structural change to IMAGE may cause unexpected problems.

5. Modifications to and maintenance of user programs tends to be more complex.

6. User-written utilities may be required.

These, then, are some of the general considerations entailed in the first phase of design: identifying the items and their relationships. The second phase of design investigates the trade-offs associated with optimization for specific characteristics.

III. Data Base Design for Optimization

Many IMAGE data bases are intended to be central to on-line applications for which performance, specifically, response time to human interaction, becomes a primary concern. Since particular IMAGE performance considerations have been discussed elsewhere, we will not attempt to do more than relate general performance issues here. By doing so, we do not wish to minimize the priority of performance as a criterion in data base design. Indeed, that is not our choice to make. We do, however, wish to emphasize other areas of optimization, which, if neglected, can be even more damaging to the success of a data base system.

The pressure of production in the real world obtain, by default if not consciously, the decision to design a data base in order to optimize application development time. The benefits derived from such optimization are, at best, a product to show the end user in a relatively short amount of time. This benefit is almost invariably short term. If design for rapid development is obtained at the cost of other optimizing criteria, it is not long before catastrophes, constant reprogramming, and general dissatisfaction ensue. This is not recommended since effective design does not necessarily preclude quick development. Indeed, the opposite may be a truer consequence.

The relative inexpense of hardware has lessened the demands for optimization of storage space. It is typically cheaper to buy another disc drive than to re-design, or require herculean programming efforts in the interests of mass storage. The disadvantages of optimizing for space usually entail a minimization of redundancy. Even though this is a generally recognized goal of data bases, the realistic anplication gains increased performance due to more flexible access in the form of, say, redundant search keys in automatic masters. Decreased redundancy also typically entails larger data entries which imply grouping of unrelated items and grouping of volatile with static items. This generally results in larger impacts on application programs if structural changes are required. One benefit which accrues to smaller data bases is by no means negligible, however: the loading and unloading of data bases to magnetic tape for archival or for recovery is a time-consuming task which is directly related to the size and, in the case of structural reloading (DBLOAD), the organizational complexity of the data base.

Designing a data base with a view towards optimizing performance can be stated as one general rule: Reduce the total amount of work required by the system to process along the primary paths. For IMAGE/3000 data bases this translates typically into minimizing I/O's required to process along the critical paths. Complying with this rule requires, first, that the designer identify the critical paths. This is accomplished by understanding the flows of the applications contending for data base and system resources concurrently. Minimizing I/O activity becomes, first a matter of deciding who can, in fact contend. Can an application be batched if it contends with necessary on-line activity? Setting programming standards may obtain performance gains: locking strategies and standards; item-list processing; inefficient or unnecessary DBMS calls; use of internal record numbers for directed access; use of implicit relationships. Image or HP3000 peculiarities can impact performance: security settings, synonym chains; sorted paths; multiple paths into volatile data set disc drive placement; primary path contiguity on disc; the number of buffers; the sizes of buffers. The costs of performance consists primarily in: knowledge level; redundancy of data and, hence increased storage space requirement; complexity of programs; decreased flexibility in structure resulting in costlier impacts if changes are made; implicit relationships; possible data integrity and update problems due to redundancy and batching of applications not essential to on-line activity.

An appreciation of flexibility as a data base design criterion is, unfortunately, almost non-existent. It is unfortunate because the penalties meted out for failures in this aspect of design are rarely anticipated and often expensive. While performance of data base systems is a highly visible attribute, an inflexible data base structure typically displays its weaknesses suddenly and dramatically. It is usually triggered by an external change in the environment, perhaps as simple as an account number format (zip codes?) or as subtly complex as a slight modification to a standard corporate procedure. The solution may entail a simple organizational unload, modification of the schema, and reload to accomodate the structural change, or it may require many man-days and man-nights of reprogramming, or it may even force an admission that the change cannot be implemented without a total redesign.

A data base is said to be flexible if it is characterized by elastic data structures and elastic data relationships as opposed to inelastic structures and relationships. Elasticity is measured in terms of the ability to withstand change with a minimum of impact.

Redesign for flexiblity, for elasticity, can have significant effects even in trivial cases. Suppose, as is common, that an IMAGE data set is modified by adding a new item to the end of the entry. This is perhaps the simplest of changes to implement. The item is to be used only by a new application so its effects should be minimized. The data base is unloaded. modified, and reloaded. That's it! But wait! Suppose twenty or thirty or forty other programs access that set and, as is likely, for no more reason than programming ease, each program coded each call to DBPUT and DBGET with an item list of "@". Send out for beer and pizza; it will be a late night at the terminal! chances are fairly good, as well, that one or two bugs will creep into the system as a result.

Attaining flexibility in data base design is dependent on an understanding of data bonding and its implications. Bonding of data refers to the relating of data base components through either explicit or implicit relationships. Image data base components can be bonded as follows:

1. Items can be bound by grouping into entries.

2. Sets can be bound by paths.

3. Implicit relationships can form virtually any number of bonds, including those between distinct data bases.

Bonding can be described as tight or loose generally in the order indicated. The greater the number of items bound into an entry, the higher the probability that entry will be impacted by external, environmental change. The designer's first step in incorporating flexibility is minimizing the number of items in an entry. This choice is generally consistent with the functional definition of an entry which obtains an abstraction of some particular object of organizational relevance which an occurence of the abstraction describes, such as a bank account, a product, an oil well, or a transaction. Each item in the entry typically has a specific relationship to all other items in the entry, or, more commonly, to a key item in the entry. If an environmental change impacts the entry, all items in the entry are necessarily impacted naturally, and programs which process the entry will tend to be impacted naturally by the change in function. If unrelated items are bound into the same entry, environmental changes will unnecessarily impact these items as well as the programs which process perhaps totally unrelated functions.

Data is by definition inelastic in direct proportion to its redundancy. Every occurence of the items is impacted by relevant environmental change. If an item is both redundant and grouped with unrelated items, the impact of change multiplies even more. If a data functionally belongs in more than one entry, the designer needs to consider the reasonableness of combining the entries. Designing for flexibility commands that an item appear in only one data set and that items which serve to implement logically different functions should reside in logically different entries. (It should be clear that while key items are a necessary exception to these rules, the exceptions should be kept to a minimum).

For IMAGE this means that, if flexiblity is the design goal, a master/detail relationship in which the detail contains unrelated data items ought to be resolved into a master related to two or more details. Stated differently, each search item in the respective details should be an abstraction of a different functional object.

In practice these guidelines are not always so straight forward. Suppose, as in figure 5, a data base is used to maintain cost analyses by product. A master is related to three details, each containing cost figures for materials, labor, and transportation, respectively. Since each detail contains items functionally related to historical costs, a designer might reasonably combine the cost items into a separate detail set if he knows that these items are subject to frequent structural change these cost items are uniform and remain uniform through changes. The cost of doing so is an additional path and some implicit structures relating the material, labor, and transportation activity in the new set.

Designing flexibility at the data base level for IMAGE necessarily requires implicit, non-maintained relationships. The same objectives still apply, however. Separate data bases are warranted when the items and sets which comprise them are functionally dissimilar. If subsets of a data base are highly complex in terms of relationships and tightly bound, there is incentive

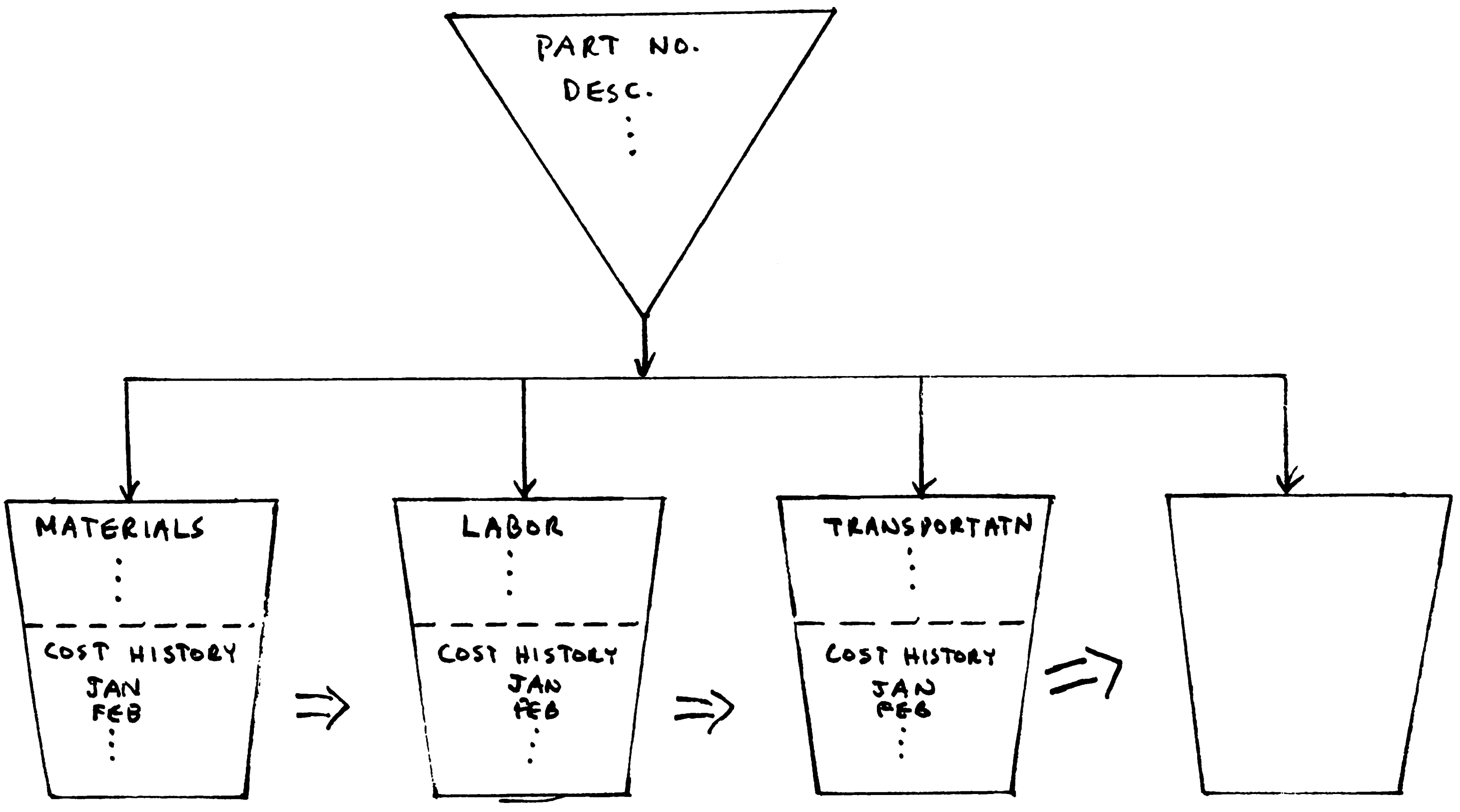

PART NO.
DESC.
MATERIALS
COST HISTORY
JAN
FEB
LABOR
COST HISTORY
JAN
FEB
TRANSPORTATN
COST HISTORY
JAN
FEB

figure 5

to consider breaking up the subsets into multiple, loosely bound data bases in order to minimize the impact of change.

In general, naturally structured data bases tend to be more flexible than data bases structured to easily implement strict user requirements. That is, natural structures tend to faithfully represent processes abstracted from the user's environment and changes to the user's environment will tend to follow naturally, whereas structures created to facilitate particular objectives will tend to be brittle and of narrow scope. The costs for flexibility are not always compatible with performance requirements or desires for ease of development and the proper balance must always be in the designer's mind.

IV. Design Review

As in any thoughtful systems work, design and analysis ought to be an interactive process. In data base design, periodic review with end users, development personnel, and management is the best method for reaching the goal of surprise-free implementation.

The reviews ought constantly to reaffirm the design priorities by evaluating both the reasonableness of the exceptions and, as clearly as can be judged, the costs in terms of other design criteria. End users should be encouraged to distinguish what they want from what they need and to understand, again, the costs associated with sundry features of the system. Analysts and programmers need to understand the requirements for standards in implementation as well as the relative weight of choices to be made during coding and testing. A detailed analysis may require a measure or count of system resource usage demanded by contending processes. Management must be persuaded to consult with DP when considering the adviseability of changes which impact the system. After implementation, periodic monitoring of system usage, performance, and standards enforcement is essential to securing on-going success.

We hope these remarks prove useful both in encouraging discussion of these matters and in proving or disproving their utility in practice. These matters would be far easier to engage if we lived in a world of perfect information with which to make informed, intelligent analysis. The fact is that we do not have anything near to such perfect information and so our tasks are characterized by artistry as much as by technical competence. We will always be artists to some extent, and so we need to appreciate that the success of the masters depended on skills and knowledge of tools as well as creativity.

References

1. Hewlett-Packard Co., Data Base Design Kit for the 9845B, C, 1980 part no. 09845-91057.

2. Callinane Corp., IDMS Data Base Design and Definition Guide, 1979.

3. Orland J. Larson, IMAGE Data Base Design and Performance Measurement, Abstracts and Proceedings of the HPGSUG, 1978.

4. Alfredo Rego, Design and Maintenance Criteria for IMAGE/3000. Journal of the HPGSUG, Vol III, No. 4, 1980.

5. Berni Reiter, Performance Optimization for IMAGE, Abstracts and Proceedings of the HPGSUG, 1980.

VESOFT CONSULTANTS
506 N.Plymouth Blvd.
Los Angeles, CA 90004
U S A
(213) 465-7453

MPEX/3000: EFFECTIVE USE OF MPE FILESET CONCEPT

By Eugene Volokh, Vladimir Volokh

Presentation to the HPGSUG
1980 International Meeting
Orlando, Florida, USA

ABSTRACT

This paper will discuss how one can use the concept of 'FILESETS' for general application systems development; it will describe the way in which MPE supports filesets, and discuss certain holes that are patched up by MPEX, a useful utility which greatly expands this concept.

MPEX/3000: Fileset Concept

WHAT IS A FILESET?

The MPE operating system supports the important concept of filesets. A fileset is a very convenient way to describe more than one file; e.g. just one file can be referred to as 'FILE.GROUP.ACCOUNT', but a name such as '@.SOURCE.AP' can refer to an entire SET of FILES -- in this instance all the files located in the group SOURCE of account AP; or, you can say 'AP@SRC', which means 'all the files that start with AP and end with SRC in the logon group and account'.

WHY A FILESET?

When an application program is written, it usually does not stand alone; it is logically linked to many other programs -- in short it is part of a SYSTEM of PROGRAMS. Whereas the system concept is a very good , it has its problems; very important one is that it is inherently hard to manipulate an entire system of programs. For instance, as you may well know, to store off a system of 50 programs I can use a FILESET in the STORE command, which is certainly much better than listing each one of the 50 filenames. This feature, allowed on the LISTF, STORE, and RESTORE commands is very handy. But, there are still many things to be desired -- can I list all my sources to the line printer? Can I find where in my 50-program system I refer to the item 'NUMBER-WIDGETS'? If I change my COPYLIB or make some other drastic change to my system, can I recompile it? Can I copy this system into another group, or RENAME it from AP@SRC to BU@SRC? With the current MPE facilities, the answer to this is unfortunately NO.

MPEX

These are the problems that confronted us in our application system development tasks. We found that after a substantial amount of alteration, our system needed a total recompilation; we found that when a new programmer arrived, we needed to give him a complete listing of all the sources in our system; we found that it was often necessary to find or change all occurences of a string in our system. We saw that MPE's fileset handling as it is implemented now is not adequate and not consistent. MPEX/3000 remedies this unfortunate inadequacy. It allows the fileset concept to be applied to several MPE commands/subsystems Thus, you can do a COBOL compile on a fileset; you can do a RELEASE/SECURE of an entire fileset; you can do a purge of an entire fileset; you can execute an EDITOR command upon an entire fileset; you can FCOPY/RENAME an entire fileset; you can list all the files of a certain filecode (PROG, EDTCT, etc.) that exist in the fileset.

MPEX even allows you to use the powerful SET OF SETS; i.e., you can perform an operation on several filesets at a time!!!

MPEX USAGE EXAMPLES

For instance, the following are some of the situations in which MPEX can prove to be an essential programmer's tool.

* A bulky 50-program system was written. It was then determined that a COPYLIB file must be changed, and thus the system must be recompiled. Ordinarily, you would need hundreds of MPE commands to do this. With MPEX, you can type one command and presto! a job is streamed that will compile and prep all of those progrsm, while you can continue productive work at your terminal.

* It was decided that a certain data base must be changed (e.g. with Alfredo Rego's ADAGER/3000). It would take about 3 hours to perform the actual structural changes; it would take days to modify the associated programs! With MPEX, you can get a complete listing of your system; you can find where an altered item/set is referenced; you can even automatically change the name of some item or set in all of your programs; and then, once all the modifications are completed, you can quickly and easily recompile the entire system.

* It is often very desirable to get a listing of the entire system of programs. Without MPEX, this task will be very time-consuming and error-prone. With MPEX, one MPEX EDIT command will create a listing of all those programs.

* It was decided that all the sources in the system are to be transferred from a crowded PUB group into the SOURCE group. Ordinarily, it will take many hours and many mistakes before this job is finished. With MPEX, one FCOPY or RENAME command will do the trick.

* Due to numerous system failures, many EDITOR K-files were generated. With MPEX, you can purge all those files in the entire system, in an account, or in a group; optionally, you can ensure that you will purge only the files you want to by asking MPEX to perform the PURGE with Y/N verification; i.e. for every file it will ask you whether you REALLY want to purge it.

* The system manager determines that disc space is very low because people are building huge, space-wasting data bases. He can do a LISTF of all the data- bases in the system and see which ones are necessary and which are not. Conversely, if an installation has decided to convert to IMAGE from KSAM, a system manager can insure that nobody is still using KSAM files by using MPEX's LISTF command to find all the KSAM files in the system.

* A set of source files must be secured against some programmers or must be released so everyone CAN look at them. Just run the MPEX RELEASE or SECURE command.

The list can go on and on.

MPEX supports many different commands. Following are some brief descriptions: (ALL THESE COMMANDS CAN BE EXECUTED ONLINE!!!)

COMMAND NAME	DESCRIPTION
COBOL	This command, similar in syntax to the MPE COBOL command, will compile and :PREP one fileset (the source-set) into another (the object-set), online or offline.
COBOLII	This command is identical in syntax to the COBOL command except that it uses the COBOLII compiler.
EDIT	The EDIT command will perform a specified EDITOR/3000 command upon a fileset.
FCOPY	This command will copy one fileset into another via FCOPY/3000. This command also has features that allow it to copy using SUPRSORT/ROBELLE, a product which provides faster file copying than FCOPY.
FORTRAN	This command is identical in syntax to the COBOL command except for its use of the FORTRAN compiler.
LISTF	MPEX provides an improved version of the :LISTF command which allows you to LISTF all the files with a certain file code (e.g. EDTCT, PROG, PRIV, USL, etc.) in the specified fileset.
PURGE	The PURGE command of MPEX lets you purge an entire set of files. This operation can be performed in a stream or ONLINE. It is suggested that the Y/N verification feature of MPEX (SEE BELOW) be used with this command.
QEDIT	This command will execute any QEDIT/ROBELLE command on a specified fileset. QEDIT is a ROBELLE Consulting product that is a fast, easy-to-use, and powerful replacement for the HP/3000 EDITOR.
RELEASE	This command lets you RELEASE (suspend security provisions) for a fileset.
RENAME	RENAME allows you to rename a fileset into another; this can be useful for renaming entire groups or accounts (RENAME @.@.OLD, @.@.NEW).
RPG	This command is like the COBOL and FORTRAN commands, except that the RPG compiler is invoked.
SECURE	SECURE is the opposite of the RELEASE command; it allows you to restore default security provisions to files that had previously been RELEASEd.

SPL	The SPL command is syntactically and functionally identical to the COBOL, FORTRAN, and RPG commands (except that SPL is used).
HELP	MPEX features an online HELP facility similar to that of MPE; it can be used for getting information on any MPEX commands and key features.
SP or SPOOK	All $STDLIST outputs of jobs streamed by MPEX stay in the MPE spooler, and can be looked at, printed out, and/or deleted via the spooler (SPOOK.PUB.SYS) subsystem. MPEX allows you to enter the spooler subsystem directly using this command.
ED or EDITOR	MPEX allows you to enter the EDITOR sysbsystem directly.
QE or QEDIT	MPEX allows you to run QEDIT.PUB.ROBELLE directly.
[:]mpe-command	Specifying any command other than the ones above directs MPEX to execute it as an MPE command; almost all MPE commands (e.g. :RUN, :PREP, :EDITOR, :BASIC, etc.) are allowed. As MPEX uses some MPE command names for its commands, some commands like PURGE, LISTF, and others will trigger their MPEX equivalents. To avoid ambiguity, prefix these commands with a ':' if you want them executed as an MPE command.

ADDITIONAL FEATURES OF MPEX/3000 COMMANDS.

Some other things you can do with MPEX commands:

Any command prefixed by a '!' is executed ONLINE as opposed to offline (in the background, the default). IT IS NOT SUGGESTED THAT COMPILES OF A LARGE NUMBER OF PROGRAMS BE DONE ONLINE!

A command prefixed by a '?' is executed ONLINE with Y/N verification; i.e., for every file selected the name of the file is printed, and you are asked to reply YES or NO. If you hit a carriage return or type anything other than a 'Y', that file will NOT be used. This is primarily useful with the PURGE command.

MPEX/3000 AS AN 'OPERATING SYSTEM' ON ITS OWN.

With all the above features, plus the capability to run, prep, and compile programs from MPEX, MPEX can be viewed as a self-contained operating system; it has been found to be possible and beneficial for programmers to 'live' in MPEX, running EDITOR or QEDIT when they must edit a program, compile single programs or sets of programs from MPEX, go into the spooler to retrieve the results of the job streams that MPEX streams off, etc. Moreover, if you choose not to use MPEX in this way, there are special hooks that can be installed that will allow you to run MPEX out of EDITOR rather than EDITOR out of MPEX; thus in EDITOR, you can say:

```
%COBOL AP@SRC, AP@OBJ, $NULL
```

and the command 'COBOL AP@SRC, AP@OBJ, $NULL' will be executed as if you were in MPEX.

AN EXAMPLE MPEX SESSION:

(all the lines that you type are marked '<<you>>').

```
:RUN MPEX.UTIL.SYS                                                   <<you>>

This is MPEX/3000 Version 1.0 (VESOFT Consultants)
For help or in case of problems type HELP

%HELP COBOL                                                          <<you>>

[?]COBOL source-set,program-set,[list-set],[master],[new-set],
                    [usl-set],[prep-time-parms]

Compiles COBOL programs

%FILE COPYLIB=COPYLIB.SOURCE                                         <<you>>
%COBOL GL@SRC, GL@OBJ, XLP                                           <<you>>
 #J17
See the spooler after 'DONE' message
%
FROM/J17 USER.ACCT/ GL003SRC COMPILE FAILED!
FROM/J17 USER.ACCT/ GL012SRC PREPARE FAILED!
FROM/J17 USER.ACCT/ COBOL DONE!

SPOOK                                                                <<you>>
SPOOK B.00.03   (C) HEWLETT-PACKARD CO., 1976
> SHOW                                                               <<you>>
#FILE    #JOB    FNAME    STATE  OWNER
#034     #J17    $STDLIST READY  USER.ACCT
> TEXT 34        << look at spool file #034 >>                       <<you>>
> LIST 100/200   << list lines 100/200 >>                            <<you>>
...               << the Spooler lists the lines >>
> EXIT                                                               <<you>>
```

```
%EDIT GL@SRC, LIST ALL,OFFLINE << print all files to LP >> <<you>>
 #J19
See the spooler after 'DONE' message
%
FROM/J19 USER.ACCT/ EDIT DONE!

FCOPY GL@SRC, BU@SRC, NEW      << create online backup >>  <<you>>
 #J20
See the spooler after 'DONE' message

%EDITOR                                                     <<you>>
/TEXT GL003SRC                                              <<you>>
/<< this is the program which gave a compile failed; modify it >>
/KEEP                                                       <<you>>
/EXIT
%COBOL GL003SRC, GL003OBJ, *LP                              <<you>>
 #J21
See the spooler after 'DONE' message
%
FROM/J21 USER.ACCT/ COBOL DONE!
RUN GL003OBJ         << check it out >>                     <<you>>
...    << output from GL003OBJ >>
%?PURGE TEST@.JUNK+TRY@.JUNK                                <<you>>
TEST1.JUNK (y/n)? Y  << yes, purge (you reply) >>
TESTXX.JUNK (y/n)? N<< no, retain (default) >>
TRY.JUNK (y/n)? Y    << yes, purge >>
TRY2.JUNK (y/n)?     << CONTROL-Y pressed; operation stops >>

%SHOWTIME                                                   <<you>>
FRI, DEC 17, 1980,  2:33 PM
FROM/J20 USER.ACCT/ FCOPY DONE!

%LIISTF @.@.PROD,PRIV               << typo! >>             <<you>>
UNKNOWN COMMAND NAME (CIERR 975)
%REDO                                                       <<you>>
LIISTF @.@.PROD,PRIV
 D                                                          <<you>>
LISTF @.@.PROD,PRIV
                                                            <<you>>

ACCOUNT=  PROD        GROUP=  PUB

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

TEST      PRIV   212W  FB           5          5   1        1  1  1
TEST01    PRIV   333W  FB       29111      29111   1     5053  1  1
TEST02    PRIV   767W  FB       35957      35957   1     7671  1  1

%TELLOP PLEASE PURGE DATA BASE TEST.PUB.PROD - DISC WASTER.<<you>>
%EXIT

END OF PROGRAM
```

TECHNICAL ASPECTS OF LARGE GEOGRAPHIC DATABASES ON THE HP3000

By Kenneth Berkun, Penny Evers, Thomas Juhasz

SCS Engineers

ABSTRACT

This paper will present and discuss a number of technical aspects relating to large (on the order of 150 Megabytes) geographic databases on the HP3000. Topics to be covered include design considerations, problems that were encountered and overcome, the use of HP2648 graphics terminals and HP2721 plotters for producing and editing maps, large IMAGE and KSAM file techniques, and the traversal of tree structures using recursive procedures.

SCS Engineers has designed and implemented a hydrologic database covering the continental United States. The database contains information on surface water -- rivers, streams, and lakes -- including locational information, segment names and numbers, and information on industrial dischargers located on the river segments. In addition, a KSAM file has been created which contains the complete digitized data for the hydrologic system. This data is used for producing computerized maps. These maps are first displayed on interactive CRT's and edited for optimal appearance, then plotted on a 4 color pen plotter.

During the design and implementation of this system numerous considerations had to be kept in mind, and many problems were encountered. Some of these were due to the large amount of data, and others were caused by the problems with maintaining accuracy while mapping. Several interactive and batch tools were developed to aid this project.

This database and associated programs are of interest to state, local, and federal agencies for managing water quality. The special techniques developed to work with the HP3000 are of interest to a wide range of programmers and designers.

TORONTO: (416) 678-1841 **info-boutique** MONTREAL: (514) 337-5007

EXPERTS EN ORDINATEURS - COMPUTER SPECIALISTS

The Obsolessence of Programming - GENASYS/3000.

----

by: Ian Farquharson

President - Info-Boutique Ltd.

Part 1. The Obsolessence of Programming.

In order to commence this discussion, we first address two questions viz:

i) IS IT POSSIBLE? that we can live without programming.

ii) WHAT IS PROGRAMMING?

i) Firstly, we must say that if one does not think it is possible for programming to become obsolete AND one is not prepared to listen to persuasion then ....HALT.

Otherwise , let us continue......

ii) After some research into defining "programming", I have come to the conclusion that the world outside of data processing is pitifully ignorant still of basics. For example, in one noted dictionary the following definition is given.

PROGRAMMING: DATA FOR A COMPUTER.

As a result, I have attempted to define programming, in a way in which we can relate.

Is it informing the computer a) WHAT you want it to do.
b) HOW to do what you want.
c) Both of the above.

I believe it is c). ie. PROGRAMMING IS INFORMING THE COMPUTER BOTH WHAT YOU WANT IT TO DO AND HOW TO DO IT, IN A WAY IN WHICH IT CAN UNDERSTAND.

This is acheived through the use of various 'languages' which, as we know, ultimately translates down varying levels to the binary notation of the computer (COBOL, FORTRAN, RPG etc).

Within computer programs we find both the WHAT and HOW of our definition above.

The WHAT invariably becomes actions such as PUT, GET, DISPLAY, PRINT, UPDATE etc. The HOW is the logic that pulls together these actions and other programs to acheive the desired results. IF-THEN-ELSE-(GO TO?) etc.

Most of us concerned with this discussion will at one time or another, *have inherited programs which resemble 'spaghetti with meatballs'; programs where the actions (meatballs) are interconnected by logic so messy (spaghetti) that understanding it is 90% of the work and modifying it only 10%.*

I would also like you to give thought to the following, related, questions:

a) where is most programming time spent?

b) where are the worst "bugs" found?

c) which "bugs" create the worst problems?

answer any of the following.......

Answer 1). Incorrect or invalid actions.

Answer 2). Syntax errors.

Answer 3). Logic flaws.

My answer is 3).

With both invalid actions and syntax errors, the results are usually obvious and fast to resolve especially with an on-line system. In fact with syntax errors, the solution is often just to re-compile.

With flaws in the programs logic, the symptom is often intermittent. With spaghetti programs (& many are!) the time involved is significant.

Also true of logic flaws, is that the larger the program, the more difficult it is to fix, and the fact that the logic involved is that of another person and each person thinks in a different way!

We must conclude from the above points that the current status is not very acceptable and we should look for alternatives.

Alternatives.

1) back to the quill pen.

2) there is no solution and we must live with it.

3) structured programming.

4) program generators.

5) application generators.

6) robotics.

For myself, I believe that ultimately we will see the day when we can talk to robots and tell them what we want. However, I do not beleive that the technology is yet ready although it is progressing rapidly. I therefore select 5) - application generators, as my solution and goal.

Before discussing my reasoning, however, let me first comment on the other alternatives:

1) & 2). I expect no supporters of these alternatives.

3). Structured programming. If one must program computers, then let them be structured. Unfortunately, too many of us are of an 'old school' where we have left behind us years of programming to become

Senior Analysts, DP Managers or Consultants and as a result have carried forward too many bad habits. We think we understand structured programming but many of us would have problems knowing where to start. Fortunately, the Universities and Colleges are now producing people who do not know any other way to program other than structured.

Structured programming definitely REDUCES logic problems but does not eliminate them. It is still susceptible to costly conversions with new machines and languages.

4). Program generators. examples are SL1, GENASYS Inc. NB. this product is not to be confused with Info-Boutiques product called GENASYS/3000 which is an application generator.

These products allow us to tell the computer WHAT we want with out detailing the logic. The computer reacts by producing COBOL programs for you which are nicely structured and commented.

Magic you say; wonderful! OR IS IT.

The advantages are obvious, of course, but let us analyse the disadvantages:-

a) COST...... typically $100,000 to $200,000!

b) COBOL.....	is usually the result. You may hate COBOL. COBOL standards are becoming less and less compatible with previous versions, and it is too inflexible for fine actions like string handling.
c) EFFICIENCY	COBOL is not necessarily the most efficient language.
d) DATABASE..	there are no database standards in COBOL.
e) PEOPLE....	you still need people trained to program.

Application Generators.

Examples of this are our own GENASYS/3000 (Generator of Application Systems) and APL.

APL is mentionned because it is an attempt to develop systems by entering specifications and avoiding detailed logic, although most people think of it as a programming language. Its biggest disadvantage is that it uses special symbols which bear no resemblance to English.

The CONCEPT is: Tell the computer WHAT you want, NOT HOW to do it.

In other words, avoid time-wasting with logic.

In as simple a form as possible, we should be able to enter specifications into the computer relating to any new application that we wish to build (eg. with an on-line EDITOR). END OF STORY.

The computer should then be able to interprate these specifications and do what we require without any further action on our part.

It is possible and if we look at the following broad spectrum of *programming areas we can assess each individually:*

```
               ***************
               * COMPUTER    *
               * DATABASE    *************** Documentation
               ***************
               *         *
               *         *
               *         *
               *         *
           Terminal     Batch
           oriented     reports
               ¶
               ¶
            Speech
           oriented
```

The computer can do all of the above without having to write programs. (except for speech, which is in the near future)

I believe in letting the computer do the work.

It is the last necessary step before we enter the age of robotics. If we can not just tell the computer WHAT we want, how can we tell a robot?

PART 2. GENASYS/3000

GENASYS/3000 is an application generator which was designed and produced by Info-Boutique Ltd.

Of the three main program categories listed earlier, it currently satisfies the terminal oriented functions (and would be easily adaptable to speech) and the documentation. A report-writer is planned soon, however it will intereact easily with QUERY, QUIZ etc. It is a database oriented product (ie IMAGE/3000) although much can also be accomplished with KSAM/30000.

At this point it would be useful to reiterate point 5) from PART 1.

Tell the computer WHAT you want. ie enter your specifications with an on-line EDITOR.

Thats exactly what we do. Referred to as a SPECIFICATIONS file (SPECS file) we use the standard HP EDITOR and enter our system specifications in a relatively free-format but using key-words.

NB. We can have any number of SPECS files. eg. 1 containing all *applications or many containg different applications.*

The concept is really the same as QUERY but the idea is carried much farther. eg.we can create an XEQ file to produce a report with just a few lines. The traditional approach would have been to write a program which would have been a few pages of code. We would have depended on each programmer to use correct logic for such repetetive functions as numbering pages, printing the date, totalling etc.

```
GENASYS                       QUERY


.......                       .......

.......                       .......

....... SPECS                 ....... XEQ

....... file                  ....... file

.......                       .......

.......                       .......


:FILE MENU=specsfilename      :RUN QUERY.PUB.SYS

:RUN GENASYS.GS.INFOSYS       >XEQ filename

......                       ......

order entry,                 outstanding orders report.

accounts receivable,

file maintenance,
```

accounts payable,

inventory control,

etc, etc.

NB there are no programs to be written in either of the above.

I am sure that most people will agree that QUERY is extremely limited but for some strange reason HP have never improved it. I believe that the concept is right and the proof is the wealth of comparable products on the marketplace. eg ASK, WIZARD, QUIZ, REX, etc. All of these products use a similar concept to QUERY but succeed in all areas where QUERY does not. (mainly multiple data-sets.)

There are some points that must be clarified now with respect to GENASYS/3000:

1) It should NOT be confused with V/3000.

V/3000 maintains forms. You still have to write programs.

2) It should NOT be confused with data-entry programs.

GENASYS/3000 can enter data but it has two way communication with your database. It does most of the functions that your programs would have done.

3) You do NOT need either block mode or even HP CRT's.

4) Yes there must be some limitations but very few.

See the following list of features.

Main features:

1) Consistent & compatible with IMAGE, KSAM etc.

2) Terminal independant.

3) Multilingual.

4) Powerful in-built help feature.

5) Ability to branch out to standard subroutines.

6) Ability to run other programs (interface with packages).

7) User control over batch STREAMS.

8) Multiple data-sets per program function.

9) Multiple screen forms per program function.

10)multiple screen forms per data-set.

11)Verification option (double-entry) for keypunch.

12)Self-test, self-demo options.

13)STACKED input.

14)MENU control.

15)MENU-level security.

16)SCREEN level security.

17)FIELD level security.

18)Subroutine library included.

After assessing the above features the obvious question that will be raised is WHAT ABOUT EFFICIENCY?

The fact that the HP3000 automatically makes programs re-entrant means that GENASYS/3000 automatically becomes re-entrant. eg. if all your applications were done with GENASYS only, and let us say for example you had 50 terminals running 30 different program functions, there would still be only 1 copy of 1 program in memory - GENASYS!

Bearing this in mind, it is important that GENASYS itself is not cumbersome. FACTS... if GENASYS is completely locked in memory the total requirements are 35k bytes plus user data-segments. Alternativley, if it is not to be locked in memory then segments of *function code are swapped in as required. The largest segment is 4k bytes!*

GENASYS/3000 is written in SPL.

Documentation

Documentation warrants a special section in its own right.

If all your system specifications are in the computer and field statistics are in the IMAGE root file AND the MODUS OPERANDI is consistent, THEN it is feasible that the computer can write your complete system documentation for you!

We have acheived this with an optional GENASYS product called the DOCUMENTOR.

You don't have to write a word. It will present you with sizeable well written, consistent documentation in a matter of minutes which is....

1) Word processed.

2) Table of Contents.

3) Standard Operating procedures.

4) All MENU's used.

5) All SCREEN layouts used.

6) All FIELD specifications.

7) All EDITING rules used.

8) Explanation on operating each function.

9) IMAGE schema and SPEC's file.

10) Complete cross-index of words.

The Bottom Line

Where does this discussion lead us finally?

I think that perhaps a look at one companies experience with an application generator such as GENASYS is worthy of consideration.

According to the M.I.S. Manager of B.A.S.F. (Canada), Hank Van Leuwen, the following was true:

B.A.S.F. installed an HP3000 series III in 1979. The mandate given to DP was an overwhelming amount of new on-line systems and requests which were growing faster than results could be obtained.

The initial action taken was to hire people and contract much of the work to outside individuals/companies.

The standard approach was used ie. COBOL with V/3000 and either KSAM or IMAGE., until the discovery of the GENASYS prototype. After purchase of GENASYS and a report-writer from Info-Boutique, B.A.S.F. progressed at an outstanding rate and they concluded the following after a detailed analysis:-

** 2,500% improvement in development over COBOL & V/3000 (ie 25 times faster)*

** greater m/c efficiency (purchase of extra memory was delayed after seeing GENASYS performance).*

** immediate cost savings by not contracting out.*

** high moral of users, getting action fast from DP.*

** high moral of programmers, can now concentrate on interesting problems and progress to systems design.*

** less cost in terminals by switching to HP2621 in many areas.*

** documentation & systems standardized and always up to date.*

** luxury features available to users at no cost eg HELP (?)*

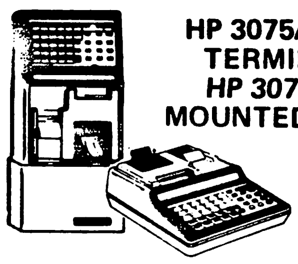
** their projects, which were estimated to take 14 man-years with COBOL & V/3000, took just 4 man-years with applications generation.*

CONCLUSION.

The obsolessence of computer programming is not a dream about the future. Neither is it a complex system of COBOL program generation, only available for hundreds of thousands of dollars on IBM 370's.

It is a reality, here and now, on your own HP3000 computer.

It's biggest enemy is the closed mind.

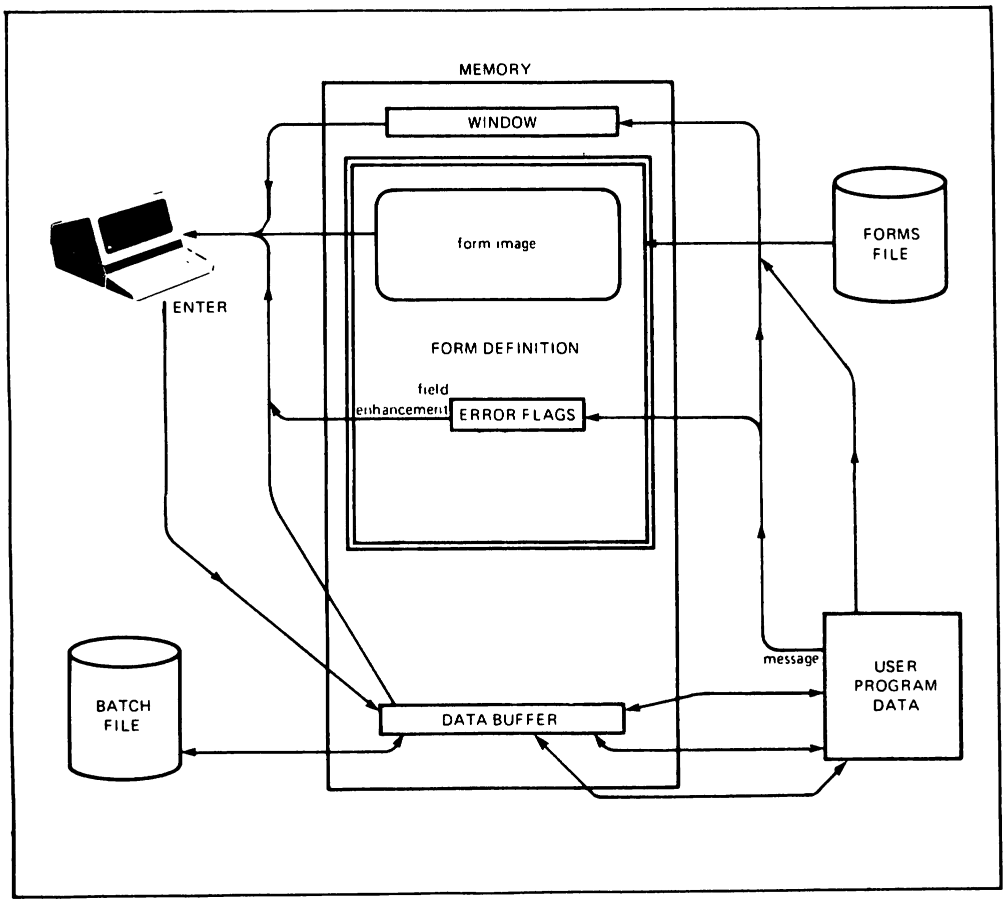

HP 3075A DESKTOP TERMINAL AND HP 3076A WALL MOUNTED TERMINAL

DATA CAPTURE ON THE HP 3000

by: Jutta Kernke
Product Manager
Hewlett-Packard Company
Information Systems Div.
19420 Homestead Road
Cupertino, CA 95014

Data Capture on the HP 3000 is performed through VPLUS/3000 which was designed to address the marketplaces from dedicated source data entry to data entry, data manipulation and terminal management of a general purpose computing system. The support of the HP 3075/6 data collection devices extends the capabilities to the end-user without terminal experience. NO new product needed, NO other rules to learn, NO additional training--just ONE VPLUS/3000 to manage it all, CRT's and Data Capture terminals. Data Capture design and data collection is a natural extension of V/3000!

VPLUS/3000 simplifies the design of terminal applications since it provides a single, friendly, consistent method for specifying data entry and validation. VPLUS/3000 manages the interface between a user program, CRT terminals and the specialized data collection devices, a forms file (where applicable), the entered data, and, for data entry, the batch file to which entered data is written. The high-level procedures take care of reading data from the terminal or data collection device, transferring data to it, and provide a convenient method of handling errors that might occur. No programming is needed using the ENTRY utility for a quick and simple application.



FEATURES for Data Capture

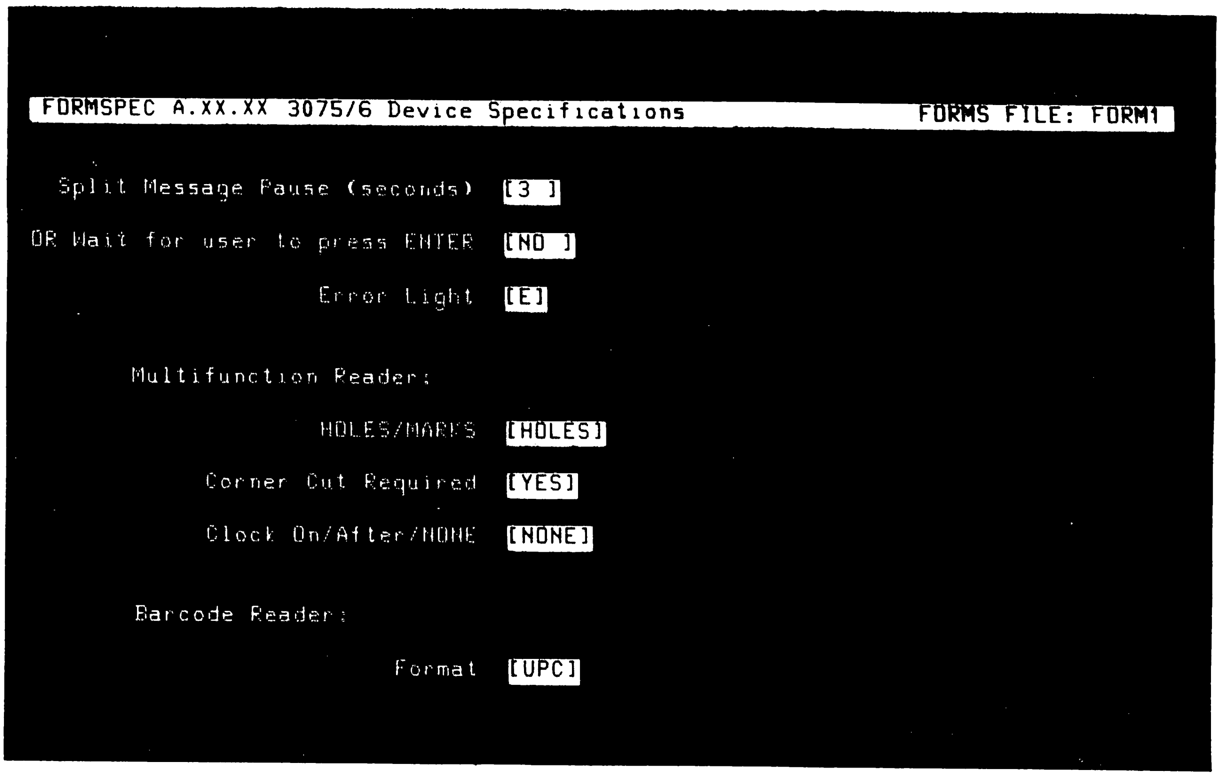
- Management and Support of HP 3075/6 Data Capture terminals
- Menu-driven, conversational dialogue to design terminal screens and specify multiple input/output devices and up to 17 prompting light sequences
- Comprehensive editing without programming
- Operator-terminal interaction
- No programming for simple application and source data collection
- High-level procedures callable from COBOL, COBOL II, FORTRAN, RPG, BASIC and SPL programs
- Convenient and customized error message handling
- Support of three connection alternatives, including Factory Data Link

USER BENEFITS

Design Flexibility:

VPLUS/3000 offers great flexibility through the easy use of FORMSPEC to configure HP 3075/6 Data Capture devices or design forms for the 5 inch optional CRT. The designer can, for example, specify from a single menu which light is to be lit when an error is detected, specify the device configuration for Multi-function and Bar Code readers, or specify how a message longer than 24 characters is to be presented on a single-line display. The configuration command can be used to determine the input/output devices and prompting lights for use on the HP 3075/6 terminals.

For example, if the user selects the HP 3075/6 terminals on the Terminal Selection menu, the Device Specification menu will follow. On this menu, the user can specify how a message longer than 24 characters should be presented on a single-line display, or specify which light is to be lit if an error is detected.



```
FORMSPEC A.XX.XX 3075/6 Device Specifications                    FORMS FILE: FORM1

  Split Message Pause (seconds)  [3 ]
OR Wait for user to press ENTER  [NO ]
                    Error Light  [E]

       Multifunction Reader:

                    HOLES/MARKS  [HOLES]
            Corner Cut Required  [YES]
            Clock On/After/NONE  [NONE]

       Barcode Reader:

                         Format  [UPC]
```

The Field menu is used with the configuration commands to specify input devices and the use of prompting lights. A simple example is as follows:

```
CONFIG
DEVICE printer,keyboard, display,barcode
(this will enable the input media device)
                    - or -
CONFIG
LIGHT A,B,D,G,N,P
(this specifies which light will be lit during
 field presentations.)
```

Data Verfication

High-level procedures can be used from a user-written program to verify data immediately against information in an IMAGE data base, KSAM or MPE files.

User Applications Can Be Customized

Applications may be customized for the terminal features available. The HP 3075/6 devices support a numeric or alpha-numeric keyboard, numeric or alpha-numeric display, a compact 5-inch, high resolution CRT, function keys, punched card and badge readers, magnetic stripe reader, and a bar-code reader. FORMSPEC makes it easy to specify the addressing of any of the terminal options at any one time.

HP Data Capture Communications

The HP 3075A and 3076A Data Capture terminals possess identical data communications capabilities. A choice is available in each terminal from three data communications modes:

- point-to-point
- multi-terminal daisy-chained
- factory data link

The Factory Data Link is a data communications link used to interface a computer and a large number of terminals. It is ideally suited for applications involving the collection of data from a large number of widely separated sources in the same building.

ROBELLE CONSULTING LTD.
#130-5421 10th Avenue
Delta, B.C. V4M 3T9
CANADA
(604) 943-8021
Telex 04-352848

Experiences With Pascal

Technical Report, February 1981, By

DAVID J. GREER

Robelle Consulting Ltd.

Summary

This paper reviews the history and possible future of the Pascal programming language on the HP 3000 computer system. It reviews some of the currently available compilers, and discusses the features that are likely to be included in an HP-supported Pascal compiler. The presentation will cover practical problems encountered while using Pascal, including problems that arise in transporting Pascal programs from other machines to the HP 3000.

Contents

1. Introduction

2. Pascal-V

3. Proposed HP Pascal

4. Pascal in Applications

5. Performance of Pascal Programs

6. Appendix I - Making Pascal Portable - Three Examples

7. Appendix II - Implementation Notes on Pascal-V

8. Appendix III - Sample Data From the Sequential Read Test

9. Bibliography

Introduction

This paper concentrates on one person's experience using and implementing the Pascal language on the HP 3000. The information presented was accumulated during approximately two years of work with Pascal on the HP 3000 and on an AMDAHL/V6 at the University of British Columbia. The major points covered by this paper are: 1) a brief history of Pascal, including implementation on the HP 3000; 2) the state of the contributed Pascal compilers; 3) an approximation of what the HP-supported Pascal may look like; 4) a look at Pascal in applications; what to avoid and what to be prepared for; and, 5) a preliminary evaluation of Pascal's performance.

History

Pascal was developed by Niklaus Wirth in Zurich. The principal aims of Pascal were to provide: 1) a language in which structured concepts could be taught easily; and, 2) a language that would be relatively easy to implement on many machines [18,15,1].

These original aims have direct application to business and scientific programming. The language provides constructs which emphasize structured coding concepts. Programs written in Pascal tend to be structured, which in turn makes them easier to maintain and understand. Because Standard Pascal has been implemented on many different machines, including the HP 3000, it is a good vehicle for writing portable software.

Another reason for the current interest in Pascal is that many secondary institutions, universities and colleges are now using Pascal as their principal programming language. As this trend continues, there will be increasingly more incentive to write software in Pascal, since the new work force will already be trained in the language.

Portable Pascal-P4

In order to make Pascal available on many machines, Urs Ammann, K. Nori, Ch. Jacobi, K. Jensen and H. Nageli wrote the portable Pascal-P4 compiler, which is itself written in Pascal [15]. Approximately 80% of the Pascal compilers in the world are based on this compiler.

The P4 compiler takes Pascal source code and compiles it into symbolic code for a hypothetical stack machine called a "P-machine". The individual implementer of Pascal writes an assembler or interpreter for the "pcode". Later, the source program of the original compiler is changed to generate the host machine's object code directly. The following is a schematic description of how pcode works.

```
Pascal -----> PCODE -----> Object
Source   |            |     Code
         |            |
      Compiler     Assembler or
                   Interpreter
```

HP 3000 Pascal-P4

In the HP 3000 contributed library, there is a version of Pascal developed by Grant Munsey, Jeff Eastman and Bob Fraley. This compiler is a modified version of the Pascal-P4 compiler; it fixes several bugs in the original P4 compiler, and provides extensions to Standard Pascal. Some of the important extensions are: built-in procedures to do direct access to MPE files, an "otherwise" label in the <u>case</u> statement, calls to Pascal procedures which were compiled externally, and calls to procedures written in other languages.

The process of compiling Pascal programs on the HP 3000 is slightly different from the one described above. Instead of assembling pcode into object code, the assembler of the contributed version assembles pcode into SPL, which is then compiled into USL files.

```
Pascal -----> PCODE -----> SPL -----> USL
Source   |            |          |
         |            |          |
     Compiler     Assembler     SPL
                              Compiler
```

While much of the work of implementing Pascal was simplified because of the P4 compiler, the early work done to transport Pascal to the HP 3000 was still difficult. Each of the P4 "pcode" instructions had to be translated into one or more SPL instructions. Also, the basic Pascal-P4 compiler only allowed for character constants of ten characters or less - a severe restriction [5]. See Appendix II for more details.

Pascal-V

History

Pascal-V is another version of Pascal for the HP 3000, developed in Vancouver as a project in compiler design at the University of British Columbia. This compiler is a modified version of the one that is available in the contributed library [3,4,5].

Major Problems

Two of the problems in using the contributed library's Pascal are: it is somewhat difficult to specify all of the command sequences for invoking the compiler (although this has been fixed with UDCs), and, the compiler is very slow. Both of these problems stem from the long process required to get from Pascal source code to USL files.

Pascal-V does away with the assembly stage of the compilation process. Instead, Pascal source code is translated directly into SPL code. The Pascal compiler then invokes the SPL compiler to produce the USL file.

```
Pascal -----> SPL -----> USL
Source   |           |
         |           |
     Compiler       SPL
                  Compiler
```

By eliminating the pcode stage of the process, a 20-40% savings in elapsed compilation time was realized. Also, by having all of the compilation stage in the compiler, it was easier to use the compiler. The SPL stage of the compilation process was not eliminated, because it was too difficult to work with USL files directly.

Minor Problems

The original contributed compiler did not print error messages. The compiler now prints a summary of all of the error numbers which occurred during compilation, along with their associated error messages. In addition, a compiler option has been provided which causes all Pascal reserved words to be underlined. This greatly enhances the readability of Pascal programs.

Athena Compatibility

With the Athena MIT release of MPE (2011), all Pascal programs on the HP 3000 ceased to work, including all of the contributed versions of Pascal. The cause of this was that, as of the Athena release of MPE, Qinitial (the initial setting of the Q-register in the program's data stack) was two words higher in the stack. Since Pascal made certain assumptions about where Qinitial would be in the stack, Pascal programs stopped working.

Pascal-V fixes this bug by making no assumption about where Qinitial should be.

Standard Pascal

There is a world-wide effort to provide a comprehensive standard for Pascal. Any compiler which claims to compile Standard Pascal must compile all parts of the defined standard correctly. In order to help implementers and users of Pascal find out whether their particular compiler meets the standard, A. Sale and R. Freak have written a suite of Pascal programs to test Pascal compilers.

The suite consists of approximately 300 Pascal programs. Each program is compiled and executed by the Pascal compiler. The results of each program are analyzed for errors. If a compiler meets Standard Pascal completely, there will be no errors from any program in the suite.

Pascal 2.4-V was tested using this suite. It had as many errors as other compilers based on the original P4 compiler (that is, the compiler was average). Several of these errors were fixed in Pascal 2.5-V and later versions.

Usage

The Vancouver version of Pascal is currently used in about forty installations around the world. Many installations are also using Pascal-S (see Appendix I) for teaching Pascal. Two installations are using Pascal to convert software from other machines to the HP 3000.

Availability

Because we need a Pascal compiler now to develop programs at Robelle Consulting, I have been and will be maintaining this Pascal compiler. It is not a supported Robelle product; but, for $200 U.S. (to defray costs), we will send a magnetic tape copy of the latest version to interested users ($100 if you send payment with your order and do not ask for 800 BPI). As of January 1981, the latest release was Version 2.8-V. A bug was fixed that did not allow compiles while logged on as MANAGER.SYS. Enhancements were made in error messages on file opens, in the use of Control-Y and in the run-time support (so that all Pascal programs can read QEDIT-format files). Inquiries can be directed to me at Robelle Consulting Ltd., #130-5421 10th Ave., Delta, B.C., V4M 3T9, Canada. Phone: (604) 943-8021. Telex: 04-352848.

Future Enhancements

Three major problems (as well as many minor ones) remain to be fixed in the compiler, if it is to be used in a commercial environment. The first is that character strings are stored as one character per word, rather than one character per byte. This needs to be fixed, so that variables will use less memory space, and so that Pascal programs can communicate directly with HP subsystems such as IMAGE.

The second problem has to do with Pascal's definition of parameters to procedures and SPL's definition of parameters to procedures. SPL is known as a programming language with weak type checking. This gives the programmer more flexibility, but provides more opportunity for making mistakes. It also allows IMAGE parameters to be defined very loosely. For example, a data set can be defined by a character-string containing the name of the data set, or by an integer variable with the set number. Since Pascal does not allow such flexibility in parameter passing, some mechanism must be established to allow procedure calls to subsystems such as IMAGE.

The third problem is that Pascal integers are implemented as single-word integers. Many subsystems such as IMAGE require that double-word integers be passed as parameters. At the same time, there must be a way of declaring single-word integers, since other subsystems require both single- and double-word integers to be passed as procedure parameters.

Proposed HP Pascal

Recently, there have been several rumors that HP would provide a supported Pascal compiler for the HP 3000 some time in the future. The following is an educated guess as to what this compiler may look like when, and if, it arrives.

The HP compiler is to be based on an internal HP standard for Pascal. The HP 1000 Pascal compiler, which was recently introduced, is also supposed to follow the internal HP Pascal standard. Any comments I make about Pascal for the HP 3000 are based on how things are done on the HP 1000 [9].

If Pascal/3000 looks very similar to Pascal/1000, we can look forward to an excellent implementation of the language. Pascal/1000 provides features which take advantage of the HP 1000 operating system, yet still retain the "spwirit" of Pascal. Of special importance is the inclusion of a compiler option which permits the compilation of Standard Pascal only. By turning this option on, only Pascal source that followed the standard would compile.

Storage allocation in Pascal/1000 is done in a very flexible manner. Two restrictions of the contributed versions of Pascal are that neither double word-integers, nor sets with greater than 62 elements are allowed. Pascal/1000 permits sets with up to 32767 elements, and only allocates as much storage as is necessary. Similarly, both single- and double-word integers may be declared in a way that is natural for the Pascal language.

Pascal/1000 also allows character strings to be stored as one character per byte, instead of just one character per word. Procedure calls are allowed to external procedures written in either Pascal or HP 1000 assembler.

Assuming Pascal/3000 will follow the lead of Pascal/1000, it should be a very successful compiler. The only problem to which I've no solution is how Pascal/3000 will permit calls to HP subsystems like IMAGE, while working within the confines of Pascal type checking.

Pascal in Applications

This section attempts to describe some of the common pitfalls to watch for when using Pascal. It also does a step by step examination of each of the Pascal types, in the context of their use in future versions of Pascal and with other subsystems. For those just learning Pascal, [18,8,17] may be useful. In particular, [8] gives a complete and readable treatment of Pascal.

Pascal Types

The concept of types in Pascal is one of the most powerful features of the language. Because so much of Pascal operates from the concept of types, it is one of the main areas where problems can occur, especially when dealing with different Pascal compilers.

Integer

The integer type is one of the simplest and most common types in Pascal. The defintion of integer is as follows:

```
    type

        integer = -maxint ..  +maxint;
```

The notation '-maxint .. +maxint' is called a subrange; it defines integer to be a type that can take on values from a lower bound (-maxint) to a higher bound(+maxint). The first question that one usually asks is just how large is maxint? The answer is that it varies from compiler to compiler. In Pascal-V it is +32767, but in the future it will likely be +2147483647. In the first case, storage will be allocated as a single- word integer, and in the second case, storage will be allocated as a double-word integer.

Suppose that the compiler used the second value for maxint. How could you declare a type that only used single-word storage? It could be done as follows:

```
    type

        int = -32767 ..  32767;
```

If the compiler is "smart" in storage allocation, int would only use single-word storage. The int declaration also has the advantage of telling the reader exactly what range any variables of type int should have.

Real

The problem of a value range for real numbers (and the amount of storage to allocate) is similar to the problem of single and double integers. The main difference is that reals cannot be used in subrange notation. The storage allocation and size attributes of reals are left totally up to the implementer of each Pascal compiler.

Since there are many instances where different size reals are needed, there is a general solution. But, this solution is not part of Standard Pascal. The predefined type "Real" is usually taken to mean single precision floating-point numbers, where the size of single precision floating-point numbers is defined for each host machine. On the HP 3000, it is a 32-bit number with approximately seven decimal positions. For larger real numbers, the predefined type "Longreal" is provided, which is usually taken to mean double precision floating-point numbers. On the HP 3000, these would be represented by 64-bit long-real quantities. Currently, all of the HP 3000 Pascal compilers supply only the type Real. Since future compilers will probably allow for larger real numbers, the following declaration could be used to localize the changes at a later date:

```
     type

          Longreal = Real;
```

Whenever "Longreal" became available, this type declaration could be deleted, and the program would just need to be recompiled.

Boolean

Logical values in Pascal are represented by True(1) and False(0). On the HP 3000, Boolean is implemented as a single-word integer. The Pascal-V compiler checks for False = 0 (True = not 0) when examining the value of a boolean expression; but, this may change, and shouldn't be counted on.

Characters

One of the main reasons more people are not using Pascal on the HP 3000 is that characters are packed one per word, instead of the regular one per byte. This means that a Pascal array[1..n] of char cannot be passed to procedures in other languages, without first packing the character array.

The Pascal type packed array[1..n] of char will someday have characters packed one per byte. For this reason, packed should be used wherever possible. It also requires less storage. One word of caution: it is very likely that packed types will not be able to be passed as var parameters to a procedure or function. To change a packed array to an unpacked array, the built-in procedure unpack should be used, and the built-in procedure pack should be used to convert in the other direction.

Set

Sets are one of the more unique concepts available in Pascal. By using sets, it is possible to have a single variable take on more than one value at the same time. This could be very useful in certain application areas (for example, on a customer status field, where a customer could be in two different status classes simultaneously).

Again, there are few guidelines regarding the implementation of sets. The limiting factor in declaring sets is the number of distinct elements that can be in one set type. With Pascal-V, there can be up to 62 elements in a set. Unfortunately, this rules out the following useful type:

```
type

     charset = set of char;
```

With Pascal-V, set types occupy four words of storage, where each bit in the four words represents one of the values in the base type. Other versions of Pascal are more likely to allocate as many words as necessary to represent the base type. This means that writing out set types to files, or calling procedures in other languages with parameters of type set is very unwise. At the very least, such calls should be isolated so that they can be changed easily.

Many implementations of Pascal allow only 48 distinct elements in set types. This should concern anyone who intends to write Pascal software for other machines. For portable software, you should use sets with a small number of elements. While this is unfortunate, it is likely to be the case for many years.

Record

Record structures allow similar things to be grouped together and optionally given a name. This feature is similar to the COBOL level structure. With COBOL, a level structure is allocated space in the order that the various levels are allocated. If we were to map the following record structure into COBOL it would look like this:

```
----------------------------------------------
| integer A | string B | integer C | integer D|
----------------------------------------------

01  RECORD.
    05  A                     PIC S9(4) COMP.
    05  B                     PIC X(10).
    05  C                     PIC S9(4) COMP.
    05  D                     PIC S9(4) COMP.
```

And the equivalent structure in Pascal-V would be:

```
type

     int    = -32767 ..  +32767;
     string = packed array[1..10] of char;

     cobol_record = record
                      d : int;
                      c : int;
```

```
                    b : string;
                    a : int;
                end;
```

Note that the Pascal record structure is exactly opposite to what you would expect. This is because Pascal-V allocates storage elements in reverse order to their declaration. This is implementation-defined; other Pascal compilers may do exactly the opposite, or even something in-between. For this reason, all of your record structures should be declared in one place, using type and $include filee (this facilitates changes, when necessary).

IMAGE/COBOL/Pascal Table

The following table gives equivalences among IMAGE, COBOL and Pascal types. [10] The table assumes the following Pascal types are available:

```
type

     int     = -32767 ..  32767;
     integer = -2147483647 ..  2147483647;
     real    = real;     (* single precision floating-point *)
     longreal= longreal; (* double precision floating-point *)
```

IMAGE	COBOL	Pascal
J1	PIC S9(4) COMP	int
J2	PIC S9(9) COMP	integer
I1	PIC S9(4) COMP*	int
I2	PIC S9(9) COMP*	integer
K1	PIC 9(4) COMP*	boolean*
K2	PIC 9(9) COMP*	integer*
P4	PIC S9(3) COMP-3	packed array[1..2] of char*
P8	PIC S9(7) COMP-3	packed array[1..4] of char*
R2	PIC X(4)*	real
R4	PIC X(8)*	longreal
Z N	PIC S9(N)	packed array[1..N] of char*
X N	PIC X(N)	packed array[1..N] of char

* - Storage is allocated correctly, but the types do not really correspond to the IMAGE types.

Note that the concept of packed decimal does not exist in Pascal. The only way to handle packed type data is to have a set of SPL procedures which do the conversion from packed decimal to some internal format, and the reverse, as well as providing for the actual arithmetic. The most likely data type to hold packed decimal numbers would be a packed array[1..N] of char to hold the packed decimal numbers.

The COBOL zoned-decimal type is not supported directly in Pascal. While the numbers can be read in as a packed array, they will have to be converted to integer by hand. The last byte of the zoned-decimal array will contain the digit value, as well as the sign. For some sample solutions to these problems see [6,7].

Files

In general, the only type of file that is compatible between Pascal and other HP 3000 languages is:

```
     type

          filetype = file of array[1..n] of char;
```

All other file-types are likely to be incompatible, or at best, difficult to interpret. While it is certainly possible to declare a file of the type cobol_record above, the results are not what one expects. In the first place, all of the elements of the record will be reversed, so that the COBOL and Pascal record layouts for the file must be reversed. Further, the string that is part of the record will be packed one character per word with Pascal-V. (Pascal does not currently pack strings, even if packed is included in the declaration.) In order to reduce difficulties with program maintenance, it is recommended that there be one common input and output interface to an external file from Pascal programs. These routines should be included in any program that is to use the file.

IMAGE

All of the problems mentioned above apply even more to IMAGE and Pascal. The layout of a data set and the layout of the Pascal record must be reversed with Pascal-V. In addition, there are currently no double-word integers or packed arrays, so calling IMAGE procedures would be difficult.

As mentioned above, the definition of IMAGE procedures does not agree with that of Pascal procedures. Until a Pascal compiler on the HP 3000 recognizes all of the HP intrinsics, it will be necessary to code calls to IMAGE in some other way. The suggested solution is to provide one SPL procedure for each data set that is to be used in Pascal.

Each SPL procedure would have a mode parameter (read, chainread, write, update, etc.), as well as the other IMAGE parameters (the base name, the status area and a buffer). It would make sense for the SPL interface to use the "@" list in all DBGET calls (as well as assuming the set name). Each interface routine would transform the IMAGE record structure into the equivalent Pascal structure (i.e., unpack character arrays, reverse the order of records, etc.). This is a general solution which should work for different versions and implementations of Pascal on the HP 3000.

Performance of Pascal Programs

Introduction

One of the most frequently-asked questions about the implementation of a programming language is: how fast are the programs it generates? On a machine like the HP 3000, where many factors contribute to the performance of a program, it is a difficult question to answer.

Programs written in the host programming language are generally the fastest, because the host programming language was custom-designed for the particular machine. Most of the constructs in the language translate directly into hardware instructions. This is true of SPL on the HP 3000. Other languages (like Pascal or COBOL, designed for use on many different machines), must translate the language constructs into a combination of hardware instructions and procedure calls to a run-time library. Since the procedures in the run-time library are generally slower than hardware instructions, there are certain language constructs to watch out for.

What to Watch Out For

Not all high-level language constructs are slow. Many of the language features will be as fast as a lower-level language like SPL. On any particular machine there are certain things to watch out for and avoid, because they are unusually slow. Pascal is no exception to this rule.

One of the most common operations is to open a file and read it sequentially from beginning to end. In Pascal, this is often accomplished by using the built-in procedures reset and read. The reset procedure opens the file, and the read procedure reads the file as if it were one long string of characters. Since files are normally organized into records, it seems obvious that reading the file one character at a time will be inefficient.

An Example

Pascal provides another built-in procedure, get, which will read a file one record at a time, if the file is declared correctly in the Pascal program. In order to test the time necessary to do a sequential read, the following Pascal declaration was made:

```
     type

          filetype = file of array[1..80] of char;
```

The file input was declared to be of this type, and get was used to read the file sequentially. Figure I and II give a graphic demonstration of the elapsed and CPU time of four programs which read a sequential file.

Elasped Time vs. Number of Records Read

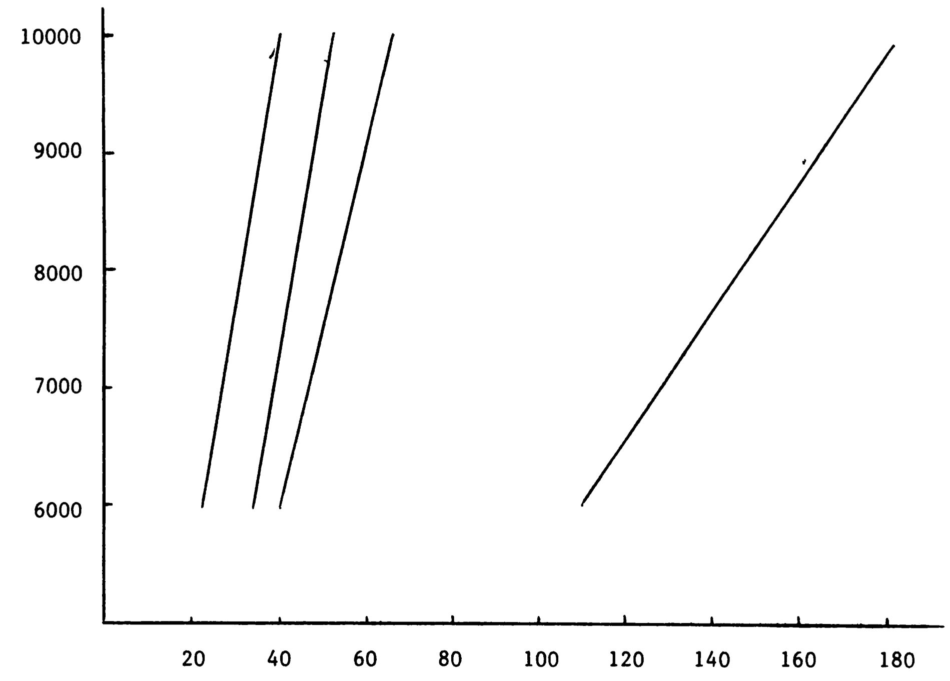


Figure I

CPU Time vs. Number of Records Read

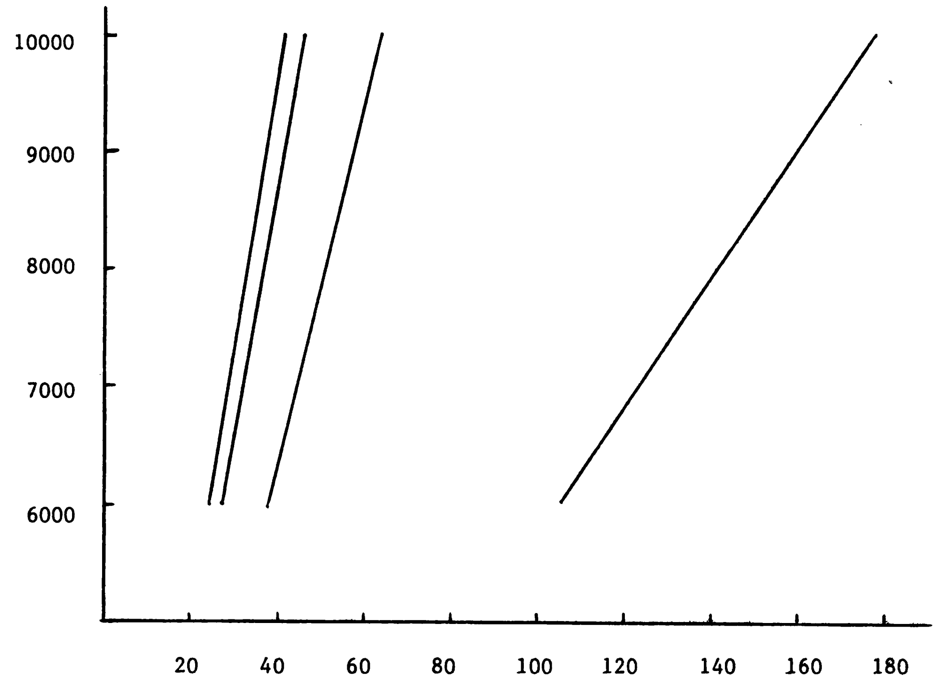


Figure II

One program was coded in SPL, using the FREAD intrinsic. Another was coded using the COBOL read statement. One Pascal program was coded using get, and another was coded using read. As expected, the SPL program was the fastest, followed by the COBOL program. The Pascal program using get was slower than the COBOL program, but substantially faster than the Pascal program using read.

Explanation of the Results

SPL was the fastest, because it communicates with the operating system directly, doing a FREAD of 80 bytes and checking the condition code for end-of-file.

The COBOL program interfaces to FREAD through the COBOL run-time support. The run-time support is general-purpose; it must handle all different file types and file sizes and it also does more error checking. The extra time consumed in the run-time support routines makes the COBOL program slightly slower than the SPL program.

The Pascal program suffers all of the same problems as the COBOL program. The Pascal run-time support must be prepared for all situations. In addition, the buffer that is read for each record must be unpacked, since the current version of Pascal does not support packed files. The extra time to do the unpacking shows up dramatically in Figure II, where there is a large difference in CPU time between the COBOL program and the Pascal program using get.

Finally, the Pascal program using read was much slower than any of the other programs. The reason for this is that an extra procedure call is made for every single character in the file. All of this extra overhead results in a Pascal read being much slower than a Pascal get. (The only thing that could be slower is the FORTRAN formatter with a format of 80A1, which calls the formatter for each character).

The Moral

Each implementation of a high-level programming language has certain constructs which are inefficient. For example, COBOL programmers use COMP variables when the value to be represented is less than ten digits long, and they avoid the use of the COMPUTE verb, since it is very slow in COBOL68. Pascal programmers should be aware of certain Pascal constructs that are slow on the HP 3000. In particular, this example shows that Pascal get should be preferred over Pascal read, when sequentailly reading a file. See Appendix II.

Appendix I

Making Pascal Portable - Three Examples

I. Pascal-S

Pascal-S is a subset of standard Pascal. [18,19] The compiler itself is a complete system which compiles the Pascal program, and, if there are no errors, executes the program (interpretively). Pascal-S was written in Pascal by Wirth.

The major problem in getting Pascal-S running on the HP 3000 was the difference between the character set on the CDC series of machines and the character set of the HP 3000 (ASCII). In the CDC character set, blanks collate after letters; but, in the ASCII character set, blanks collate before letters. Also, the internal numeric representation of characters differs between the CDC and the HP 3000, and this caused further errors in the compiler.

The lesson to learn from this is that writing portable programs, even in Pascal, takes some thought and effort. Even when using Standard Pascal, problems can arise. One tip: make no assumptions about character sets, as they vary widely from machine to machine.

II. PROSE

PROSE is a text formatter published in Pascal News No. 15 [16]. It is written entirely in Standard Pascal and pays particular attention to the character sets of different machines. PROSE stores all text internally using the ASCII conventions, and each implementation (of PROSE) must have routines to convert from the external character set to the internal one.

PROSE is approximately 3500 lines long, and is just now being completed on the HP 3000. The main problems encountered in implementing PROSE were finding and eliminating typing mistakes, and understanding the conversion from the external character sets to the internal set. Even though the external and internal character sets are the same on the HP 3000, it took some time to find and change the conversion routines accurately. The coding of these routines assumed an understanding of how the CDC character sets work, since the version of PROSE written in Pascal News No. 15 was for a CDC computer.

Another problem encountered with PROSE was the definition of carriage control on output files. Many implementations of Pascal take the first character written to each line of an output file as the carriage control character. For example, to write a page eject, the following Pascal code would be written:

```
writeln; writeln('1');
```

The "1" in the second writeln would be interpreted as a carriage control character, which, on most line printers, causes a page eject. Pascal 2.8-V uses standard built-in procedures to do carriage control. The procedure page(output) causes a page eject on the file output. Because PROSE was implemented using the first method, it was necessary to change each carriage-control write statement to a similar or equivalent Pascal 2.8-V statement. While this is relatively straightforward with page ejects, it is much more difficult with overprint and underlining.

The lesson to learn from this is that the meaning of carriage control on output files is implementation-defined. In order to make a Pascal program portable, all carriage control should be done by a few, easy-to-modify procedures, rather then by any implementation features. These procedures can then be modified for each Pascal installation.

III. LISP

LISP is a small implementation of the interpretive language LISP. It was written at the University of British Columbia using Pascal/UBC, an extended version of Pascal. There were only two major problems in getting LISP to work on the HP 3000.

The first was that Pascal/UBC runs on an AMDAHL/V6, using the EBCDIC character code. On most IBM terminals that use EBCDIC, there are no "]", "]" or "^" characters. Because of this, Pascal/UBC accepts "(." as "[", ".)" as "]" and "@" as "^". Each of these character sequences had to be converted to their ASCII counterparts.

The second problem centered around the use of the Pascal *forward* declaration. Since LISP is essentially recursive in nature, all of the procedures of LISP were declared *forward* to avoid the mutually recursive problem. The use of *forward* in Pascal is not clearly defined; but most implementations of Pascal require that the parameter list to a *forward* *procedure* or *function* be declared when the *procedure* or *function* is delcared *forward*, and that the actual parameter list be left off when the *procedure* or *function* is actually declared.

Pascal/UBC permits an extension whereby the parameter list may be declared both when the *procedure* or *function* is declared *forward* and when the actual *procedure* or *function* body is declared. Pascal 2.8-V does not allow this extension, so each actual declaration of a *procedure* or *function* had to have the parameter list deleted.

The lesson to learn from this is that non-standard Pascal features must be avoided, if a Pascal program is to be made portable. Most Pascal compilers have a compiler option which permits only standard Pascal to be compiled. Before declaring a Pascal program portable, be certain to turn the standard compiler option on, recompile and rerun the program, to ensure that it is free from any non-standard Pascal features.

Appendix II

Implementation Notes on Pascal-V

Introduction

This appendix is intended for the interested reader who wishes to know more about the actual implementation of Pascal. It assumes that the reader is already familar with Pascal, the HP 3000 operating system, SPL/3000, and the HP 3000 instruction set [18,15,5,1,11,12,13].For those interested in compiler design in general, [20] gives a readable introduction to recursive-descent compiling, and [2] covers the general topic of compiler design thoroughly.

Basic Compiler Design

The compiler itself is written in Pascal. It uses recursive-descent compiling techniques. The original compiler compiled code for a hypothetical stack machine [15]. The principal modules and their functions are as follows:

nextsym - reads the input text and returns the next lexical token. This module is also responsible for output, including error messages, and all input, as well as $INCLUDE files. If an identifier is encountered, it also looks up the identifier to see if it is a reserved word, but it does not check to see if the identifier was already defined by the user.

table management - these routines store all user-defined identifiers into the various compiler tables, and provide lookup of identifiers from the same tables. Table storage is organized as an unbalanced binary tree for each program level [20,14].

block - this is the syntax recognizer of the compiler. It also generates the "pcode" pseudo-instructions, and handles all parsing and semantic definition. The structure of block is broken down as follows:

```
     label          label-declaration-part
     const          constant-declaration-part
     type           type-declaration-part
     var            variable-declaration-part
     procedure-and-function-declaration-part
     statement-part
```

The statement-part handles all of the various statements available in Pascal. The structure of block follows the syntax diagrams in the User Manual and Report [18].

initialization - this module handles all static and dynamic initialization. This includes the reserved words, input/output and nextsym variables, as well as the predeclared types and procedures. Predefined procedures and types are stored just like regular user identifiers, so that they may be overridden by the Pascal programmer.

SPL code generation - these procedures translate the "pcode" statements into valid SPL statements. A combination of regular SPL, ASSEMBLE and TOS is used in the translation to SPL. Note that the basic code generation is still based on the "pcode" machine. However, in Pascal-V the "pcode" translation stage, which generated an external file and ran the program ASSM, has been eliminated. The code that is generated still has some of the basic deficiencies of the "pcode" machine.

Stack Frames

Usually, when implementing Pascal, a major problem is the provision of the procedure call and return mechanism. Fortunately, most of the tools required are already available on the HP 3000, in the form of the PCAL and RETURN statements of the HP 3000 instruction set.

When a procedure is called using PCAL, a four-word stack marker is loaded onto the top of the stack before control is transferred to the new procedure. In SPL, this stack marker is sufficient to save and restore the entire SPL environment. However, SPL provides only single-level addressing, while Pascal provides multiple-level addressing. To implement "up-level" addressing, each Pascal procedure call loads a five-word "stack frame" onto the top of the stack, instead of a four-word stack marker. The stack frame is structured as follows:

```
        |------------------------------|
  Q-4   |  Address of previous level   |
        |------------------------------|
  Q-3   |        Index Register        |
        |------------------------------|
  Q-2   |        Return Address        |
        |------------------------------|
  Q-1   |        Status Register       |
        |------------------------------|
  Q-0   |           Delta Q            |
        |------------------------------|
```

This is the Pascal stack frame. Words Q-3 through Q-0 are loaded automatically by the PCAL instruction. The DB-relative address of Q-0 of the previous level is placed onto the top of the stack by the Pascal compiler, before the PCAL is executed.

Since all procedure levels are established at compile time, it is possible to compute the address of a variable that is neither local nor global. The Pascal program walks back through the correct number of stack frames to reach the beginning of the level where the variable to be used is located. Then the Pascal program works forward from the computed address to obtain the actual address of the variable in question.

Parameter Passing

Most parameters are passed to Pascal procedures in the same way that parameters are passed in SPL. For reference parameters, the word address of the parameter is loaded onto the top of the stack before the procedure is called. Once the procedure is called, it uses Q-indirect addressing to obtain the value of the reference parameter.

Simple scalar value parameters are also passed as in SPL. The actual value of the parameter is loaded onto the top of the stack before the procedure is called. Set, record and array value parameters are permitted in Pascal, but not in SPL. The principle remains the same: space is reserved on top of the stack for the value parameter, then the actual value of the set, record or array is copied into the reserved space. Once the procedure is called, it uses Q-negative addressing to obtain the value of any one of the variables.

Each Pascal procedure must know exactly how many words of storage are taken by both its value and reference parameters. When the Pascal procedure returns to the invoking routine, it deletes all of its parameters from the stack.

Function Return

The last problem involved in calling Pascal procedures or functions is where to leave the result of a function when it returns. The calling routine reserves enough space for the result on top of the stack, before loading the parameters or the stack frame.

Since only scalar, subrange or pointer types may be returned as functions, the compiler reserves either one or two words on top of the stack for the function result. Two words in the case of reals, and a single word in all other cases. Once the function is called, it uses Q-negative addressing to store the function result. Upon return, the space for the result is the only space not deleted from the stack. Therefore, the result is always on top of the stack after a function call.

Addressing

In general, Pascal uses DB+ addressing for global variables, Q+ addressing for variables declared at the current level, and Q-negative addressing for procedure or function parameters. When a variable from another level, other than global, is needed, the X register is used.

One problem encountered in Pascal that is not present in SPL is that Q-relative addressing is only allowed from Q-63 to Q+127. Since value parameters of unlimited size can be passed in Pascal, and it is easy to declare a record structure that is more than 128 words long, provision had to be made for larger Q-relative addresses. In order to work around this problem, Pascal uses a combination of the Q register and the X register.

Whenever a reference is made to a local variable whose address is larger than Q+127, the following algorithm is used.

The nearest multiple of 128 is stored in the X register. The difference between the address of the variable and the value in the X register is then used as the Q-relative address. Any reference instruction then uses combined Q-relative and indexed addressing. Q-negative addressing is similar, except that the value stored in the X register is a multiple of 64, because Q-negative addressing only allows for addresses up to Q-63.

For example, take a variable whose address would be Q+130, and assume that a single-word load to the top of the stack was to be done. The following SPL code would do the actual load:

```
X := 128;                 <<CLOSEST MULTIPLE OF 128>>
ASSEMBLE(LOAD Q+2,X);     <<130-128 = 2>>
```

A similar situation exists with DB-relative addressing, but the limit is DB+255. The same solution is used as in Q-relative addressing, except that the closest multiple of 256 is loaded into the X register, and the difference between the desired location and the X register is used as the DB-relative address. A combination of DB-relative and indexed addressing is then used to reference the global variable.

Why Didn't Pascal Work Under Athena?

The contributed Pascal compilers would not run with the Athena (2011) release of MPE, because Qinitial was two words higher in the stack. If the Pascal compiler used DB-relative addressing for all of its global variables, why would it matter where Qinitial was?

The answer to this question is that IF Pascal always used DB-relative addressing for global variables, it would not matter. Unfortunately, over time, and through various changes to the compiler, some references to global variables became Q-relative. Since the structure of the initial Pascal stack was precisely defined, it did not matter whether DB-relative or Q-relative addressing were used for global variables, so long as the addresses were computed properly at compile time. As long as Qinitial was always exactly four words higher in the stack than secondary DB, there was no problem.

With the Athena MIT of MPE, Qinitial was six words above secondary DB (or more, if INFO= was used in the :RUN command). But Pascal continued to use Q-relative addressing on some global variables, and, under Athena, these were not the correct variables. The fix to the compiler consisted of tracking down every global reference and ensuring that it used DB-relative addressing. Once this was done, Pascal was no longer concerned with the position of Qinitial, and all Pascal programs could run on any release of MPE.

Dynamic Memory-Allocation

One of the most powerful Pascal features, for both applications and teaching, is the concept of pointer-variables and dynamic memory-allocation. Whenever the built-in Pascal procedure new is used, storage must be allocated for the new variable. The amount of storage allocated is determined by the type of the object passed as a parameter to new.

Pascal maintains an area called the heap. The heap is an area of memory, logically separate from the "stack", which can be used for dynamic memory-allocation. On the HP 3000, the heap is implemented as the DL area. A dynamic counter of the current size of the heap is maintained in every Pascal program. When the <u>procedure</u> new is used, the counter is incremented, and the resulting address is stored in the variable passed to new. If the counter has passed the current limits of the DL area, the DL area is expanded by 1024 words (using the DLSIZE intrinsic).

In order to give the Pascal programmer some control over the size of the heap, two built-in <u>procedures</u>, mark and release, are provided. Mark stores the current size of the DL area in the variable passed to mark. This same variable is passed to the <u>procedure</u> release, which shrinks the DL area back to that original size. If the value of the variable used to mark the heap is changed before the call to release, the DL area will be shrunk by an unpredictable amount.

Run-Time Libarary

The run-time library gives Pascal significant power to deal with files, and other features of the HP 3000 which are external to the Pascal program. From a Pascal program, it is a simple task to write out an integer, real or character value, especially when compared with the effort needed to do the same thing in SPL. The run-time support allows Pascal this "friendly" type of communication with files.

The run-time library consists of approximately 1800 lines of SPL code. The main entry points to the run-time library, and their functions, are listed below:

PASCAL'FERR	Writes out Pascal file error messages.
PASCAL'ERROR	Writes out general Pascal run-time errors.
PASCAL'CLOSE	Closes an open Pascal file.
PASCAL'CLOIO	Closes the standard files INPUT and OUTPUT.
PASCAL'OPEN	Opens a Pascal file for either read or write.
PASCAL'GET	Gets the next record from the file. This procedure is called when the built-in procedure get is used.
PASCAL'GCH	Returns the next character in a file buffer.
PASCAL'PUT	Writes out the current file record to the file. This procedure is called when the built-in procedure put is used.
PASCAL'PCH	Adds a character to the output file buffer.
PASCAL'POS	Positions a file to a particular position.
PASCAL'FILE'POS	Returns the current file position.
PASCAL'RESET	Equivalent to the built-in procedure reset.
PASCAL'REWRITE	Equivalent to the built-in procedure rewrite.
PASCAL'CY	Pascal Control-Y trap.

PASCAL'GETHEAP	Expands the DL area by 1024 words.
PASCAL'INIT	Initializes the Pascal environment and opens the files INPUT and OUTPUT.
PASCAL'RDI	Reads an integer from a text file.
PASCAL'RDR	Reads a real from a text file.
PASCAL'WRI	Writes an integer to a text file.
PASCAL'WRR	Writes a real to a text file.
PASCAL'WRS	Writes out a string to a text file.
PASCAL'WRB	Writes out a boolean value to a text file.
PASCAL'DATE	Obtains and formats today's date.
PASCAL'TIME	Obtains and formats today's time.

These functions are stored in the RL file and are copied into the Pascal program by means of the RL= parameter of the :PREP command.

File Buffers

Each of the file-handling procedures above is passed a pointer to an array which acts as the Pascal file-control block. The structure of the file-control block is as follows:

Word	Use	
1	MPE File Number	
2	Current Record Length of the Buffer	
3	Maximum Record Length of the File	
4	Control Bits	Carriage Control
5	Length of the Buffer	
6	Current Character Position in the Buffer	
7	Current Character If TEXT File	
8	Next QEDIT Block	
9	Next QEDIT Index	
10	QEDIT File Flags	
11	QEDIT Linenumber of Current Line	
12	Variable Length Buffer Size depends on the type of file. For TEXT files, this buffer is 128 words long.	

When a Pascal program uses a read statement, characters are not obtained one at a time. Instead, a buffer containing the the line most recently read from the file is used, along with a character position in the buffer. If all of the characters in the buffer are read, the next line of the file is read automatically. Since only 256 bytes are reserved for the file buffer of a text file, this is the largest record size that the actual MPE file attached to the text file may have.

When a file is declared in a Pascal program, the local storage for the file-control block is allocated at the point where the file is declared. This method doesn't work for the standard files input and output, since they are predefined automatically in every Pascal program. To get around this problem, the file-control blocks for the standard files input and output are declared in the DL area. When the Pascal program is first run, these file-control blocks are allocated and initialized. One of the reasons that a Pascal procedure cannot be called from another language is that this allocation and initialization of the input and output files would not be done properly. If the Pascal program does any reading from input or any writing to output, then it would fail, due to the lack of the correct file-control blocks.

Summary

The only software more complicated than compilers are operating systems; yet compilers are vital to our continued use of computers. Pascal-V is approximately 7500 lines of Pascal, and it has been modified by at least three different people. Despite this, many useful programs have been developed using Pascal-V. The Pascal-V compiler is a reasonably solid and reliable system, which provides a good base for incremental enhancements in the future.

Appendix III

Sample Data From the Sequential Read Test

I. Elasped Times

No. Records	SPL	COBOL	Pascal/Get	Pascal/Read
6000	26.34	34.41	39.77	109.44
7000	30.69	40.13	46.31	127.55
8000	35.01	45.78	52.87	145.79
9000	39.36	51.50	59.44	164.03
10000	43.70	57.20	65.99	182.15
y-int	-70.97	-42.82	-68.36	-14.94
slope	230.47	175.59	152.60	54.97
Corr. Coeff.	0.9999	0.9999	0.9999	0.9999

II. CPU Times

No. Records	SPL	COBOL	Pascal/Get	Pascal/Read
6000	25.34	28.03	38.62	108.03
7000	29.56	32.69	45.01	125.98
8000	33.78	37.35	51.48	144.03
9000	38.00	42.01	57.93	162.11
10000	42.21	46.66	64.35	180.04
y-int	-8.06	-18.03	4.06	4.56
slope	237.08	214.68	155.33	55.50
Corr. Coeff.	0.9999	0.9999	0.9999	0.9999

All of the data was compared using a least squares fit of a linear line. The resulting slopes and y-intercepts are listed. Notice that all programs had a near perfect correlation coefficient; this indicates that the file system has a linear increase in time as the number of records sequentially read increases.

Bibliography

[1] Addyman, A. M.
"A Draft Proposal for Pascal"
SIGPLAN Notices Vol. 15 No. 4, April 1980
Association for Computing Machinery,
1133 Avenue of the Americas, New York, NY 10036

[2] Aho, Alfred; Ullman, Jeffrey
Principles of Compiler Design
Addison-Wesley, Don Mills, Canada, 1977

[3] Earls, John
"Pascal for the HP 3000"
HPGSUG Journal, Vol. 1, No. 5

[4] Fraley, Robert
"Pascal-P on the HP 3000"
HPGSUG 1980, San Jose Proceeedings

[5] Fraley, Robert
"The Pascal Programming Language"
HPGSUG 1980, San Jose Proceeedings

[6] Green, Robert M.
SPL Aids, Sofware Package
Robelle Consulting Ltd.

[7] Green, Robert M.
SPL/3000 in a Commercial Installation
Robelle Consulting Ltd.

[8] Grogono, Peter
Programming in Pascal
Addison-Wesley, Don Mills Canda, 1979

[9] Pascal/1000 Programmer's Reference Manual

[10] IMAGE Data Base Management System Reference Manual

[11] SPL/3000 Reference Manual

[12] HP 3000 Machine Instruction Set Reference Manual

[13] MPE Intrinsics Reference Manual

[14] Knuth, D. E.
The Art of Computer Programming, Vol. III
Sorting and Searching
Addison-Wesley, Reading, Mass, 1973

[15] Nori et. al.
The Pascal(P) Compiler: Implementation Notes,
Revised Edition
Order from William Waite
Software Engineering Group

Electrical Engineering Department
University of Colorado
Boulder, Colorado 80309

[16] Pascal News
c/o Rick Shaw
Digital Equipment Corporation
5775 Peachtree Dunwoody Road
Atlanta, Georgia 30342
Subsrciption rates are $6.00 per year

[17] Pollack, Bary and Greer, David
A Programmer's Introduction to Pascal
Available from Robelle Consulting Ltd.

[18] Wirth, Niklaus; Jensen, Kathleen
Pascal User Manual and Report Second Edition
Springer-Verlag, New York, 1974

[18] Wirth, Niklaus
Pascal-S: A Subset and its Implementation
See order information above

[19] Wirth, Niklaus
Systematic Programming: An Introduction
Prentice-Hall, Englewood Cliffs, New Jersey, 1973

[20] Wirth, Niklaus
Algorithms + Data Structures = Programs
Prentice-Hall, Englewood Cliffs, New Jersey, 1976

THE FIELD SOFTWARE COORDINATION PROCESS

by

Brian M. Perkin

Systems Engineer, HP

Over the last few years, growth of the Hewlett-Packard 3000 customer base, through strong sales and successful installations, has caused substantial growth in Hewlett-Packard manufacturing divisions and in the field support operations. In addition, Hewlett-Packard has developed many new and innovative software and hardware products.

With this exciting growth, Hewlett-Packard is still very concerned about its customers. We want to be sure that a product will work correctly and reliably the first time it is installed on a customer site. Many of these products have complex interactions with other products, making quality assurance a difficult task. In order to maintain Hewlett-Packard's high standards of reliability, a new program has been created in cooperation with the Computer Support Division.

Field Software Coordinators have been designated for each sales/support area. Generally, a software coordinator will use local and factory resources to verify problem solutions, identify potential problems and weaknesses and to act as a distribution and integration point for software distribution. Software coordinators also have responsibility for training and acting as a resource center to other HP support groups.

FULL TEXT WILL BE DISTRIBUTED AT THE SESSION

DISTRIBUTED PROCESSING IN HIGH VOLUME TRANSACTION PROCESSING SYSTEMS

A paper presented to the 10th International Meeting
of
The Hewlett-Packard General Systems Users Group at Orlando, Florida, U.S.A.
April 27 through May 1, 1981

By

Thomas L. Fraser
Systems Research Inc.
2400 Science Parkway
Okemos, Michigan
USA

I. Introduction

Distributed processing concepts are frequently at the center of the implementation of transaction processing systems. The strengths of the HP3000 in these types of systems make it a desirable alternative from several standpoints. However, limitations of the HP3000 require that new approaches be used if successful high volume transaction processing systems are to be implemented.

One innovative solution applies the theory of distribution to the HP3000 itself. By vertically distributing the processing to be done on the HP3000, it becomes a viable solution to a much broader range of business problems. Systems Research Incorporated has effected this vertical distribution through the development of an HP1000-based front-end to the HP3000.

The SRI front-end dramatically expands the power of the HP3000 both in data communcations and data processing. Moreover, it makes new options available both in on-line linkages to other processors in distributed systems, and in other areas of data communications and terminals.

II. A Definition of Transaction Processing

A TRANSACTION is a phenomenon of the real world. It is a logical unit associated with an event or series of events in a system external to the computer. When one makes an airline or hotel reservation, a transaction occurs. Invoicing a customer, booking a suspect into jail, or inquiring on the status of inventory are all examples of transactions. Many transactions have nothing whatsoever to do with computers (e.g., buying the morning newspaper from a vendor on the street).

As can be seen from the examples, transaction processing is nothing new. Transactions have been occurring since the beginning of civilization, have been processed manually for thousands of years, and have been batched and processed on computers for decades. What is relatively new, is on-line transaction processing on mini-computers. When a system is on-line, transactions are processed individually as they occur in the real world. One can readily see that having an airline reservation system on-line is desirable, and there are many other on-line systems which will provide benefits that are not possible in batch systems. In the past it has been impossible or too costly to put most systems on-line. Today the improvements in hardware and software technology have changed this a great deal. Specifically, the HP3000 provides an environment which can be used to develop very high performance and effective on-line transaction processing systems.

III. The Transaction Monitor

A transaction, being an external occurrance, must be input to the computer system before it can be processed. In an on-line system this is usually accomplished through a terminal, frequently a CRT terminal. A transmission from a terminal to the host computer is defined a MESSAGE. A transmission to the computer and a response back to the terminal is a MESSAGE-PAIR. A transaction may involve one or multiple messages or message-pairs. Most transactions which are entered from CRT type terminals will involve multiple messages, and are frequently referred to as interactive.

There are several functions which are common to transactions in general, and do not depend on the specific application itself. Therefore, it is possible for these functions to be handled by software or firmware external to the application program. Examples of these functions are control and management of the network, data communications between the terminals and the host computer, logging of transactions, queueing of messages, routing messages, controlling access to the computer, handling screen formats, spooling of output print files, and providing facilities for restart and recovery in the event of a system failure. Some of these functions may be performed by the host operating system, but frequently these functions are grouped in a program (or programs) which is logically between the terminals and the application program, and is called a transaction monitor.

IV. Transaction Processing vs. Timesharing

In timesharing while user capabilities can be restricted, in principle users have broad latitude in what they can do, and the activities of the others on the system are transparent. The system is user and terminal oriented and provides an environment where each user can be simultaneously performing totally different tasks.

In transaction processing user capabilities are pre-defined and very limited. The system is transaction oriented and provides an environment where several users can perform the same or similar tasks simultaneously. Our earlier example of the airline reservation system illustrates; the reservation clerks are executing very similar transactions and are very restricted in capability (e.g., none of them can execute a compile or use Query).

The HP3000 uses MPE, a timesharing operating system. MPE associates a user with a stack. There is a one-to-one correspondence between users (terminals) and stacks. Each time another user logs on, another stack and another process provides for maximum flexibility and power for the users, but severely limits the number of users (terminals) because of the load that is placed on the resources of the HP3000.

V. Performance Limitations in the HP3000 Environment

Performance will vary from system to system depending on many factors such as what needs to be done, how well software is written, how well data structures are defined, etc. but some general measures are useful.

There are many ways to measure performance, but in a transaction processing environment it is the number of transactions which can be handled per unit time and the number of concurrent users which can be supported with acceptable response time.

The performance limitations of the HP3000 as measured by these two means are discussed in the "Hewlett-Packard Performance Guide"[1]. An approximation of two charts from that publication are included for convenience (Figs 1 and 2). Based upon these data, even a large Series 44 (with 4MB) reaches capacity at a transaction rate of 10,000/hour, and response time gets bad at about the 60 terminal level. For a 2MB Series III with MPE III, these numbers are about 4,000 transactions per hour and 30 terminals. High performance is defined for use here as that beyond what the native mode HP3000 and MPE can yield.

The resources which limit performance can be generally classified as CPU, memory, datacomm capacity and available disk accesses. (They are not independent of one another. They are considered separately here for simplicity as we are only illustrating a point, not presenting a study on performance). If we are to achieve higher performance, we must either add to these resources, or make better use of those already available. As will be seen, the vertical distribution of some functions to a front-end will accomplish both.

Footnotes:
1 Hewlett-Packard Performance Guide, March 1981

Figure 1

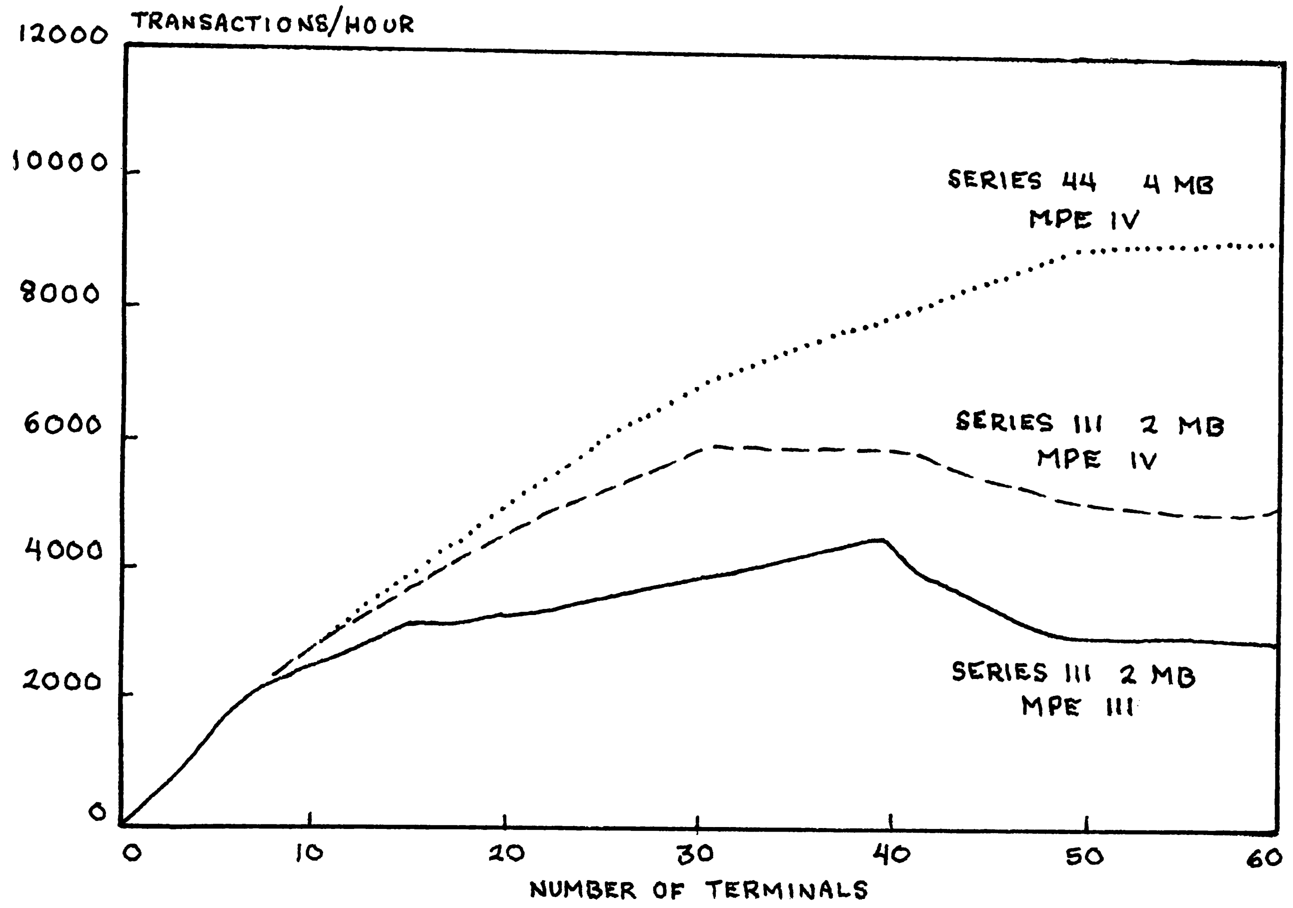
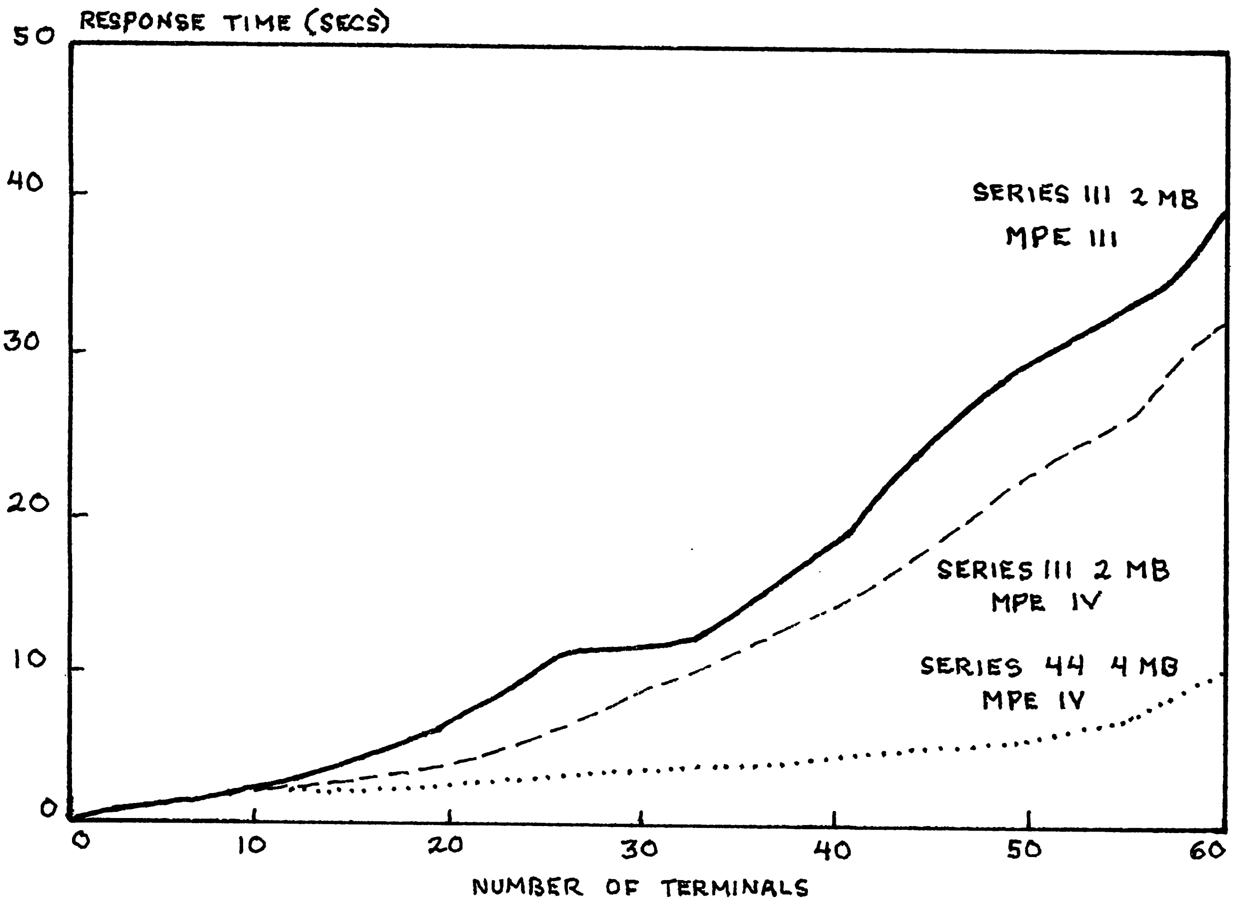


Figure 2



VI. Vertical Distribution Vehicle

The logical candidates for distribution to the front-end are of course those functions referred to earlier which are common to most all transactions. That is precisely what was undertaken.

The HP1000-E was chosen over alternatives because of reliability, price/performance, vendor quality, and support for user microcoding. All software and firmware was written specifically to perform the datacomm and transaction monitor functions. In effect, a specialized and dedicated operating system was written.

The HP1000 is interfaced to the HP3000 through the Universal Interface Card for Series III hosts and through HP-IB for Series 30, 33 and 44. Although the mechanics of the two interfaces differ, both achieve very fast and broad transfer of data between host and front-end with very little demand on host resources. An HP7906 disk is used by the front-end for large operating system tables and queue space, but because the HP1000 has 256KB (or more) of memory, queueing is usually done in memory. The datacomm line controllers complete the front-end hardware, and since they are also HP manufactured the entire system can be maintained by HP field personnel. Up to 240 high speed asynchronous lines or up to 70 high speed synchronous lines can be supported.

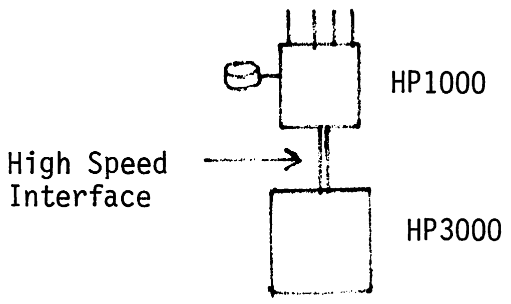


Figure 3: Basic Configuration

VII. Message Flow and Application Structure

When timesharing was discussed earlier, we saw that users and terminals were associated with processes and stacks. This is neither desirable nor necessary in a transaction processing environment. It impedes performance, control, and software development.

The front-end operating system associates transactions with processes. All executions of a particular transaction type can be routed to a single (or a small number of identical) process(es). The front-end treats all messages independently and transmits only data to the host. All forms, control characters and etc. are stripped off or inserted by the front-end. Since the application sees only data, it is as if there were only one terminal of unspecified type in the network.

A typical transaction will begin with a message from a terminal which begins with a slash, followed by a transaction mnemonic, followed by optional data. The front-end will check for transaction validity, as well as user and terminal authorization. If all checks-out, routing information is extracted from transaction tables, and the host process is checked for availability. If necessary the message is queued. When available, the message (data only) is sent to a switch program in the HP3000, which moves it to an extra data segment, and activates the application process. The user and station are logically linked to the process, and will be until the transaction is completed.

For detailed descriptions of structure, refer to the MCS3000 programming manual[2].

VIII. Features and Performance

The division of labor and specialization of resources, together with the addition of resources resulting from the vertical distribution, and the adoption of a transaction oriented architecture greatly expand the capacities and capabilities of the HP3000. Moreover, flexibility is enhanced and many otherwise unavailable features are brought to the HP3000.

Footnotes:

2 MCS3000 Programming Manual, Systems Research Incorporated, June 1980

As measured earlier, performance is several times what an HP3000 alone can do. Transaction rates of up to 40,000 per hour have been achieved on Series IIIs with MPE III, and although no Series 44s have been installed with front-ends at this writing, even higher rates are likely then. The maximum number of terminals currently installed on a single front-end is about 600 (see Appendix 1), but many times that could be supported.

Other performance limitations are 250,000 bits per second throughput, up to four simultaneous hosts (HP3000's or Burroughs Medium Systems) and up to 500,000 characters per second transfer between host and front-end.

Front-End Resident Messages Control System - The front-end message control system offers: Transaction-based message routing and/or predefined message linkage support; application data-save areas; a five-level user and terminal security system; front-end forms files, with automatic data-fill; application program control; and dynamic network management. With the front-end host resources are directed toward the support of user applications, not data communications or message control functions.

Protocol Flexibility - Supports most standard data communication protocols; also, SRI can develop specialized protocols tailored to specific user requirements. This flexibility allows price and capability to become the key criteria in selection of terminals and remote devices. (See Appendix 2.)

Application Independence - The front-end totally insulates application programs from data communication and message control functions. The front-end also provides a prototype application handler for simplified development of new application programs. Together, these yield efficiencies in application program development and maintenance.

Device Independence - The front-end control character mapping and automatic forms-fill features allow device-independence at the application program level. Specific application programs can be developed without regard to the characteristics of a given terminal. Device-independence reduces application program development time; in addition, changes in terminal hardware no longer obsolete application programs.

Dynamic Network Configuration and Maintenance - With the front-end, the network is defined using on-line transactions. Most network parameter changes can be made dynamically; network down-time thus is eliminated for recompilation of front-end or host application programs. Refer to MCS3000 Reference Manual[3].

Other Features - The front end provides: Audit and recovery; on-line forms generation and maintenance; remote print spooling; and on-line access to summary network statistics.

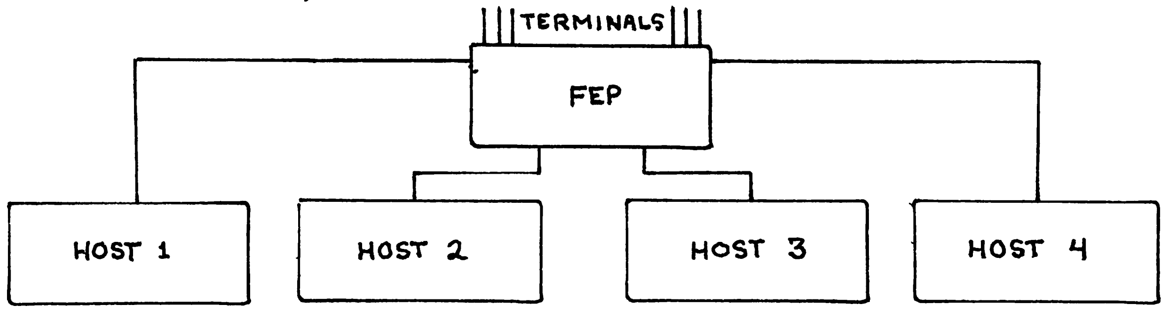
Multiple Configurations for On-Line Processing

The basic front-end is configured with a single front-end processor (FEP) interfaced to a single host computer; alternative configurations include:

- Simultaneous interfaces to as many as four hosts.
- Multiple front-end processors interfaced to a single host.
- Redundant front-end processor configurations for single- or multi-hosts.

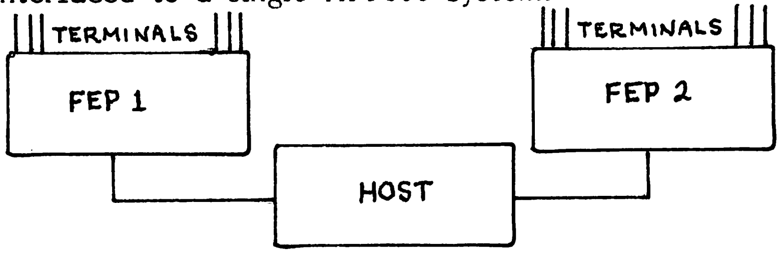
Single FEP to Multiple Host Configurations

The front-end provides an integrated front-end data communication and message control system for the multiple host environment. Any combination of HP3000 Systems can be interfaced to a single front-end. Alternatively, one or more Burroughs Medium systems can be used in conjunction with the HP3000 host(s).

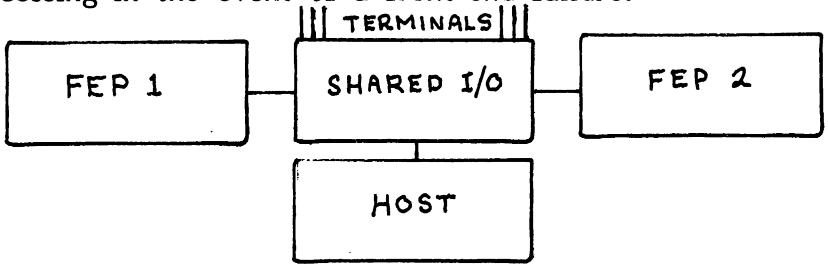


Footnotes:
[3] MCS3000 Reference Manual, Systems Research Incorporated, June 1980

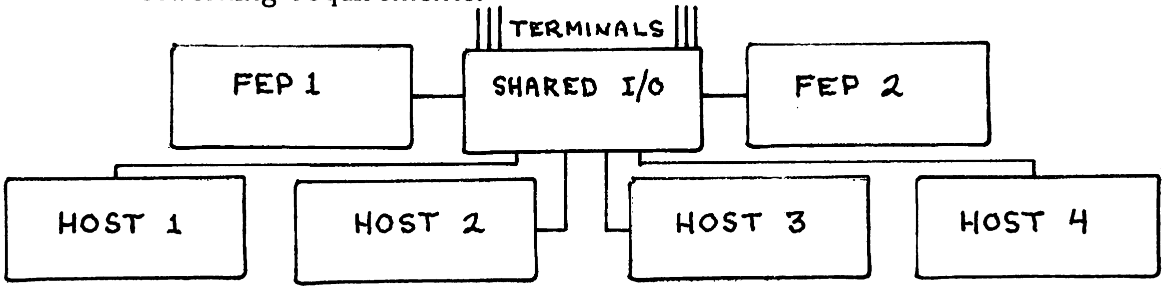
Multiple FEPs to Single Host Configuration - Where network requirements dictate discrete front-end processors, multiple FEPs (limited only by the number of HP host interface slots available) can be interfaced to a single HP3000 System.



Redundant FEPs to Single Host Configuration - Redundant front-end processors, sharing all I/O interfaces (SHARED I/O) to the terminal network and the host, provide automatically switched front-end processing in the event of a front-end failure.



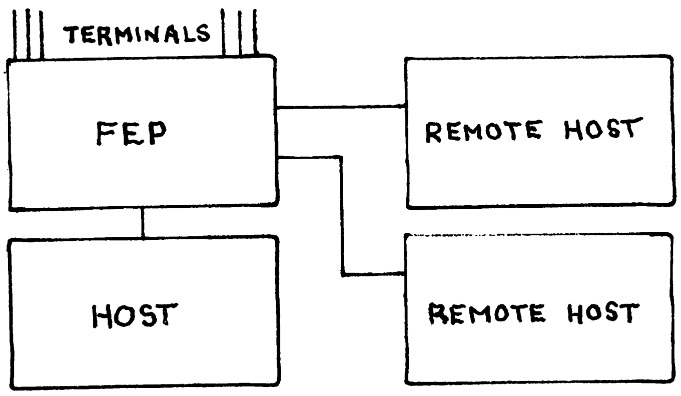
Redundant FEPs to Multiple Host Configuration - Combining the unique front-end features of redundancy and multiple host support provides a configuration suitable for the most demanding of networking requirements.



The front-end has been designed to provide maximum flexibility. It supports the above configurations, but alternative configurations are available to meet specific networking requirements. For example:

Store and Forward - With a data communication link, the front-end can store and forward messages to remote processors. These messages can originate from: 1) Terminals interfaced to MCS3000 Model 200; 2) host resident application programs; or 3) other remote processors.

By utilizing "inverse protocols", the front-end appears to the remote host processor as a terminal or terminal controller (e.g., IBM 327X, IBM 2780, Burroughs TD830, etc.). The store-and-forward feature enables an HP3000 user to interface with a remote (IBM, Burroughs, etc.) host network.



Remote Diagnostics and Maintenance - To enhance SRI technical support the front-end can provide remote diagnostics and maintenance which enables SRI technical support personnel to monitor, isolate and correct front-end code, memory and disk files remotely, without halting on-line operations.

IX. Summary

The SRI front-end, MCS3000, significantly enhances the HP3000. The capacity of the HP3000 to do transaction processing is greatly increased because of the distributing of data communications and transaction monitor functions to a specialized processor. Moreover, many options are opened to the implementers of systems, such as the multiple host configurations, redundant configurations, and the ability to interface foreign terminals and processors; all new options for distributed processing with HP3000's.

Appendix I

In August of 1978 a major brokerage firm headquartered in the midwest began a process to replace its aging communications system. The company was then communicating with its 200 branch offices via low speed teletypes. During the inquiry, the firm came to realize that they might do much more than just replace their TTYs. Thus in January of 1979 they selected Systems Research of Okemos Michigan to install an online turnkey computer system and terminal network which was to handle both communications and data processing functions.

SRI, working closely with the brokerage firm's staff, began immediately to design a comprehensive system centered around MCS3000, SRI's HP1000 and HP3000 based communications and transaction processing product. The resulting plan called for a phased implementation which would have as its first goal the replacement of the existing "network". Installation of the Hewlett-Packard hardware began in April of 1979.

By the fall of 1979 all of the old teletype equipment and slow asynchronous lines had been replaced by CRT and printing terminals multidropped on 2400 baud synchronous lines. By the end of 1980 due to the growth of the firms business, there were over 250 offices being supported, and the network consisted of more than 600 terminals. Additionally, an on-line datacomm link to a non-HP remote mainframe is supported for accessing a highly volatile and time-critical data base. The entire network is controlled and supported from a single HP1000 minicomputer with SRI's MCS3000 software. (See figure A).

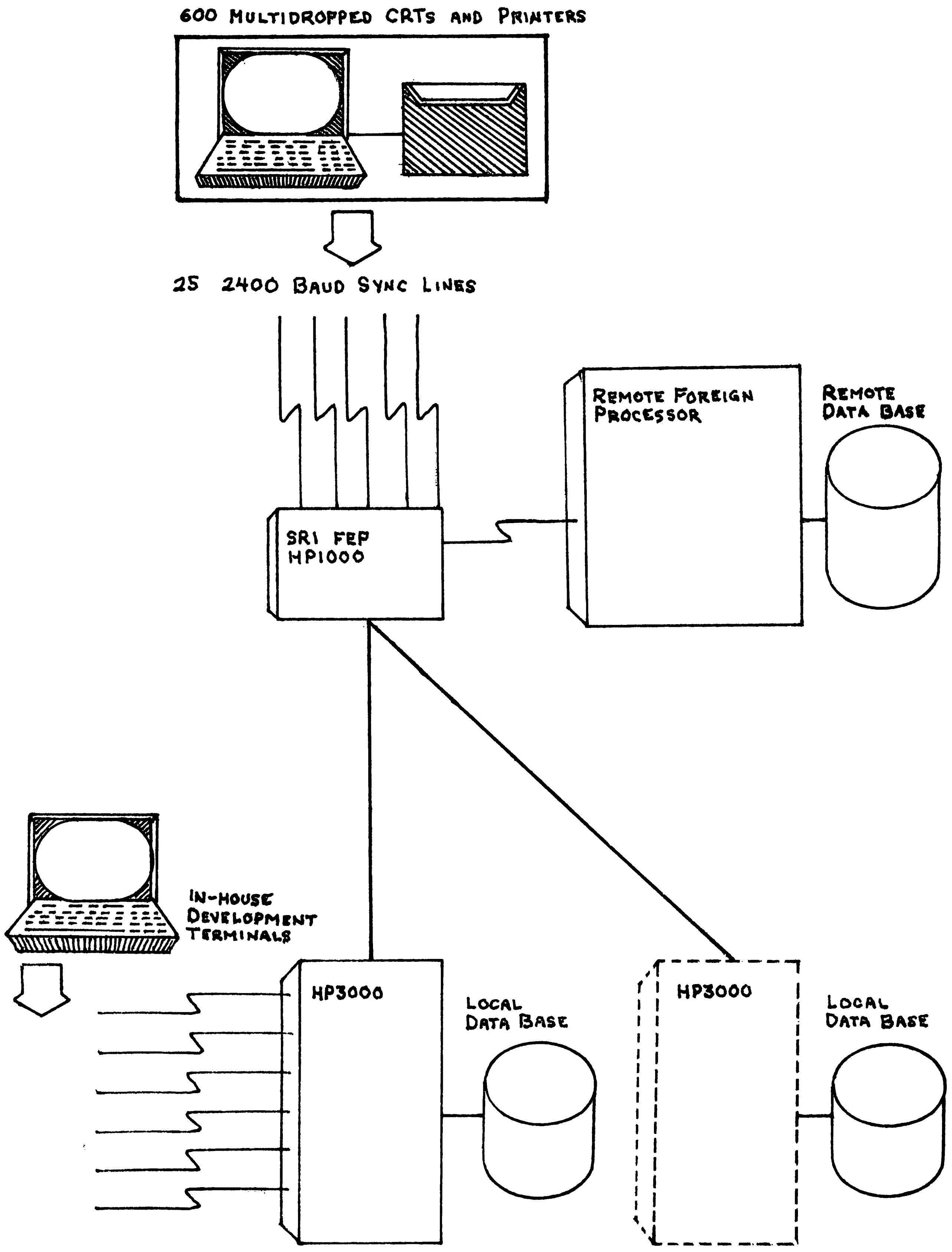
The HP1000 is linked through high-speed (400,000 char/sec) HP interfaces to an HP3000 Series III which handles the data processing functions. This vertical distribution provides, in effect, a division of labor which makes possible much higher throughput on the HP3000, as well as the obvious expanded data communications. The HP1000 is configured with 256KB of memory and 20MB of disk; the HP3000 has 1.5MB of memory and 375MB of disk, as well as a pair of tape drives and a high speed line printer.

Appendix 1 Cont.

There are four major applications currently being handled by the system, trades, quotes, research, and branch communications and administration. Each of the major systems has a single program in the HP3000 which handles all transactions for that application. The message volume being processed by the current configuration exceeds 80,000 daily and has reached a level of 15 per second on peak trading days.

Current development plans include the addition of another HP3000 to the configuration in 1981 to accommodate between 5 and 10 additional applications. At least three of these applications will require on-line/realtime computer to computer links with various securities service organizations. These links will be effected through the HP1000 front-end. Additionally the number of offices supported is expected to reach the 350 mark in 1981 demanding a minimum of 750 terminals to be supported by the configuration.

Appendix I, Figure A.



Appendix II.

SRI PROTOCOLS

Name	Description
PTTY	Character Mode Teletype
P264X	Block Mode HP 264X
PBIPP	IBM Bisync Point-to-Point (2780/3780)
PBIMP	IBM Bisync Multi-Point
PBPS	Burroughs Poll/Select
PADM	Lear-Siegler ADM-2
PBRJE	Burroughs Remote Job Entry (761)
PBPPCT	Burroughs Point-to-Point/Contention
PBPPBT	Burroughs Point-to-Point/Batch
PBPPCV	Burroughs Point-to-Point/Conversation
PNCR270	NCR 270 Basic Variant
PTTEL	Bell 407C Transaction Telephone
PAZUR	Azuredata Scorepad Terminal
PTIC	TI 742 Cassette Station
PVSET	VUSET device at Pan Am Bank
PNCR270A	NCR 270 Variant A
PNCR270B	NCR 270 Variant B
PNCR796	NCR 796 Basic Variant
PNCR796A	NCR 796 Variant A
PIDEN	IDENTICON Bar Code Reader
PIBM2260	IBM 2260
PVTRXI	Votrax In
PVTRXO	Votrax Out
PCARD	DATA CARD (IBM 2740 Subset)
PBNCR	Bisync Transparent N.C.R. (3270 var)
PCLETS	California L.E.T.S. (2780.3780 var)

INCREASE PROGRAM PRODUCTIVITY/
FULL SCREEN EDITOR

By:

Gurdo VanBempt
Jaak VanDamme

Syvas, Inc.

Paper to be presented
at Conference

USING HARDWARE MONITOR TO SOLVE PROBLEMS ON HP3000 - III

Ivan Loffler
Senior DP Staff Analyst
GTE Service Corporation
P. O. Box 1548 - F071
Tampa, FL 33601

I. ABSTRACT

This paper describes a case study concerned with solving a severe response time problem on an HP3000 large data base system. A major tool used on this project was the hardware monitor.

The paper states the problem, the objectives, and describes the problem solution. To arrive at a solution it was necessary to define levels of utilization of the troubled system, for which the hardware monitor was used directly. Also, the application of capacity planning guidelines is described here, especially how the guidelines were used for the definition of system utilization.

Also, software tools such as: SOO, SHOWME, and Event Monitoring Facility were used. These tools were calibrated for accuracy (i.e., capture ratio) and for overhead by comparison to hardware monitor data.

The results of this project, including conclusions and recommendations are contained in this paper.

II. INTRODUCTION

This paper describes the problems and the solutions as they occurred in one of our data centers within GTE. The main problem was an extremely long response time (up to 30 minutes) experienced by users of a large data base system, run on a Hewlett-Packard HP3000, Series III minicomputer.

The configuration consists of the CPU, multiplexor channel, selector channel, 1 megabyte of main memory, 8 disk drives, 2 tape drives and about 30 terminals connected via a multiplexor channel. The disk drives are attached to the system via the selector channel[1] (See Figure 1).

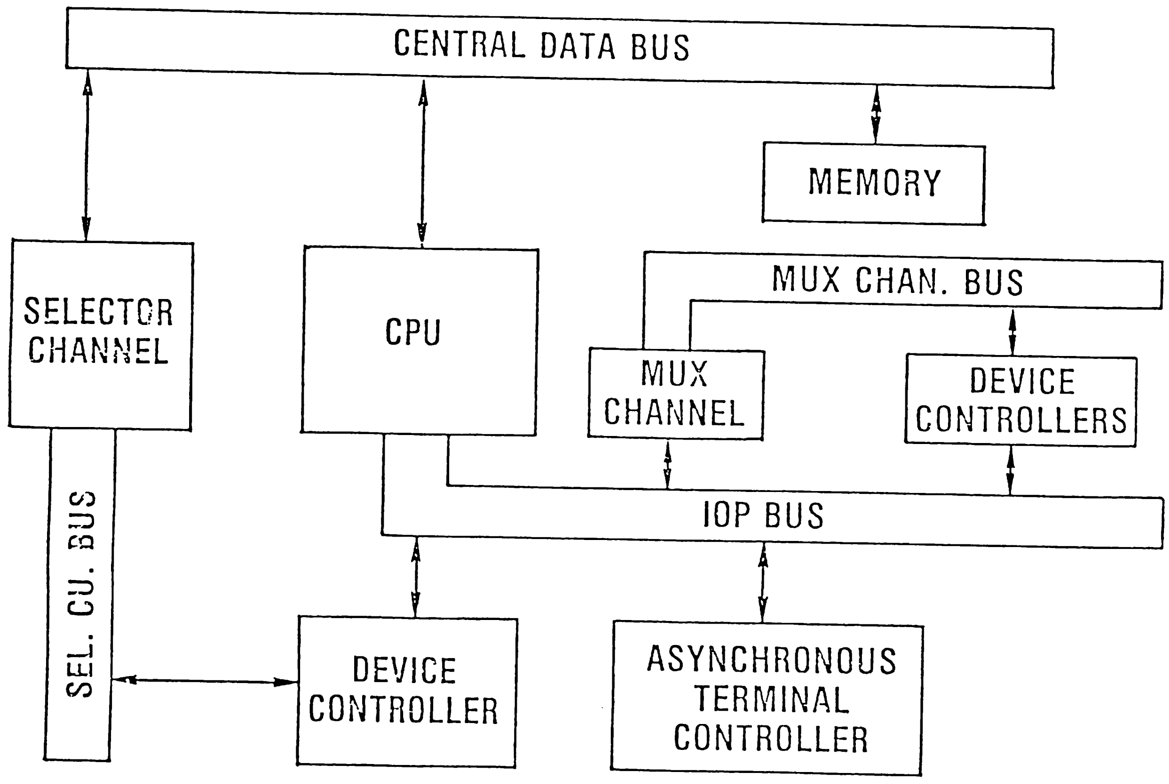


FIGURE 1

The hardware is controlled by the MPE-III operating system. The data base is supported by the IMAGE subsystem.

Also provided in this paper is an analysis of the problems and a discussion on selected diagnostic tools. A description of a major tool used in this project, the hardware monitor, is also included. The paper lists solutions designed to rectify the problem.

III. DEFINITION OF PROBLEMS

The HP3000 data center started to experience a severe degradation of the response time on its large online data base system. This system has been designed to keep track of the telephone circuits and related equipment, including trunk and facilities. It provides for such details as circuit orders and line assignments. The system requires an interaction with the engineering department, and has to absorb intensive data entry and frequent inquiries.

The response time delay of up to 30 minutes impaired the productivity of the operators, clerks, and engineers, who use the system. This resulted in delays in equipment ordering and could result in delays in service to telephone company customers.

After the management of the center recognized this as an emergency, several technical groups were involved and an informal task force was formed with an objective to improve the service levels of the system. The group included application development people, vendor's hardware and software engineers, the data center technical support and a corporate planning group, of which the author is a member.

This task force divided the investigation into appropriate areas of interest. The development team looked at the way the application design might be improved, while the Hewlett-Packard people investigated the load on the system using the Event Monitoring Facility, a software measurement tool. The planning group brought in the Tesdata hardware monitor to measure the hardware components of the system, to compare their utilization against the recently developed capacity guidelines for minicomputers, to calibrate all software tools used in this project, and to observe the impact of remedial changes on the investigated system.

IV. HARDWARE MONITORING[2]

Since the hardware monitor became a major diagnostic tool in this project, it deserves more explanation. The hardware monitor is a measurement device, independent of the monitored system called the host. It is electrically attached to the host's backplane pins and collects specific discrete electronic signals. Because it does not interact with any part of system's software, it is totally independent of host activities and thus does not impose any overhead on the measured host.

Figure 2 describes the hardware monitor in a block diagram. The electronic signals, such as CPU Busy, Selector Channel Busy, and Byte Count are captured by high speed probes and brought to the hardware monitor's capturing registers via a system of cables and concentrators. The signals are processed inside the hardware monitor by its internal Boolean logic circuitry. The hardware monitor's internal minicomputer organizes the signals into the memory in the form of data. The data are either displayed online on a CRT or recorded on a magnetic tape or disk for postprocessing and for printing of reports.

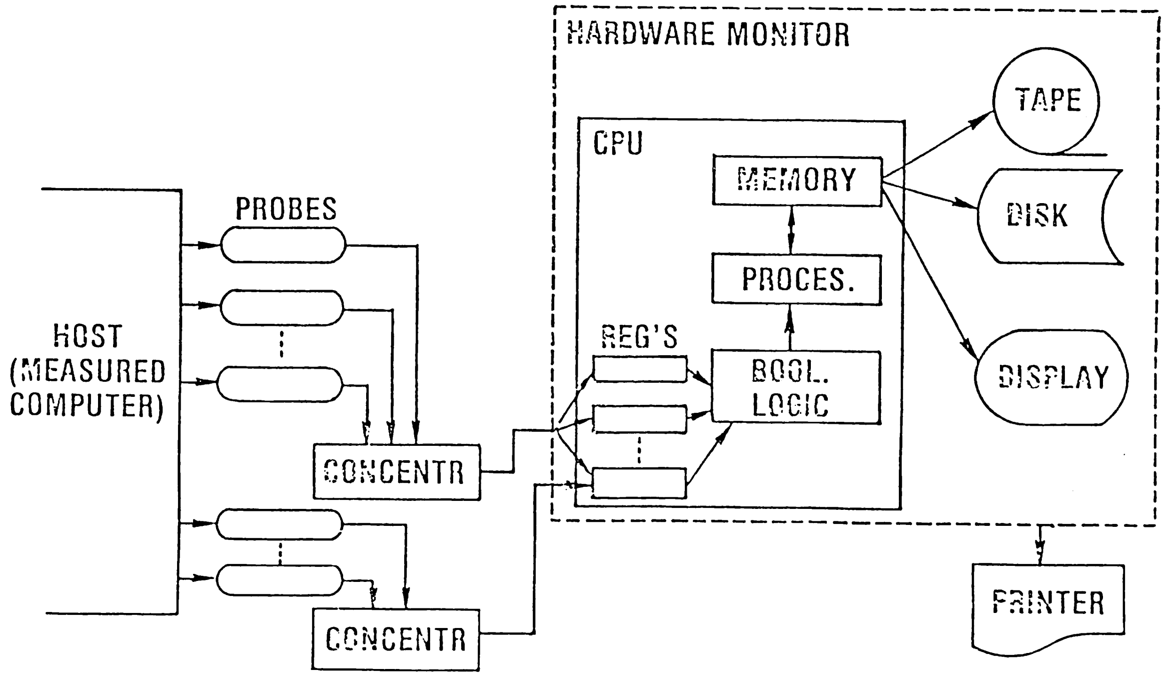


FIGURE 2

Basic information required about the system being measured is called the System Profile[3]. It basically consists of:

- CPU Busy
- CPU in Privilege State (Overhead)
- Multiplexor Channel Busy
- Selector Channel Busy
- Instruction Count
- Disk Drive Busy

The hardware monitor is the best suited device for capturing this information, since it does not skew the data by its own processing overhead, as software monitors do. Also, the Boolean logic of the hardware monitor allows for composite measures such as CPU/Selector Overlap, CPU Only, System Busy, and CPU Only (See Figure 3).

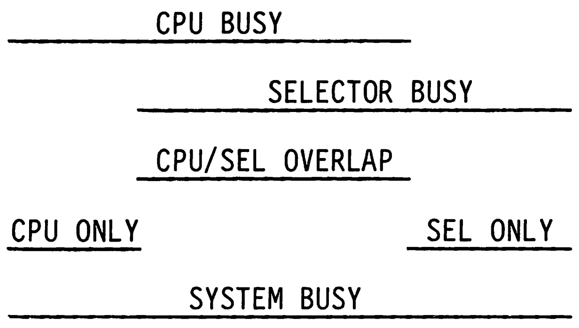


FIGURE 3

These measures are available from the hardware monitor only. They are crucial for the investigation of system performance. For example, the CPU/Selector Channel Overlap is commonly low on machines operating under the MPE-III operating system. This results in inefficient operation and decreases the usable capacity of the system.

V. ANALYSIS OF PROBLEMS

Analyzing the collected data from all the available tools, such as the hardware monitor, Event Monitoring Facility, SOO and SHOWME software monitors, and also from the physical observation of data center operations, there were several apparent areas for improvement:

1. Over 60% of the disk I/O activity was concentrated on one disk drive, creating a high amount of conflicts and long queues on this particular drive. Since this drive also contained a system data set, the situation was complicated even further by adding to the response time delay.

2. The system swapping amounted to 7.2 swaps of 8 KBytes per CPU second. Comparing this number with our Capacity Planning Guidelines, the swapping (or paging in IBM terminology) should not exceed 10 pages per second on a comparably sized IBM machine. Since every HP swap equals to two IBM pages, the maximum paging was 14 per seconds, exceeding the guidelines by 40%.

3. The CPU activity was low, compared to the amount of I/O activity. This suggests an I/O bound workload. Since some percentage of the CPU Busy time has to be attributed to the I/O processing, it was necessary to concentrate the tuning effort on the I/O area.

4. Some transactions contain up to 71 disk I/O's. This obviously is a design and data base deficiency, and it is adding considerably to the response time delay. There is an obvious correlation between the amount of I/O processed and the response time. The disk (Model Number 7920) timings are described in Table 1.

Average Random Seek Time:	25.0 ms
Average Rotational Delay:	8.3 ms
Data Transfer Time (937.5kB/Sec):	8.5 ms
Total per Average 8K Block	41.8 ms

TABLE 1

41.8 ms is the time required to process one I/O within the disk drive. There are some delays added: the contention for the drive, controller and channel, the CPU involvment (which may not always be overlapped with the I/O operation), and the memory contention. These factors may very well increase the time for processing an I/O to 1 second.

5. Number of active users was observed at a maximum of 28, which exceeded the previously recommended maximum of 15 active terminals. Previous monitoring of the simulated application system by the hardware monitor determined that on the average, one terminal would utilize the system at about 6% of its usable capacity.

6. The HP3000-III does not have adequate capacity, using current versions of utility support software, to reorganize our very large data base in a timely fashion. A vendor's software engineer estimated that this task would take about 10 days. Currently, secondaries account for 30% of the records in portions of the data base. This fact is resulting in unnecessary disk I/O operations. Decreasing and maintaining a reasonable amount of secondaries requires more disk space, which is an area where maximum capacity is also being approached. If this capability to reorganize remains unresolved, it will keep the system on a continually degrading course.

7. Some miscellaneous observations were also made; such as, very long searches for certain transactions (resulting in a high number of I/O operations), and inefficient algorithms for resolving synonyms in the data base (resulting in a 30% level of secondary records).

VI. SOLUTIONS

1. Disk I/O Contention

Table 2 describes the number of disk I/O's on each disk drive at the beginning of our investigations. These numbers were reported by the Event Monitoring Facility.

Drive	Disk I/Os	%
1	6,331	62.0
2	1,056	10.3
3	54	0.5
5	953	9.3
65	200	2.0
66	927	9.1
67	511	5.0
7	180	1.8
TOTAL	10,212	100.0

TABLE 2

The obvious disparity created overutilization of drive 1, resulting in contention, queues, and long delays on this drive.

This drive contained the system data set plus some application data sets. The trivial system response time (i.e., pressing the RETURN key) was about 30 seconds.

Reorganization of disk data sets, with an objective to decrease contention was recommended and implemented. Drive 1 was dedicated to the system data set. The result was an immediate response to the RETURN key. Since the production response time was of a magnitude higher, the result of this improvement was not measurable.

2. Memory Contention

The contention for space in the main memory leads to swapping. Each swap results in at least one I/O operation. In the heavily I/O bound workload the 7.2 swaps per second (measured by the Event Monitoring Facility) contributes to the I/O load.

The suggested remedy was to increase the size of memory. The original system had 1 Mbyte of main memory. Thanks to Hewlett-Packard's cooperation we were able to experiment with two 0.5 Mbyte increases.

The system with 1.5 Mbyte of memory decreased the amount of swapping to 22% of the 1 Mbyte swapping level. The second 0.5 MByte decreased the remaining swapping in half, but since swapping had already been decreased considerably, the second memory increment did not provide much improvement in performance.

Also, the hardware monitor utilization measurements supported these results. Table 3 describes all three steps. The CPU Busy and the Selector Channel Busy are basic measures. The third measure, Utilization Level[4], is an arithmetic sum of the first two measures. The Utilization Level was calibrated previously, and it was found that the maximum laboratory achievable value is 180 (MPE-III). The practical maximum utilization of the system was found to be 135.

Memory Size	CPU Busy	Selector	Utilization Level
1 Mbyte	61.8%	64.2%	126.0
1.5 Mbyte	49.2	49.2	98.4
2 Mbyte	43.2	45.0	88.2

TABLE 3

Although these comparisons were not based on results of a rigorous benchmark, an effort was made to compare data taken during periods of comparable workload execution. The type of transactions and number of users was comparable, and all measures were made on the same day, during the same shift.

The memory size testing was performed after some tuning had already been implemented, thus the original high utilization (over 130) was not reached.

3. I/O Operation

The high amount of I/O operation was attributed to the following:

- Disk I/O contention, as discussed previously.
- High swapping rate.
- Very large database operations.

The application design team implemented changes to alleviate the I/O bound workload. They were:

- Multiple copies duplication was moved from the disk to memory.
- Optional restriction of conditional search to a smaller subset.
- Elimination of unnecessary summary reports displayed on a terminal.
- Ability of the user to bring less information to the screen.

The results of this effort are shown in Table 4.

Date	CPU Busy	Selector	SEL/CPU Ratio
Jan. 6, 1981 (prior to changes)	58.3%	60.8%	1.04
Feb. 10, 1981 (after changes)	43.9	42.8	0.97

TABLE 4

4. System's Capacity

The capacity of HP3000 was exceeded by the operation of this data base. Based on previous monitoring on a simulated workload, the "Utilization Level" increased 9 units with each added terminal. This would permit a maximum load of 15 active terminals. It was observed in actual practice that as many as 28 users were active.

Having up to 28 active terminals on the system resulted in contention of all the system's resources (except Multiplexor Channel), and delays in long search response time of up to 30 minutes.

One of the initial changes, and perhaps the most significant to date, was to split the workload over 2 shifts. After this change, the number of users on each shift usually do not exceed 15. This decreased the response time for transactions requiring long searches to a maximum of 30 seconds (from the original maximum of 30 minutes). Other types of transactions (those not requiring long searches) now required only a few seconds.

VII. CALIBRATION MEASUREMENTS

1. MPE-III/MPE-IV Comparison

During the hardware monitoring sessions, we also discovered a shortcoming of the system which influences its capacity. It is the lack of the ability of the operating system MPE-III to overlap CPU and I/O activity. The HP engineers were aware of this problem and claimed that the MPE-IV operating system should rectify it.

Table 5 compares the hardware monitoring measurements of MPE-III and preliminary released MPE-IV operating systems. A single stream serial batch benchmark consisting of five steps was used for comparison. The first step is CPU bound, the second to fourth steps are CPU and I/O mixed with increasing I/O portion, and the fifth step is I/O bound.

STEP	MPE-III			MPE-IV		
	CPU	SEL	OVERLAP	CPU	SEL	OVERLAP
1	41.54s	8.04s	0.00s	41.54s	6.70s	0.00s
2	63.65	18.76	9.38	61.64	18.76	10.05
3	50.92	34.84	25.46	48.91	32.83	26.13
4	55.61	50.25	20.77	54.27	48.24	20.77
5	60.97	65.66	28.14	56.95	63.65	29.48
TOTAL	272.69s	177.55s	83.75s	263.31s	170.18s	86.43s

TABLE 5

The same benchmark job consumed 3.5% less CPU time and 4.5% less selector channel time, when run under the MPE-IV operating system. However, the most significant improvement is the ability of MPE-IV to overlap the CPU and the selector channel activities. The formula (1) describes the method of calculation of the Overlap Factor:

$$\frac{\text{CPU + SEL}}{\text{OVERLAP}} = \text{Overlap Factor} \qquad (1)$$

The improvement of the Overlap Factor under the MPE-IV operating system was 7.1%. Since the overlap consists of two values (CPU and Selector) only one half of the Overlap Factor improvement should be considered for improvement in the system's capacity.

Since this benchmark is serial in nature and does not heavily load the system, the improvement is marginal. Therefore, another measurement of the system, heavily loaded with a series of engineering test programs is described in Table 6. This time an average percentage utilization is described in time samples.

NUMBER OF PROGRAMS	MPE-III			MPE-IV		
	CPU	SEL	OVERLAP	CPU	SEL	OVERLAP
1	41%	65%	9%	70%	47%	47%
2	50	66	16	98	57	56
3	50	78	26	99	79	78
4	49	90	41	99	57	56
AVERAGE	48%	75%	23%	92%	60%	59%

TABLE 6

In this case, the system under MPE-IV could provide an average of 24% more capacity, and 39% in an extreme case (3 programs). The overlap factor calculated by using formula (1) improved by a factor of 2 above MPE-III.

It should be noted, that all measurements were conducted on a pre-released version of the MPE-IV operating system using a benchmark technique. The performance results on the final supported version in a production environment might be different.

2. "SHOWME" Calibration[4]

The CPU time measured by the hardware monitor is considered an absolute measure. This allows it to calibrate any software package.

Table 7 describes the CPU times of the hardware monitor (H/M) and the SHOWME software for both MPE-III and MPE-IV operating systems. Again all measurements were taken while the above described benchmark job was executed.

	MPE-III			MPE-IV		
STEP	H/M	SHOWME	CAPTURE RATIO	H/M	SHOWME	CAPTURE RATIO
1	41.5s	39s	0.94	41.5s	38s	0.92
2	63.6	60	0.94	61.6	57	0.93
3	50.9	45	0.88	48.9	40	0.82
4	55.6	47	0.85	54.3	41	0.76
5	61.0	50	0.82	56.9	42	0.74
TOTAL	272.6s	241s	0.88	262.2s	218s	0.83

TABLE 7

SHOWME has 88% capture ratios under MPE-III and 83% under MPE-IV operating systems. These factors must be used whenever SHOWME is used to define the actual CPU utilization.

VIII. SUMMARY

It is obvious that we did not measure all parts of the system with the hardware monitor, the main storage and buses for example. This would require much more effort in installation, operation, and analysis and additional benefits would be marginal. Also, the urgency of the project required swift action.

The hardware monitor was not the only tool used in this project. There were software monitors, accounting data, physical observation and chiefly the dedication and effort of all personnel involved, which helped to solve this difficult and complex problem. However, the hardware monitor did play a major role in pointing out otherwise obscure bottlenecks and inefficiencies in a timely fashion, and thus saving many manhours, which would have been used otherwise. It also helped to calibrate the software monitors so they can be used in the future instead of the hardware monitor, which is an expensive and labor intensive tool.

The capacity of the HP3000-III and IMAGE/3000 to maintain the large data base remains a problem. This problem should be resolved since it has the potential to become worse with regular operation of the data base.

The effort to improve the service level of the HP3000-III system is not yet completed. The improvement of response time from the maximum of 30 minutes to 30 seconds was only the first step. The next step will be capacity planning with an objective to maintain an acceptable service level as the workload grows. Installation of MPE-IV, further changes in the application, extension of main memory to 2 Mbytes, and possible installation of the Model 44 are some of the available actions for future consideration.

References

(1)	Hewlett-Packard:	"OEM Computer Products Catalog; 5922-0151(D)," November 1980.
(2)	Tesdata Systems Corporation:	"MS58 Computer Performance Measurement System Reference Manual", McLean, Va.
(3)	Buzen, J. P:	"A Survey of System Tuning Tools and Techniques; System Tuning", Infotech State-of-the-Art Report, pp. 229 - 241, Bershire, England, 1977.
(4)	Loffler, I:	"Capacity Planning for Minicomputers", INTELCOM 80, Los Angeles, Ca., November 1980.
(5)	Wenig, R. P:	"Effective Use and Application of Minicomputers", IMS, Inc., Framingham, Ma., 1979.

The Development of a large Application System for the HP3000 Computer

Nancy Federman
Robert Steiner
Manufacturing Systems Operation

INTRODUCTION

Design, development, and market analysis are only the initial steps in producing a profitable product. Manufacture of the product is the next step. Do you really know what is involved? The first step is to specify the engineering data. What is the structure of the product? You need to determine the parts and subassemblies that comprise the product and the quantities of each required per unit. Which of these components will you manufacture? Which will you subcontract out to other manufacturing companies, and which will you purchase?

Now that you have some of the engineering specifications the next step is to determine the manufacturing procedure - how is the product to be made? You will need to decide the manufacturing operations necessary to construct the product and then specify the location in your shop where the work will be done (workstations). The sequence of these production steps, called the routing, plus the labor and material required at each operation must be detailed.

The product is specified and the manufacturing plant and process are established. You are now ready to begin production. How many products should you build? You need to consider customer demand, current and forecast, as well as the capacity constraints of the factory. Once your production plan is established you need to determine your material requirements. How many component parts do you need to meet your production schedule? You also need to determine the schedule of production for the manufactured components. When do the purchased parts need to be ordered? You need to balance the desire to have components always available with the economic necessity of keeping inventory levels as low as possible.

Controls are needed to monitor the flow of materials in the stockroom and on the factory floor. When do the component parts need to be issued to the production lines and what quantities are required? You will want to monitor the shop floor and track the work in process. How much material is wasted? How efficient is your work force? When the products are finally completed you need to keep them in a finished goods inventory and track their sales.

Finally, you will be concerned with calculating the costs - labor, material and overhead. The production costs will help you decide your pricing policies and determine your profitability.

From this brief and simplistic overview of a manufacturing operation it should be clear that there are many intricate relationships to deal with. Effective and efficient management of a manufacturing operation is difficult to achieve and many neighborhood businesses as well as sophisticated manufacturing companies are turning to computers for help.

MATERIALS MANAGEMENT/3000

Hewlett-Packard is among the vendors providing application systems for manufacturing management - a manufacturing company offering a solution to the problems of manufacturing companies. Materials Management/3000 is an interactive material planning and control system designed to make it easier to deal with the complexities of operating a manufacturing company. It is primarily designed for manufacturers who build standard products to stock in discrete manufacturing steps (fabricators and assemblers) These companies have a significant investment in inventory. Materials Management/3000 can help them balance their inventory levels with customer demand for timely shipments to optimize their dollar investment.

The complexity of the manufacturing environment can also be seen in the internal complexity of the software product which provides the solution. Materials Management/3000 consists of over 400,000 lines of SPL code which make up 161 transactions referencing 9 data bases. There are 291 screens, both transaction screens and menu screens. There are also 168 batch programs. The application data bases and screens can be customized by the user. Materials Management/3000 operates on any of the HP3000 family of computers and uses the 264X family of terminals. The system uses HP's IMAGE data base management system and HP's V/3000 screen handler.

Materials Management/3000 provides a solution to the previously outlined problems of manufacturing companies by supplying 10 major modules. Their inter-relation is diagrammed in Figure 1.

Master Production Scheduling: MPS is an on-line management planning and production scheduling tool. It is used by the master scheduler to generate a production schedule for the plant's marketable products and to set the mix for the product options. Input to the module includes the current customer orders, forecast customer orders, the current production schedule and the current level of product inventory. The output of the MPS calculations is called the master production schedule. This schedule contains suggested manufacturing orders including quantities and starting dates. The schedule is then input to the Material Requirements Planning (MRP) module to plan the manufacture and purchase of the required component parts. MPS also includes a "WHAT IF" simulation capability which allows the user to generate tentative schedules, and view the impact of modifications on the current schedule.

Rough Cut Resource Planning: Rouch cut resource planning (RPP) is a management tool used by the master scheduler to compare the resources needed to implement the master schedule with the available critical resources to help produce a realistic master schedule. The user specifies the critical resource requirements for each master schedule part, and the maximum capacity of these resources. Examples of critical resources are labor hours, floor space, dollar investment in work in process inventory, material supplies, etc. The RPP reports highlight the capacity constraints and help the user to resolve competing demands for the critical resources. An on-line RPP report can also be used to help evaluate simulated master schedules by comparing their resource requirements to the requirements of the current schedule.

Material Requirements Planning: MRP simulates the complex flow of materials required to manufacture products and generates a material plan. MRP planning starts with up-to-date information about the current inventory levels and the planned production requirements. Using part and bill of material information the material requirments for each part are calculated. The plan is started with the highest level assemblies and then proceeds through the lowest level parts. MRP will reschedule current work and purchase orders and suggest new orders as necessary to meet the demand. MRP is a regenerative system - a complete material plan is generated every time MRP is run.

Parts and Bills of Material: This module provides on-line maintenance of engineering, accounting, and planning information about each part and product, and information on how the parts relate to one another to form the product structure (bill of material). Responsibility for maintaining this data will normally be shared among several departments - accounting, engineering specifications, and planning. Part and structure data can be reviewed on-line or through printed reports. The part and structure data is used by many of the other modules in Materials Management/3000, including MPS and MRP.

Routings and Workcenters: The Bill of Material defines the parts and subassemblies that comprise a product but doesn't document how the various components are actually assembled. The routings and workcenter module maintains information that describes the locations where the products are made (workcenters) and the proper sequence of manufacturing (routings) This information is used to generate cost information for the Standard Product Cost module, and to help develop detailed production schedules. Responsibility for this data usually resides with manufacturing specifications.

Standard Product Cost: SPC provides manufacturers with the capability to accurately calculate the standard costs associated with the manufacture of each product. All current cost information may be edited and reviewed on-line. The standard cost of a product is determined by accumulating all relevant material, labor and overhead costs for the components of the product as well as the costs associated with the actual construction of the finished product. These standards can be used to determine product pricing and profitability. Marketing as well as the material manager would use this data.

Material Issues and Receipts: This module helps to control stockroom inventory by maintain timely and accurate records of all actions that affect inventory balances. Ths data includes receipts of work orders or purchase orders, material issues from stock to a particular work order, filling of a backorder, or an unplanned issue. All record keeping and updating is done on-line and a record of all inventory activity is kept on-line for a user-specified period of time as an audit trail. Stock room personnel are the primary users of this system.

Inventory Balance Management: Inventory balance management is a module to maintain information about inventory balances and the warehouse locations where the inventory is stored. The current inventory status can be affected by three types of transactions - material movement, inventory counts, and stock adjustments. All three types of transactions will trigger an immediate update of the inventory counts as well as create an audit trail record. All udpates are done automatically in an on-line mode. Current inventory balance data from this module is used by MPS and MRP to determine the next master schedule and the next material plan. All activity that affects inventory status can be reviewed on-line. An inventory value report is also available. Materials Management/3000 also allows for multiple stock locations - a separate on-hand balance can be maintained for each stock location in each warehouse. This system also helps with the actual counting of inventory which is periodically used to verify the inventory totals.

Work Order Control: A work order is an internal factory authorization to build a specified quantity of a particular subassembly by a specified date. All work orders require the issue of on-hand inventory for their completion. Prior reservation of on-hand inventory is the best method of preventing shortages at the time of issue. Allocation, or logical reservation, of on-hand inventory will help predict and prevent these shortages. The timely notification of exceptions to the material plan can allow corrective action before the results become disastrous. The output of this tracking system is the reports noting exception conditions. The materials manager can then act on these reports. The actual issuing of parts and work orders, and the actual receipt of finished products is accomplished by using the material issues and receipts module. MRP is a prime user of information from this system.

Purchase Order Tracking: A purchase order represents a scheduled receipt for purchased items. Entering a purchase order requires the entry of more information than that required on a work order - e.g. vendor information, shipping information, price information. It is also possible to group multiple delivery dates and/or multiple parts on the same purchase order. Purchase Order Tracking system monitors these scheduled receipts and also maintains vendor information. Users can get a report on the current orders for a particular vendor, or the value of outstanding purshases by scheduled receipt date. The purchasing department and the materials manager would normally use this module.

What distinquishes Materials Management/3000 from other materials management systems in the marketplace is not just the product features but the design and implementation philosophy behind it. This philosophy evolved from previous experience with application systems, a knowledge of the competitive marketplace, and first-hand experience with manufacturing company operations. The design philosophy can be summarized as providing a functionally complete solution which fits the business practices of the user, is friendly and easy to use, and is supportable by HP.

APPLICATION STRUCTURE

Materials Management/3000 evolved from a previous product, MFG/3000, which was released in December of 1977. MFG/3000 began in the HP3000 manufacturing area as a computerized solution to HP's internal materials management problems. As the system was completed and put into use it became clear that other medium to large manufacturing companies were having the same sorts of problems. It was decided to to turn the home-grown system into a product.

MFG/3000 was sold both as an object code product and a source code product. A source code product is one in which the actual computer programs are sold to the user. The user then can modify the code directly if they wish to implement any modifications. A source code product is difficult for the factory to enhance and puts a support burden on the customer. The bug fixes and factory enhancements must be sent to the user in source code format. The customer must then implement the changes manually. If the customer has made modifications to the source code the changes from HP may not be compatible. It is also very difficult for the factory to support a source code product. If the user reports a bug the source of the error is difficult and time-consuming to detect since the fault could be in the original code or in the user-modified code.

An object code product is one in which only the executable code is sent to the customer. The user cannot make any modifications to this product and so has gained factory support at the price of flexibility and local user control. The idea of an object code product is a difficult one to "sell" to the customer. The application cannot be tailored to fit the individual needs of the customer. On the other hand, it is not possible for the factory to anticipate all the detailed and distinctive capabilities peculiar to any particular customer.

HP's solution was to develop an object code product that was user customizable. The big advantage is that HP can fully support and enhance the product while the user can tailor the application to suit their individual needs.

Most of MFG/3000 customers had purchased the object code version. The majority of those customers that did purchase the source code were interested in changing the field edits, the data item characteristics, and the data items in the reports, not the program logic. The implementation strategy was to include standard and accepted functions in the program code and provide software tools to allow the customers to tailor, or "customize," the system to fit their own individual requirements. So the program code was separated from the data item characteristics, the screen formats, and the other parameters that characterize each particular installation of the product. This would allow the factory to maintain the logic of the programs while the user could tailor the edits and data item characteristics, as well as modify the appearance of the application.

The next problem to tackle was to develop an efficient mechanism to "interpret" the user-supplied execution-time parameters. The first design decision was to put the customizable information into tables. A table-driven application was chosen over a compiled application because implementation of the latter design meant the development of a new language, a challenging task in itself. And control over the customer use of the language would be difficult. Data item customization via re-compilation of all the source code programs would be time-consuming and prone to error. The factory would also lose some control over the integrity of the application once the source code was distributed to the customers. A table-driven applciation seemed the wise choice.

Experience with MFG/3000, with its rudimentary edit table, led to the decision to expand this table and add to it a data dictionary which would contain tha structure of the data base, the specification of the data items itself, and the format of the screens and reports. Now that the contents of the tables had been agreed upon the next problem was to provide efficient execution-time access to the data in the tables.

Tables implemented in files or data bases would be too slow to access at execution time. If the tables were places on the stack too much memory would be used. Extra Data Segments could provide efficient execution-time access with good memory utilization. The design agreed upon put the "source" version of the application parameters into a data base, so the user could edit them. This data was them compiled into extra data segments for execution-time access.

The only problem with extra data segments is that they are not sharable across sessions. An important underlying assumption that the applciation would have many users. So the application had to be designed to allow for multiple users. The solution was to develop a control program to manage all the Materials Management/3000 user terminals so that they would appear to the MPE operating system as just one session. This solution fit in nicely with the design goal to have a dedicated system - the user would interface with a program that was optimized for the non-computer professional instead of with MPE. This control program would automate some of the standard control functions, such as scheduling terminals and initiating batch jobs.

Some of the tools necessary to implement an object code user-customizable applciation were already available. IMAGE/3000, the data base subsystem, eliminates data redundancy and resulting maintenance problems. V/3000, the forms data entry subsystem, makes it easy to design and implement a friendly, consistent user interface. The MPE message system provides a facility for creating customizable report headings and user error messages.

To meet our objectives it was necessary to develop two more tools, the Application Customizer and the Application Monitor. The Customizer provides a method for the customer to tailor Materials Management/3000 to fit an individual environment, and the Monitor automates many of the day-to-day administrative functions ususally performed by an operations staff. The Monitor accomplishes its function by starting and stopping terminals at predetermined times and scheduling background jobs such as MRP to be run on a regular basis. System security is controllable because users may not use the application (i.e., Materials Management/3000) unless system administrator has instructed the Monitor to start the application on a specific terminal. The Monitor also includes review capability of the application-generated error messages and other system activity, such as the background job schedule or current terminal activity. To the application program, the Monitor provides many services normally associated with operating systems. The application programs may request services such as process initiation, interprocess communcation, and resource allocation for on-line terminals and printers. The application designer can concenterate on solving application-oriented problems and call on the monitor to provide other functions that are necessary but not directly involved with materials management functions.

The key componenet of a customizable application is the application data dictionary, which serves as a repository for application-dependent information such as data item characteristics, data base schemas, V/3000 form descriptions, security passwords, terminal configuration, and background job schedules. The Application Customizer was designed to maintain the data dictionary, and it performs two major functions. The first is a facility for customers to alter or customize the application system using a simple menu-driven fill-in-the-blanks sequence of forms. Since this is the part of the Customizer most visible to the customer, the bulk of the design effort went into making customization functions simple and easy to understand by nonprogrammers. The second function performed by the Customizer is to transform the information present in the data dictionary from data structures suitable for run-time access by the application programs. These transformed data structures, collectively known as the run-time application ta dictionary, are used by the application programs to determine the values of all customizable parameters in the system.

Fig. 2 shows how the Customizer, the Monitor, IMAGE/3000 V/3000 and the application software interact.

CUSTOMIZATION TECHNIQUES

The rest of this article describes some of the methods used by the designers of Materials Management/3000 to design programs that can operate efficiently in a customizable environment. Because Materials Management/3000 is a customizable application, the customer has the ability to change many of the characteristics of the system by modifying items in the application data dictionary, rather than using the traditional time-consuming and error-prone method of modifying source code and compiling programs. Designing customizable applications is therefore complicated by the fact that many assumptions traditionally made by application programmers are not true. Customers may modify data item characteristics, add and delete items, modify the on-line user interface, and define additional processing.

Changing Data Item Characteristics

An assumption traditionally made is that once a data item is defined, its characteristics will not change. In a customizable environment that assumption is no longer valid. Because it is possible for the customer to alter the length, type and precision of any field, the application program has no idea what the characteristics of fields will be until the program is executing. For example, there are three broad categories of data type used by Materials Management/3000: alphanumeric strings, numeric fields, and date fields. An application designer may assume a data item is one of these three general types, but cannot know the specific format of the field. Numeric fields may be any of five numeric data types: display numeric (with explicit sign and decimal point), zoned numer (with implicit decimal point and sign overpunch), packed decimal, 16-bit integer, and 32-bit integer. Any numeric field may be changed to any other numeric type and the length and the precision (number of deicmal places) of display numeric, zoned numeric, and packed deccimal numbers may also be altered by the customer.

The solution is to place field definitions in tables that are accessed by the application program at execution time. These tables form the run-time application data dicitonary generated by the Application Customizer and are accessed only by a set of Application Customizer routines called intrinsics. This enables the designer to code the application without specific knowledge of the structure of the tables. As the Application Customizer is enhanced, the tables may change, but the application programs will not have to be modified because the intrinsics insulate the application from the Application Customizer.

A field may have several occurrences in an application, each having slightly different characteristics. For example, a numeric field may be present on an IMAGE data set, and also on a data entry screen defined for a transaction that updates the data set. The item on the screen will be defined as being display numeric type, with a length of ten digits including two decimal places, the same item on the data set will be defined as being packed decimal, with a length of 15 digits including four decimal places. The designer can develop customizable programs without concentrating on these differences because of the intrinsics provided by the Application Customizer to handle all arithmetic and data movement operations.

A table lookup is required every time a data item is maniputlated by the application. Materials Management/3000 is structuared to provide the the best response time for users who perform the same transaction many times, using few or no other transactions. An example is loading dock personnel who perform"receive stock" transactions almost exclusively. When a transaction is entered, only that portion of the customizer tables that contains data item defiinitions are used by the trasnaction is moved to the program data area. The data item definitions remain in memory until the user branches to another transaction. With the needed data item definitions in program data memory, Customizer intrinsics may access data defininitions with a minimum of overheaad. This conserves memory and provides fast response time for subsequent executions.

Since there are Customizer intrinsics that perform data movement and arithmetic operations, instead of coding SPL statements to manipulate data, the application designer codes calls to intrinsics that add, subtract, multiply, or divide numeric data items, and move numeric or alphanumeric items. These intrinsics reference the data item definition tables, performing data validation, decimal point normalization, data type conversion, and security checking. If an error prevents proper processing, the intrinsic returns an appropriate error code, and the user can be informed.

Modifying Fields

In addition to changing data item characteristics, it is possible for the customer to add and delete some fields appearing on screens and data sets. Materials Management/3000 is designed to perform specific inventory control functions, so a working set of data items must be present for the application to perform its function properly. These data items are defined as critical to the application and may not be deleted by the customer. Other data items in the released product are included for optional processing and may be deleted by the customer for reasons of efficiency or to prevent user confusion. On the other hand, a customer may want to adapt the application to perform additional functions not anticipated by the application designers. This will require the addition of data items to data entry screens and data sets. A method must be used to represent the association of data items with screens and data sets to the application programs.

Fortunately, much of the processing in Materials Management/3000 and many other data processing applications involves the movement of complete records from place to place. For example, "add" transactions simply construct a record from the data items entered on a data entry screen, and after appropriate validation edits, move the record to an IMAGE data set. "Change" transactions retrieve a record from a data set, update it with fields entered from the screen, and then move the record back to the data set. When adding or deleting a data item on a data set or screen both the designer and the customer must associate the item with a specific record format. Record formats are nothing more than collections of data item definitions that correspond to the fields on a data set or data entry screen record. A data entry screen record and the corresponding data set record it will update will contain many of the same data items, although they may have different characterisitics. Since it is unknown until execution time exactly what items will be present on a given record, the Application Customizer provides an intrinsic that moves corresponding data items from one record to another.

The operation of the MOVE CORRESPONDING intrinsic is very simple. The intrinsic is passed the record format definitions present in the source format, the instrinsic searches the target format for a corresponding item definition. If a match occurs, the data is moved from the source to the target record, changing the data type, length, and precision if necessary. This process continues until all corresponding fields have been moved from the source to the target record. The MOVE CORRESPONDING intrinsic allows the designer to think on a record level, not being conerned with individual data items. This makes it possible for the customer to add and delete noncritical data itmes at will.

Fig. 3 shows an example of MOVE CORRESPONDIN operation. Each record is described by a format maintained by the Application Customizer. Every item is assigned a unique item number by the Customizer. This item number is used to identify all occurrences of an item. Each format consists of a format header, which contains pointers and information concerning other control structures, and a collection of item definitions, organized in ascending item-number order. The MOVE CORRESPONDING intrinsic performs its function for each item in the source format (in this case the screen format) which has a matching item definition in the target format (the data set format). The intrinsic locates the field and determines its length, type, and precision, using information stored in the item definition. In this example, the source field is located at byte 0 and is ten bytes long. An item type code of 3 indicates that the field is in display numeric format and the precision is two decimal places.

The target field is located at byte 54 of the data set record and is eigth bytes in length. An item type code of 5 indicates that the field is in packed decimal format, and the precision is four decimal places. MOVE CORRESPONDING copies the field from the screen record to the data set recod, changing the type, length, and precision of the data according to the item definition.

Changing Screens

In addition to changing field and record characteristics, the customer has the ability to modify the appearance of the application itself. Data entry screen appearance, and even the sequence of screens may be altered by the customer.

V/3000 provides a relatively simple method for altering screen appearance. Screens may be redesigned by repainting them using a few control character sequences on HP's 26xx series terminals. This gives the customer the power to alter screens so they look like forms that are presently in use, lessening the technology shock thay many users experience. Screen alterations are then entered into the run-time application dictionary via the customizer and translated into updated record format definitions. The application program is thereby insulated from cosmetic changes to screens. The MOVE CORRESPONDING and other Customizer intrinsics handle changes in data field order as easily as additions and deletions.

In Materials Management/3000, screens corresponding to transactions are at the bottom of a large tree of menus. The 26xx terminal series has eight dynamically definable softkeys. These keys are used by the application as the primary method of moving from screen to screen. The top of each screen in Materials Management/3000 contains eight labels, each corresponding to a data entry screen or a menu. The user may navigate through the menu tree by pressing a softkey that will cause the application to transfer to the desired transaction, or to a menu that will list seven other choices. The eighth function key is always labeled EXIT and takes the user to the screen's parent.

The customer has the ability to modify these labels through the Customizer, creating subtrees for different users. For example, security reasons may require that a customer prevent stock room personnel from altering any engineering data. By removing any labels that identify transactions dealing with engineering data, it is possible to restrict the stockroom personnel to a closed set of transactions.

The application determines softkey definitions by looking up values in a screen sequence table, which is part of the run-time application dictionary and is accessed by Customizer intrinsics. An entry in the sequence table is associated with every screen. Before displaying a screen, the corresponding entry is moved to the program data area. If the user presses a softkey, the application looks up the value that corresponds to the key pressed and transfers control to the appropriate screen or menu. This allows the customer to be very flexible in tailoring the system and relieves the designer of the burden of determining the screen structure while coding.

An additional feature becomes very powerful for experiendced users of Materials Management/3000. A 16-character input field called the command window is present on all menu screens. If the function desired by the user is not directly accessible from a menu, the desired function name may be entered into the command window and the corresponding screen will be accessed directly, eliminating the need to navigate through the menu tree. Whenever the application detects an entry in the command window, a Customizer intrinsic retrieves the appropriate value, effectively providing a ninth softkey. The command window may be altered via the customizer and V/3000 to accept only selected labels. This provides an additional measure of security, while providing the means for the experienced user to travel rapidly from screen to screen.

Processing Logic Customization

It is impossible for the designers of a general-purpose application to anticipate the needs of every customer. Customers will almost always want the application to do some additional processing, beyond the capabilities of the standard product. With noncustomizable applications, the customer would either have to purchase source code and modify it, or live with the standard product. Materials Management/3000 provides two methods of modification. The first involves V/3000 and the second involves the Application Customizer. V/3000 provides a set of powerful functions, including: checking for minimum length, data type checks, range checks, pattern checks, and data formatting. However these functions apply only to data entered on the screen records. To allow customer-defined manipulation and movement of data between screens and data sets, a set of functions called processing specifications may be entered using the Customizer.

Processing specifications are defined by the customer for each transaction where additional processing is desired. Simple commmands allow the user to add, subtract, multiply, divide, and move data items. These commands are compiled and placed in tables that are accessed by Customizer intrinsics at execution time. In most of the product, each transaction is structured so that after all normal processing occurs but before any data sets are updated, the processing specification interpreter is called. This is a Customizer intrinsic that performs the operations indicated by the customer-entered statements. It is possible to alter almost any data item on any data set that is to be updated by a transaction. This tool allows the customer to extend the usefulness of the appplication program to areas that were not originally anticipated by the designer.

Fig. 4 shows how customer processing specifications are implemented. The format header of the screen format contains a pointer to any processing specifications the customer may have defined for the transaction. All processing specifications are generated by the Customizer and placed in a processing specification table, which resides in an extra data segment. The processing specification interpreter uses the pointer and length fields in the format header to locate and move the processing specifications defined for this transaction to the stack. The Customizer generates an intermediate language in the form of triples, which consist of an operation code and two operands. Each operand field is either a constant, a register, or a format/field number combination. In this example, the customer wishes to convert the value entered in field 135 from pounds to grams and accumulate the result in field 222, which is described in format 14. This might occur in the situation where the customer wishes to record the year-to-date quantity ordered for management reporting. Field 135, described in format 22, corresponds to the quantity-ordered, which is accumulated on some other record for use in preparing periodic management reports. The normal unit of measure for ordering is pounds, but for some reason, management has decided to accumulate the total quantity in grams. The first triple moves the constant 454 to register 1. The second triple multiplies the contents of field 135 by the contents of register 1, and places the result back in the register. This converts the value of the field from pounds to grams. The contents of the register are then added to the contents of field 222 in the third triple. Upon returning from the processing specification interpreter, the transaction will update all of the effected data sets. This method of implementation allows the customer to add to or override the processing specified by the application designers.

Local Languages

HP's market for manufacturing applications is worldwide. The application designer cannot assume that the users of an application understand the English language. Materials Management/3000 is designed to be completely localized to any language supported by the 26xx series terminal without reprogramming. Localization may be accomplished by translating the screens using V/3000, by modifying report headings and error messages stored in message catalogs, and modifying other literals maintained in the application data dictionary. Materials Management/3000 uses many single-word literals to control processing. For example, a user may enter engineering information about a part, such as whether it is normally purchased or fabricated. The English version of Materials Management/3000 codes this information as P or F on the data base. The literals P and F will have different interpretations in other languages. Therefore the customizer maintains another table containing all literals defined by the application designer. When manipulating literals entered by users, the application must first look up the current value of the literal. The table is loaded into the program data area and accessed by Application Customizer intrinsics. Because the table is located in the program data area and accessed directly, there is very little additional overhead. NonEnglish-speaking customers have an application product that is easily understandable by their users, and the support burden is minimized for HP because only one version of an application system needs to be supported instead of one for each language.

Security Checking

An advantage of manipulating data in an interpretive mode is that other functions may be added with a minimum of effort by the designer. One example is security checking. Many auditors demand that security access be carried down to the data item level. In Materials Management/3000, each user is assigned a password that will grant that user access to only the data items that he or she is expected to review or update. The password is entered only once on a special security screen. The user may view only screens that contain data items for which that individual has access, and on screens that may be accessed, not all data items may be reviewed or updated. This allows the user to see and manipulate only the authorized items.

This type of security would require a lot of design and coding effort in a conventional system. In Materials Management/3000 the Customizer intrinsics that manipulate data also perform a security check. The Customizer maintains a table containing all valid passwords along with a list of data items to which that password grants access. Each time a Customizer intrinsic accesses a data item, a table lookup is performed. If the user does not have access to the item, an error message is displayed on the terminal. This powerful feature is implemented with a minimum of overhead and design effort.

CONCLUSIONS

Materials Management/3000 is HP's first user-customizable, factory-supportable application system. The team that designed and implemented Materials Management/3000 has verified that an object code user-customizable product is a good idea and that it works. The performance of such an application can be acceptable. As the installed base expands we are gaining a better understanding of our customer needs. In general, the customers really like the product. They just want HP to expand its capabilities. Two examples are discussed next.

We are looking into providing more customization features. A frequent request is to allow the user programs greater access to the application data base. This could be accomplished by allowing the user access to our customizer instrinsics. The data base would only be accessed through these intrinsics and therefore the integrity of the data base could be insured. Users are also requesting the ability to add data sets, and gain more flexibility in associating data items. Even with expanded capabilites, though, some customers will still want more flexibility. For example, they want to write their own programs to perform expanded data validation and they want the application to call these programs. Another feature request is program logic customization. The user would be able to select among pre-coded algorithms.

In another area, the design team is exploring the implications of a distributed application. Currently, Materials Management/3000 is a single application system that runs on one HP3000 machine. A distributed application system would involve functions and data spread out among several machines. We need to understand how to distribute the data and how to handle the customization of data distributed throughout an application network. These are only two areas of research. There is a lot to do.

Built on the technology of an application monitor and an application customizer, Materials Management/3000 is HP's first step toward providing a total solution to the problems of manufacturing companies.

ACKNOWLEDGMENTS

We would like to recognize the lab and marketing teams who spent two and one-half years designing, implementing, documenting, and marketing the product. Special thanks go to Steve Baker, Dick Dolan, Beth Eikenbary, Mike Kalashian, and Bob Poulos for their contributions to this article.

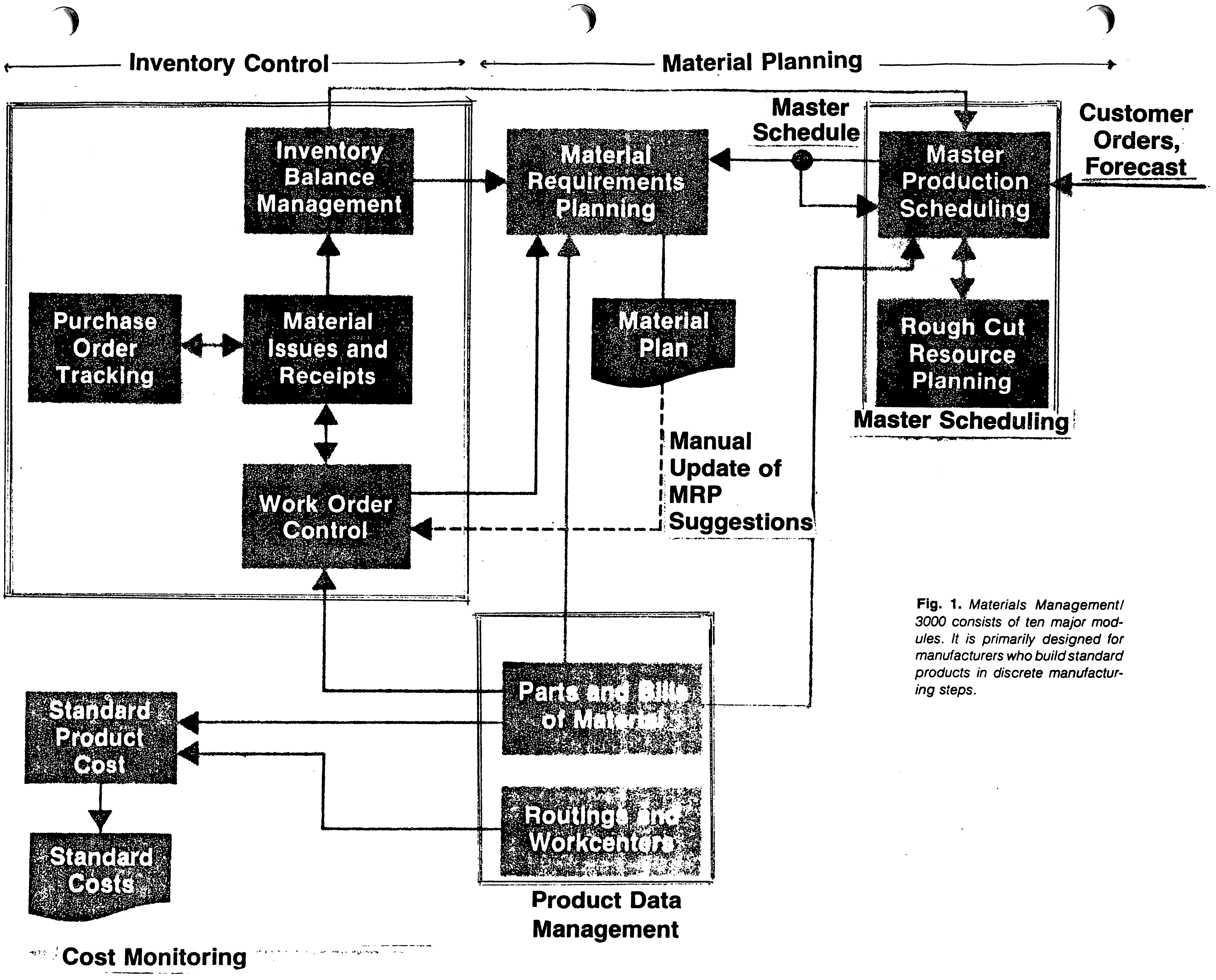

Inventory Control
Material Planning
Inventory Balance Management
Material Requirements Planning
Master Schedule
Master Production Scheduling
Customer Orders, Forecast
Purchase Order Tracking
Material Issues and Receipts
Material Plan
Rough Cut Resource Planning
Master Scheduling
Manual Update of MRP Suggestions
Work Order Control
Parts and Bills of Material
Standard Product Cost
Routings and Workcenters
Standard Costs
Product Data Management
Cost Monitoring

Fig. 1. *Materials Management/3000 consists of ten major modules. It is primarily designed for manufacturers who build standard products in discrete manufacturing steps.*

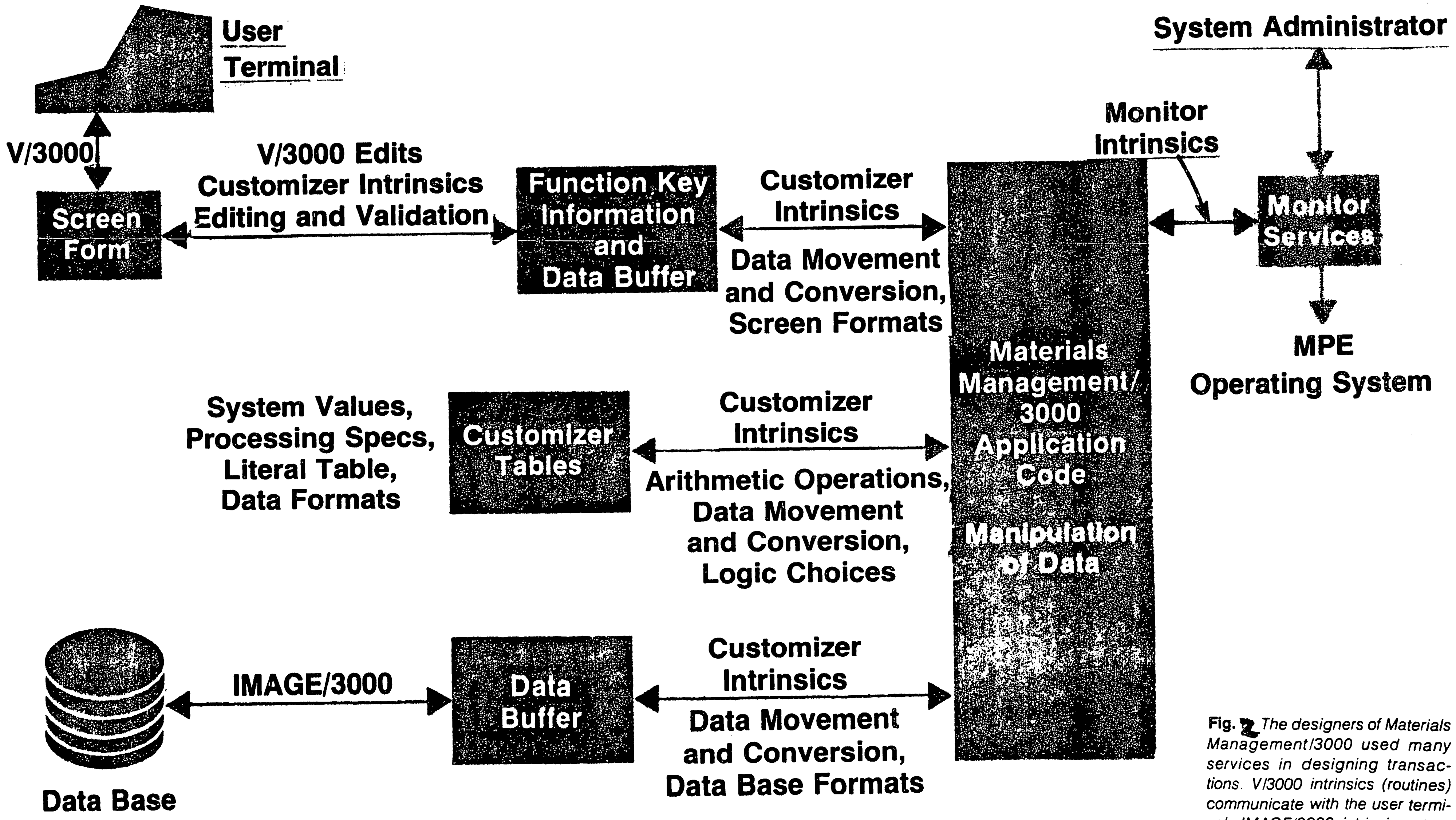

User Terminal
V/3000
Screen Form
V/3000 Edits
Customizer Intrinsics
Editing and Validation
Function Key Information and Data Buffer
Customizer Intrinsics
Data Movement and Conversion, Screen Formats
Monitor Intrinsics
System Administrator
Monitor Services
MPE Operating System
Materials Management/3000 Application Code
Manipulation of Data
System Values, Processing Specs, Literal Table, Data Formats
Customizer Tables
Customizer Intrinsics
Arithmetic Operations, Data Movement and Conversion, Logic Choices
Data Base
IMAGE/3000
Data Buffer
Customizer Intrinsics
Data Movement and Conversion, Data Base Formats

Fig. 2 *The designers of Materials Management/3000 used many services in designing transactions. V/3000 intrinsics (routines) communicate with the user terminal. IMAGE/3000 intrinsics store and retrieve data. Application Customizer intrinsics retrieve data item definitions, screen formats, data set formats, and customer-added processing specifications. Customizer intrinsics also manipulate any data items whose characteristics are unknown to the application designer and must be looked up [illegible]he Customizer tables.*

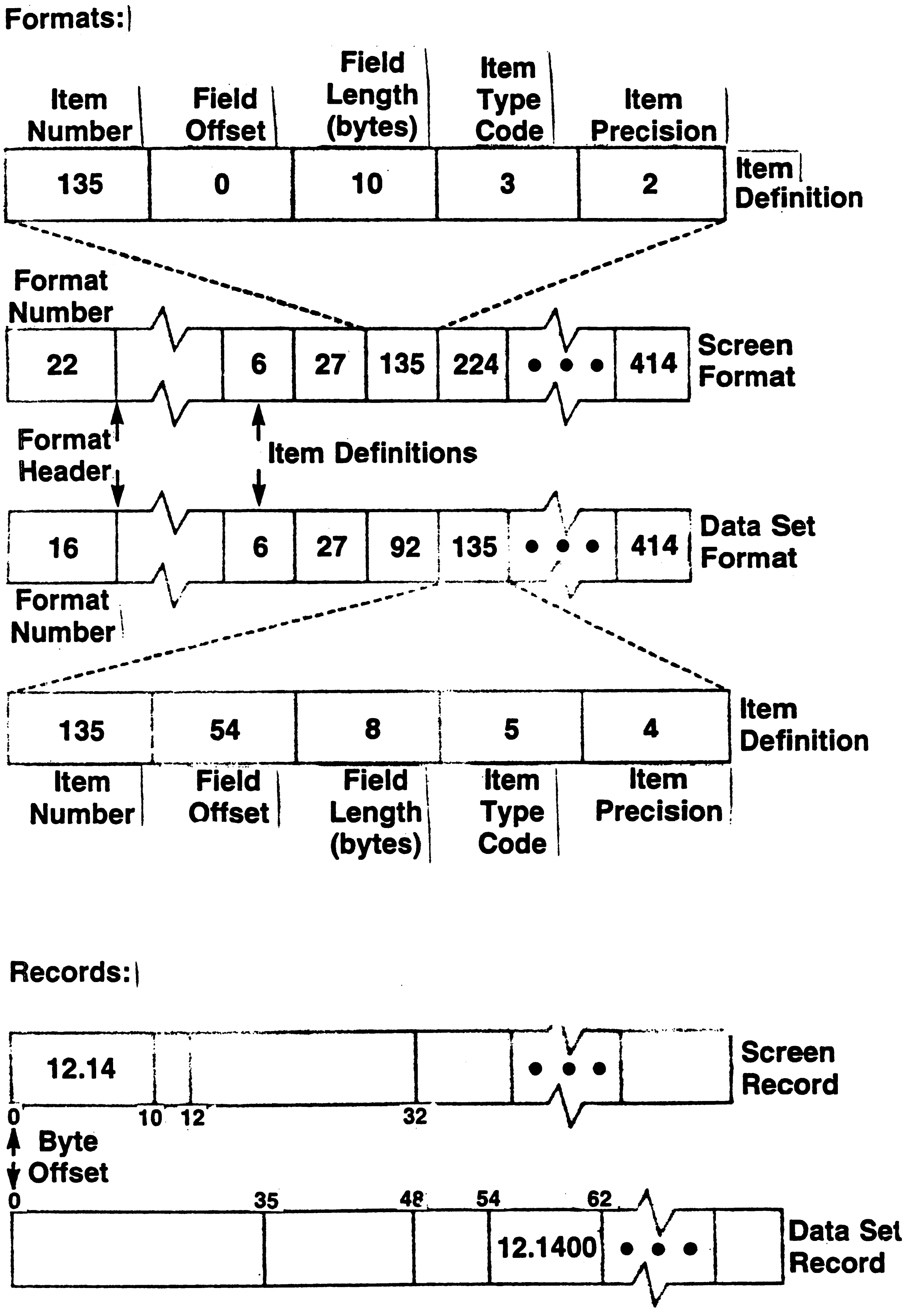
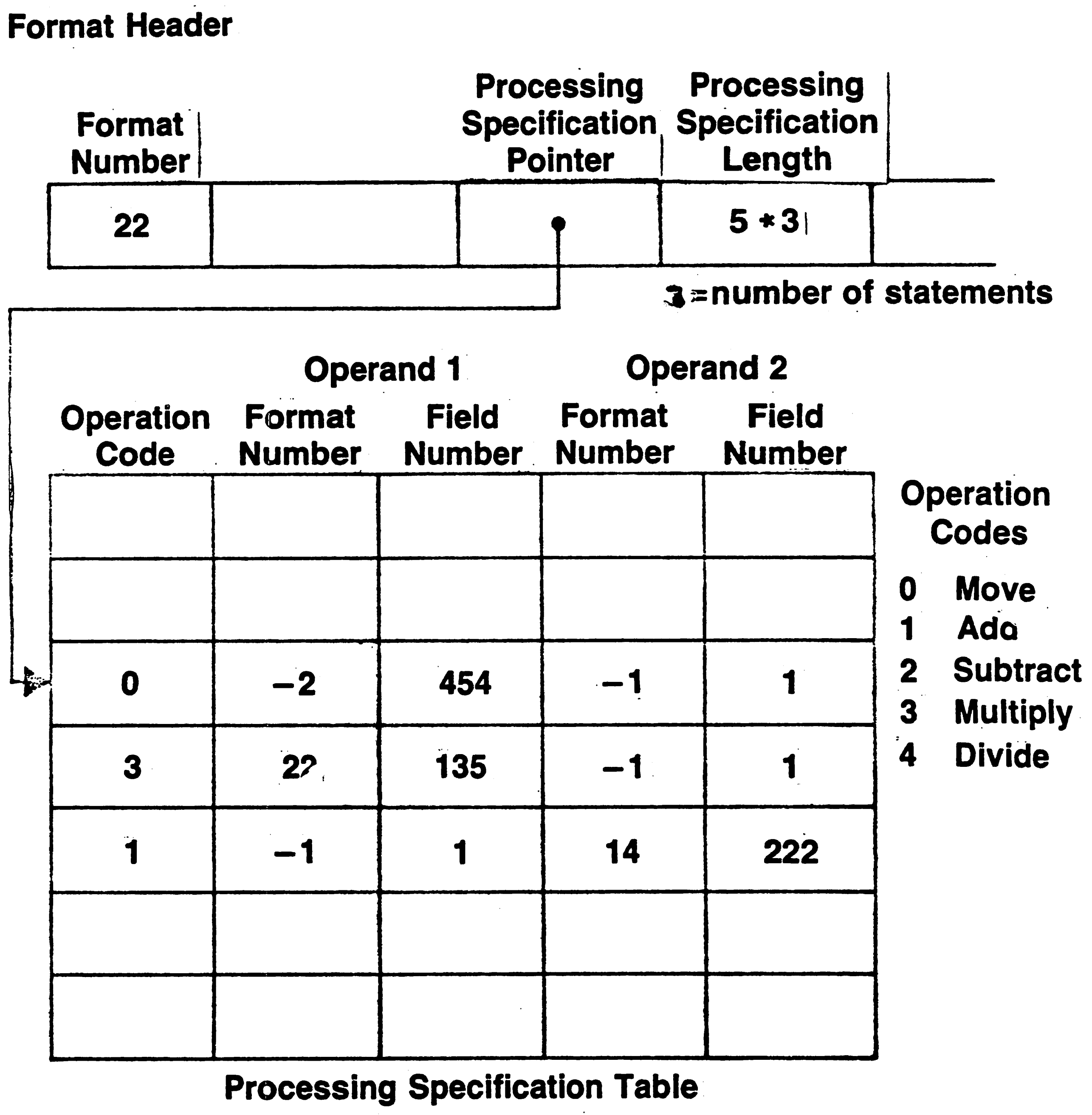


Fig.3 *An example of the operation of the MOVE CORRESPONDING Customizer intrinsic. See text for details.*



Format Header

Format Number		Processing Specification Pointer	Processing Specification Length	
22		●	5 * 3	

3 = number of statements

	Operand 1		Operand 2	
Operation Code	Format Number	Field Number	Format Number	Field Number
0	−2	454	−1	1
3	22	135	−1	1
1	−1	1	14	222

Processing Specification Table

Operation Codes

0	Move
1	Add
2	Subtract
3	Multiply
4	Divide

Fig. 4 *An example showing how customer-specified processing is implemented. See text for details.*

MOVING TOWARD INFORMATION MANAGEMENT IN AN INTEGRATED ENVIRONMENT

Benjamin J. Ventresca, Jr.
Manager
Touche Ross & Co.

Government and business leaders both in our Nation and in other nations have been stressing the need for a raised awareness to overcome the pressing economic problems of today such as the steady upward inflationary spiral and the uncontrolled drop in productivity. In our own country, the campaigns of last year's election have helped to raise our awareness of the need to revitalize America and to increase our productivity. The present outcry for economic renewal and for sound fiscal policies are perhaps the greatest since the Great Depression. As we all know, there are no easy solutions to the problems which face us, but a growing concensus points to existing and future technologies to hold the key. However, technology alone is not the answer. Today much job design in the United States is better suited to robots than to mature adults because of the increased use of technology and automation which has made work more simplified, standardized and routine. In the past, advances in technology with their resultant economics, and greater managerial control have increased productivity, which in turn has contributed to a general increase in affluence, education, and the level of aspirations of Americans.

As a result of these benefits many people rightly want jobs that allow them to make greater use of their education and skills, and that provide intrinsic work satisfaction while affording the opportunity to enjoy the luxuries of our society. In effect the nation may have arrived at a point in conflict with itself and the result of this conflict is that productivity is plummeting to an all time low.

At the same time, America's leaders, as well as their counterparts in other industrialized nations throughout the world, are advocating a renewed commitment to the free market system. Their expectation is that the return of business incentives will spur greater investments back in people and products thereby helping to increase productivity and job satisfaction, and to reverse the current inflationary spiral. This commitment mirrors an international trend towards deregulation and the elimination of obstacles to entrepreneurial freedom and corporate growth.

We are all aware of the effect of inflation in our own perlonal lives and are reminded of this every time we watch the evening news or return from a trip to the supermarket. These effects are also felt in all aspects of today's business operations. Increasing material and inventory costs, steadily rising labor costs, and high costs of financing are but a few of the factors combining to add external pressures to our nation's businesses. These pressures affect all industries, but naturally the effect varies with each. Additionally, these pressures impact all aspects of business whether they be income-producing or revenue-consuming. Traditionally, management has invested disproportionately in certain of these aspects. During the past ten years, management has increasingly turned to data processing to provide monitors and controls over the production-oriented functions (such as Inventory Control, Order Processing, etc.). Investments in hardware and software have been heavily justified by projected increases in sales and by anticipated savings in production. This philosophy has created an imbalance between capital investments in income producing functions versus typical "overhead" functions. In the latter, investments have been of secondary importance because of an imperception of the need to stimulate and support "white collar" activities. No doubt you've seen those figures comparing the low state of office productivity with a better record chalked up by industry and agriculture. They get repeated often in management circles and have been verified (with small discrepancies) by a various number of sources. Basically these figures are as follows:

	Capital Investment Per Worker	Productivity Increase During the Past Years[1]
White Collar Worker	$ 2,000.00	4%
Industrial Worker	$25,000.00	90%
Farm Worker	$35,000.00	185%

While these figures are sometimes disputed in terms of their detailed composition, their value should be viewed on a broader scale. The challenge for today's executives is to make

---

1 "Editorial", Walter A. Kleinscrod, Administrative Management, October 1980, pg. 23

the same level of commitment to the administrative and professional functions as it has to the operation functions. This commitment must be made in order to improve the effectiveness of workers in the office as we have improved the effectiveness of workers in industry or on the farm. In order to help fulfill this commitment we, the "experts" of the information industry, must demonstrate the importance of information in today's business world. To do this, we must first acquaint ourselves with the presence of information in various forms throughout our organizations and understand the role that it plays in the organization. This means that we must strive to understand those technologies and applications which traditionally have not been an integral part of Data Processing (such as Text Processing, Telecommunications, Reprographics, Micrographics, etc.)

The Information Resource

Our primary challenge must be to educate both management and staff in the value of the Information Resource. To do this effectively it is critical that we establish a common base of understanding and a common set of expectations. This challenge is compounded by the fact that the Information Resource is a concept rather than a tangible product. Webster defines a resource as:

- An available means or property
- Natural advantages or wealths
- A capacity for finding or adapting means of achieving
- The power of achievement
- A skill or ingenuity in meeting any situation

These definitions really don't help us describe the Information Resource any more than a few words can properly describe the six senses. But since the Information Resource is a concept and not an object, it can best be described in terms of the unique environment in which it applies. The Information Resource is different to a manufacturer than it is to a service organization. Therefore, in order to educate management and staff in the value of the Information Resource, we must first define the scope and breadth of the Information Resource within the individual organization. Once the discreet components of the Information Resource has been identified, it's our challenge to envision the role of this cohesive base of information in the organization's current and future operations. To do this we must describe that role in terms of its impact on the firm with regards to the business environment, to the operating procedures, and to the bottom line. To be effective, the Information Resource must be perceived by the people in the organization to be an extension of themselves and should help foster their association with the organization itself. This will result in an improved awareness of their value to the organization and their impact on everyone's success. The keystone to the vitality of the Information Resource is its ability to match the organization's strategy and structure and to provide a medium for the organization's success in meeting its corporate missions.

Distributed Information Processing

If we look at the Information Resource as being a broad based commodity which serves all aspects of an organization, we have to think in technical terms of its orientation to a distributed/integrated information processing environment. To many technicians distributed data processing simply means a dispersement of computer hardware and data to multiple sites around an organization. The shortcoming of this definition is that it overlooks a wide range of non-data processing activities and issues that help make information systems work. Similarly, it does not satisfy the need of an Information Resource to align itself with an organization's strategy and structure. A broader definition of DDP acknowledges that information processing is an organizational resource composed of many areas of activity which are performed and controlled by various and diverse individuals. In reality, DDP can only truly exist in an integrated information environment. By that we mean that Distributed Data Processing is the composition of interdependent functions which draw and build upon a common base of information. These functions can vary from traditional data processing applications to newer word processing functions, and looking to the not to distant future, to extensive use of electronic mail and computer-based communications. With this broader definition and application, the term Distributed Information Processing becomes more representative.

Basically, in order to appreciate the value and potential of Distributed Information Processing in an integrated environment we must first accept the fact that today's technologies transcend the traditional structures of business ten years ago. With today's technologies, these different functions are converging and are often indistinguishable. Distributed Data Processing operations based on localized minicomputers are bringing their capabilities closer to the administrative user, and are becoming a more responsive and attractive alternative for traditional Word Processing applications such as text management. At the same time, increased list and records management capabilities of word processing involve activities which were once the sole domain of data processing. The challenge in making distributed systems work is to design and implement systems which tie together the diverse function of an organization and break down the communications barriers which naturally exist in any organization, thereby providing a more natural use of information.

The objective of this presentation is to identify the value of integrated information, to surface the major concepts of integrated information, and to highlight those issues which are critical to successfully implementing an integrated information environment.

<u>Why Distributed/Integrated Information Processing?</u>

In order to understand the value of integrated information in today's business environment it is helpful to take a quick look back at how technology has changed the way we process information. To do this let's look at both the development of traditional data processing and at the development of automation in the modern office. In the mid nineteenth century the "modern office" was graced with the invention of the adding machine and the manual typewritter. Each of these products promised a high potential for increased staff productivity. During the earlier part of the twentieth century (from 1900 to 1940) the main focus of scientists and inventors was in developing and perfecting mechanical devices to improve on the adding machine and the manual typewritter. Generally, little increase in productivity was realized. Even though it was 1946 when the first electronic computer was put into operation, it wasn't until 1964, when IBM introduced the System/360, a new concept in commerical computers, and also introduced word processing in the form of Magnetic Tape Selectric Typewritters that real increases were seen. After this time, in the late 1960's and early 1970's, the ability to achieve and maintain a high level of productivity was largely dependent on the prudent use of technology. During that period, the impact and effectiveness of technology in the business environment was generally clearly defined. Computers were used to perform data manipulation and processing to improve accounting and management effectiveness. These computers were maintained in a controlled environment pretty well divorced from the mainstream of daily office life. From a historical perspective, the computer was thought of as a tool (granted more sophisticated than its earlier counterparts) but a tool none-the-less. Meanwhile out in the office there were other tools - typewriters (some manual, some electric, some with memory), copiers, adding machines, file cabinets, etc. which sought to increase staff productivity through speeding the process of performing their functions. Hence, a newer, better typewriter simply replaced an older, slower typewritter. A bigger filing cabinet replaced or supplemented a smaller one. If you were to ask someone to describe the typical office chances were that you'd get a list of some of the equipment in it. This list would probably include the items mentioned earlier, and most of them would also have been used to describe the "modern" office of 20, 50, or even 100 years ago. In reality, businesses, with exception of course, really hadn't changed much in quite a while. Electric typewritters replaced manual ones, calculators replaced adding machines, telephone switchboards were automated, and everything looked more streamlined but most tasks in the modern office are performed much the way the have been for decades. The focus on using technology never really touched on changing the way that tasks were done and how information was used.

From a data processing perspective, the centrallized DP environment of the 60's reflected the image of corporate thinking towards information processing. Information was a discrete commodity; that which belonged to accounting was processed by accountants and clerks, that which belonged to management was processed by administrative support personnel and very rarely did one function interface with the other. This led to an increased sense of propriety and helped insulate the different departments within an organization.

Distributed/Integrated Information Processing is a concept which has only recently come of age. Technological, economic, and educational developments now allow us to design information systems that may achieve the objectives of matching the organizational structure, supporting the business strategy, and reflecting the character of the organization itself. The effect of distributing information in an organization is one of breaking down some barriers, of establishing communication channels that never existed, and of raising the awareness of users in different departments within an organization of what other departments are doing with information. With this awareness comes a realization of what information may be available for mutual use and what benefits might be realized. It's the recognition of the potential use of information and the acknowledgement of the resultant benefits through sharing and communication that allows the concept of the Information Resource to become a reality. Without integration, the Information Resource is merely a collection of discrete bits of information.

What's Involved in Establishing the Integrated Information Resource?

When embarking upon integrating information and developing the information resource, we have to first sit back and look at the way the organization conducts itself to be sure that the scope of the resource is comprehensive and change-oriented. Technologies such as process control, computergraphics, laser printing, intelligent copying and integrated telephones and switching units, don't always apply to every organization. However, the actual or potential need for any of these or other technologies could be a deciding factor in the structure of the Information Resource in terms of the nature of the information base and the type of equipment used to process and store the information. Just as in designing a data base it is prudent to anticipate and allow for future enhancements and/or modifications, in planning the Information Resource it is imperative to consider and evaluate future issues. This is why we must develop and review long range strategies and be aware of trends in business and technology. Five years ago, people planning an information system could be content with assessing the needs and direction of their own organization. Today, however, we must be mindful of external concerns such as inter-company communications and information exchange. The impact of advances in the areas of communications and human system interfaces mandate a closer, more deliberate look to the future. For example, while the concept of using electronic mail may not be feasible for a given organization today, it is our responsibility as the planners and designers of this information resource to determine the probability of the need to tie that capability in at some future point. Additionally, when defining the nature and extent of integration amongst functions, it is important to probe into the current methods of operation, into the current use of information, and examine the effectiveness of the current approach versus potential alternatives. For example, trends in office organization have promoted the use of functional centers and word processing centers as opposed to individual secretaries. These centers receive work, process the work and return it to the originator in the same fashion that a data center would process work from various users and sources. Many organizations which operate under this approach are presently evaluating the merit of centralized dictation which replaces the dictating equipment on a professional's desk with the use of the telephone into a dedicated processor/storage device. This device digitizes the dictation and treats it like any other stored information on magnetic media. This information can then be manipulated by the processing center or can be shared with other processing centers in the same way that stored information can be manipulated by any data processor in a more traditional environment. While this is an individual example, the intent is to show the potential breadth of application and technology which the Information Resource must address. These actual and potential needs should be surfaced during the design process (which will be briefly discussed later in this presentation).

Issues of Integration

Once the decision to integrate has been made the next step is to address the major issues that are associated with integrated information. Briefly, those major issues are:

1. Management and Control of the Information Resource
2. The Mode of Functional Integration Required
3. The Functionality and Accessibility of the Information
4. The Skill and Experience of the Various Users

Management and Control

As data processing matured, the organization which housed data processing also matured. In the early stages, the purpose of data processing was to provide an automated means of performing established functions, such as processing payroll, reporting and maintaining accounts receivable information, etc. The rationale for data processing was basically to lower cost and time, and to achieve a higher degree of reliability and accuracy. To provide these services, the data processing facility was operation-oriented and became a function within finance. The EDP manager reported directly to the executive of finance and was normally on an organizational par with the controller or assistant controller. In the later stages of data processing development, the purpose expanded to include automating the processing of business functions and to provide comprehensive management information. The domain of data processing expanded from just accounting functions to incorporate business functions and management reporting and control. The rationale for data processing was to improve efficiency, to sustain reliability, and to increase profitability. In essence, it also provided a decision-support mechanism. To support these new objectives, the data processing facility broke away from the direct control of the executive of finance and became an anomaly of sorts in that it was too unique to be treated like any other division in the organization. The director or manager of information systems was still not perceived to businessoriented enough to merit executive status or participate in the corporate planning process. During this time, this maturing group began to concern themselves with information planning as well as systems development and systems operation and with the spread of computer support throughout the organization, operations now began to concern themselves with issues like communications and user interface. In the 80's, the purpose of information processing is to establish and maintain a base of information which reflects all key aspects of the organization and which provides a catalyst for continued growth and effectiveness. And in keeping with the Information Resource we have been talking about, we see that it indeed touches on all aspects of the organization. The rationale for doing this is that it then provides a means to sustain growth and effectiveness in all phases of the business operation and it minimizes the effect of external contraints. In order to support these objectives and to truly be a part of the organization's plan, information processing has finally come of age and has established itself as a corporate entity in its own right. As such, it merits its own executive with a role in corporate planning, and has a structure which spans information services, communications, advanced office systems, and customer services. The key to the success of this structure is that the management of the Information Resource must pass to someone who can effectively relate well in all levels of the organization (horizontally) as well as in all functional areas of the organization (vertically).

One of the greatest challenges in managing the Resource may arise from balancing the needs and approaches of the traditional MIS function with those of the traditional WP function. This is due to the idiosyncrasies of each and is compounded by the pre-established misgivings between them. Characteristically, the major problems of word processing have not been in the area of technology or systems design. The majority of the problems have been and continue to be people problems. Word processing totally changes the social structure of an organization. If it is not total or if management's commitment is not total, there is usually chaos which results in a waste of time, energy and money. Therefore, the manager of the Information Resource must be more people-aware than the manager of a data processing facility ever had to be.

Modes of Integration

Depending on the type and volume of information being processed throughout the organization, the modes of integration will vary in each instance. Word processing hardware is more function-oriented than data processing hardware. Therefore, the acceptability of one model to an end user is much more critical in the WP environment than in the DP environment. This fact often negates the use of mainframe hardware to perform WP functions thereby eliminating some otherwise obvious options. Basically, there are three modes of integrating data processing and word processing. They are:

- Multifunction System - which is capable of storing and processing information in either a traditional data processing or word processing sense. Typically, this would be a mainframe computer with capability of interfacing with data processing or word processing CRTs as well as with a wide range of peripherals (from Laser printers to intelligent copiers) as well as with other processors in either a direct or a remote sense.

- Shared Resource - which is a cluster of single or multifunction systems which have primary functions that would share some common resources such as storage, printers, and other peripherals. This is akin to a distributed environment with a series of minis commun- icating in varying degrees of compatibility.

- Standalone - single function computers with no automatic interfaces and along which information is integrated on a batch assess and retrieval basis only.

Functionality and Accessibility

The third major issue revolves around the required functionality of the system and of the information. This touches on the design of the database as well as the selection of the particular hardware used to perform various functions, and is closely related to the above issue.

Unlike data processing, word processing applications are typically preprogramed and are function key oriented. This presents a major obstacle in retraining support personnel from one type of equipment to another. Therefore, in terms of hardware functionality, the type and volume of function that is being performed is of critical concern. The data base should be designed to allow access through a series of paths either directly or through logical record relationships.

Skill and Experience

The final major issue revolves around the skill and experience of the users themselves. As distributed data processing became commonplace in the DP environment, data processing professionals have learned to be more aware of nontechnical people such as clerks, managers and executives. The priorities, expectations, and needs which face this diverse group of users is greatly different from those which face data processing professionals and technicians. For example, most data processing managers don't have to devote their energies and skills to three and four cycles of revisions of two and three page memos and would have a hard time in adjusting to the environment The psychology of dealing with the varied demands and responsibilities is critical to the success of an integrated information environment.

Naturally, there are no easy solutions to these and there are other issues and pitfalls that are associated with integrating functions and information in today's business environment. These are as varied as the organizations themselves. The key element is that in launching into an integration effort, with functions as well as information being addressed, it is of critical importance to realize that the requirements definition and user review process and the proper organizational plan is vital to the success the entire effort.

An Approach to Creating an Integrated Environment

Traditionally in the office environment, and to a lesser extent in a data processing environment, users have been "sold" someone else's perception of a solution to an immediate need; rather than "buy" a solution which satisfies their immediate needs and is compatable with the overall information plan of the organization. In order to develop and implement such a plan it is important to take a multi-phased approach which is designed to establish and maintain the common expectations and objectives of the various users throughout the planning, development and implementation process. One such approach is composed of four major phases. These are:

- Planning Phase
- Design Phase
- Development Phase
- Implementation Phase

Planning Phase

The purpose of this phase is to define the information objectives, to formalize operational concepts, to establish common expectations, and to determine the feasibility of integrating functions and information. The two major steps within this phase are: an Initial Investigation and the Feasibility Study. The Initial Investigation looks to evaluate general needs throughout the organization and to establish the value of proceeding with the Feasibility Study. The Feasibility Study should develop the conceptual characteristics of an information solution through different alternatives, and to define the projected costs benefits for each function within the organization for each alternative which has been defined.

Design Phase

The focus of this phase is to develop the detailed foundation for the potential solution. This solution will be a blend of manual and computerized system procedures which satisfies the individual users requirements and support the goals and character of the organization as a whole. The major steps within this phase are: Business Analysis, Operations and Information Analysis, Technical Environment Analysis, a Conceptual Design, a Systems Review and Evaluation, and the Creation of the Development Plan.

The Business Analysis is intended to identify the corporate goals and objectives and define how those goals and objectives cascade through the various departments and functions within the organization. The result of this analysis is to define the structure within which the information needs of the organization should be viewed. The <u>Operations and Information Analysis</u> is made up of two major components; the first being an organization survey which is intended to define the individual job functions and responsibilities, needs, and work patterns throughout the organization and to identify how these functions are qualified and how they support the corporate goals. The second component is a technical survey conducted to define the objectives, functions and inter-relationships of information in a technical sense (i.e., in terms of systems and operations). The <u>Technical Environment Analysis</u> is intended to determine the requirements and constraints of the organization, which will affect the environment within which the solution must function. The next step in this process is the <u>Conceptual Design</u> within which the objectives and requirements determined above will be addressed in the confines of the technical environment defined previously. This design will show how each function will be performed and will generally define the interaction level of other functions under the new approach. The <u>System Review and Evaluation</u>, which looks to identify and review the hardware and software alternatives, will support the conceptual design and will assess each alternative in terms of its satisfaction of the objectives, constraints and expectations.

The final step in this phase is to Create the Development Plan for the implementation of the proposed solution defining target dates and deliverables through the completion of the development process.

Development Phase

Based on the design developed in the prior phase, the focus here is to develop the actual system and prepare for implementation. This is done through the following steps: A Manual Procedures Design, An Application Specifications Development, User Orientation and Training, Implementation Planning, and System Testing.

The first step, Manual Procedures Design, looks to develop the detailed manual systems design and to review these with the users. This design will provide the blueprint for orientation and training, and for the system testing which follows. The second step, Development of Application Specifications, Coding and Testing of the applications themselves looks to develop the programs, through module testing, and supporting documentation. The third step, User Orientation and Training, is intended to serve two purposes. The first is to help the user understand the integrated information concept, how it relates to their particular responsibilities and how their efforts blend with those of others; and secondly to train them in the use of the new system. The fourth step, Implementation Planning, seeks to identify the test criteria and test cases which will be used to validate the accuracy of the system. Secondly, to develop the schedule for the implementation of the system from system testing through the start of production processing. The final aspect of this step is to develop the plan for the conversion from current operations to the new system. The last step in this phase is Systems Testing which provides an integrated test of manual and computerized systems in a controlled environment.

Implementation Phase

Upon successfully concluding the system testing, the development process enters into this phase, which is intended to establish an operable environment for the "production" use of the new system. The major steps within this phase are the Phased Conversion and Implementation and the Refinement of Operating Systems.

The first step, Phased Conversion and Implementation, brings each function into a production mode according to the implementation schedule and expands the integrated information base by its own functionality.

The second step, the Refinement of Operating Systems, is intended to identify areas of need for modification and fine tuning and to allow management to prioritize and schedule these needs for implementation without impacting the progress of the main development effort.

There is an advertising slogan that states, "...the future belongs to the efficient ..." While the intended thrust of that slogan is directed toward conservation of energy its merit holds true for businesses as well. The challanges that face us both in our business and personal lives will require our full understanding and reaction. Information will play an increasingly important role in how we meet those challenges, and we cannot afford to waste time and money gathering or sifting through data in order to get it.

America's businesses are being charged with a mandate that requires a commitment to improved business effectiveness and profitability. Now more than ever, their success in doing this will depend on their ability in motivating and satisfying their employees and extending their skills and experience with the use of technology. We have demonstrated the ability to do this well in industry and agriculture and must continue to do so. The new challenge is to match those achievements in the office. The technology to do this is here today and is being followed with impressive advancements for the near future. The understanding of how to use these technologies and how to exploit our most abundant resource - Information - is the opportunity that we share.

MICRO-DISTRIBUTED-PROCESSING

By J. Michael Mason

Maryland Computer Services, Inc.

ABSTRACT

Exploring the expanding capabilities of intelligent peripherals and micro-processor based computer systems such as the desktop computers and small business systems, this paper will bring an awareness to the system analyst and distributed system network designer of the importance of "THINKING SMALL".

Most of us tend to relate distributed processing to the Macro Scale "DISTRIBUTED SYSTEMS" such as HP's DS/3000 and DS/1000. We should be considering the potential of any form of processing from the "dull-normal" terminal capable of performing only rudimentary field editing to the small business systems with sophisticated operating systems and data-base management software, in any distributed processing solution.

Application examples of distributed processing solutions that utilize HP desktop computers, HP 250 systems and HP intelligent terminals will be presented to illustrate how and where you might implement these systems as part of your own distributed processing environment. Considerations in cost, development and the user environment will be discussed.

Todays technology and economic environment continue to challenge those faced with developing data processing solutions. "Main-Frame-Mentality" gave way to mini-computers and distributed systems in the 70's. The Micro-Processor will have the greatest impact on systems design through out the 80's.

By examining how the micro-processor can be used in todays distributed processing environment and bringing an awareness of the potential impact on system design for the future, this paper will provide an excellent introduction to the Micro-Processor and some good reasons to "THINK SMALL".

The MPE IV Kernel : History, Structure and Strategies

John R. Busch
Member of the Technical Staff
Hewlett Packard Corporation
Computer Systems Division
19447 Pruneridge Avenue
Cupertino, California
95014

Abstract

The MPE IV kernel is the result of over three years of research and development undertaken at Hewlett Packard's HP 3000 R&D lab in Cupertino. It provides a new high performance, integrated, extensible foundation for the 3000 operating system, MPE. The project's history and the kernel's characteristics are described. Project objectives, investigation approach, implementation methodology, functional characteristics, resource management objectives and strategies, and performance results are presented.

1. Introduction

The evolution of the HP 3000 family towards large main memories, fast processors, and large and distributed configurations stressed the original MPE kernel design and implementation. It became clear that just supporting the evolution in terms of kernel data structure extensibility would become a problem. Moreover, the original algorithms could not be relied upon to exploit the performance potential offered by the larger configurations.

Project objectives for a new kernel were established. Research into the growth and performance limitations of the old kernel and into state of the art approaches to resource management policies was undertaken. Alternative designs were established, implemented and evaluated.

This process, culminating in the MPE IV C-Mit, is presented in the following sections.

2. Kernel Project Objectives

The primary project objectives for the MPE IV kernel were to provide :

(1). support for the evolution of the HP 3000 family;

(2). maximum performance across the family members;

(3). high reliability and improved fault detection and recovery;

(4). increased functionality as required by the other system components of MPE IV; and

(5). simple extensibility when unforeseen system requirements surface.

3. The Investigation

The 3000 architecture, workload, and evolution were to be matched by the new kernel.

The investigation would procede by : identifying the characteristics of the workload; determining the growth limitations and performance problems of the existing kernel; researching and consulting to determine promising approaches to kernel design; and formulating alternative designs.

Instrumentation was placed into the old kernel, and the system was measured under reproducible, representative environments.

Service requirements induced by the workload on the various system servers were determined. Distributions of segment sizes, processing requirements, and access requirements to secondary store were observed.

Resource utilization was measured. Disc, main memory and processor queue lengths, request type distributions, and overall utilization were determined.

Migration among the servers and service delays were characterized. Process stop type (eg segment fault, disc I/O, terminal I/O, time sliced, etc.) distributions and service delays for each stop type were measured.

These measurements were to be used not only in isolating invariant workload characteristics and performance problems but would also be used as the base for later comparisons. (The specific growth and performance limitations of the old kernel are addressed in later sections).

The literature was researched and academicians consulted to ensure that the lessons of the past and the academic investigations whose results offered potential were taken into account. Although extensive literature was available on resource management policies, very limited literature was to be found which directly related to the segmented architecture of the 3000 family. What was acquired from the research and consulting was a set of principles, measures and ideas which could be incorporated into the design and implementation.

The investigation phase resulted in an understanding of the problems and limitations of the old kernel, and a set of alternative strategies which eliminated the limitations and problems and offered potential in satisfying the overall objectives. Rather than coming up with the eventual design, the investigation came up with a commitment to try out the alternatives and select the best strategies for the architecture and environment based on measurement rather than intuition.

4. Performance Goals and Strategy

In this paper, a transaction is considered to be a step in an interactive session which begins when carriage return or enter is hit and terminates when the system is ready to accept further input form the session.

The global performance indices for the intended application environment are :

(1). transaction response time;

(2). transaction throughput;

(3). fairness;

(4). batch throughput.

The desired system behavior is as follows :

- For a given workload and configuration, the system should provide minimum transaction response time with maximum transaction throughput. Batch performance should be "acceptable."

- Under increasing load, the system should be stable. As the load increases, transaction response time should degrade gradually and fairly. System throughput should stay high even under very heavy loads.

- The system should dynamically tune itself to optimize performance for the current workload with the given configuration. However, explicit control over the relative service between transactions and between interactive and batch should be available to the operator and system manager.

It must be kept in mind that the bottom line performance of the system is measured by the global performance indices and not by the factors which may influence them. This suggests that the performance strategy should be directed towards optimizing the global indices and not towards optimizing indices local to each system component.

The selected overall performance strategy was to achieve maximum system performance by having the system components cooperate to optimize the global performance indices. This approach is fundamentally different from having each system component attempt local optimization and hoping the result will be good overall performance.

Measures and associated instrumentation were defined for the global and local performance indices and supported by the measurement interface. With these measures, the effects of alternative strategies could be understood and evaluated. The measurement interface through performance tools would be made available in the field so that on-site trouble shooting and tuning could be performed.

5. Implementation Methodology

In order to achieve the desired high reliability and natural extensibility, the implementation would have to be highly structured.

Interfaces between system components would be explicit, general, and adhered to. Access to kernel services and internal information would be available only through the use of explicit messages or the invocation of kernel interface intrinsics. An adequate set of interface intrinsics and a general, efficient internal message system would be required to support this structured interfacing.

Within the kernel, a structured implementation was absolutely necessary so that alternative resource management policies could easily be incorporated, coexist, and eventually be deleted.

Performance considerations at the instruction level would be of secondary concern in favor of a structured implementation. The sought after high level of system performance would be achieved through integrated, parallel policies rather than by relying on highly optimized code sequences.

The algorithm selection process and the support of MPE IV performance tools would require that complete instrumentation and an instrumentation interface be carefully designed into the new kernel.

6. The Internal Message Facility

A high speed memory resident message facility was defined and implemented. The facility is intended for the transmission of operating system status and control messages. This facility eliminates the need for supporting multiple ad hoc communication mechanisms.

The message facility associates a message harbor with each process. Each message harbor contains 32 message ports. Each message port contains a FIFO queue of messages, where a message is up to 5 words in length (maximum length is configurable).

Message intrinsics are provided to send a message to any port of any process, to determine the status of any or all message ports of a process, and to receive in a destructive or non-destructive manner the message at the head of a specified message port.

Use of the internal message facility is limited to operating system code. User level inter-process communication is available through MPE IV message files.

7. The Measurement Interface

In order to evaluate alternative strategies and to support the envisioned and yet to be envisioned MPE performance tools, an extensible measurement interface was designed and implemented.

The existing MPE measurement tools were highly dependent on the kernel implementation. They were knowledgeable of internal data structures and called very low level kernel routines or exerted direct control over resource management. Modifying the tools to support the new kernel would be an inadequate solution since the tools were inadequate for the evaluation task at hand, and future changes would create the same problem over again. The decision was made to attempt to centralize support of measurement requirements within the kernel itself, and to make the tools independent of the kernel's implementation.

The basic requirements of the existing and envisioned tools were investigated. An interface was defined which would provide the mechanisms, support structures, and the access and control intrinsics so that the information needed would be obtainable through intrinsic calls without knowledge of internal structures or policies.

The key objectives of the interface were : service (provide what's needed by the current tools); transparency (eliminate dependencies of performance tools on system internals); extensibility (meet future requirements by natural extension of the initial specification); and low overhead (so that use of the interface would minimally effect the performance of the system under test).

The resulting measurement interface supports complete system global and process local statistics gathering, selective measurement class enabling/disabling intrinsics, statistics class delivery intrinsics, and complete cleanup upon abnormal termination of a process which had enabled statistics gathering. Tools using the measurement interface require no privileged code so that system reliability is improved and the sought after independency from the kernel implementation is achieved.

8. MPE IV Kernel Resource Managers

Each resource manager operates independently through clean interfaces so that strategy or data structure changes of another resource manager will not effect him. Each resource manager is built from structured, general pieces so that alternative strategies can be easily implemented.

The management of the disc, main memory, and processor resources has the primary impact on system performance. The approaches taken towards resource management for these key resources is sketched in the remainder of this section.

8.1 Disc Management

Disc management policies have an extremely significant effect on system performance due to the workload characteristics. Transaction processing applications on the 3000 are characterized by several disc references per transaction with short processing requirements between references. In such an environment, good disc management is essential in achieving good system performance.

The goal of disc management should be to provide maximum disc subsystem throughput while minimizing the service time for the most important requests. The selection of policies for the management of disc space and the scheduling of accesses to secondary store should be based on achieving this goal.

The disc management systemi nterfaces with the file system and the memory management system when allocating disc space and servicing requests to access secondary store.

The major problems identified with the old disc management policies revolved around virtual memory management, disc access scheduling, and serial seeking.

Disc space for data segments was restricted to a single volume (i.e. virtual memory limited to the system disc). This restriction has serious detrimental effects on system growth and performance. The effect on system growth is the obvious limitation on the amount of disc space available for data segments by the size of the system disc. The performance impact of this restriction is due to the long queue length created at the system disc when the system is under memory pressure, and the resultant service delays for access requests to that volume.

Disc access scheduling did not perform requests in the order of their urgency. The scheduling policy for disc requests directed at a device was preemptive for all memory management requests and FIFO for all other requests.

The memory management replacement policy selected segments deemed not likely to be needed in the near future. In the case of data segments, a write to disc of the segment was requested, the motivation being that the write may complete before the region occupied by the segment was required so that the delay in fetching the new segment would be reduced. These "anticipatory writes" were not urgent and often unnecessary, yet the scheduling policy selected them for service before disc access requests required for the completion of important transactions. Segment fetches on behalf of batch jobs were also serviced before transaction related service requests under the old scheduling policy.

The FIFO policy within process initiated disc access requests resulted in the accesses of less urgent processes being performed before those of more urgent processes. This increased the service time for the more urgent processes requests and thereby increased the response time for the related transactions.

Disc service was entirely serial for each disc sharing a common controller (i.e. no ovelapping.) Although the controller supports overlapped seeks, this feature was not exploited. This resulted in a disc throughput limitation per controller to one access per (avg cylinder positioning delay + avg rotational latency + avg transfer time).

MPE IV disc management solves these problems. Additionally, the general approach to disc management gives broad flexibility in scheduling policies.

In MPE IV, disc space for data segments can reside on each system volume. This multi-spindle virtual memory eliminates the limitation on total virtual disc space, and helps to balance the disc queue lengths. (Balanced disc queues are required to take advantage of parallelism in I/O offered by overlapped seeks or multiple controllers).

The MPE IV disc queues are priority ordered. The priority of a disc request is determined by the priority of the process that requires the transfer. This holds for segment transfer requests issued by the memory management system on behalf of a process as well as for file system initiated transfer requests. Anticipatory writes are given the worst priority and sit at the back of the queue so that they are performed as background activity when the device would be otherwise idle. This priority queue management integrates the disc management policies with the goals of the rest of the kernel, since priority assignments reflect the global performance goal of the system.

The feature of the disc controller which allows a seek command to be sent to a unit other than the unit owning the controller is exploited in MPE IV. The seeks for units waiting for the controller are issued during the execution of the channel program for the unit currently owning the controller. This results in the heads being in position over the proper cylinder when the next unit gets the controller. The net result is a potential maximum disc throughput per controller of 1 access per (avg rotational latency + avg transfer time). Since the disc 1 access time is dominated by the head positioning delay, this overlapping approximately doubles the maximum throughput per controller. (The overlapping seek software will only be available for the Series II and III on the C-MIT).

To achieve this maximum throughput, the disc queues for the units on the controller must be kept non-empty and balanced. This requires a sustained high level of multi-programming and a proper spreading of data across the volumes. To make response times short, the more urgent disc requests have to be performed first. It can be seen how the interrelations between memory management, processor management and disc management impact system performance.

8.2 Memory Management

Memory management requirements for the 3000 architecture consist of free space allocation, segment replacement, and garbage collection.

Free space allocation is required when a segment fetch is to be performed. The free space allocation algorithm selects the hole into which the segment should be read. Alternative strategies include first fit, best fit, and buddy schemes.

Segment replacement must be performed when a segment fetch is required but a hole of adequate size is not available. Alternative strategies include working set type policies and least recently used type policies.

Garbage collection is required in a segmented system to combine holes into larger holes. A variable sized allocation policy tends to produce small, unusable holes scattered throughout memory. This is known as external fragmentation. Garbage collection attempts to minimize the external fragmentation by combining the small holes into larger usable holes.

The major problems identified with the old memory manager were its serial nature, high fault rate caused by the per program working set replacement policy, restricted garbage collection performed during critical periods, and an inefficient free space allocation policy.

The old memory manager was entirely serial. Once the memory manager was started on a process swap-in, he couldn't begin on a second (or more important) swap-in until all the disc transfers required to finish the first swap-in completed. This serial memory management service forced artificial limits on the multiprogramming level. For large main memories this limitation restricts the system from achieving its potential performance.

The working set per program policy caused processes to release each others localities resulting in a high fault and recovery rate.

Garbage collection could only be performed locally within a bank, and performed during allocation time, so that memory management service time was further increased.

MPE III free space allocation selected the first fit hole causing large holes to be used up before they were needed. This resulted in excess invocation of the replacement policy.

The memory management policies were entwined with the rest of the system so that minor strategy changes would require extensive development.

MPE IV memory management solves each of these problems and in addition presents a general, structured implementation which allows major strategy changes with minor development effort.

Free space allocation is implemented by a best fit policy using size ordered free lists. This scheme is very fast, and saves the big holes until they're needed.

The resulting external fragmentation is eliminated through background garbage collection. Main memory garbage collection is performed as a background activity using cpu cycles during which the processor would have otherwise been idle. Garbage collection attempts to move small assigned regions located between large holes into small fragmented holes. The large holes are combined into even larger holes, and the small holes are eliminated. This skews the distribution of hole sizes towards the large holes and eliminates the external fragmentation thereby reducing the frequency of application of the replacement policy. The garbage collection code is responsive to the system state, and returns to the dispatcher when more urgent activity becomes pending.

The memory replacement policy is a very low overhead implementation of a global least recently used (LRU) policy. When a hole of the required size is not available, segments not needed by the current multiprogramming set (as determined by a global LRU algorithm) are selected for replacement on a memory ordered basis. In the segmented architecture of the 3000 family, this replacement policy proved to be superior to the working set policies. The memory scanned LRU approach tends to release unneeded segments in adjacent regions of memory, thereby creating large holes with few replacements. In contrast, the working set policies were found to require many more segment replacements to satisfy placement requests since they freed up space randomly through memory when releasing a working set of segments.

Segment fetching is an unblocked parallel operation in which memory management code invoked directly by the dispatcher sets up the operation and
the disc management code finishes it off as the required transfers complete.

8.3 Processor Management

Processor management consists primarily of selecting the activity to which the cpu should be devoted. The major cpu activities include running system and user processes, swapping in processes, and garbage collection.

Processor management is implemented by assigning priorities to the pending activities and giving the cpu over to the activity with the most urgent priority. This function is performed by the dispatcher.

Priority assignment in the old kernel had a problem with batch jobs.

Batch jobs would migrate up in priority to compete equally with interactive processes during busy periods.

Activity selection was restricted in the old kernel due to the memory manager being serial. The consequence of this limitation was that even if plenty of free space was available, memory management could not be performed when needed for a process if a more urgent process was waiting for disc I/O to complete. It couldn't be risked to swap-in a less urgent process since the more urgent process might need memory management service soon and the memory manager would be busy. The dispatcher was forced to pause the cpu rather than to work on increasing the multiprogramming level.

The MPE IV processor management scheme is very flexible. Priority assignments and activity selection are directed towards optimizing the system performance and can be tuned by the operator.

Priority assignments are made to reflect the performance goals of the system. Each scheduling class (C,D,E) has a base priority and a limit priority. When a transaction begins or a job is introduced into the system, the related process gets its class' best priority, the class base priority. As the process uses more cpu time than that required for an average member of the class, the process is considered to be less urgent and its priority drifts towards the class's limit priority. The limit priority is the worst priority that a process in the class can get assigned to it.

The priorities of processes placed in the A or B scheduling classes are kept static over time. The filtering parameter which determines the migration rate for a C, D or E scheduled process from its class base to class limit is dynamically tuned for C scheduled processes only. Bounds on the filtering parameters for C, D and E classes are set in the :TUNE command.

This priority assignment scheme enables the dispatcher to apply a scheduling policy which approximates a "shortest processing time first" algorithm. This gives maximum system throughput and best response time for short transactions while slightly delaying the longer transactions.

The C, D, and E classes can be made to overlap so that the processes in the various classes compete with each other, or they can be made disjoint. By making them disjoint, D and E processes will always be preempted for C processes. This tuning causes batch work to be performed as background activity between bursts of interactive transactions. This is the default tuning setting.

The operator or system manager can control the base and limit priorities of each class, and the rate at which a process' priority moves from the base to the limit through the :TUNE command.

CPU activity selection procedes by inspecting the priority ordered queue of processes requiring cpu service. When a process is encountered which is ready to run, the process is launched. If the process requires some memory scheduling, the swap-in procedure is invoked directly by dispatcher. If there's nothing better to do, main memory garbage collection takes place.

In MPE IV, swapping-in of a process is performed by nested procedures on the dispatcher's stack. The fetching of a segment on behalf of a process is a low overhead, unblocked operation which allows an unlimited degree of parallelism in memory management. The swap-in code is responsive to the system state, and returns to the dispatcher when a process more urgent than the one that's being worked-on becomes ready or requires scheduling attention.

If the queue of ready processes is empty and there are no processes requiring memory scheduling , or increasing the multi-programming level has been determined to be dangerous at this time, the dispatcher invokes the background garbage collection code which returns when more urgent activity becomes pending.

9. Performance

The performance of MPE IV as measured by system transaction throughput and mean transaction response time is published in the "HP 3000 Performance Guide for Installed Systems." Performance tests were conducted using a standard application workload which represented a general-purpose EDP environment with a mix of online data base and program development sessions and background batch jobs.

The measurement results from these tests indicated that under light loads relative to system configuration, MPE IV showed slight performance improvement over MPE III. This was anticipated, since the MPE IV kernel seeks its performance improvements through the exploitation of parallelism, and the potential for parallelism is small under light loads. As the workloads were increased, MPE IV showed substantial improvement in both transaction response time and transaction throughput over MPE III. The performance improvements were realized across the family under the configurations and workloads measured.

The behavior of the system was exactly that which was sought. The system exhibited stability and good performance across the range of workloads, processor speeds, memory sizes and system configurations examined.

10. Conclusions

The approach to kernel design and implementation undertaken by the MPE IV kernel project resulted in an operating system foundation which naturally fits the evolving 3000 computer family to the environments it supports. The structured approach permitted alternatives to be easily implemented, and the measurement interface permitted them to be thoroughly evaluated. The final implementation consists of integrated resource managers who cooperate to provide the best performance with the given system configuration under the current workload. The validity of the approach taken to kernel design is demonstrated by the resulting kernel's reliability and performance.

Acknowledgements

Special recognition is due to those who significantly contributed to the project's success. Professor Wesley Chu of UCLA and Professor Forrest Baskett of Stanford impacted the process through their valuable consultations. Alan Hewer, Howard Morris, and Neil Wilhelm kept things on course through their careful reviews. Ron Kolb, Ray Ventura and Bruce Blinn, through their design and development help on seek-ahead, the measurement interface, and multi-spindle virtual memory respectively, helped to speed the kernel to completion. Chris Moeller with his tuning help and Marcia McConnel and Carl Sassenrath with their debugging assistence contributed to the system's performance and reliability. Ken Spalding, through his coordinating function in the late stages, helped to get the system out the door.

THE MPE IV KERNEL: A HIGH PERFORMANCE, INTEGRATED FOUNDATION FOR MPE - THE DESIGN PROCESS

By:
John Richard Busch
Hewlett-Packard Computer Systems Division

The MPE IV Kernel, available on the C MIT, is the result of over three years of research and development. It provides a new high performance, extensible, integrated foundation for MPE. The principal designer of the kernel describes the design and implementation process. Design Objectives, research approach, functional characteristics, algorithms, design methodology, and performance results are presented.

THE ROLE OF PRINTERS IN A DS ENVIRONMENT--
AN ENGINEERING FEEDBACK SESSION

Presentors
Jim Langley, Project Mgr., HP

The author will be soliciting customer input on the following topics:

1. The quality and features of HP's existing product line.
2. Data Communications requirements for printing in a DS environment.

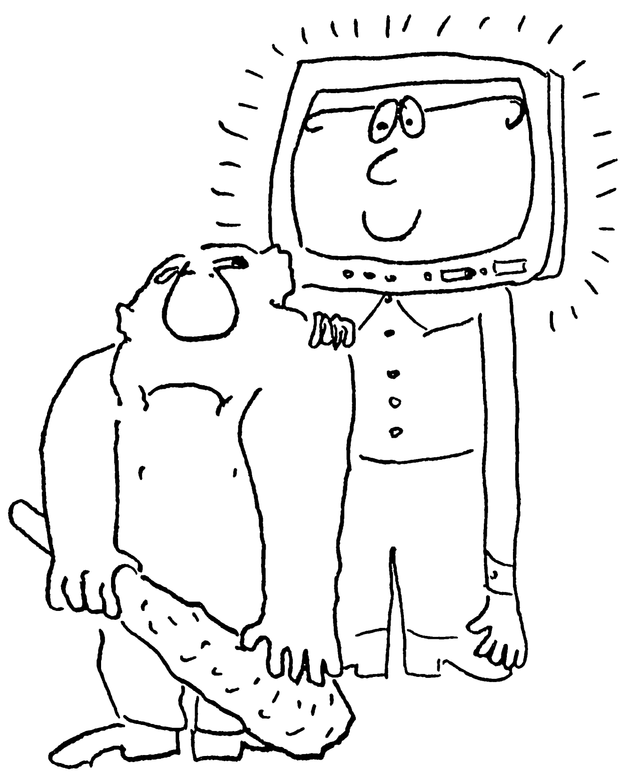
DATA COMMUNICATIONS--
A TECHNICAL ROUNDTABLE

Presentors
Tom Black, Marketing Mgr., HP
Ed Turner, Product Support Mgr., HP
Jim Beetum, Project Mgr., HP

To start off this session, HP will present the future direction of its data communications program. Following this, the panel will answer to questions from the audience the panel expertise varies widely such that virtually any data comm. question or concerns can be responded to.

USER FRIENDLY APPLICATIONS IN COMMERCIAL REALTIME DATAPROCESSING

- EXPERIENCES
- SUGGESTIONS
- PROBLEMS



JOACHIM GEFFKEN

RECHENZENTRUM
HERBERT SEITZ KG
GRÜNENSTRASSE 11/12
D-2800 BREMEN
W-GERMANY

Survey

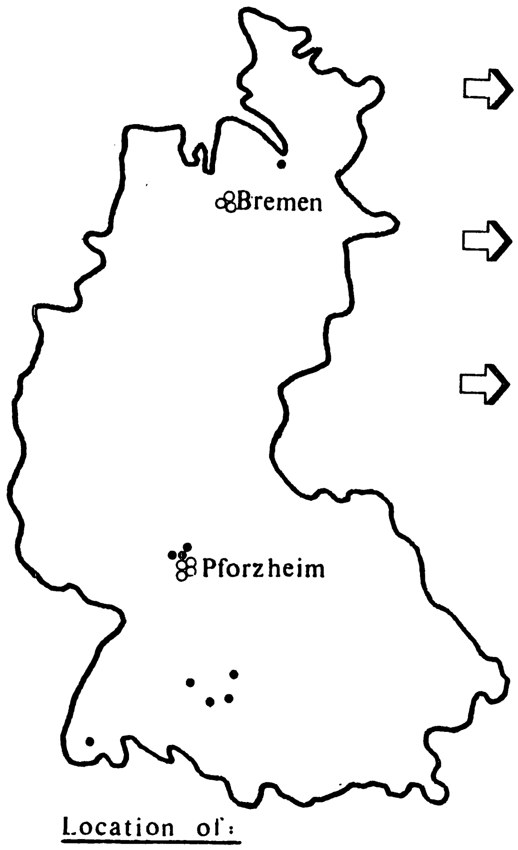
- **Introduction**
- **User Interface**
- **Application Programs**
- **User Training**
- **User Documentation**

Introduction

The Herbert Seitz Company is ...

- A REALTIME DATAPROCESSING SERVICE BUREAU
- AND SOFTWAREHOUSE
- AND HEWLETT PACKARD OEM

with (1981) ...



Location of:

o own Computers

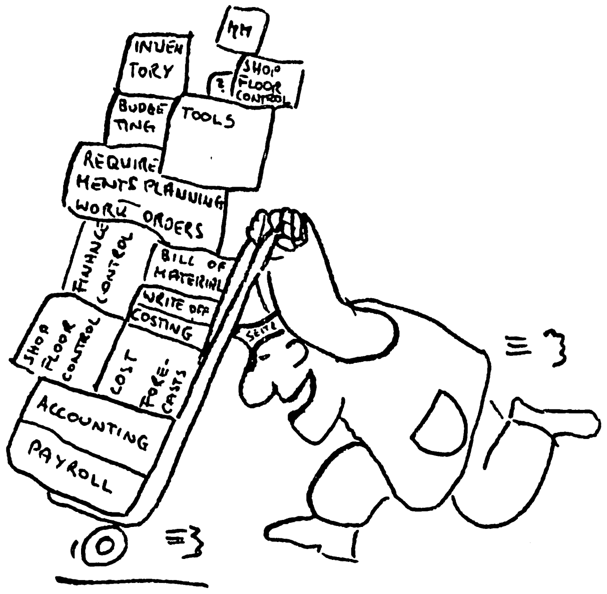
● Associated Companies

- 7 OWN HP 3000 (SERIES III AND 44) IN OUR BREMEN AND PFORZHEIM BRANCH
- AND 10 HP 3000 SERIES III IN ASSOCIATED COMPANIES
- WITH APPROX. 350 TERMINALS SPREAD OVER GERMANY CONNECTED VIA HARD-WIRED LEASED LINES/DIALED LINES

UFS2	RECHENZENTRUM HERBERT SEITZ KG	

Introduction (2)

WE PROVIDE OUR SERVICES IN GERMANY AND FRANCE FOR COMMERCIAL APPLICATIONS LIKE ...



- ACCOUNTING
- PAYROLL
- MATERIAL MANAGEMENT
- SHOP FLOOR CONTROL, CAPACITY PLANNING
- TOOLS FOR HP 3000 OPERATION, SOFTWARE-DESIGN AND DOCUMENTATION

OUR USERS ARE ...

- WORKMEN
- DATA TYPISTS, CLERKS
- MANAGERS

ONLY A FEW OF THEM ...

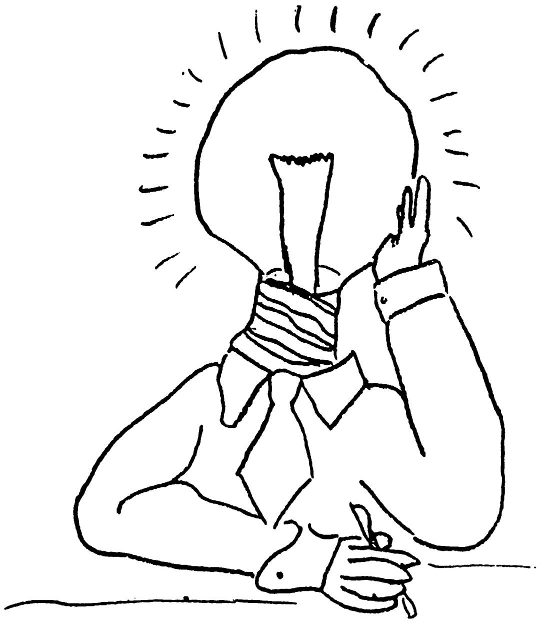
- ARE SPEAKING (HP-)ENGLISH
- HAVE DP EXPERIENCE
- HAVE SEEN ANY TERMINAL BEFORE

UFS2	RECHENZENTRUM HERBERT SEITZ KG	

Introduction (3)

for this kind of users we need ...

☞ A FRIENDLY **interface** BETWEEN THE USER AND HIS APPLICATION PROGRAMS



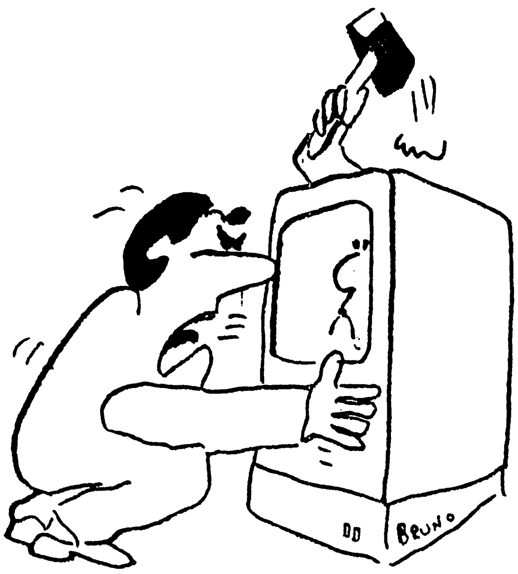
☞ EASY-TO-UNDERSTAND **application programs**

☞ A **user training** WITH REGARD TO THE STANDARD OF EDUCATION OF ITS PARTICIPANTS

☞ **user documentation** MANUALS, WHICH INVITE TO READ

Interface

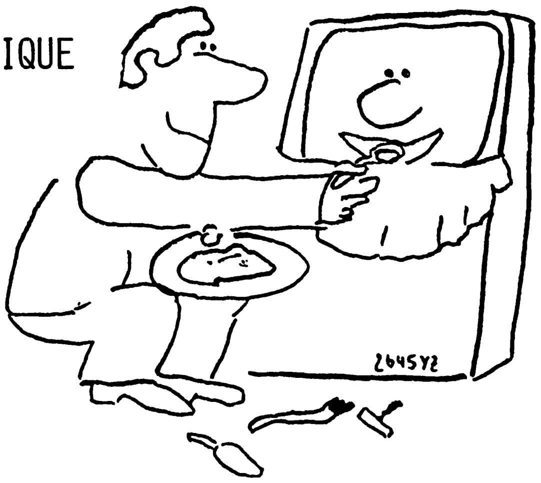
➡ KEEP YOUR USERS OUT OF MPE !



MPE IS A HIGH LEVEL OPERATING SYSTEM WITH A LOT OF POWERFUL COMMANDS, BUT IT IS NOT DESIGNED FOR THE DIRECT USE OF USERS WE ARE DISCUSSING ABOUT

➡ USE ANY KIND OF MENU-TECHNIQUE

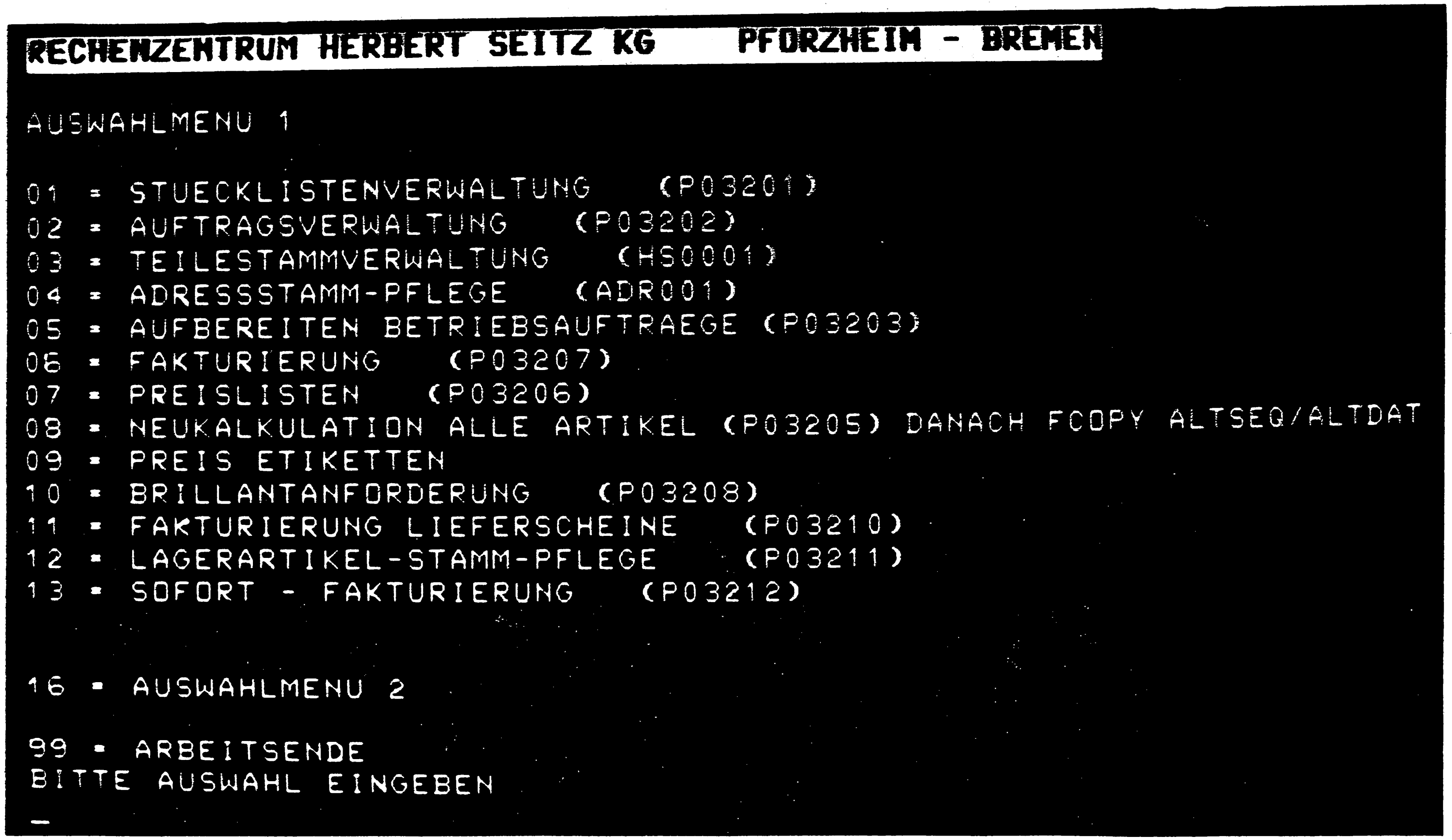
(WE CALL OURS "USER-PROFILES")



INTERFACE (2)

⇛ WE ONLY NEED 1 MPE COMMAND: HELLO

⇛ MENUES EASILY CREATED AND MAINTAINED WITH EDITOR IN ANY LANGUAGE

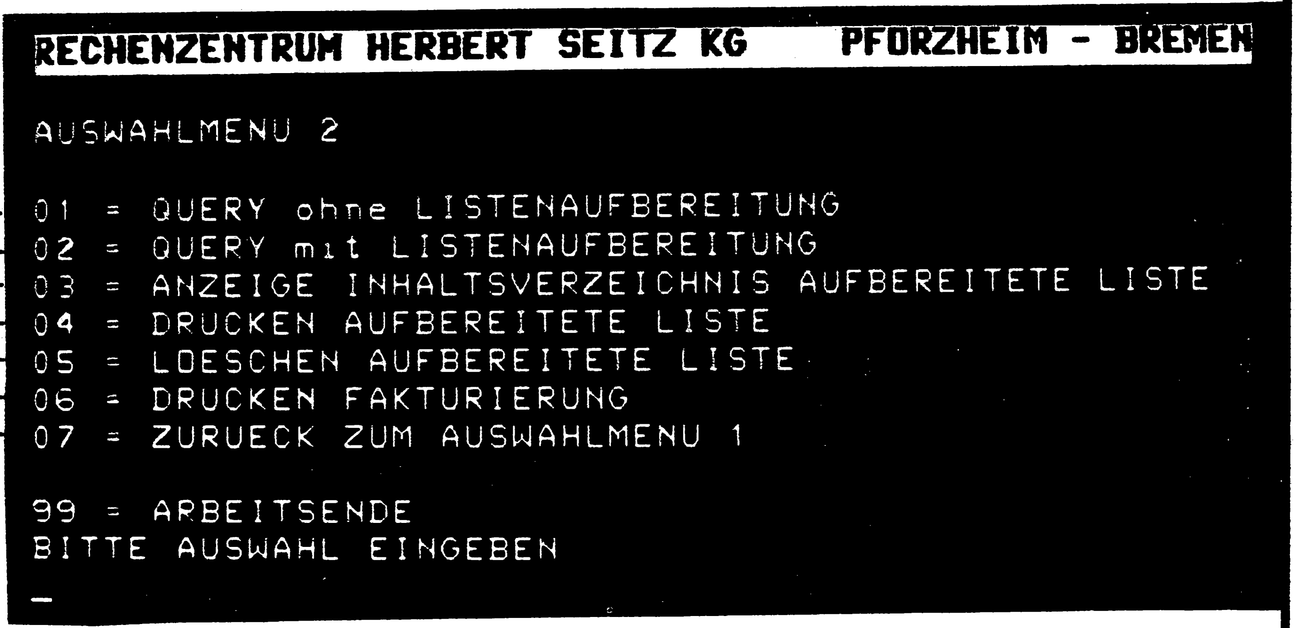


```
RECHENZENTRUM HERBERT SEITZ KG     PFORZHEIM - BREMEN

AUSWAHLMENU 1

01 = STUECKLISTENVERWALTUNG    (P03201)
02 = AUFTRAGSVERWALTUNG    (P03202)
03 = TEILESTAMMVERWALTUNG    (HS0001)
04 = ADRESSSTAMM-PFLEGE    (ADR001)
05 = AUFBEREITEN BETRIEBSAUFTRAEGE (P03203)
06 = FAKTURIERUNG    (P03207)
07 = PREISLISTEN    (P03206)
08 = NEUKALKULATION ALLE ARTIKEL (P03205) DANACH FCOPY ALTSEQ/ALTDAT
09 = PREIS ETIKETTEN
10 = BRILLANTANFORDERUNG    (P03208)
11 = FAKTURIERUNG LIEFERSCHEINE    (P03210)
12 = LAGERARTIKEL-STAMM-PFLEGE    (P03211)
13 = SOFORT - FAKTURIERUNG    (P03212)


16 = AUSWAHLMENU 2

99 = ARBEITSENDE
BITTE AUSWAHL EINGEBEN
_
```

INTERFACE (3)

- THE USER ONLY CAN DO THINGS YOU WANT HIM TO DO
- BUT HE CAN DO ANYTHING POSSIBLE WITH THE HP 3000

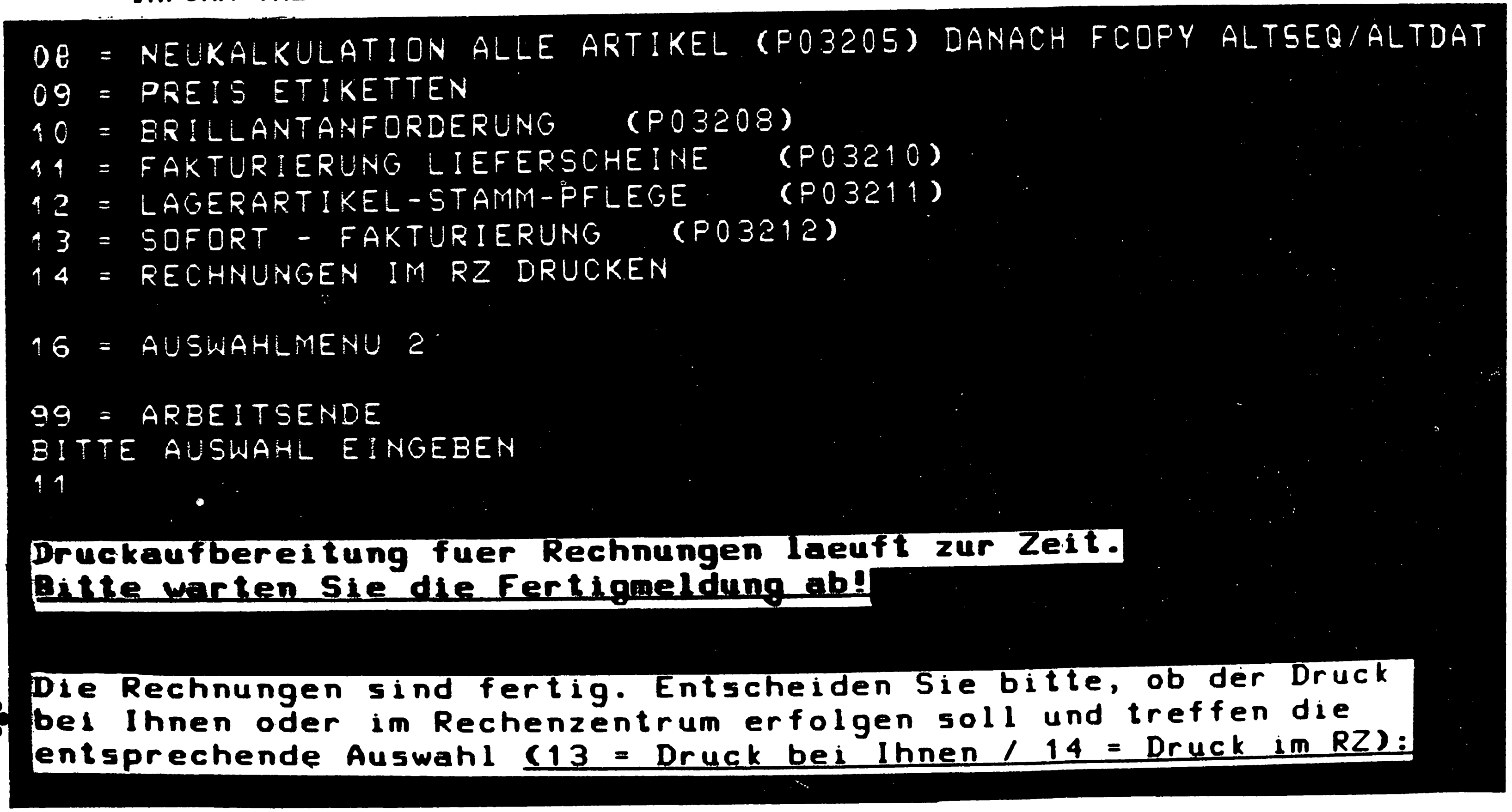
QUERY WITH TERMINAL-OUTPUT
QUERY WITH PRINT-OUTPUT
"LISTF" OF SOME FILES
HARDCOPY SPOOL PRINT
"PURGE" OF ONE FILE
START OF A JOBSTREAM
SKIP TO ANOTHER MENUE



```
RECHENZENTRUM HERBERT SEITZ KG      PFORZHEIM - BREMEN

AUSWAHLMENU 2

01 = QUERY ohne LISTENAUFBEREITUNG
02 = QUERY mit LISTENAUFBEREITUNG
03 = ANZEIGE INHALTSVERZEICHNIS AUFBEREITETE LISTE
04 = DRUCKEN AUFBEREITETE LISTE
05 = LOESCHEN AUFBEREITETE LISTE
06 = DRUCKEN FAKTURIERUNG
07 = ZURUECK ZUM AUSWAHLMENU 1

99 = ARBEITSENDE
BITTE AUSWAHL EINGEBEN
_
```

INTERFACE (4)

- CONTROL OF REQUIRED SEQUENCE OF DIFFERENT MENUE-CHOICES REFERRING TO TIME, SITUATION OR KIND OF DATA-ENTRIES
- INFORM THE USER WHAT THE COMPUTER IS DOING FOR HIM

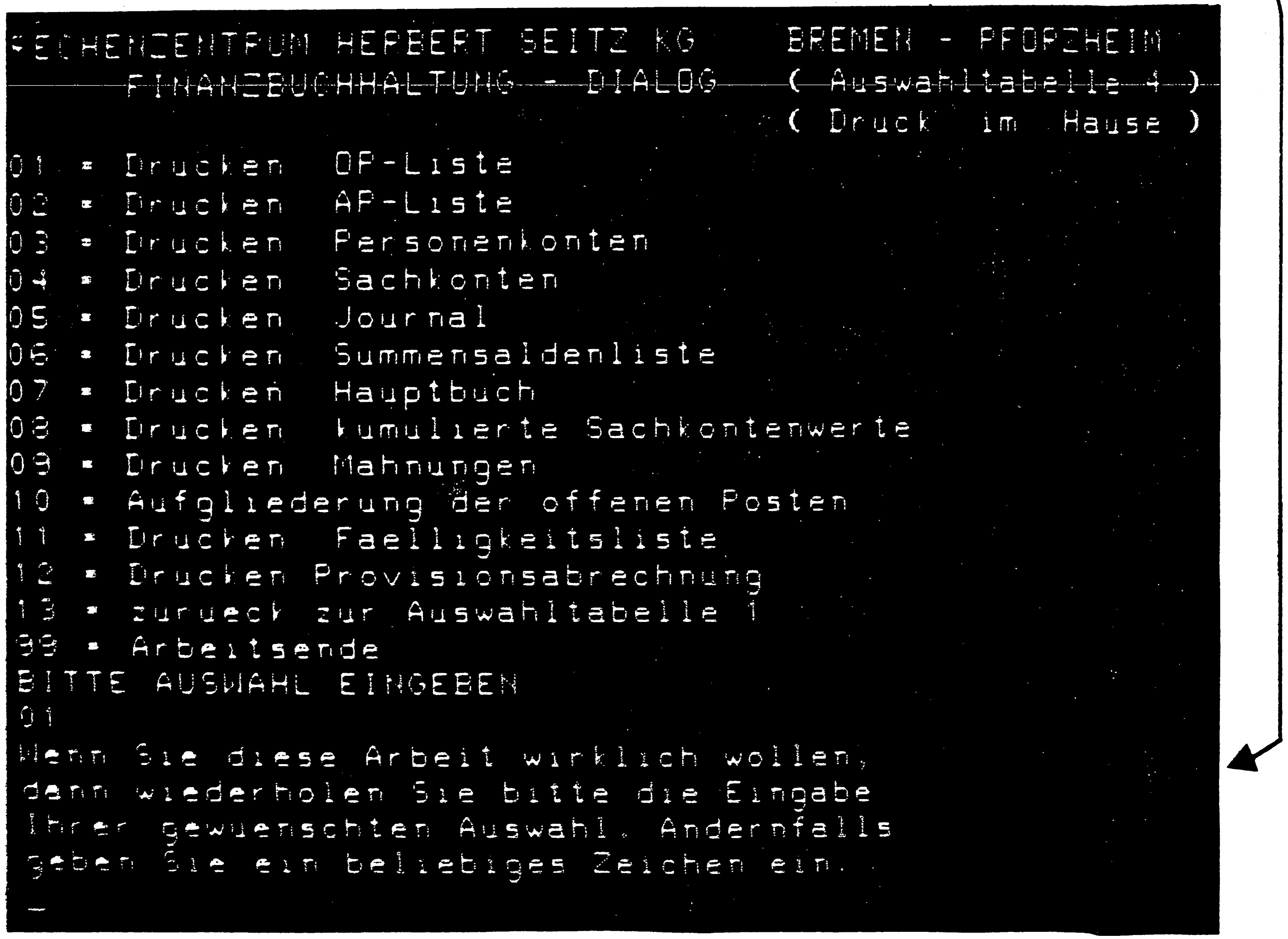


```
08 = NEUKALKULATION ALLE ARTIKEL (P03205) DANACH FCOPY ALTSEQ/ALTDAT
09 = PREIS ETIKETTEN
10 = BRILLANTANFORDERUNG   (P03208)
11 = FAKTURIERUNG LIEFERSCHEINE   (P03210)
12 = LAGERARTIKEL-STAMM-PFLEGE   (P03211)
13 = SOFORT - FAKTURIERUNG   (P03212)
14 = RECHNUNGEN IM RZ DRUCKEN

16 = AUSWAHLMENU 2

99 = ARBEITSENDE
BITTE AUSWAHL EINGEBEN
11

Druckaufbereitung fuer Rechnungen laeuft zur Zeit.
Bitte warten Sie die Fertigmeldung ab!


Die Rechnungen sind fertig. Entscheiden Sie bitte, ob der Druck
bei Ihnen oder im Rechenzentrum erfolgen soll und treffen die
entsprechende Auswahl (13 = Druck bei Ihnen / 14 = Druck im RZ):
```

"YOUR INVOICES ARE READY. ENTER 13 FOR HARDCOPY OR 14 FOR LINE-PRINTER OUTPUT"

INTERFACE (5)

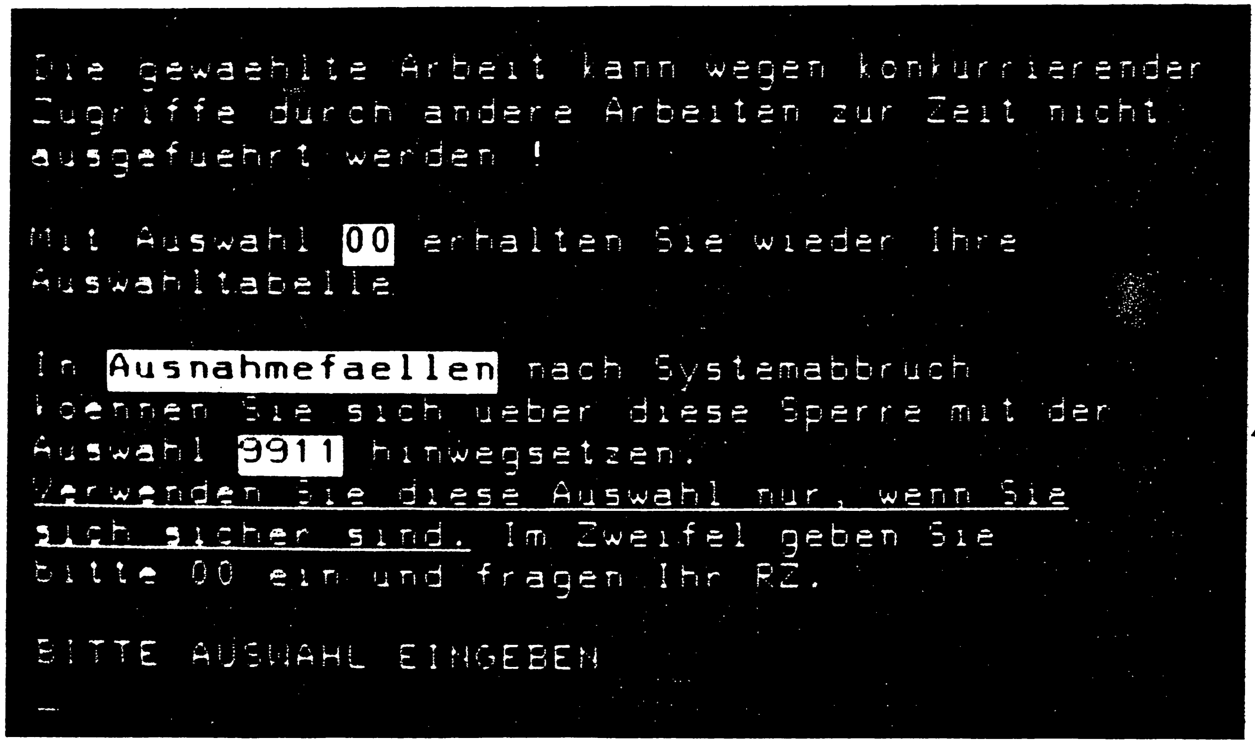
⇛ ASK FOR RECONFORMATION OF CRITICAL CHOICES



```
RECHENZENTRUM HERBERT SEITZ KG    BREMEN - PFORZHEIM
     FINANZBUCHHALTUNG - DIALOG   ( Auswahltabelle 4 )
                                  ( Druck  im  Hause )
01 = Drucken  OP-Liste
02 = Drucken  AP-Liste
03 = Drucken  Personenkonten
04 = Drucken  Sachkonten
05 = Drucken  Journal
06 = Drucken  Summensaldenliste
07 = Drucken  Hauptbuch
08 = Drucken  kumulierte Sachkontenwerte
09 = Drucken  Mahnungen
10 = Aufgliederung der offenen Posten
11 = Drucken  Faelligkeitsliste
12 = Drucken Provisionsabrechnung
13 = zurueck zur Auswahltabelle 1
99 = Arbeitsende
BITTE AUSWAHL EINGEBEN
01
Wenn Sie diese Arbeit wirklich wollen,
dann wiederholen Sie bitte die Eingabe
Ihrer gewuenschten Auswahl. Andernfalls
geben Sie ein beliebiges Zeichen ein.
_
```

INTERFACE (6)

AVOID ANY CONFLICTING ACTIVITIES
(AS OTHER ACTIVE SESSIONS OR JOBS,
SUCCESSFUL COMPLETION OF OTHER SESSIONS OR JOBS ETC.)

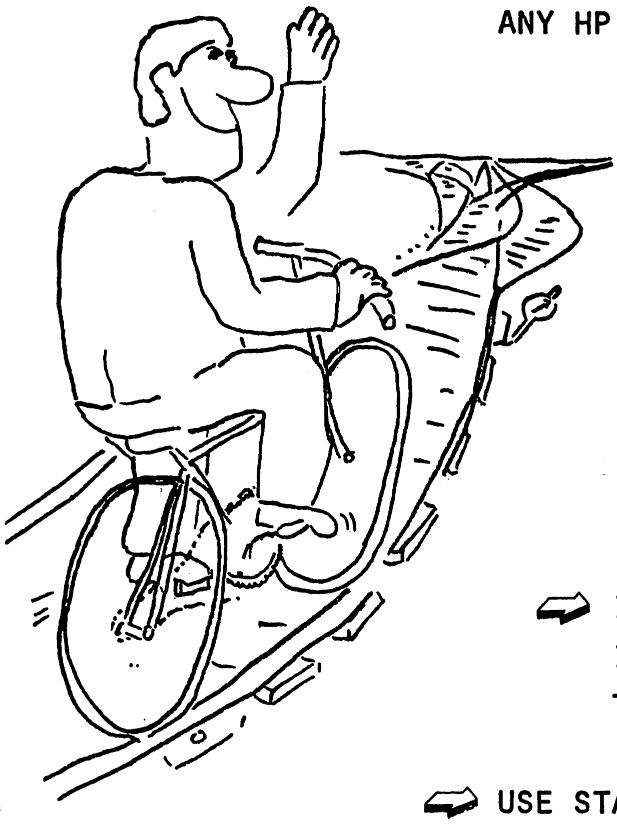


```
Die gewaehlte Arbeit kann wegen konkurrierender
Zugriffe durch andere Arbeiten zur Zeit nicht
ausgefuehrt werden !

Mit Auswahl 00 erhalten Sie wieder Ihre
Auswahltabelle

In Ausnahmefaellen nach Systemabbruch
koennen Sie sich ueber diese Sperre mit der
Auswahl 9911 hinwegsetzen.
Verwenden Sie diese Auswahl nur, wenn Sie
sich sicher sind. Im Zweifel geben Sie
bitte 00 ein und fragen Ihr RZ.

BITTE AUSWAHL EINGEBEN

_
```

Application Programs

GENERAL GUIDELINES FOR OUR PROGRAMMERS:

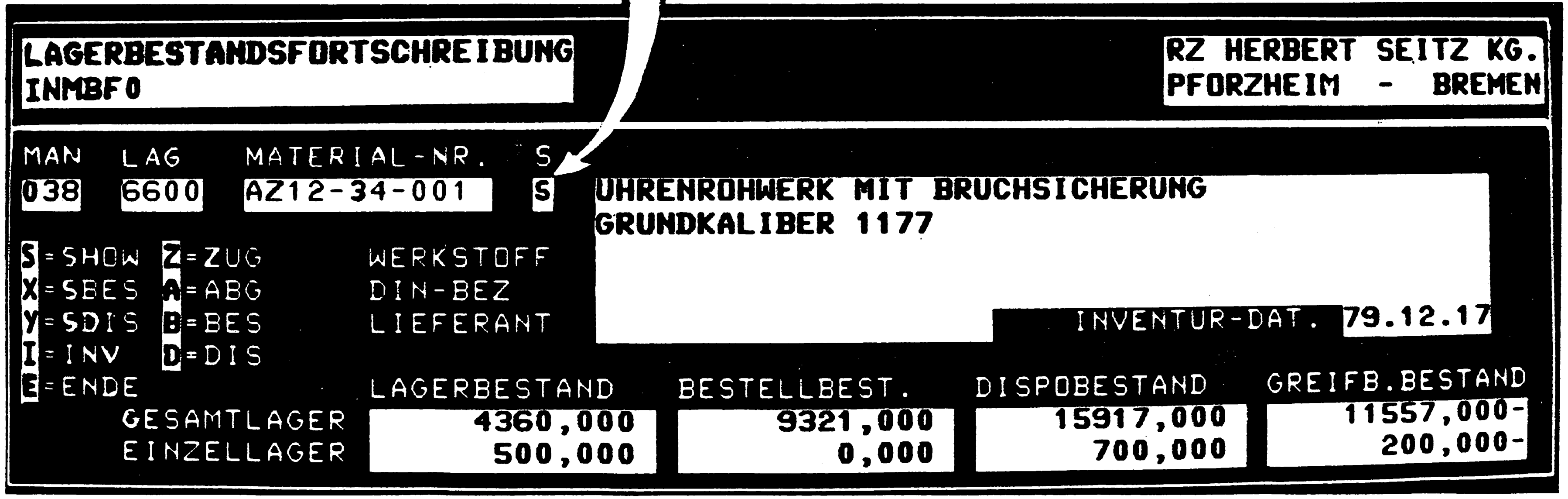
- ONLY BLOCK MODE PROGRAMS (V/3000)
- TRY TO DESIGN GOOD READIBLE FORMS
- AVOID (ENGLISH) ERROR MESSAGES FROM ANY HP SUBSYSTEM
- ALWAYS POINT TO WRONG ENTRIES AND PROVIDE A MEANINGFULL ERROR MESSAGE
- ALLOW A REGULAR PROGRAM TERMINATION OR A CANCELLATION OF THE LAST ENTRY AT ANY TIME IN ANY SITUATION VIA THE F7/F8 KEYS
- IF POSSIBLE, USE THE FIELDNAMES IDENTICAL WITH THE ITEM NAMES OF THE CORRESPONDING DATA BASE
- USE STANDARD FORMS AND TRANSACTION CODES IN ALL PROGRAMS AND SYSTEMS



SOME EXAMPLES ...

APPLICATION PROGRAMS (2)

SAME TRANSACTION CODES IN ALL PROGRAMS

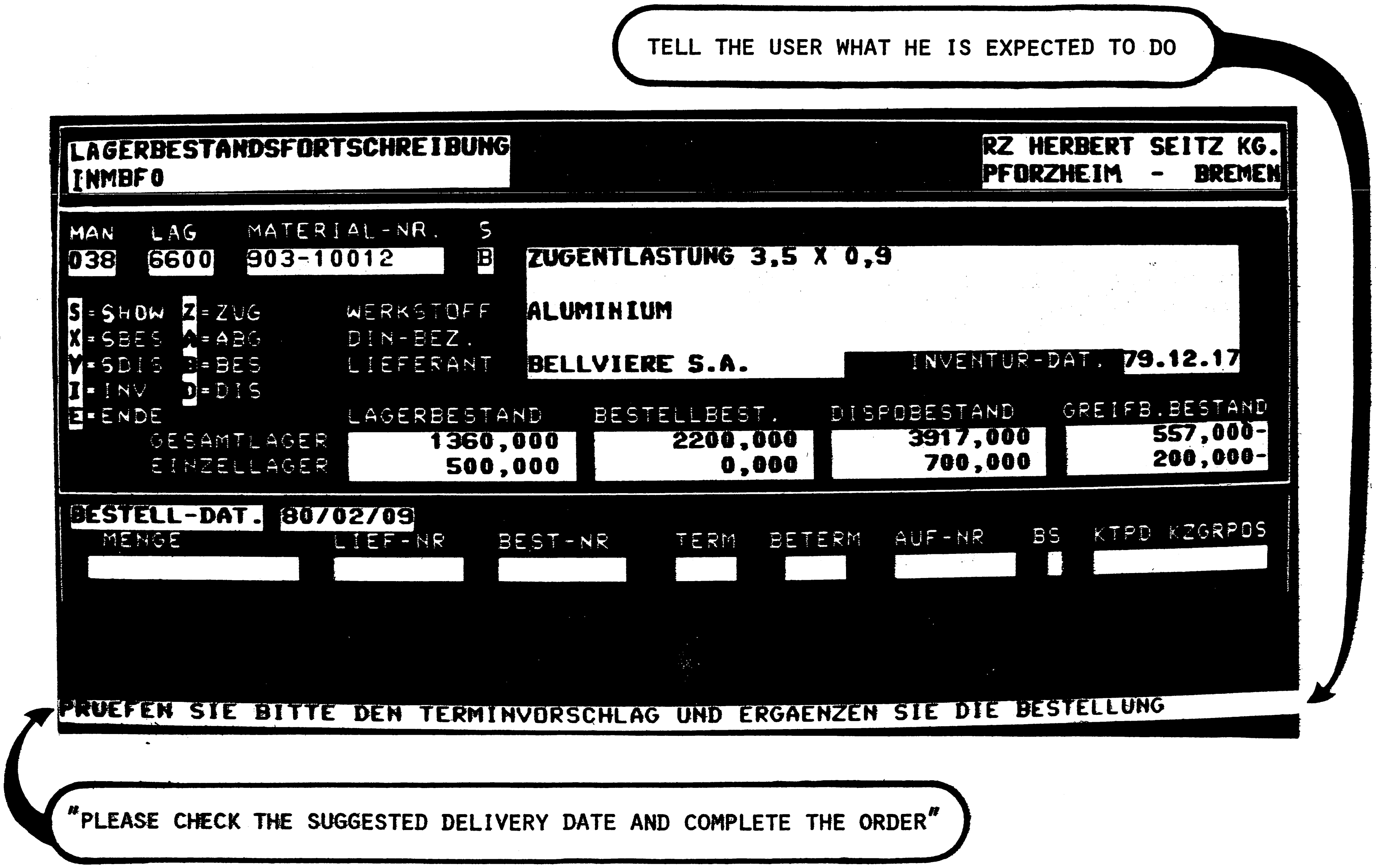


```
LAGERBESTANDSFORTSCHREIBUNG                                  RZ HERBERT SEITZ KG.
INMBF0                                                       PFORZHEIM  -  BREMEN

MAN  LAG    MATERIAL-NR.    S
038  6600   AZ12-34-001     S  UHRENROHWERK MIT BRUCHSICHERUNG
                               GRUNDKALIBER 1177
S=SHOW  Z=ZUG      WERKSTOFF
X=SBES  A=ABG      DIN-BEZ
Y=SDIS  B=BES      LIEFERANT                  INVENTUR-DAT. 79.12.17
I=INV   D=DIS
E=ENDE             LAGERBESTAND   BESTELLBEST.   DISPOBESTAND   GREIFB.BESTAND
     GESAMTLAGER       4360,000       9321,000      15917,000      11557,000-
     EINZELLAGER        500,000          0,000        700,000        200,000-
```

SHOW - IF POSSIBLE - THE AVAILABLE TRANSACTION-CODES

APPLICATION PROGRAMS (3)

TELL THE USER WHAT HE IS EXPECTED TO DO

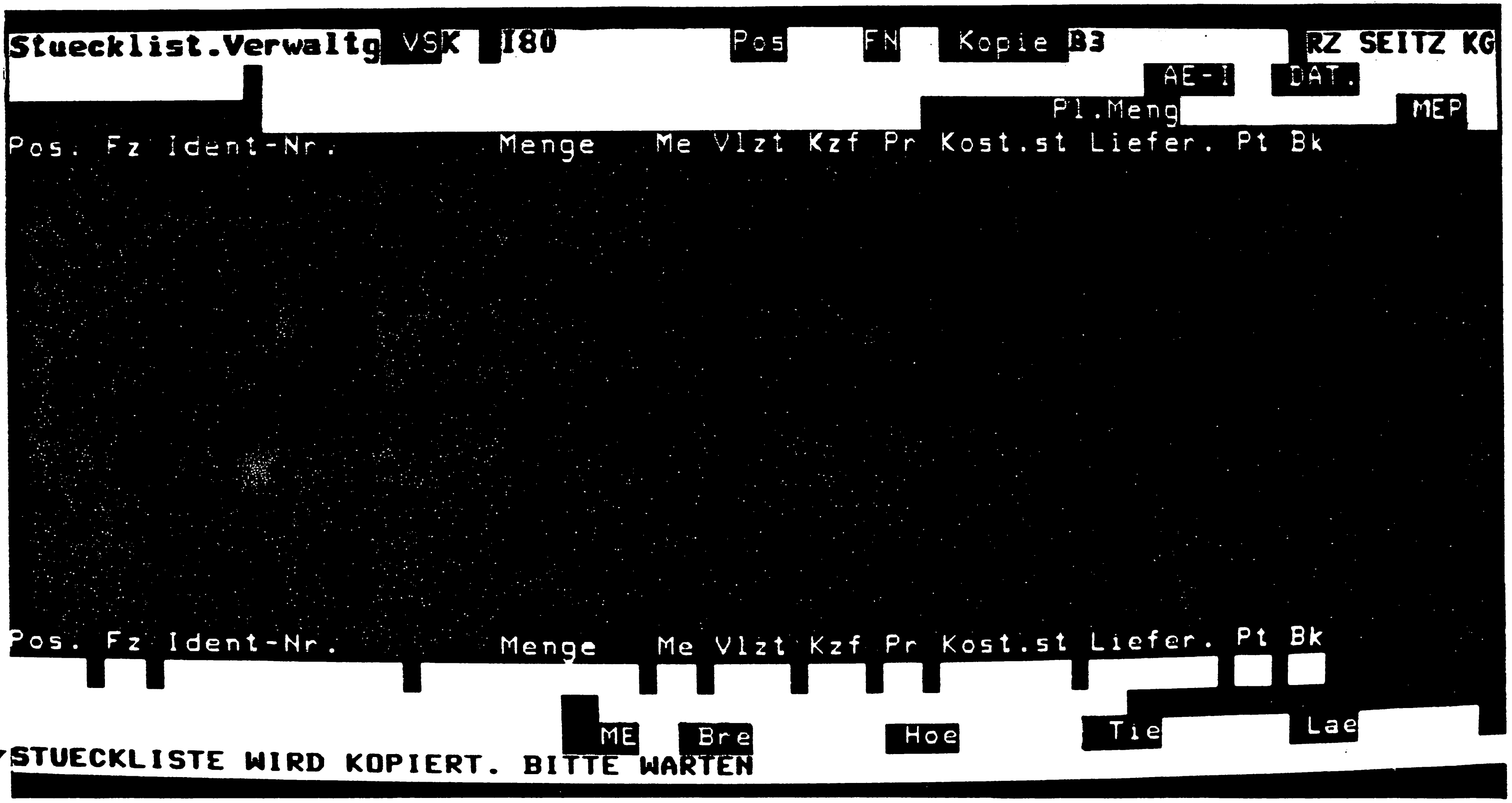


```
LAGERBESTANDSFORTSCHREIBUNG                                    RZ HERBERT SEITZ KG.
INMBFO                                                         PFORZHEIM  -  BREMEN

MAN  LAG   MATERIAL-NR.   S
038  6600  903-10012      B   ZUGENTLASTUNG 3,5 X 0,9

S=SHOW  Z=ZUG        WERKSTOFF  ALUMINIUM
X=SBES  A=ABG        DIN-BEZ.
Y=SDIS  B=BES        LIEFERANT  BELLVIERE S.A.          INVENTUR-DAT. 79.12.17
I=INV   D=DIS
E=ENDE               LAGERBESTAND   BESTELLBEST.   DISPOBESTAND   GREIFB.BESTAND
     GESAMTLAGER         1360,000       2200,000       3917,000        557,000-
     EINZELLAGER          500,000          0,000        700,000        200,000-

BESTELL-DAT. 80/02/09
   MENGE             LIEF-NR    BEST-NR     TERM   BETERM   AUF-NR   BS   KTPD KZGRPOS

PRUEFEN SIE BITTE DEN TERMINVORSCHLAG UND ERGAENZEN SIE DIE BESTELLUNG
```

"PLEASE CHECK THE SUGGESTED DELIVERY DATE AND COMPLETE THE ORDER"

APPLICATION PROGRAMS (4)

TELL THE USER, IF A TRANSACTION TAKES MORE THAN THE NORMAL RESPONSE TIME

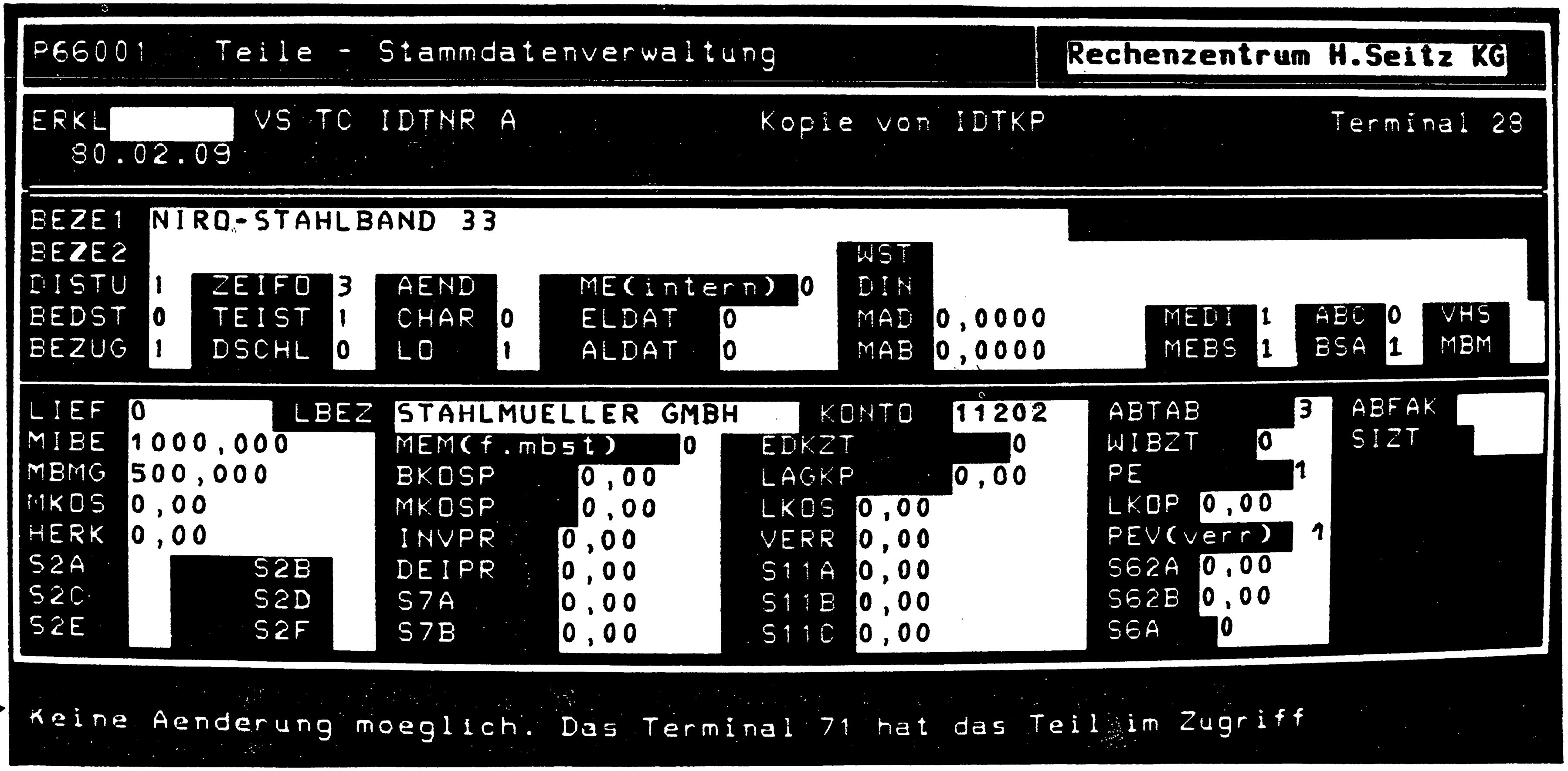


```
Stuecklist.Verwaltg VSK  I80        Pos    FN   Kopie B3              RZ SEITZ KG
                                                            AE-I    DAT.
                                                     Pl.Meng           MEP
Pos. Fz Ident-Nr.       Menge   Me Vlzt Kzf Pr Kost.st Liefer. Pt Bk

Pos. Fz Ident-Nr.       Menge   Me Vlzt Kzf Pr Kost.st Liefer. Pt Bk
                          ME   Bre        Hoe       Tie        Lae
STUECKLISTE WIRD KOPIERT. BITTE WARTEN
```

"BILL OF MATERIAL WILL BE COPIED. PLEASE HOLD ON"

APPLICATION PROGRAMS (5)

PROVIDE INFORMATIONS ABOUT LOCKS AND THEIR REASON

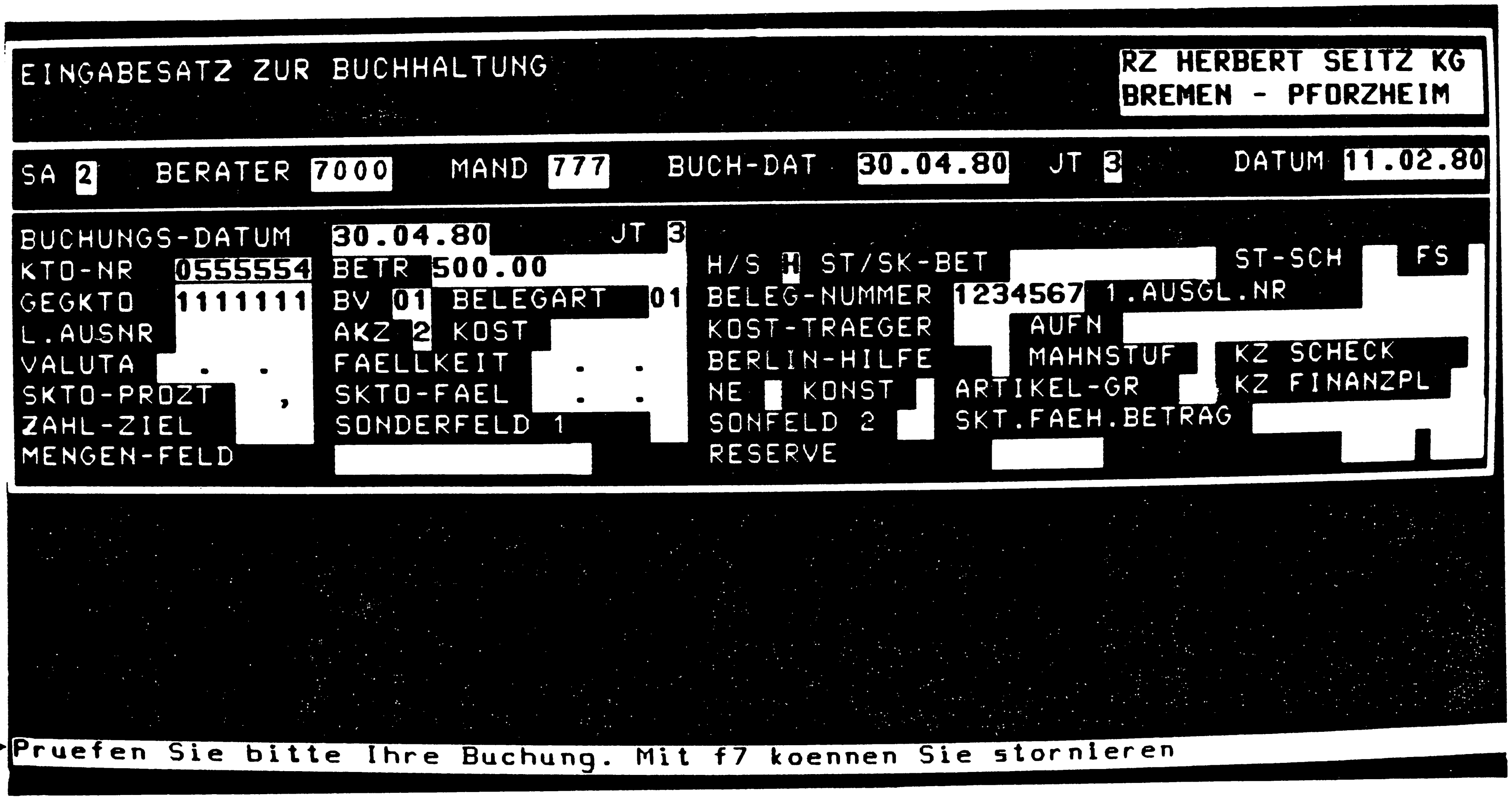


```
P66001   Teile - Stammdatenverwaltung                    Rechenzentrum H.Seitz KG

ERKL        VS TC IDTNR A              Kopie von IDTKP                Terminal 28
  80.02.09

BEZE1  NIRO-STAHLBAND 33
BEZE2                                                  WST
DISTU  1  ZEIFO  3  AEND        ME(intern) 0           DIN
BEDST  0  TEIST  1  CHAR  0     ELDAT      0           MAD 0,0000     MEDI 1  ABC 0  VHS
BEZUG  1  DSCHL  0  LO    1     ALDAT      0           MAB 0,0000     MEBS 1  BSA 1  MBM

LIEF  0          LBEZ  STAHLMUELLER GMBH   KONTO 11202     ABTAB       3   ABFAK
MIBE  1000,000         MEM(f.mbst)    0    EDKZT      0    WIBZT  0        SIZT
MBMG  500,000          BKOSP     0,00      LAGKP   0,00    PE          1
MKOS  0,00             MKOSP     0,00      LKOS  0,00      LKOP  0,00
HERK  0,00             INVPR    0,00       VERR  0,00      PEV(verr)   1
S2A            S2B     DEIPR    0,00       S11A  0,00      S62A  0,00
S2C            S2D     S7A      0,00       S11B  0,00      S62B  0,00
S2E            S2F     S7B      0,00       S11C  0,00      S6A   0

Keine Aenderung moeglich. Das Terminal 71 hat das Teil im Zugriff
```

"NO MASTER ITEM CHANGE POSSIBLE. THE TERMINAL 71 HAS EXCLUSICE ACCESS TO THIS ENTRY"

APPLICATION PROGRAMS (6)

FORCE USER TO CHECK HIS ENTRY, WHERE IT MAKES SENSE

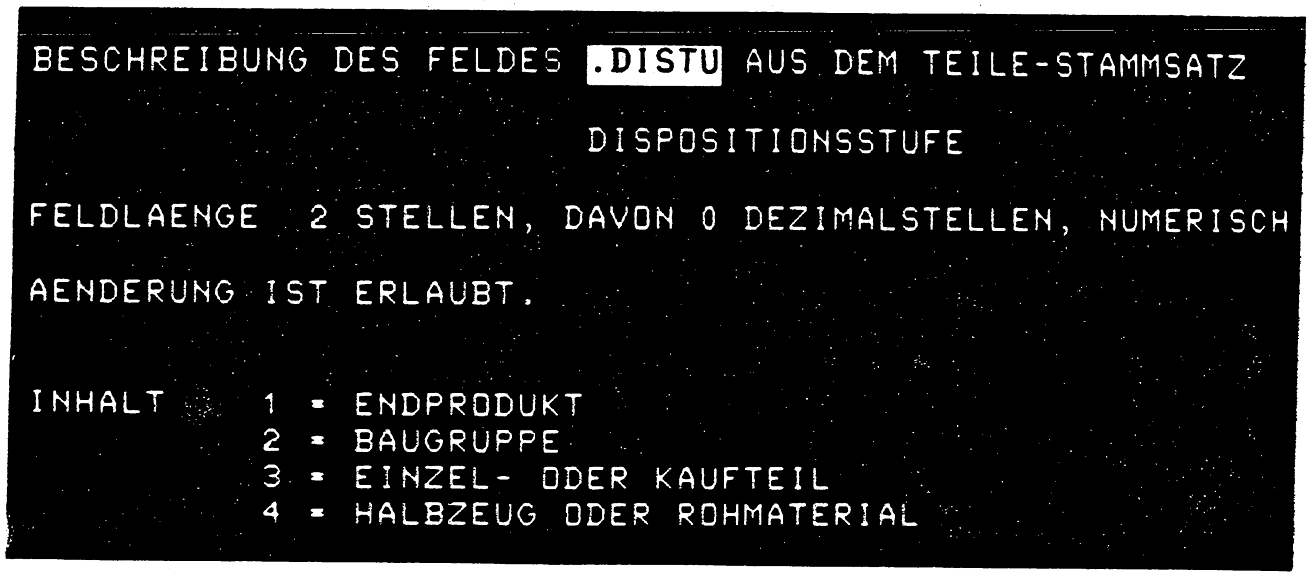


```
EINGABESATZ ZUR BUCHHALTUNG                                  RZ HERBERT SEITZ KG
                                                             BREMEN - PFORZHEIM

SA 2  BERATER 7000  MAND 777  BUCH-DAT 30.04.80  JT 3       DATUM 11.02.80

BUCHUNGS-DATUM  30.04.80       JT 3
KTO-NR   0555554  BETR 500.00          H/S H  ST/SK-BET              ST-SCH     FS
GEGKTO   1111111  BV 01  BELEGART  01  BELEG-NUMMER 1234567  1.AUSGL.NR
L.AUSNR           AKZ 2  KOST          KOST-TRAEGER          AUFN
VALUTA      .  .  FAELLKEIT    .  .    BERLIN-HILFE          MAHNSTUF    KZ SCHECK
SKTO-PROZT     ,  SKTO-FAEL    .  .    NE    KONST     ARTIKEL-GR        KZ FINANZPL
ZAHL-ZIEL         SONDERFELD 1         SONFELD 2       SKT.FAEH.BETRAG
MENGEN-FELD                            RESERVE

Pruefen Sie bitte Ihre Buchung. Mit f7 koennen Sie stornieren
```

"PLEASE CHECK THE RESULT OF YOUR ENTRY, IF NECESSARY CANCEL WITH F7"

APPLICATION PROGRAMS (7)

⇛ HELP FACILITY AND FIELD EXPLANATION WITHIN PROGRAMS



```
BESCHREIBUNG DES FELDES .DISTU AUS DEM TEILE-STAMMSATZ

                       DISPOSITIONSSTUFE

FELDLAENGE  2 STELLEN, DAVON 0 DEZIMALSTELLEN, NUMERISCH

AENDERUNG IST ERLAUBT.


INHALT      1 = ENDPRODUKT
            2 = BAUGRUPPE
            3 = EINZEL- ODER KAUFTEIL
            4 = HALBZEUG ODER ROHMATERIAL
```

DESCRIPTION OF FIELD DISTU
FROM MASTERFILE
DISPOSITION LEVEL

FIELDLENGTH 2, 0 DECIMALS, NUMER

CHANGE IS ALLOWED

ALLOWED ENTRIES:	1 = FINAL PRODUCT
	2 = ASSEMBLY
	3 = PART OF PURCHASE ITEI
	4 = RAW-MATERIAL

★★★ OUR EXPERIENCE: THIS FEATURE IS USED VERY SELDOM AND IT IS VERY EXPENSIVE TO DESIGN AND MAINTAIN
WE DON'T EMPHASIZE IT IN ALL APPLICATIONS

User Training

OUR PROGRAM:

- INTRODUCTION INTO INTERACTIVE DATA PROCESSING

- UPDATE TRAINING FOR STANDARD USERS
- APPLICATION-TRAINING
- QUERY FOR NON-DP PERSONNEL
- UPDATE TRAINING FOR **QUERY** USERS

User Training (2)

INTRODUCTION TRAINING

▷ 1/2 DAY THEORY, 1/2 DAY LABS

▷ TERMINAL USE

▷ HARDCOPY-PRINTER USE

▷ HOW TO LOG ON

▷ HOW TO USE THE MENUES

▷ TRY SOME TRANSACTIONS

▷ HANDOUTS:

- SLIDE COPIES
- TERMINAL AND HARDCOPY USER MANUAL (SIMPLIFIED AND TRANSLATED)
- CHECKLIST FOR TROUBLESHOOTING

User Training (3)

UPDATE TRAINING FOR STANDARD USERS

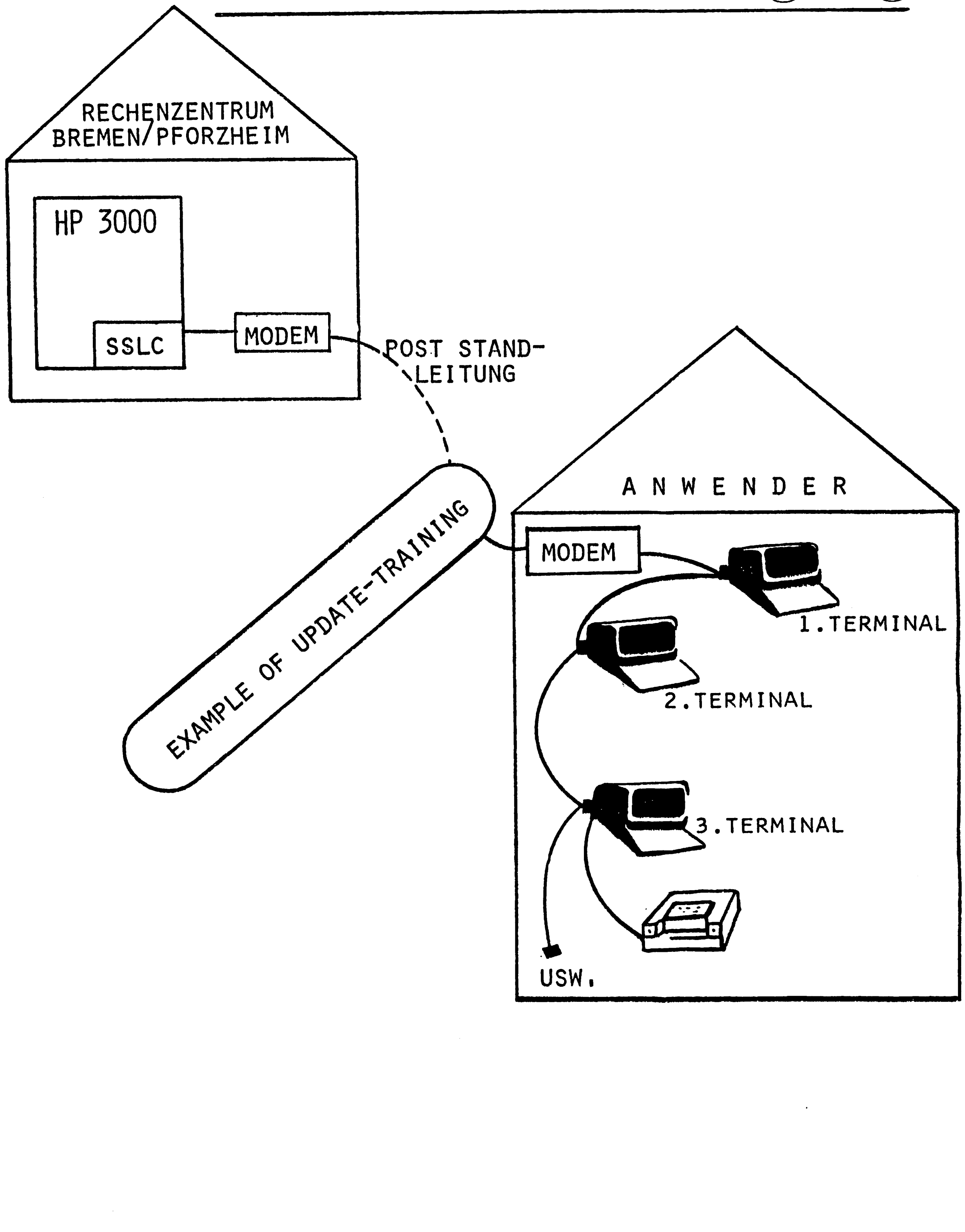
- REPEAT INFORMATIONS FROM INTRODUCTION AFTER SOME WEEKS/MONTH OF PRACTICE WITHIN 1 DAY THEORY
- HOW MTS WORKS
- HOW TO IMPROVE RELIABILITY OF DATA COMMUNICATION

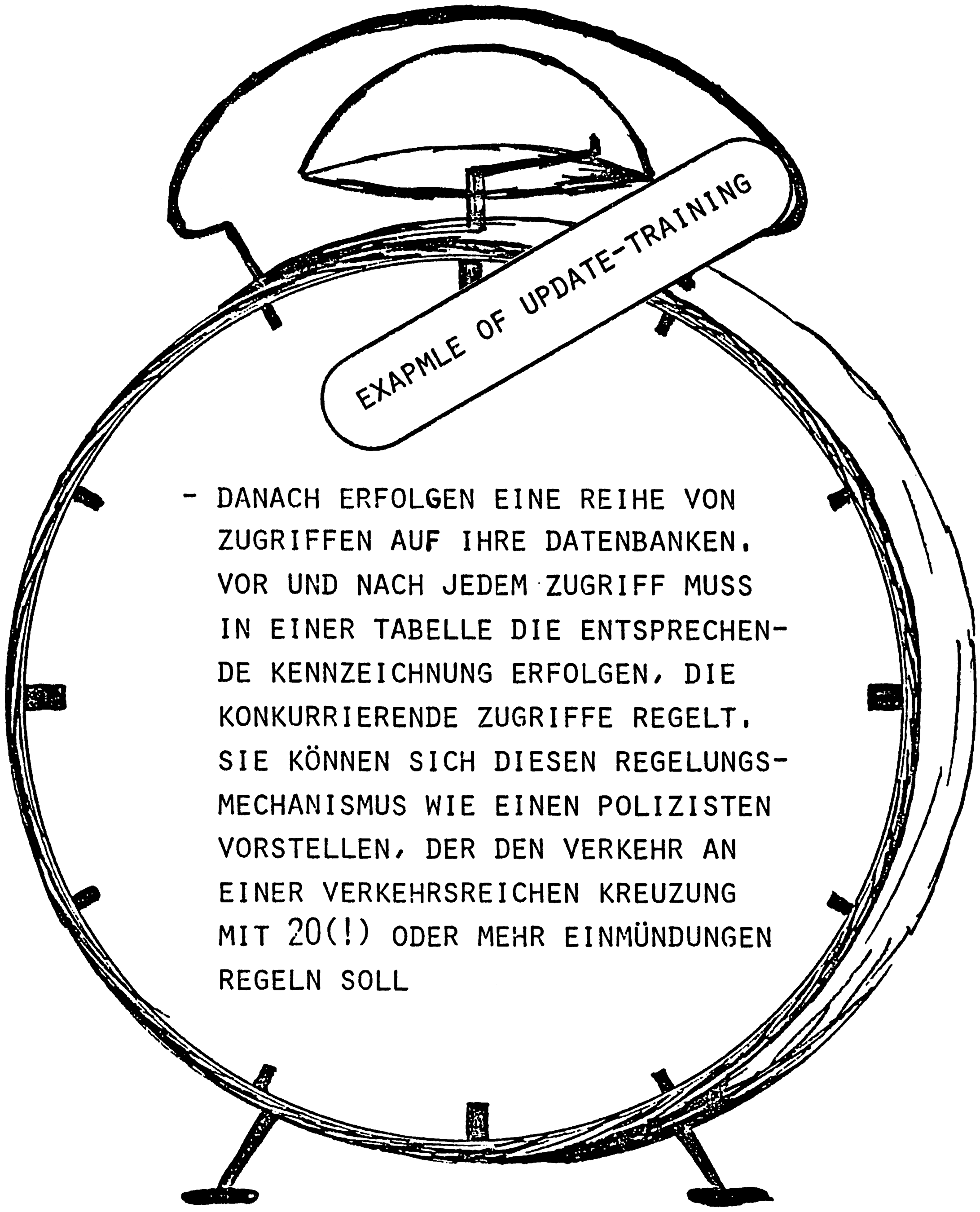
- STRATEGIES FOR LESS RESOURCE-USAGE AND BETTER RESPONSE TIMES
- HELPFUL HINTS AND TRICKS FOR TERMINAL- AND HARDCOPY USAGE
- WHAT A COMPUTER HAS TO DO FOR A "SIMPLE TRANSACTION" (OR WHY DOES IT TAKE SUCH A LONG TIME)
- DIFFERENCES BETWEEN JOBS AND SESSIONS
- REASONABLE USE OF QUERY
- MORE INFORMATIONS ABOUT TROUBLESHOOTING

SOME EXAMPLES ...

UFS2	RECHENZENTRUM HERBERT SEITZ KG	

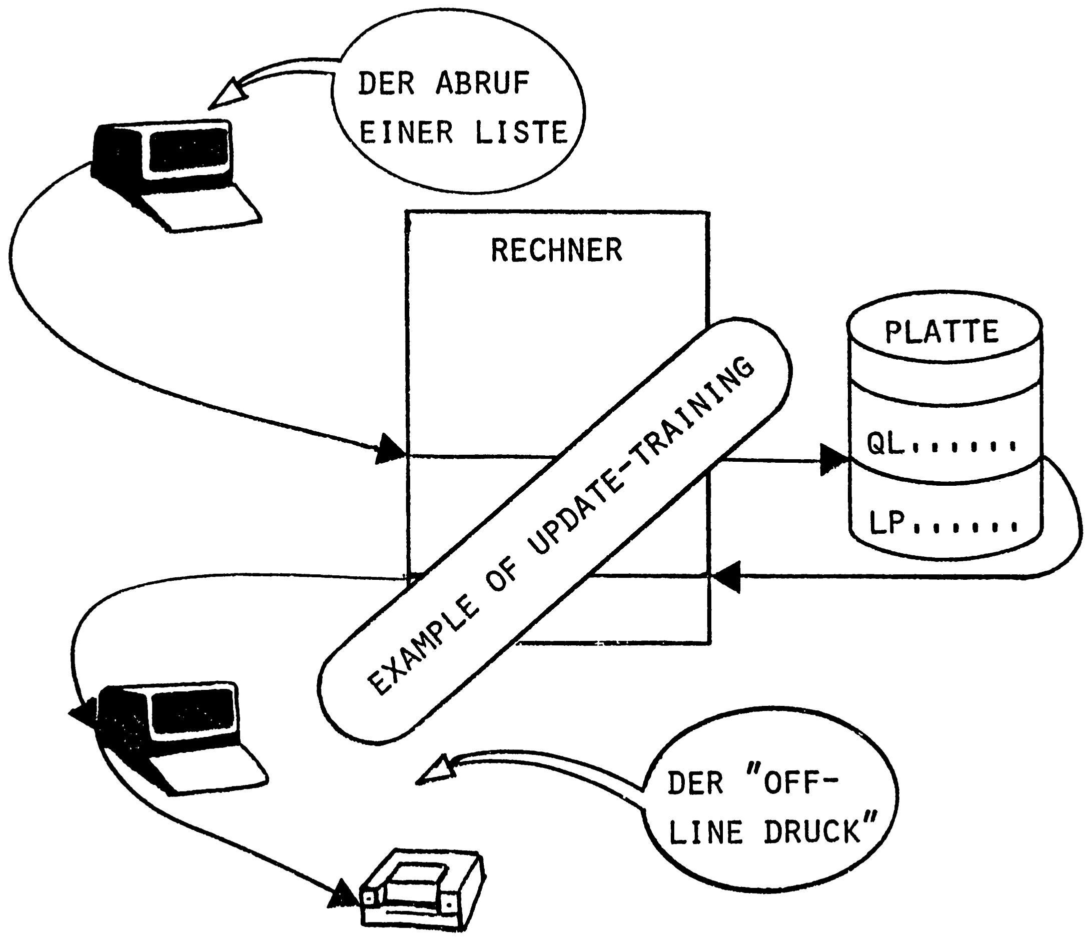
MTS Datenübertragung





- DANACH ERFOLGEN EINE REIHE VON ZUGRIFFEN AUF IHRE DATENBANKEN. VOR UND NACH JEDEM ZUGRIFF MUSS IN EINER TABELLE DIE ENTSPRECHENDE KENNZEICHNUNG ERFOLGEN, DIE KONKURRIERENDE ZUGRIFFE REGELT. SIE KÖNNEN SICH DIESEN REGELUNGSMECHANISMUS WIE EINEN POLIZISTEN VORSTELLEN, DER DEN VERKEHR AN EINER VERKEHRSREICHEN KREUZUNG MIT 20(!) ODER MEHR EINMÜNDUNGEN REGELN SOLL

LISTEN SELBST GEMACHT - WIE FUNKTIONIERT DAS ??



MIT DEM LISTEN-ABRUF WIRD EINE PLATTENDATEI ERZEUGT (KEIN PAPIER BEDRUCKT)

ERST DAS OFF-LINE DRUCKEN BRINGT DIE PLATTENDATEI AUF PAPIER

User Training (7)

APPLICATION TRAINING

- CLASSROOM-TRAINING OR TRAINING ON THE JOB
(KIND AND TIME DEPENDS ON APPLICATION)

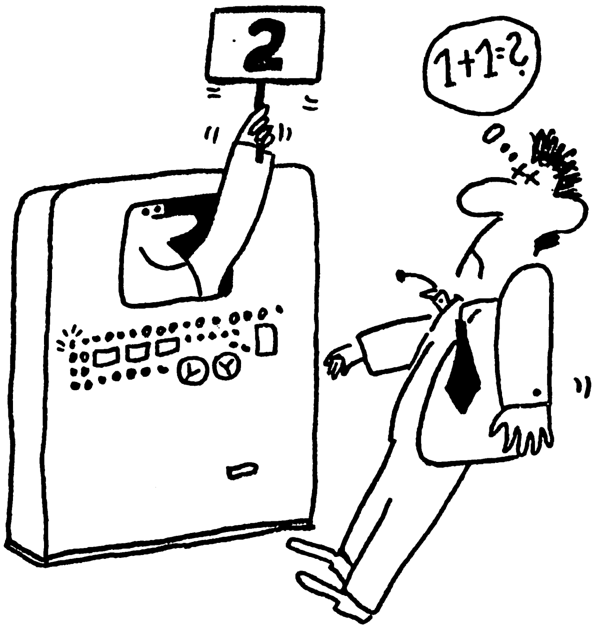
- "TRAINING COMPANIES" FOR TEST AND TRAINING IN ALL APPLICATIONS

- FREE PHONE IN CONSULTING FOR ALL USERS

User Training (8)

<u>QUERY FOR NON-DP-PERSONNEL</u>

- ⇨ 3-4 DAYS WITH 7 LABS
  QUERY FOR USERS, ONLY INFORMATION RETRIEVAL
  (NO UPDATE AND DELETE)
- ⇨ DATABASE TERMS AND THEORY (VERY LITTLE)
- ⇨ HOW TO USE "HELP, FORM"
- ⇨ SMALL REPORTS WITH "LIST"
- ⇨ "FIND" COMMAND
  (OR HOW TO TRANSMIT
  MR. BOOLE'S MESSAGE)
- ⇨ COMPLEX REPORTS



SOME EXAMPLES ...

Ü B U N G:

$$\left\{\begin{array}{l}\text{FORM}\\\text{FO}\end{array}\right\}\quad\left[\left\{\begin{array}{l}\text{DATA SET NAME}\\\text{DATA ITEM NAME}\\\text{SETS}\\\text{ITEMS}\\\text{PATHS}\end{array}\right\}\right]$$

	RICHTIG	FALSCH
FO		
FO PATHS		
FORM KDSTAM		
FOR KDNR		
FORM		
F ITEMS		
SETS FORM		
FORM ITEMS SETS		
FOR KDSTAM		

REPORT BEISPIEL

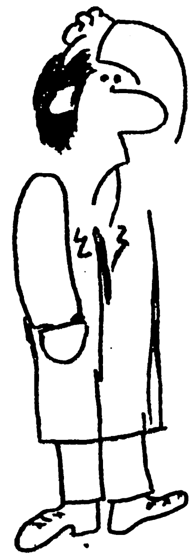
```
>D RP01

PROCEDURE: RP01

 001  R
 002  H1,"AUFTRAGSUEBERSICHT VOM",30
 003  H1,DATE,40
 004  H1,"SEITE",60
 005  H1,PAGENO,65,SPACE A1
 006  H2,"AUFTRG",6
 007  H2,"DATUM",13
 008  H2,"KDNR",19
 009  H2,"KUNDE",26
 011  H2,"MENGE",54
 012  H2,"VK-PREIS",65
 013  H2,"AUFTRGS-WERT",78,SPACE A1
 014  G1,"ARTIKEL:",8,SPACE B1,SPACE A1   } Druck der Artikel-Nr
 015  G1,ARTNR,18                          } und Bezeichnung bei
 016  G1,ARTKBZ,29                         } neuer Artikel-Nr.
 017  T1,R2                                } Löschen Reg.2 bei neuer Art.Nr.
 018  D1,AUFNR,6
 019  D1,AUFDAT,13
 020  D1,KDNR,20
 021  D1,KDKBZ,26
 022  D1,AUFMENGE,54,E1
 023  E1,"ZZZZZZZ9"
 024  D1,ARTVKPR,65,E2
 025  R1,L,AUFMENGE                 } Errechnen Auftragswert
 026  R1,M,ARTVKPR                  }
 027  R2,A,R1                       } Kumulieren Auftragswert je Artikel
 028  D1,R1,78,E2
 035  E2,"ZZZZZ9.99"
 036  E3,"ZZZZZZZ9.99-"
 037  T1,AUFNR,6,COUNT
 038  T1,"AUFTRAEGE",16
 039  T1,AUFMENGE,54,E1,ADD
 040  T1,R2,78,E2
 041  T1,ARTVKPR,65,E2,AVERAGE
 042  LINES=15
 043  PAUSE
 044  S1,ARTNR
 045  END
```

EXAMPLE OF QUERY TRAINING MANUAL

User Documentation

⇛ STANDARD DOCUMENTATION FILES
(FOR MANUAL PRINTING) ARE USER ACCESSIBLE

⇛ USER MAY SUBMIT HIS ADD ON DOCUMENTATION
INTO THE SAME DOC FILE



⇛ MANUAL FOLLOWS THE TYPICAL PROGRAM
SEQUENCE

⇛ ANY NEW INFORMATION ON THE
SCREEN IS DISPLAYED WITH PICTURES

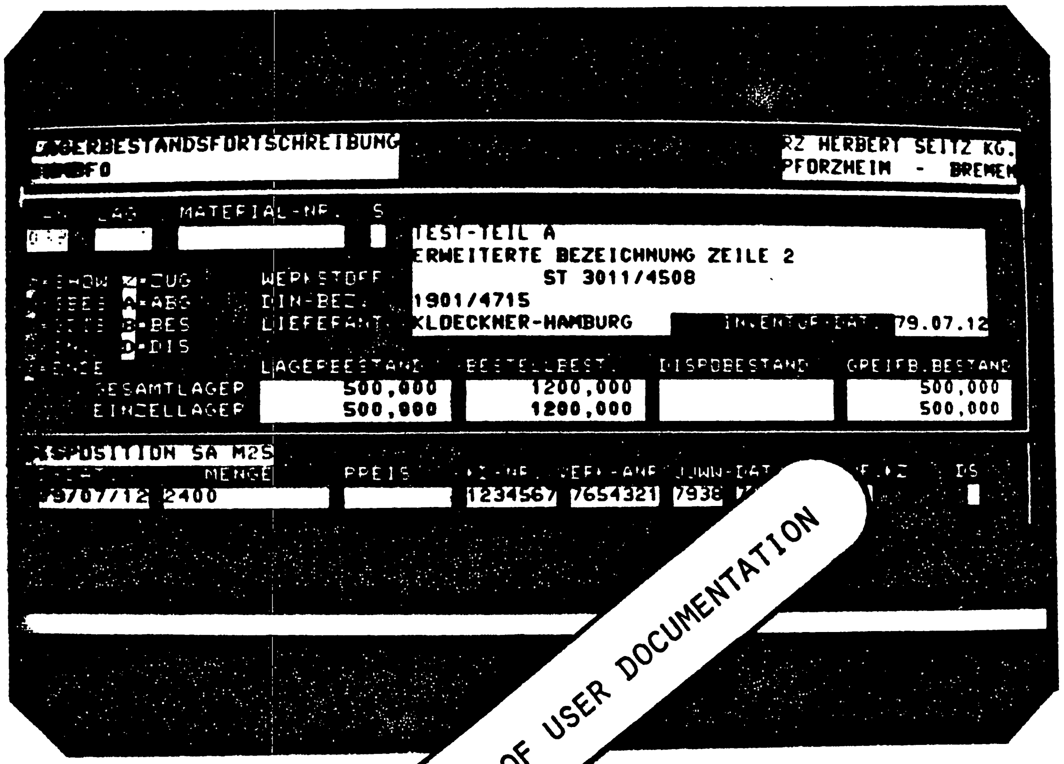
SEE EXAMPLE NEXT PAGE ...

UFS2	RECHENZENTRUM HERBERT SEITZ KG	

Sachgebiet

DISPOSITIONSERFASSUNG

WIR HABEN IN UNSERER GRUNDMASKE WIEDERUM DIE MATERIAL - NUMMER A UND DEN VERARBEITUNGSSCHLUESSEL D FUER DIE ERFASSUND DER DISPOSITION EINGEGEBEN. ES ERSCHEINT DIE ABGEBILDETE MASKE.



```
LAGERBESTANDSFORTSCHREIBUNG                          RZ HERBERT SEITZ KG.
    FO                                               PFORZHEIM  -  BREMEN

      LAG   MATERIAL-NR.   S   TEST-TEIL A
                               ERWEITERTE BEZEICHNUNG ZEILE 2
      ·ZUG      WERKSTOFF                ST 3011/4508
    A·ABG      DIN-BEZ.        1901/4715
    B·BES      LIEFERANT       KLOECKNER-HAMBURG     INVENTUR-DAT. 79.07.12
    D·DIS
                LAGERBESTAND   BESTELLBEST.   DISPOBESTAND   GREIFB.BESTAND
   GESAMTLAGER      500,000       1200,000                        500,000
   EINZELLAGER      500,000       1200,000                        500,000

DISPOSITION SA M2S
               MENGE     PREIS     KI-NR.   VERK-ANF  JUWN-DAT          DS
79/07/12  2400                     1234567  7654321  7938
```

D-DAT: DISPOSITIONSDATUM

FUER DIESES FELD GILT WIEDER DIE BEREITS GENANNTE REGEL FUER DATUMSERFASSUNG.

MENGE : DISPOSITIONSMENGE (3 KOMMA STELLEN, LOGIK WIE BESCHRIEBEN)

PREIS: IN DIESEM FELD KANN DER EINZELPREIS EINGEGEBEN WERDEN. WIRD KEIN WERT EINGEGEBEN SO HOLT DIE MASCHINE, FUER DIE BEWERTUNG DES AUFTRAGS-BESTANDES , DEN PREIS AUS DEM LAGERSTAMMSATZ.

Ausgestellt von	Datum	Kontrolle		Seite von
	01. 10. 79			

Form Nr. 000 02

- CNTL (CONTROL) DIE CONTROL-TASTE WIRD ZUR UMKEHRUNG VON NORMALFUNKTIONEN BZW. ZUR STEUERUNG VON SONDERFUNKTIONEN VERWANDT. ZUR AUSFUEH-RUNG EINER SOLCHEN SONDERFUNKTION MUSS STETS DIE CONTROL-TASTE FESTGEHALTEN WERDEN UND EINE WEITERE ANDERE TASTE GE-DRUECKT WERDEN. SO IST Z.B. DIE CONTROL-TASTE FESTZUHALTEN UND DIE TAB-TASTE ZU BETAETIGEN, WENN MAN FELDWEISE RUECK-WAERTS SPRINGEN WILL.

- SHIFT-TASTE MIT DER FESTGEHALTENEN SHIFT-TASTE KOEN-NEN DIE JEWEILS IM OBEREN TASTENBEREICH ANGEZEIGTEN ZEICHEN EINGEGEBEN WERDEN. IM BEREICH DER BUCHSTABEN A - Z DIENT DIE SHIFT-TASTE WIE DIE BUCHSTABEN-UM-SCHALTTASTE BEI SCHREIBMASCHIN[illegible] ZUR AN-STEUERUNG VON GROSSBUCHSTAB[illegible] REGEL FALL IST JEDOCH DURCH DIE [illegible] FIGU-RATION (CAPS-LOCK-TAST[illegible] SHIFT-TASTE DER GROSSBUCHS[illegible] EINGESCHAL-TET.

- LEERTASTE DIE BREITE LE[illegible] UNTEREN TASTATUR-BEREICH DI[illegible] UF DER SCHREIBMASCHINE, ZUR EIN[illegible] ERZEICHEN. DIESE TASTE HAT WI[illegible] EREN TASTEN AUCH EINE FORTS[illegible] UNKTION, DIE BEI LAENGEREM DRUECK[illegible] N- UND BEIM LOSLASSEN DER TASTE WIEDER AUSGESCHALTET WIRD.

- DEL-TASTE NICHT VERWENDEN.

- RETURN-TASTE DIE RETURN-TASTE HAT NUR UNTER FOLGENDEN BEDINGUNGEN DIE FUNKTION EINER SENDE TASTE (ZUM RECHNER):

  - DAS TERMINAL IST NICHT IN EINEM MEHRTERMINALBETRIEB (MTS) ANGESCHLOS-SEN

  - SIE ARBEITEN Z.ZT. NICHT IN BILDSCHIRM-MASKEN SONDERN IN ABFRAGE- ODER DRUCKPROGRAMMEN (MENUE, QUERY, OFF-LINE-DRUCK)

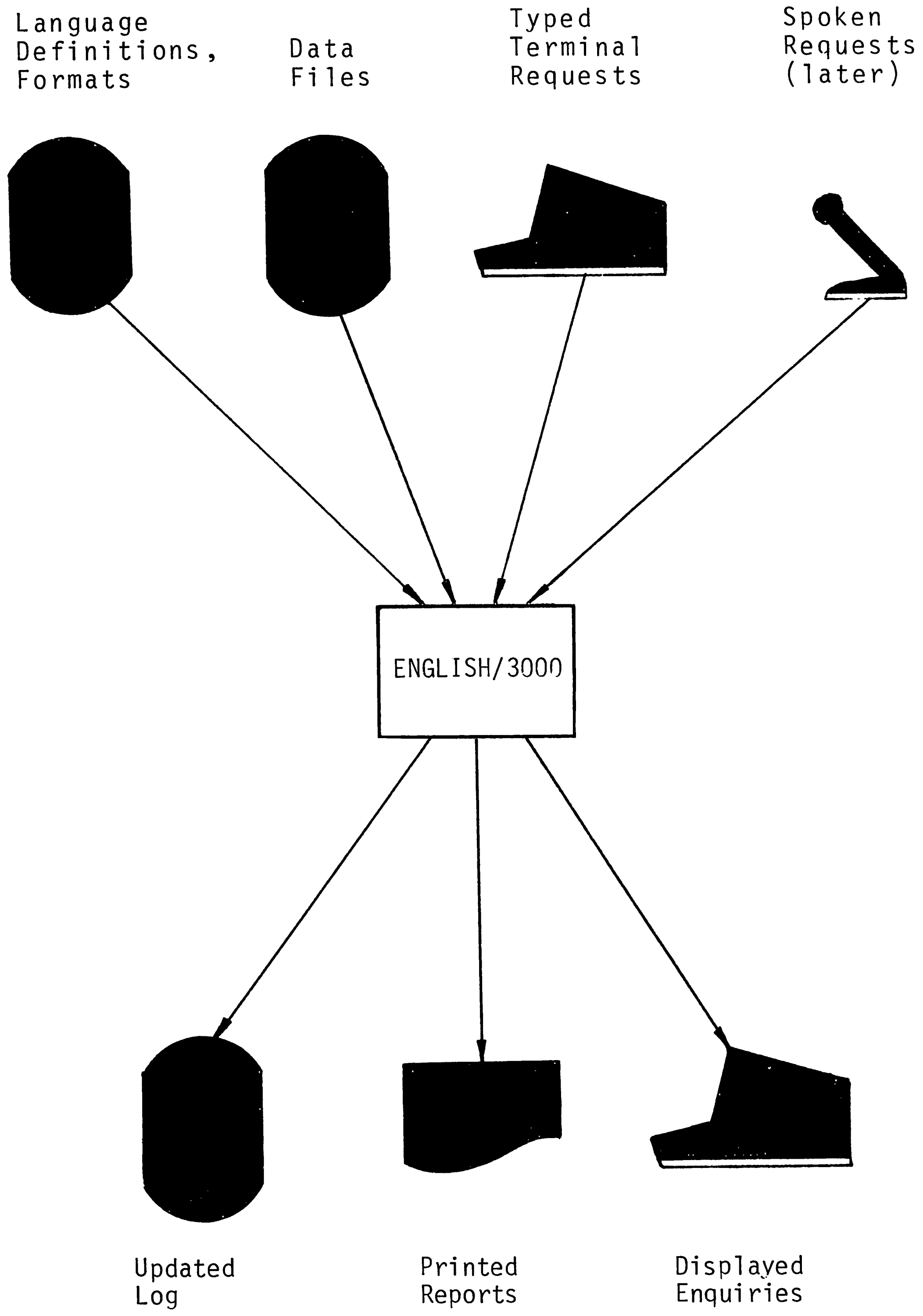
Ausgestellt von	Datum	Kontrolle		Seite von
GE-CH	01.10.79			

E N G L I S H / 3 0 0 0

A NATURAL LANGUAGE ON A MINI-COMPUTER

Doug Peckover
D.P. Concepts Inc.
98 Perodeau
Vaudreuil
Québec J7V 5V5
(514) 455-1373

ENGLISH/3000 SCHEMATIC



WHAT & WHY?

"Hardware vendors could go a long way toward eliminating the problem of computer-caused unemployment if they were somehow able to achieve a technological breakthrough in natural computer languages. Development of a powerful, easy-to-use natural language would make computing systems available for the first time to a large class of unskilled users who would otherwise find the systems forever intimidating and inaccessible".

Alvin Toffler - author of "Future Shock"

What Mr. Toffler said is equally applicable to the thousands of "unskilled"managers who already have access to computers, but find them "intimidating and inaccessible".

There is presently a furious race to produce the world's first Voice Recognition Unit (VRU) that would understand our most natural language - our native tongue. Everyone from the heavies at the IBM T.J. Watson and Yorktown Heights labs as well as the Bell labs to fairly small micro and terminal manufacturers are producing encouraging, but crude VRU's. The general feeling is that a fully "unrestricted" VRU is still 8-20 years away. However, there are many enormous benefits to natural languages that are available right now. ENGLISH/3000 will shortly permit the HP 3000 users to take the first of three phases to a fully unrestricted natural language:

Phase 1 Accept a TYPED terminal command in a NATURAL (unrestricted) format. Interpret and execute the command. ("PLEASE SHOW ME ALL FINAL PRODUCTS THAT USE A 56421 BOLT" - this can be abbreviated).

Phase 2 Accept a SPOKEN voice command in an ARTIFICIAL (restricted) format. Interpret and execute the command. ("SHOW FINAL PARTS 56421").

Phase 3 Accept a SPOKEN voice command in a NATURAL format. Interpret and execute the command. ("PLEASE SHOW ME ALL FINAL PRODUCTS THAT USE A 56421 BOLT").

Phase 3 is probably 8-10 years away. Phase 2 is 3-5 years away. Phase 1 is available now.

There are many advantages to simulating VRU's. These include:

1. User training for enquiries and reports would be reduced to a minimum. Instead of having to study your documentation and running your programs, they would explain what they require just as they would to another person.

2. Slight variations in how a request is worded could format the information in many different ways. For example, "Display the parts list..." and "Display the costed parts list..." could indicate different requirements for two separate departments.

3. You will be able to start designing for the second and third stages that will have the most profound impact on your organisation. What we learn will not only prepare you now for VRU's but will also simplify the selection criteria for them as they come onto the market.

When the information requested is ready to print, we need a powerful report generator to produce terminal enquiries and printed reports. The benefits provided by ENGLISH/3000 include:

1. Traditional programming costs for most enquiries and reports would be slashed by up to 90%. The net increased productivity to your programming staff could double their output.

2. Your managers and users will be able to write their own enquiry and report formats. This would remove much of the load traditionally placed on the data processing department.

As with all major projects, specific design goals were set for ENGLISH/3000. The following pages outline some of the prime considerations that specified the framework on which the product was developed.

EASE OF USE

There can be no easier way to train a manager or user how to use a new system than by permitting him to use his natural language, whether it be English, French, Spanish, and so on.

1. No matter how complex the data structures are or how involved the programs get, when it comes down to a one on one enquiry, the user can communicate with the computer on an equal basis. There are no programs to memorize, no keys to remember, no rules to look up.

2. The training for enquiries and reports becomes the responsibility of the System Manager as he is the one to determine who needs what. The reduced number of conventional enquiries and report programs will make new systems easier to teach. In addition, ENGLISH/3000 could significantly reduce the sales effort needed by software OEMS by giving the System Manager these additional customizing facilities. This ease of use could even lead to a lower support cost for many installations.

3. The user can mix dependent demands, such as bill of materials with independent demands, such as customer orders, without the need to understand their relationship or dependancy on the data base(s).

4. Special considerations have been taken in designing the report writer. The assumed technical knowledge of the System Manager is EDIT/3000 and QUERY/3000. An ENGLISH/3000 format (or procedure) compiles itself automatically the first time it is used after a modification has been made. When you receive each new release of ENGLISH/3000, any modifications required to your source code (if any) will be automatically made and the format will be recompiled.

5. The formats that drive each request in ENGLISH/3000 are coded in a non-procedural way. This further reduces the need for amount of technical training required to code formats.

6. ENGLISH/3000 allows full comments - both at the heading and line level. The product will soon have an automatic flowcharting facility. This will document any format, showing loops, labels, comments, and so on.

SPEED CONSIDERATIONS

Most of the advantages of a natural language would be thrown out the window if the computer took too much time to process the request.

1. As a design goal, the average response time required to accept, interpret and begin executing the command (start displaying the enquiry or start STREAMING the report) will be 2 seconds. A tuning aid is provided that will help you improve the efficiency of your enquiry and reporting load on the computer.

2. ENGLISH/3000 supports KSAM indices for IMAGE master data sets. This eliminates the need for a serial read through the entire data set and the subsequent sort. A utility for maintaining these indices is included with ENGLISH/3000.

3. All reports are automatically STREAMed from the terminal. This not only frees the terminal for the next command but also permits the system to lower the report priority and queue the reports if necessary. At the user's option, the report can be flagged to only start after hours.

4. The source code used to describe the format is automatically compiled and stored when it is first used. All subsequent requests to the same format result in a much faster execution time.

5. Most enquiries and reports on your computer (say - 50% of your workload) would be handled by one single very efficient program that can be locked in memory. This would greatly reduce the system swapping by using the re-entrant facilities in the 3000. System thruput would be increased, improving response time, not only with ENGLISH/3000 but also your other application programs. This may reduce the need for hardware upgrades and possibly reduce the need for after-hour operations.

SECURITY CONSIDERATIONS

If natural languages will enable anyone to communicate freely with the computer, then a necessary part of our design objectives must be a complete review of conventional security methods.

1. ENGLISH/3000 comes with security at two levels. The first is the security provided with IMAGE. The second has been taken from a system designed for a large military supplier that had to be very secure. The System Manager identifies everyone in the company in a tree structure. The ENGLISH/3000 formats are then assigned to the users on a need-to-know basis. ENGLISH/3000 permits the user or any of his superiors to have full access to the format. However, no one below the user in the structure has access to the format. Invalid attempts to run a format are handled conversationally by telling the user see the department head that "owns" the format ("Please see Bob Smith..."). The error is also logged for that department head's attention.

2. If the user makes a specified (default 4) number of consecutive errors in this manner the terminal is jammed and must be freed by the operator. This makes the system secure if dialup terminals are used.

3. The System Manager has additional facilities for reports. He may choose to receive a log of all occurrences of a certain report, who ran it, when, and for what reason. He may also have a special security label precede any report. This will contain CONFIDENTIAL in large letters as well as the user requesting the report, delivery instructions, and so on. He may also have a system-generated serial number label each page of the report and log it's occurance.

4. After a pre-defined period of time, ENGLISH/3000 will do a timeout and re-request the user identify himself. This means that a user who has forgotten to sign off will not leave his terminal in a ready state.

5. As we move closer to VRU's, the user's password will be needed to identify his speech patterns. Eventually, the speech patterns themselves will be used to identify the users.

EVENT LOGGING

There are many useful byproducts of ENGLISH/3000 that are available from the logging facility. These will most likely be expanded to meet the changing needs of the client base requirements.

1. One of today's trends is to treat computers as profit centers. Departments, users, clients, and budgets are charged a portion of the costs on a flat rate or as-used basis. To facilitate this, ENGLISH/3000 has a sophisticated logging facility that enables you to charge according to the enquiry and reports used. You may charge by any combination of the following: lines or pages that are displayed or printed, CPU seconds used, elapsed time, disk reads and/or copies requested.

2. Sensitive reports, such as price lists, can be logged with their serial number printed at the bottom of each page. This then enables you to keep a journal of who has what version of what report.

3. The log can be used to analyse who is using what facilities. This will enable you to use the tuning facility to improve the efficiency of ENGLISH/3000.

4. The log can note unsuccessful attempts to run a program ("Bob Smith tried to run the PRICE LIST and was told to see Bill Boss"). Arrangements can then be made to see if Bob should have access to this request.

5. ENGLISH/3000 will use the logging facility to note requests that it could not interpret. This may be periodically analysed so that changes can be made to the language definitions. Studies indicate that this improves the chances of ENGLISH/3000 understanding a request on the first attempt from 90% up to 98% of the time.

6. At a later stage, ENGLISH/3000 can be modified to log certain data from key reports (such as monthly totals). These can in turn be reported on by selectively analysing the log file. These enhancements will be defined by user response to surveys.

DATA STRUCTURES

The usefulness of a natural language facility should not be limited to crude reports that must be used to output the information.

1. ENGLISH/3000 will support any combination of IMAGE data bases (multiple data sets), KSAM and MPE files as well as remote and local computers.

2. While not specifically aimed at the manufacturing users, there will be several data structures available that will be of particular use to manufacturers. Bill of material and where-used recursive structures to a specified number of levels will be supported. In addition, the format can specify that the intermediate sub-assemblies should not print, and only the final level should print. This is commonly required to show the raw materials for a given product or show which final assemblies use a particular sub-assembly or raw material.

3. Generic names and keys will be available for record selection, conditional printing and conditional branching.

5. Optional data set and file locking will enable the enquiry or report to be completed without any unexpected events.

PRODUCT SUPPORT

The intent of ENGLISH/3000 is to not only provide you with a powerful facility now, but to eventually accept requests via VRU's in a fully unrestricted manner.

1. Normal support is provided to continuously upgrade and improve ENGLISH/3000 for VRU's as well as to fix bugs and design oversights. By purchasing ENGLISH/3000, you are guaranteed an upward compatible path that will lead you and your organisation to the world of fully unrestricted natural languages with VRU's. The newsletter (described below) will help guide you on the selection of VRU's as they become available.

2. Extended support will shortly be introduced. This will provide you with a phone-in consulting service as well as a revolutionary new service that we believe is new to the industry. For installations that do not have programmers or installations that experience peak loads, we will offer a phone-in programming service. Your enquiry or report will be designed, coded, tested, and documented on your computer within 24 hours. (The request must be within the design limitations of ENGLISH/3000 and we must have access to your computer via a good dialup line). The request will be billed on a time and material basis with an agreed to ceiling.

3. D.P. Concepts Inc. will publish a monthly newsletter to keep you, the user, up-to-date and informed. Featured will be user tips and recommendations, progress reports on the VRU developments from the major labs and hardware vendors with product testing and recommendations, developments and standards from the American Association for Artificial Intelligence, notes on evolving natural language applications, and user survey forms for proposed enhancements. This newsletter will be sent to all ENGLISH/3000 users with either normal or extended support.

SUMMARY

Besides the more obvious reasons for buying a natural language facility (with a powerful report generator) three more important reasons exist:

1. A well designed natural language facility can actually pay for itself and save you money in the following ways:

    - reduce training times
    - reduce programming times
    - reduce documentation times
    - reduce lead times for new programs and modifications
    - charge users and departments for resources used

2. In the short term, natural languages will be used mainly for enquiries and reports. Eventually, the natural languages will be used to accept input as well. In addition,development of artificial intelligence on computers will permit the user to try more powerful requests, while requiring a reducing knowledge of how the computer works. ENGLISH/3000 will pursue these trends.

3. A well designed natural language should be able to accept input, interpret, and produce enquiries and reports for virtually any language. The only limitation appears to be whether the language has a terminal and printer that supports the character set for the particular language. Within this limitation ENGLISH/3000 can be converted to any other language.

ENGLISH/3000 Example
===================

PLEASE SHOW THE CURRENT LEVELS FOR A 1736-90.

... could execute the following format ...

```
REPORT
<<   Enquiry Name:   INVENTORY LEVELS
<<   Author      :   Manual Labour
<<   Last Changed:   23 Jan 81

<< This format displays the current inventory levels for
<< the selected part number by branch. The total for all
<< branches is shown at the end of the enquiry.

<< Steps:    - Read PART-NO from ITEM-MASTER
<<           - Use PART-NO to set up STOCK-LEVEL details
<<           - Read and print each detail
<<           - Use BRANCH to set DESC in BRANCH-MASTER
<<           - Print "*" if BRANCH-MASTER STATUS is "S"

IM MEANS ITEM-MASTER                 << Abbreviate "ITEM-MASTER"
SL MEANS STOCK-LEVEL                 << Abbreviate "STOCK-LEVEL"
BM MEANS BRANCH-MASTER               << Abbreviate "BRANCH-MASTER"

H1,"Inventory Status Enquiry",26     << Enquiry Heading
H1,DATE-TIME,66                      << Set up Date-time stamp
H2,"Part Number",22,SPACE 2B         << Column Heading for PART-NO
H2,"Branch",37                       << Colunm Heading for BRANCH-NO
H2,"On Hand",52                      << Column Heading for QUANTITY
H2,"Status",66,SPACE 2A              << Column Heading for STATUS flag

D1,IM.PART-NO,20                     << Print PART-NO from ITEM-MASTER
D2,SL.PART-NO(IM.PART-NO),DUMMY      << Set chain read in STOCK-LEVEL
D2,BM.DESC(SL.BRANCH-NO),41          << Print DESC from BRANCH-MASTER
D2,SL.QUALTITY,52,"ZZZ9.99"          << Print QUANTITY from STOCK-LEVEL
D2,IF,"S",EQ,BM.STATUS,"*",63        << Print "*" if STATUS = "S"
TF,"Total:",37,SPACE 2B              << Footing literal
TF,QUANTITY,52,"ZZZ9.99"             << Print total at end
END
```

... which would display the following resuls ...

```
  Inventory Status Enquiry                WED, MAR 18, 1891,  9:04 PM

           Part Number           Branch          On Hand          Status

             1736-90             NEW YORK         142.00
                                 DALLAS             9.50              *
                                 CHICAGO           14.00

                                 Total:           165.50
```

SYSTEMS LIFE-CYCLE - A FRAMEWORK FOR SUCCESS

By M.B. Foster

M.B. Foster Associates Limited

ABSTRACT

The objective of this talk is to describe the systems life-cycle as a software development plan. It is the author's opinion that a planned software development project will result in better quality, less costly and (with luck) on-schedule programs.

The areas which are covered in this talk are the stages of the systems life-cycle:

1. Conception - everything has to have a beginning
2. Feasibility Study - technical, organizational and economic
3. Functional Specification - what are the business needs
4. Detailed Specification - how are these needs solved technically
5. Programming, Testing and Implementation
6. System Installation
7. Post-Installation Review - an on going process

In each stage the resources (human and HP3000) required the major activities and the benchmark documents produced are discussed.

The conclusion drawn is that the software development process can use the systems life-cycle methodology as a framework for successful completion of the projects, which because of the framework will be better controlled on time and in-budget.

SUCCESSFUL CONVERSION FROM TWO IBM SYSTEM/3 TO A HP3000

by:
Jack McQuillen
Dundee Cement Company

THE PURPOSE OF THIS PRESENTATION IS TO RELATE THE DUNDEE CEMENT COMPANY EXPERIENCE IN CONVERTING FROM THE IBM SYSTEM/3 MODEL 10 AND MODEL 15 COMPUTERS TO A HP3000.

WE STARTED WITH 500 RPG PROGRAMS IN DUNDEE,MI AND 400 RPG PROGRAMS IN HOLLY HILL,SC. WITH ONLY THREE PEOPLE, WE HAD HOLLY HILL RUNNING LIVE IN 5 MONTHS AND DUNDEE RUNNING LIVE IN 10 MONTHS.

OUR MAJOR PROGRAMMING EFFORT WAS THE ON-LINE KEYPUNCH VERIFICATION USING V/3000. OUR SECOND MOST TIME CONSUMING EFFORT WAS THE JCL PREPARATION AND JCL DEBUGGING.

IN THIS PRESENTATION THE SUBJECTS TO BE COVERED ARE:

A. IBM SYSTEM/3 VS HP3000 BENCHMARK
B. PROGRAM CONVERSION
C. OCL TO JCL CONVERSION
D. DATA ENTRY PROBLEMS
E. HP3000 ACCOUNTING STRUCTURE
F. CONVERSION PLUS TWO YEARS

1. BENCHMARK

A. TIMINGS - 100 HOURS/MONTH TO 35 HOURS/MONTH
B. SOURCE LINE NUMBER IDENIFICATION FOR PROGRAM ABORTS
C. DEVICE INDEPENDENCE

2. PROGRAM CONVERSION

A. TRANSPORT SOURCE TO HP3000 VIA MAGNETIC TAPE OR RJE BYSYNC
B. DEVICE CONVERSION IN THE FILE STATEMENTS
C. ISAM TO KSAM
D. PACKED FIELDS TO UNPACKED ASCII
E. DATECARDS BY APPLICATION
F. EZRPG FOR PROGRAM CODING IN FREE FORMAT
G. HANDLING HALT INDICATORS

3. OCL TO JCL CONVERSION

A. CONVERSION AID PROGRAM
B. EXPLICIT DEFINATION OF EACH FILE STATEMENT
C. IBM S/3 SORT PROGRAM
D. "XX" JOBS
E. SPECIAL FORMS ALIGNMENT
F. REMOTE HP2631 PRINTERS
H. JCL LISTINGS IN SPOOK

4. DATA ENTRY OR KEYPUNCH FUNCTION

A. V/3000 AND THE ENTRY PROGRAM
B. "SKIPPER" AND COBOL VERIFICATION
C. MULTIPLE LINES (LOGICAL RECORDS) PER SCREEN
D. SPEED OF KEYPUNCH VS HP2645

5. HP3000 ACCOUNTING STRUCTURE

A. GROUPS BY APPLICATION,EX. PAYROLL,SALES SATAISTICS
B. DATAENTRY GROUP
C. JOB GROUP FOR JCL
D. SOURCE GROUP FOR PROGRAMS
E. DOCUMENT GROUP

6. CONVERSION PLUS TWO YEARS

A. HOLLY HILL RUNNING LIVE IN 5 MONTHS FROM AN IBM S/3 MODEL 15
B. DUNDEE RUNNING LIVE IN 10 MONTHS FROM AN IBM S/3 MODEL 10
C. REMOTE ORDER ENTRY IN 24 MONTHS
D. AVERAGE USERS IS 18 PER DAY
E. OS TO HP3000 SERIES 30
F. 50 SALESMEN USING DIAL UP TO A MARKETING DATA BASE
G. ON LINE INVENTORY DATA BASE
H. FORTRAN CONVERSION FROM AN IBM 1800 PROCESS COMPUTER TO HP3000

VTEST/3000 ON-LINE TEST HARNESS - USER VIEW

By:

Peter Byers
Glaxo Operations UK LTD

Glaxo operations UK Limited manufacture Pharmaceuticals at eight factories and have a network of eleven linked-HP 3000's using VIEW, IMAGE and DS. They are rapidly developing a large integrated production planning and inventory control system. They recognized the need for a test harness which:-

Facilitates repetitive testing of real time view programs:-

Tests multi-user database contention:

Provides screen-like hard copy of input and output:

Is streamable in non prime time:

Provides timings.

The package was developed by Wick Hill associates with advice from Glaxo. It reads input from script files held on disc. Results are written to an MPF file (usually a printer). This permits input and standard output to be stored on magnetic media, instead of bulky hard copy.

The programs can be run either as a session from a terminal or streamed as a batch job. It established one or two sessions which can run any programs able to be run from a terminal. The only limitations are that control Y and break cannot be used. The great strength is that programs using View in block mode can be run.

The paper explores the problems of testing and estimating the response time of on-line and real-time programs running in a large distributed network: explains how VTEST works: and illustrates some of the advantages to be gained from the use of the test harness, e.g.:-

Overnight Testing, thus

- Releasing terminals in prime time
- Improving response in prime time
- Improving utilization of a capital asset:

Better program specs, less program amendments:

- Quicker programming:
- Test data keyed-in once, speeds testing after modifications;
- Printed results aid user education

INTRODUCING THE HP ON-LINE PERFORMANCE TOOL (OPT/3000)

Robert L. Mead Jr.

Member of Technical Staff

Hewlett-Packard Company

Computer Systems Division

Robin P. Rakusin

Product Manager

Hewlett-Packard Company

Computer Systems Division

INTRODUCTION

The question of whether or not a computer system is being effectively utilized is often difficult, if not impossible, to answer. Equally difficult can be the identification of a bottleneck when the performance of a system is less than expected. These difficulties typically arise due to a lack of information on which to base a judgement or decision. Even in those situations where information is available, it is often the case that information is incomplete, or possibly inaccurate or misleading, thus forcing the analyst to make a "best guess" as to the true situation. When detailed and complete information is available, it is frequently difficult to separate the useful information from the vast amount of data provided. In this paper we describe an interactive software product designed specifically to aid in the analysis of HP 3000 computer system performance, and which addresses the problems just described.

This product, the HP On-line Performance Tool (OPT/3000), is Hewlett-Packard's first performance measurement software product, and can be used to identify performance problems or bottlenecks, to characterize the workload on an HP 3000, to collect information required for capacity planning activities, to analyze system table configurations, and in some cases, to tune the performance of individual applications. OPT/3000 provides information in 23 separate interactive displays in the following areas: CPU utilization and memory management activity, memory usage, I/O traffic, program and process activity, and system table usage. Although each display is designed to be quickly and easily understood, the assumption is made that the user has been trained on the internal operation of MPE IV, the newest version of the HP 3000 Multiprogramming Executive operating system. OPT/3000 is designed to operate in conjunction with MPE IV and can be used on any HP 3000 Series II, Series III, Series 30, Series 33, or Series 44.

This paper presents an overview of the HP On-Line Performance Tool, and discusses some intended applications of OPT/3000. The information reported by OPT/3000 is also reviewed in-depth, as well as the techniques used to obtain the information.

OVERVIEW OF OPT/3000

The HP On-line Performance Tool is a software product that provides performance related information in an interactive environment. As mentioned earlier, OPT/3000 can generate 23 different displays containing performance related information, in addition to seven menu displays. These displays are grouped into six categories, called display contexts, each of which is associated with a different type of system resource. The six contexts are: Memory, CPU/Memory Management, I/O, Process, System Tables, and Global (a little bit of everything). Within each context, displays are available at successively greater levels of detail. This structure allows the user to progress from summary level information to more detailed information as the situation requires. In many cases, the summary level information is sufficient.

Once a display has been generated, it is automatically updated at periodic intervals, with the length of the time interval under control of the user. A display can also be updated upon demand, simply by entering a carriage return. All commands within OPT/3000 consist of a single ASCII character, and a different set of commands are available in each display context. Certain global commands are available in all contexts. In addition, the pound sign character (#) is used as an escape character to access a set of control operation commands. These commands perform such operations as changing the current display context and suspending the updating of the current display. With this simple user interface, the generation of a different display within the current context is accomplished via a single keystroke, and the generation of a new display within a different context with a minimum of three keystrokes. Menu displays are available within each context, and list the commands available within that context.

An extensive on-line help facility is also available as an integral part of OPT/3000. With this facility, documentation explaining any command or display can be quickly displayed. In many cases, interpretation guidelines are also provided to aid in the identification of performance problems.

OPT/3000 utilizes the features of the HP 264x series of terminals to generate displays with a graphical format, where practical. The terminal features used include the four available video enhancements (blinking, inverse video, underlining, half-bright), the line drawing character set, and the cursor addressing capabilities. OPT/3000 automatically checks to verify that an appropriate terminal is being used, and warns the user if an incompatible terminal is in use.

A hard copy of any display can be generated on the line printer (device class LP) with a single keystroke. The hard copy displays are similar in layout to the interactive displays, but some reformatting is necessary to convey the same information, due to the lack of video enhancements on a line printer (e.g. paper cannot blink).

Although the HP On-line Performance Tool is primarily designed for interactive use, it can be executed in batch mode to collect summary information about system activity. These summary reports can be used to provide data for capacity planning activities, and can be generated interactively as well. Once activated, the summary reports are generated independent of the interactively generated displays.

There is no limit on the number of copies of OPT/3000 which can be executing simultaneously. OPT/3000 obtains much of its information via a new internal measurement interface facility incorporated within MPE IV. This facility maintains a set of measurement counters accessible by multiple users. Additional information concerning the measurement interface, and the techniques used by OPT/3000 to collect information, will be discussed in a subsequent section.

APPLICATIONS OF OPT/3000

There are several anticipated uses for the HP On-line Performance Tool. Among these uses are the identification of performance problems and bottlenecks, the analysis of system table configurations, characterization of the system workload, capacity planning, and performance tuning of applications. Each of these activities utilizes some or all of the capabilities of OPT/3000. We will now briefly discuss each of these application areas before describing in some detail the information provided by OPT/3000.

The ability to quickly move between displays and the variety of information available through OPT/3000 facilitates its use in identifying performance problems and bottlenecks. In particular, it is expected that a system clearly bottlenecked by CPU, memory, or I/O will be quickly identified. OPT/3000 can also be used to determine if disc accesses are unbalanced between multiple drives. Poorly behaved application programs can also be identified, in terms of programs which use excessive numbers of files and extra data segments and those which waste stack space.

A second application area is that of system table configuration analysis. Inappropriately configured system tables can degrade system performance, either by wasting memory if the tables are unnecessarily large, or by causing processes to delay while waiting for an entry in a table that is configured too small. In the latter case, system failures may also result if the table size is exceeded. OPT/3000 allows the user to quickly identify those tables which are not properly configured, and to determine (through utilization statistics) a more appropriate value.

The characteristics of the workload on an HP 3000 can be determined using OPT/3000. The names of all active or allocated programs on the system can be easily determined, as well as the users of each program. The CPU usage, disc I/O rate, and memory usage characteristics of an individual application can be determined if the application is running stand-alone on the system.

The summary reports which can be generated by OPT/3000 in either batch or interactive mode can be used to provide data for capacity planning activities. These reports indicate the CPU usage of the system, memory management activity, and the I/O traffic on individual discs, line printers, and magnetic tapes. The information used to generate the summary reports can also be logged to an OPT/3000 log file (disc or tape), which could be processed to provide input for generating plots. In this manner, OPT/3000 can gather trending information that can be used to determine when additional peripherals or systems are needed, or to detect changes in the day-to-day processing load.

Although OPT/3000 is oriented towards the measurement and analysis of the system as a whole, it can be of some value when tuning the performance of individual applications. In particular, OPT/3000 can provide information relating to an application's usage of files and extra data segments, plus detailed information about the application's use of its stack. CPU usage information is also available.

INFORMATION PROVIDED BY OPT/3000

As mentioned earlier, the HP On-line Performance Tool provides information in six different display contexts: global, memory, CPU/memory management, I/O, process, and system tables. In this section we describe the types of information available in each context. In general, the information provided by OPT/3000 can be divided into two basic classes. The first class of information shows the state of some aspect of the system at the moment the display update is generated. Examples of this class of information include the current contents of main memory and the current list of active programs. The second class of information summarizes activity within the system during some interval. CPU utilization and disc I/O rates are examples of this second class of information. In most cases, this summary class information is reported for two types of intervals: the interval between the previous display update and the current display update, and the interval encompassing all update intervals since the start of OPT/3000 execution. These two intervals are herein referred to as the current interval and the overall interval, respectively. The user can clear all totals associated with the overall interval at any time in order to start a new overall interval. We will now discuss the information available in each of the display contexts.

Global Context

The global context is automatically entered upon execution of OPT/3000, and it provides summary level information concerning CPU usage, memory utilization, disc I/O rates, and process activity. The generation of summary reports is also controlled from the global context. The two displays in the global context can be used to quickly determine potential problem areas (e.g. memory bottleneck), and then more detailed displays in the other contexts used to isolate and verify the problem at hand. The global context can also be used to monitor general system activity, in order to detect fluctuations in resource usage. The CPU and disc I/O information summarizes the activity for both the current and overall intervals, whereas the memory and process information describes the situation at the time of the update.

Memory Context

The memory context consists of eight displays, and provides information related to the usage of main memory and segment sizes. Three of the displays provide information related to the current contents of memory and the remaining displays consist of histograms depicting distributions of segment sizes or free areas in memory. The highest level display concerned with the contents of memory allows the user to determine the current percentage of memory containing code segments, stacks, and extra data segments. Additionally, the user can determine the type of code and data segments in memory. For example, the user can determine the percentage of the extra data segment memory usage that is due to IMAGE/3000, KSAM/3000, the file system, or system tables. Likewise, code segment memory usage is separated into segments originating from program files, and those from segmented libraries.

The user is also able to generate an image of the current contents of main memory, either for all of memory or for a single bank (64K words). This image indicates the type of each segment (e.g. file system data segment, stack, program file code segment), the approximate size of the segment (in either 1K or 64 word increments), and other miscellaneous information about the segment (e.g. is it locked or frozen?, is it an overlay candidate?). These images are generated by utilizing the display enhancement capabilities of the HP 264x series of terminals, and consist of a sequence of alternating white and gray rectangles (generated using inverse video and half-bright). Each rectangle represents an individual segment.

The histogram displays depict the distribution of segment sizes, in either 1K or 512-word increments. The highest level display depicts separate distributions for code, stack, and extra data segments. The remaining four histogram displays generate higher resolution histograms for each of the above three segment types, plus one for free areas in memory. The histograms are generated using the line drawing character set of the terminal, so as to provide maximum resolution.

CPU/Memory Manager Context

The CPU/memory manager context includes three displays with information related to CPU usage and memory management activity. The highest level display provides information about both CPU and memory management activity, while the remaining two displays provide more detailed information about each of these areas. All information provided in this context is of the interval summary class, with information for both the current and overall intervals.

The CPU information provided allows the user to determine the percentage of time the CPU is in various states, as well as the rate at which processes are being allowed to execute in the CPU. The reported CPU states include CPU busy executing processes, CPU time for memory management, CPU time on background memory "garbage collection", CPU on overhead processing (e.g. handling interrupts, dispatcher time), CPU waiting for user disc I/O to complete, CPU waiting for memory management I/O to complete, and CPU idle. Information for process launches and process preemptions is reported as the number of occurrences of the event per second (i.e. as a rate). Reported rates include the current interval, the overall interval, and the maximum rate observed in a single interval since the start of the overall interval. These three rates are depicted with a one-line bar on the terminal screen, utilizing inverse video and half-bright inverse video to indicate the current and maximum rates, plus an asterisk to denote the mean rate over all intervals.

Event rate information is also reported for memory management activity in this context. The events reported include memory allocation, memory management disc I/O write, memory management disc I/O read, release code segment from memory, and release data segment from memory. Information is also available concerning how the memory manager satisfies requests for absent segments. When a segment absence fault occurs in MPE IV, the algorithm used by the memory management routines can terminate with one of five possible outcomes, ranging from recovering the segment from the list of overlay candidates to temporarily postponing the request to avoid thrashing. OPT/3000 shows the percentage of memory allocation attempts terminating with each of the five outcomes. These percentages are shown for both the current and overall intervals, utilizing a bar with alternating white and gray areas.

I/O Context

The I/O context provides four displays regarding I/O completion rates for discs, line printers, and magnetic tapes. The highest level display indicates the I/O completion rate per second for each type of device, for both the current and overall intervals. The remaining three displays provide more detailed information about individual devices within each device category. These displays indicate the completion rate for three types of I/O operations (read, write, and control) on each individual device. This information can be used to determine if the I/O traffic is balanced between the devices on the system, or to identify times of peak activity.

Process Context

The process context includes four displays with information concerning process and program activity on the system at the time the display is updated. The highest level display provides information about all active or allocated programs. This information includes the fully-qualified program file name, the size of the program file in words, the number of segments in the program, the number of current users of the program, and limited working set information.

Once the above display has been generated, a second level display can be used to determine more detailed information about each process sharing a program file (or for system processes or command interpreter processes). The more detailed information includes the user name and account of the user, the process number (PIN), the size of the process stack in words, the CPU time used by the process, the number of open files and extra data segments, and the job/session number.

Additional information about a specific process can then be obtained by generating a third level display. This display contains all of the information present in the second level display, plus more detailed information about how the process is utilizing its stack space (e.g. the size of the DL area, size of the global data area). Also included are the names of all open files, a list of son processes, and a list of explicitly obtained extra data segments and their sizes.

Some of the information reported in the second and third level displays could be used to circumvent the security aspects of MPE. For this reason, these two displays cannot be generated by all users of OPT/3000. The security provisions within OPT/3000 allow a user with either system manager (SM) or operator (OP) capability to generate these two displays for any program file. Any other user can only generate the displays for a program file if they are the creator of the file, or are the account manager for the account in which the program file resides.

The remaining display in the process context provides information about the number of processes in various states. For example, the total number of processes waiting for blocked I/O, number of processes waiting for RINs, and the number of processes in the dispatch queue are reported. This information indicates the state at the time the display is updated, and no averages or totals over time are reported.

System Tables Context

The system tables context contains two displays indicating the current and maximum utilization of configurable system tables. One display provides only the current and maximum utilizations in a graphical format, using inverse video and half-bright inverse video bars. For almost all tables reported, the maximums are for the time since the last system warmstart. For the remaining tables, the maximum is that observed by OPT/3000. The second display provides more detailed information in a tabular format. This detailed information includes the configured number of entries, entry size, and maximum utilization observed by OPT/3000, as well as the current and maximum table utilizations.

MEASUREMENT TECHNIQUES

The HP On-line Performance Tool obtains the information used to generate its displays from two basic sources. The first source is the new internal measurement interface facility within MPE IV, and the second is internal MPE data structures and tables. The measurement interface provides all information related to CPU usage, memory management activity, and I/O traffic. All other information reported by OPT/3000 is obtained by examining internal MPE tables and data structures.

The measurement interface facility in MPE IV provides OPT/3000 with a formal mechanism for accessing instrumentation within MPE IV. When the facility is enabled by OPT/3000, the measurement interface obtains an extra data segment to be used as a set of counters. This segment is then locked and frozen in memory and its location stored in a global cell. As events occur, the appropriate counters within the extra data segment are incremented by code within MPE IV, and accessed in a consistent manner by OPT/3000. The CPU state time information is maintained in a similar fashion. OPT/3000 determines the activity during an interval by comparing the current sample to the previous sample, and computing the change in each counter. A count of the number of processes that have activated the interface is maintained by MPE IV. When the count falls to zero, the extra data segment is released and the instrumentation disabled. This mechanism allows multiple copies of OPT/3000 to use the same shared instrumentation. As MPE continues to evolve, both the measurement interface facility and OPT/3000 will be modified to reflect any changes within MPE.

The overhead within MPE for maintaining the counters has been determined to be approximately 0.3 to 0.8 percent of available CPU time, depending upon the amount of activity within the system. OPT/3000 can collect data from the extra data segment, and update all of its internal totals with the change in each counter in approximately 40 milliseconds. As can be seen from this data, the measurement interface facility provides a very low overhead method for obtaining performance information.

All other information reported by OPT/3000 must be obtained by examining internal MPE data structures and tables. The information concerned with the current contents of main memory is obtained by scanning all of memory, examining each region and sub-region header (these are similar to the memory links in MPE III). The segment size histograms are produced by processing the segment tables. The information concerning program files and processes is obtained by examining the loader segment table directory, process control block table, and the process control block extension area in the stack of a process. System table utilization information is partially obtained by examining information maintained in the header portion of each table.

The overhead required to gather any of the information just mentioned varies depending upon the system configuration. In general, the CPU time required to collect the necessary information and update any display ranges from 300 to 800 milliseconds, depending upon the display.

This normally translates into a total CPU overhead for OPT/3000 ranging from 1 to 3 percent of available CPU time, depending upon the displays generated and the frequency of display updates. An update interval of 15 seconds is the default used, and results in overhead in the 1 to 2 percent range.

SUMMARY

The HP On-line Performance Tool is part of Hewlett-Packard's integrated approach towards offering HP 3000 users alternatives in performance measurement analysis. In addition to OPT/3000, the first HP 3000 software performance measurement product, a new System Performance Evaluation package and a new MPE Internals and System Performance Analysis course are being offered for HP 3000 users.

The recently introduced HP 3000 System Performance Evaluation Consulting package offers an alternative to OPT/3000 for HP 3000 users who want system performance analysis conducted by HP Performance Specialists. These Specialists have in-depth training on the internals of MPE and on the performance characteristics of the HP 3000. They also have a number of HP-supplied software tools at their disposal, such as OPT/3000, IOSTAT, and the MPE IV Data Collection Program (MPEDCP), for collecting and analyzing performance measurement information on the HP 3000.

A new MPE Internals and System Performance Analysis training class is being offered in conjunction with the HP On-line Performance Tool. The first part of the course discusses the areas of MPE IV that are necessary for understanding the performance measurement information presented in OPT/3000, in particular, the new MPE IV memory manager, the dispatcher, scheduler and I/O areas in MPE IV, and the process structures. The second part of the course reviews the inter-relationships of the performance measurement variables discussed in the first part, and presents operational guidelines for OPT/3000. In addition, case study workshops will be used to share HP Performance Specialist techniques and experiences with class participants.

HP's Manufacturing Software: Engineering Feedback

By:

Barry Kurtz, H.P.

Nancy Federman, H.P.

Bob Steiner, H.P.

Engineers will have a discussion agenda for the session

Saving the Precious Resource -- Disc Accesses

by
Jim Kramer
HP3000 Performance Specialist
Hewlett-Packard Co.
St. Louis, Missouri

I. Introduction

Disc accesses is, along with CPU and memory, one of the three major resources on a computer system.

On a 3000 it is the most likely of the three to be the critical resource -- the one in short supply. This is because the 3000 is used primarily for business processing, which is usually I/O intensive.

This paper presents some ways of saving this resource when it is appropriate to do so.

II. What is a Disc Access

By disc access I mean the entire process of reading a block of data from disc or writing it to disc. This includes such activities as software setup, disc head seek to the required track, wait for the required block to pass beneath the disc head (latency), and transfer of data between disc and main memory.

As a rule the longest part of a disc access is the seek; the following figures which pertain to an average access on an HP 7925 disc drive show why:

Software setup	--	small value -- varies with CPU
Seek	--	25.0 ms
Latency	--	11.1 ms
Transfer time	--	1.4 ms/kbyte
Total	--	about 40 ms

The total of about 40 ms is what leads to the rule of thumb that the 3000 can maintain an average of about 25 disc accesses per second. It is important to notice the assumptions behind such a statement, however. One assumption is that only one disc drive at a time is active, a situation that may not hold under MPE IV for systems with multiple discs. Another assumption is that an average access takes takes 40 ms, whereas the true average depends on the actual distribution of data and the sequence in which it is accessed. This assumption is probably close to being true for a time-sharing multi-user environment because of the randomness of access in such an environment, but unlikely to be true for a stand-alone batch environment.

III. What Resource is Critical

Before any optimization is done to save accessess it should be determined that accesses is indeed the limiting resource. The reason is that most techniques of saving accesses are trade-offs -- their price is the increased use of some other resource (usually main memory). If that other resource is the limiting resource then saving accesses may actually undermine performance.

Unfortunately there is no easy method to determine the limiting resource other than to purchase performance consulting from HP. An HP performance specialist has access to measurement tools which are not normally available to customers. If this is done, great care should be taken to assure that measurements are taken during a period which is typical of the problem to be solved.

There are some tools available to customers which could probably be used to do a fairly reliable diagnosis by someone experienced in their use. I will mention these tools here, but a discussion of their use is beyond the scope of this paper.

Some tools are: the busy lights on the disc drives; the current instruction register on the Series I, II, and III; the status display on the console of Series 30's and 33's; the SHOWQ command; the contributed program SOO; the log files.

One tool which should be mentioned specifically is the contributed library program FILERPT written by Chuck Storla, an SE from the Rolling Meadows office. FILERPT can show which files are being accessed most and therefore where optimization should begin once it has been decided to optimize for disc accesses.

IV. Some Design Suggestions

A few things should be kept in mind during the design stage of an application.

Load balancing is often an effective way to alleviate performance problems. The idea is to redistribute the work load of the computer system to reduce the load during the problem period.

One form of load balancing in a transaction processing environment is the technique of batch updating -- collecting transactions into a batch for updating of the master files at a later time by a batch program. This technique has one major disadvantage -- the data in the master file is not completely up to date. Among the many advantages are:

1. The high-overhead posting operations can be done at a non-peak period.

2. The sharing of the master files for transaction processing is read only -- a more efficient sharing environment than update (no locking is required).

3. Transaction logging and recovery is generally a much simpler problem.

Another major design consideration is the amount of structure -- keys, sort items, etc. -- to use in files. Structure requires considerable machine work to establish during the writing of data, and justifies itself only by the work it saves during reading. Therefore the file must be relatively static, i.e. there must be more reading than writing of it, with respect to whatever structure it has. In terms of a catch phrase:

"A Static File Supports More Structure".

Every piece of proposed structure should be examined in this light. Further, when balancing work saved against work expended it must be considered whether either the expended or saved work is during a peak period. For example a key intended to save work during batch reporting may not be justified only because it adds to the peak load during daytime transaction processing. Or a sort item may justify itself by speeding up transaction processing even though it adds significantly to the nightly batch load.

V. Increased Blocking

General Discussion

Using a larger blocking factor is a simple and effective technique for reducing accesses if access to the file has "good locality", i.e. if the next record required is likely to be near within the file to the one just fetched. This is true when the file is being accessed sequentially, and rarely true otherwise. With MPE files it is obvious when access is physically sequential; we will discuss below when access is sequential for Image and KSAM.

It is hard to over-emphasize the importance of the technique of using larger blocks: it is generally very easy and very effective. The only price paid is increased memory requirements for buffering the larger blocks.

With MPE files it is usually possible to double the block size -- thereby cutting disc accesses in half -- without paying a memory price. This is done just by reducing the number of buffers from two (the default) to one by putting "BUF=1" on the file equation. The function of the second buffer -- allowing overlap of processing and I/O -- is usually obviated by MPE's multiprogramming environment. Even if it is not, halving the number of accesses should be a greater benefit than the second buffer.

Examples of Sequential Access

Sequential access is very common, and thus the opportunities for applying this technique are many. Examples of sequential access are: most batch applications; sorting; the text and keep operations of file editing; source program compilation; and MPE's reading of UDC files at logon and SETCATALOG time.

Image

Image does not allow control over the blocking of each data set separately -- the BLOCKMAX parameter of DBSCHEMA sets blocking for all data sets. Therefore it may be difficult to take good advantage of the technique. However if it is determined that a large proportion of access to a data base is physically sequential, it may be worthwhile to increase BLOCKMAX above the default.

In Image the most important instance of sequential access is backward or forward serial access to a data set. If the data base has been reorganized by doing a chained unload followed by a load, then chained access on primary keys is also sequential. If there are a large number of secondaries in a master, then accesses can be reduced by larger blocks; however it is probably more appropriate to investigate why the hashing scheme is not working well.

KSAM Data Files

Chronological access to a KSAM file, including addition of new data to the file, is by definition physically sequential. A very important mode of KSAM access -- keyed sequential -- will be physically sequential if the file was originally loaded in key sequence. Because of this, accesses can be saved during keyed sequential access by using large data blocks and loading in key sequence.

KSAM versus Image

If a KSAM file is being used only for keyed access, not keyed sequential, Image should probably be used instead. This is because Image's hashing techniques generally fetch a record in a single access, whereas KSAM's B-tree technique requires a search through the multi-level key tree before the data can be read. (On the other hand KSAM is very fast for keyed sequential access which Image cannot do at all).

KSAM Key Files

KSAM key files can benefit from larger blocks. Perhaps surprisingly the greatest benefit is for random keyed access rather than keyed sequential. The reason is that keyed sequential access is very sparing in its use of the key file regardless of key blocking.

However the opposite is true for random keyed access. Each fetch of a record by key requires a top to bottom search through the key tree. This results in H-1 disc accesses, where H is the height of the tree. (Unless there is locking, the root block does not have to be read -- it is already in the buffers). The fetch of the data record requires another disc access, making H accesses the total number required.

The possible benefit of larger key blocks is that the height of the tree may be reduced. A reduction of tree height from three to two would save one third of the accesses.

Altough it is possible to calculate whether a larger block will actually reduce tree height, it is probably just as simple in most cases to simply try it. The KSAM manual discusses how to specify key block factors.

VI. Increased Buffering

General Discussion

Increased buffering is of doubtful value in a sequential access environment, but can under certain circumstances be very helpful in a random access environment. These circumstances are that there are repeated accesses to the same blocks of the file and these blocks can be held in memory. As with increasing block size, the price paid for increasing buffering is main memory.

It some situations it may be possible to use MPE's buffering to implement this technique. Suppose for example that an application makes thousands of accesses to a file which is only 100 records long, each record being 100 bytes long. The total length of the file is only 10,000 bytes. Since an MPE buffer can be as large as 16K bytes (8K words) it is possible to hold the entire file in a buffer. It doesn't make too much difference how the file is blocked, but it may as well be blocked 100 -- the entire file will be brought into the buffers with a single access, and that is the only access required for the entire run.

The above example is an extreme case, but the principle applies for less extreme cases as well. Suppose that the file were 50000 bytes long -- five times as long as the previous file. Now only about one third of the file can be held in the buffers at one time, but this is still worthwhile: the chances of finding the required block in the buffers is one out of three, and we reduce our disc accesses by one third.

In situations where not all the data in each record is needed, it might be worthwhile to create a new file with smaller records, just so that a larger percentage of the file can be kept in buffers.

It is not necessary to accept MPE's restriction of a 16K buffer segment. A user can do his own buffering with extra data segments and fill as much of main memory as desired with blocks from his file. This requires some sophisticated programming but it might be the solution to some difficult performance problems.

The Effects of Locking

It is very important to keep in mind that for both MPE and KSAM files file locking can negate the value of buffering entirely and drastically increase disc access requirements. The reason is that for these file types (but not Image), each user has his own set of file buffers. To assure that all users have an accurate view of the file, it is neccesary that a file lock cause clearing of the user's buffers (forcing him to go to the file) and that an unlock post any changed blocks to the file.

It is best to avoid sharing MPE and KSAM files at all, unless access is read only for all users. Otherwise locking is required for proper sharing of the file. If a locking environment cannot be avoided, its harmful effects can be minimized by doing as many operations as possible between each lock and unlock.

Buffering and Image

Image performance can be helped by increased buffering in certain circumstances. An example is repeated random access to a master which is sufficiently small to allow a signficant part of it to be held in memory. Interestingly perhaps, sequential access can also be helped by additional buffers if they prevent the buffer containing the sequentially accessed block from being overwritten by another block.

In general it is very difficult to know how many buffers is "right" for an Image application. Probably the best advice is to try different numbers of buffers and check the effects on disc accesses by observing the file close log records.

For a batch application the right number is generally the maximum (specify 255). This increases the memory requirements for the batch job but gets it out of the way sooner. In many cases batch jobs are run at night when there is not much competition for memory anyway. Batch jobs which do posting should generally run with buffer posting deferred (invoked by calling DBCONTROL with mode = 1). This causes Image to postpone the posting of a block in a buffer until the buffer is needed for some other block. In many cases the reduction in the number of disc accesses is dramatic. If the system should fail during the running of the batch job, the data base is almost guaranteed to be invalid -- it will be necessary to restore the data base and rerun the job in its entirety. Of course this is the usual tactic for any failure of a batch job.

DBLOAD, by the way, will by default run with the maximum number of buffers and with buffer posting deferred.

Buffering and KSAM

The buffers for both the key and data files of a KSAM file are in a single data segment. There is always exactly one buffer for the data file, but the user has control over the number of key block buffers. KSAM allows as many as 20 key block buffers, but if key blocks are close to the maximum allowed length of 4K bytes, the maximum number of buffers allowed may be less than this -- as few as 15. Please note that to specify the number of buffers with a file equation, the DEV= keyword is used; the parameter is the one which specifies number of copies when writing to a spooled device. Thus to specify 5 key block buffers, add ";DEV=,,5" to your file equation. (I know this sounds strange, but what reason would I have to lie about it?).

Increased buffering has little if any value for keyed sequential access but may reduce disc accesses dramatically for random keyed access if a significant proportion of the key file can be held in the buffers (assuming no locking, of course). This can be determined easily from the KEYINFO command of KSAMUTIL, which tells the number of blocks in the key file.

VII. Increasing the Resource

All of the above suggestions have been oriented toward reducing the demand for disc accesses. It is often possible as well to increase the supply of them, by reducing the time it takes to do them, in particular by reducing seek time.

The easiest tactic is a reload. This consolidates the files to one edge of the disc and brings the extents of a file together, thus reducing average seek times.

Another major tactic is to reduce head movement be avoiding the situation in which a head is forced to move back and forth between two or more files on the same disc. Here are some suggestions based on this approach:

1. Avoid putting any frequently accessed files (including spool files) on the system disc, which contains the directory and the swapping area, both of which are frequently accessed.

2. Keep the input and output files of batch activities (including sorts) on separate discs.

3. Keep heavily accessed data sets from a data base on separate discs, especially if they tend to be accessed together.

4. Keeping the key and data file of a KSAM file on separate discs.

TERMINAL I/O INTERFACE--
AN ENGINEERING FEEDBACK SESSION

Presentors
Jim Beetem, R&D Project Mgr., HP

Hewlett-Packard is currently accessing user needs and problems in the area of MPE interface to terminals; and is seeking ideas and suggestions. This area of interest includes any device interfaced to the HP 3000 through terminal controllers.

In the first part of the session, results of the recent joint HP/Users Group "Membership Questionnaires 1981" will be presented for confirmation for additional comment. In the second half, needs, plus problems not suggested by the questionnaire, will be solicited. Finally, all suggestions will be prioritized.

VIP/3000 AND VIP-TNC/3000

by
Earl E. Colmer, Jr.
Automated Sciences Group, Inc.

Automated Sciences Group, Inc.
700 Roeder Road
Silver Spring, Maryland 20910
301/587-8750

VIEW IMAGE PROCESS 3000

VIP is a completely Screen Driven Multi-Terminal, Multi-Data Base, Multi-Data Set, Multi-Screen View Image 3000 process. VIP needs only a Data Base Information file and the user defined View Screen, thus paying for itself in one application, saving you as much as 90% implementation time of your on-line Data Base Application. VIP was developed to be machine efficient, data and code stack efficient, and completely user oriented. There are no extensive forms to fill out, VIP will prompt the user via an interactive Data Base Menu Screen of the Data Base he/she wishes to access. Once the Data Base has been established VIP will prompt the user again via an interactive Data Set Menu Screen of the data set you wish to ADD, DELETE, INQUIRE, or UPDATE.

Data Screen Technicians do not have to spend valuable time editing numeric data in View/3000, since VIP automatically right justifies numeric data and makes the necessary numeric data checks (depending upon the Data Base data type). VIP offers you a choice of editing procedures from the SL of your choice or program creation and activation of edit programs. Complete Data Base, Data Set, and Data Item Security is a function of VIP alone with user option DBLOGGING and momentary DBLOCKING and DBUNLOCKING. VIP allows the user chain reads, time out serial reads, and implied and's of the entire View Screen.

VIP's Data Base Information File format;

Record 1.	Fully qualified Data Base name.
Record 2.	Fully qualified View File name.
Record 3.	Data Set, Primary Key for this Data Set.
Record 4.	View Form for previous Data set, View Options.
Record 5.	etc. -NEXT DATA SET DESCRIPTION-
Record 6.	etc. -CORRESPONDING VIEW FORM & OPTIONS-

VIEW IMAGE PROCESS/3000 TERMINAL NETWORK CONTROLLER

VIP-TNC is a Terminal Network Controller especially designed to control VIP. VIP-TNC will open up as many terminals as your application desires, without logging users on the specified terminals, thus saving system overhead and virtual memory on each VIP process. It is especially useful for on-line registrations, order entry, or any other transaction processing applications. No View Image Process can be 100% efficient for every on-line application, but due to the Modular Design and SPL Compiler IF Statements VIP can be tailored to your specific on-line Data Base Application in a matter of minutes, thus making each process as efficient as possible.

Under ASG's Monthly Maintenance Contract, VIP alterations will be done for only the cost of a tape, plus postage and handling. Without the Monthly Maintenance Contract, there will be a minimum charge.

REAL-TIME, ON-LINE, DISTRIBUTED IN THE MANUFACTURING SYSTEMS ENVIRONMENT

Presentors

Lee R. Kneppelt, Vice President, Arista
Jerry L. Sneed, Programmer Analyst, Arista

The terms real-time and on-line have become as popular as MRP within the manufacturing user community. In many cases, Data Processing will respond with complex hardware/software solutions to implement manufacturing application functions in a real-time mode. Often, this is done with little regard to whether an application function needs to provide instant turnaround. The manufacturing planning and control functions can be broken into strategic, tactical and action functions for guidelines on user turnaround requirements.

The approach is to review the major manufacturing systems (Master Scheduling, Material Requirements Planning, Capacity Requirements Planning, Manufacturing Standards, Inventory Control, and Shop Floor Control) with respect to a break out of planning verses control functions and their use within the time zones of manufacturing cycle. The result is a blueprint for what functions are enhanced by real-time processing as well as what system-wide rules can lead to a distributed systems approach based on application concepts.

ALTER/3000

QUICK MODIFICATION PROGRAM

by:

Earl E. Colmer, Jr.

Automated Sciences Group, Inc.

Automated Sciences Group, Inc.
700 Roeder Road
Silver Spring, Maryland 20910
301/587-8750

ALTER/3000 - QUICK MODIFICATION PROGRAM

ALTER/3000 is a high speed multi-function EDITOR which can be used to modify existing MPE files. Unlike most EDITORS, ALTER/3000 (Quick Modification) can be used to modify any file including IMAGE root files and datasets. Rather than the standard 'text into workfile' approach, ALTER/3000 uses direct disc access routines to inquire or modify files, thereby saving file space and CPU time. Any MPE file can be accessed in less than 300 milli-seconds, onced OPENED any record can be accessed in less than 100 milli-seconds.

Unlike EDIT/3000, ALTER/3000 can accommodate files with record lengths exceeding 256 bytes. When printing such a file to the printer, no data will be striped off or lost. ALTER/3000 also contains an optional logging facility which maintains a record of how many times each user has accessed ALTER/3000.

Since ALTER/3000 does not check the file code for files, ALTER can be used to modify files that the EDITOR will not handle: i.e., program file, VIEW files, and QUERY procedure files. ALTER has 38 commands in its command interpreter, including CLOSE, EJECT, REDO, PRINT, and SOFTKEY. ALTER/3000 also supports Editin, Editout/Editlist, and Use files, with one major advantage of an Editin file, if your last command of an Editin file is 'USE $STDINX', control will be returned to your terminal instead of program termination.

When ordering ALTER/3000, please specify;

(1) Maximum anticipated record width (in bytes)?

(2) Number of 'LINE' printers on your system?

(3) Give 'MANAGERS' the ability to open PRIVILEGED FILES?

(4) Give 'MANAGERS' the ability to run in any SUBQUE?

(5) Enable ALTER/3000 user logging?

(6) Machine HP-?

E-ZV THE EASY WAY
TO USE V/3000

By:

David Kalman

Gentry, Inc.

Paper to be presented

at Conference

DIALOGUER/3000: FORMS MANAGEMENT FOR EVERY SCREEN

Presentor

Dr. Joseph Vignalou, President, HP

DIALOGUER/3000 extends the forms management concept to virtually any kind of screen terminal, from smartest to dumbest.
After an overview of the various terminal supported, the presentation concentrates on the terminal operator aspects - response time, formatted output, field echo, default values, screen refresh and user commands - as well as the programming aspects - form and field definition, editing routines, form stacking, procedure calls and debugging aids.

TRACS 3000
TIME AND RESOURCE ACCOUNTING SYSTEM

by:
Arthur Sera
Automated Sciences Group, Inc.

Automated Sciences Group, Inc.
700 Roeder Road
Silver Spring, Maryland 20910
301/587-8750

TRACS 3000
TIME AND RESOURCE ACCOUNTING SYSTEM

The TRACS system is composed of 3 data bases and 17 programs. The TRACS system utilizes the system log files, reports command output, and a subque logging file as input to record resource utilization on the HP 3000.

The SCHED data base is used by the job scheduler subsystem to store job submission requests. The job scheduling system is a general purpose system which is capable of submitting jobs: once, daily, weekly, monthly, or yearly. The TRACS system utilizes this capacity to run the nightly posting of the days activities and the month end transfer, posting, and reporting programs. The job scheduling system includes a data entry program to ease use.

The LOGDB data base is used to store the system log file information in a structure which permits the association of resource utilization with the originating groups and accounts. The information retained includes CPU and CONNECT times by logon subque (i.e., BS, CS, DS, ES) distributed between time consumed during prime vs. non-prime times. Also retained are disc, tape, card, terminal, and printer I/O's, again, distinguishing prime and non-prime times, cumulative disc storage, and a variety of memory usage information. Memory usage information ranges from code and data stack words through virtual memory sectors utilized again recorded as prime and non-prime times usage.

This data base additionally maintains daily summary records of total system usage and contains the billing rates to be charged for the above mentioned resources. There are three levels of rate assignments available: (1) system default rates (used in absence of other explicit rates), (2) account default rates (used in absence of explicit group rates), and (3) group rates (defined as rates for particular ACCOUNT.GROUP combinations). The LOGDB data base also records I/O errors, log errors, console messages, and line closes.

The BILLS data base contains customer information. Included in this are customer name and addresses, customer/account associations, and the monthly invoice detail lines. The BILLS data structure allows for simple, straight forward definition of a customer's account structure for billing purposes. This makes possible the assignment of one entire account plus only one group of another account to an individual customer with just two entries.

The invoice posting program runs monthly and posts the detail charges for total CPU, CONNECT, MEMORY, DISC STORAGE, and I/O charges accumulated for each individual customer. Additional detail lines for other categories of charges may be entered with QUERY and will therefore be included in the printed invoice by the invoice printing program.

Budgeting and Profit Planning on the HP 3000

Presentor

Jack Damm, Principal, HP

Budgeting and Profit Planning on the HP 3000 is a presentation about how the HP 3000 can be used as a powerful tool for company planning. The talk focuses on a particular "model" which we use in our consulting business to help companies do their financial planning. The presentation includes many of the experiences we have had in 10 years of business planning using the computer.

QUASAR, INC.

THE GREAT COMPANION (QUIZ)

BY:

MARSHALL WARWARUK

Paper To Be Presented

At Conference

Implementing Manufacturing Systems/Diffulty of Task

Presentor

Lee R. Kneppelt, Vice President, Arista

Jerry L. Sneed, Programmer Analyst, Arista

A few years ago APICS had a modern-day crusade on MRP which was followed by a few years with the theme "back to the basics". Today, the number of companies attempting to upgrade their manufacturing systems is greater than ever before. Manufacturing Resource Planning (MRP II) has been coined as the latest term for systems to support the management of manufacturing systems. Yet, the number of successful installations of techniques such as planning bills, master production scheduling and MRP still remains relatively small as compared with the potential.

The problem may be the result of not having a basic understanding of how to plan and execute a project of the scope of manufacturing systems implementation. It does not take long to realize that in implementing manufacturing systems you are digging around in the very heart and soul of management. The data is really the raw material for management decision making and for evaluation of their performance. Taken lightly, the implementation can be an exceedingly dangerous pastime.

Areas of concern in management of an implementation include objectives, realism, organization, cost/benefit, transition and training. Excuses for failure include changing specifications, user did not know what he wanted, lack of commitment by top management, vendor slippage of software ana hardware, or we wanted a bicycle and built a Cadillac. What the excuses really address is the lack of project plan, defined scope, meaningful reviews, the "not invented her" syndrome, and maybe, courage to change. Often, it is the violation of good implementation planning and execution which negates the impact of sound production and inventory control techniques.

LOGOFF/3000

by:

Earl E. Colmer, Jr.

Automated Sciences Group, Inc.

Automated Sciences Group, Inc.
700 Roeder Road
Silver Spring, Maryland 20910
301/587-8750

LOGOFF/3000

Unlike the big mainframes, and many other minis, Hewlett Packard does not have the capability to logoff jobs or sessions that have been inactive for a given period of time. This causes such problems as inaccurate bills for time sharing customers, Data Processing students running out of connect time, and of course your favorite MPE error message 'UNABLE TO OBTAIN VIRTUAL MEMORY.'.

LOGOFF/3000 can be tailored to your individual site at run time. The Systems Manager can specify the number of CPU seconds to be checked at every suspend interval. That is, activate the program once every interval (say in this case 60 minutes) and check to see if the job or session has exceeded the check CPU time (in this case 5 seconds). With these parameters every job or session that has not exceeded 5 CPU seconds in 60 minutes would be logged off. LOGOFF/3000 also allows you five entries in an Identification Table. One such entry would look like this ',MANAGER.SYS,', which means do not logoff MANAGER.SYS from the system no matter what his CPU time might be. Another such entry would be ',.sys,' which means do not logoff anybody from the SYS account. All jobs or sessions are checked against the system, if a process associated with that job or session has an outstanding request such as UCOP, RIT, etc. that job or session will not be logged off.

All jobs or sessions logged off will have an entry put into a log file (ABORTLOG.PUB.SYS). A report program is provided (LISTALOG.PUB.SYS) to report on all entries put into this file. Information retrieved from this file is as follows;

> jobname, user, account, group logon date & time, logoff date & time, last CPU check, this CPU check, check CPU time, and program suspend time.

LOGOFF/3000 can be run at a session (not advisable), streamed as a job, or created and activated by a process handler. System requirements are minimal, the program runs with an 800 word stack (128 word DL MINUS included!), and uses approximately .2 of a CPU second every hour, (assuming that your suspend time is 60 minutes).

REX/3000 AS A PROGRAMMER PRODUCTIVITY TOOL

Lance Carnes

Consultant

Mill Valley, CA

March 1981

ABSTRACT.

Programmer productivity is an important issue today: programming effort is the single largest factor in the design, implementation and maintenance of software systems. This paper surveys the major aspects of productivity: reducing programming effort, increasing program reliability, providing run-time efficiency, and reducing the cost of software production. Each of these aspects and its relationship to the others is explored. A unique high-level language, REX/3000 is recommended as a solution to boosting programmer productivity. REX/3000 programs are used to illustrate points made.

I. INTRODUCTION.

As the costs of labor and money increase, so do the pressures to control and reduce operating costs. In data processing departments programming personnel costs are the single largest cost factor. Organizations which have dealt with this problem attribute much of their success to the use of productivity tools [1].

Productivity tools are designed to reduce the time and cost required to produce software. The project manager can implement software more economically and with more efficient use of personnel through appropriate use of these tools. Equally as important is the user satisfaction with the software product developed using the productivity tool.

The interest in increasing programmer productivity is evident from the number of articles in trade journals and conference proceedings dealing with the subject. Most software departments have a larger backlog of programming requests than they have staff to do the work. Of the time spent in programming, typically 60% to 70% is involved with maintaining existing systems, while only 30% to 40% is utilized in new development. The cost of a person-month of programming effort is high and will continue to increase. New hardware is being developed faster than the software it will run.

Typically, productivity tools meet the traditional productivity requirement: decrease the amount of code required to implement the system and, therefore, reduce the programming effort. This is an obvious path to take and has proven successful. However, the result is often that with the reduced effort comes a reduction in the quality of the software. These tools are frequently specialized for certain applications and have limited scopes; once the limits are exceeded, the application must be redone using a different, usually less reliable and less productive, method. The system produced with the productivity tool is often less efficient than the same product implemented using standard methods.

REX/3000 is a high-level language and compiling system designed to meet the requirements of increased productivity. It has specialized constructs for report writing which result in a 50% to 90% reduction of effort over using general-purpose languages. The language was designed to encourage structured programming and thereby increase program reliability and correctness. As a compiling system it generates efficient program modules. There is sufficient scope in the range of applications which can be implemented so that it is rare that programs must be redone using more general-purpose languages.

In Section II the concept of productivity is defined and expanded. Section III is a brief introduction to the REX/3000 language. Section IV examines productivity quantitatively, i.e. reducing the effort to produce software. Section V deals with productivity qualitatively, i.e. maintaining reliability and efficiency. Section VI summarizes the cost savings realized with reduced quantity and improved quality. Section VII is a summary of the points made.

II. WHAT IS PRODUCTIVITY?

Traditionally, programmer productivity is the rate of software production, i.e.

$$\frac{\text{\# lines of code}}{\text{person-months}}$$

This ratio is derived by taking the total number of lines of code required to produce a software system and dividing by the total personnel time used. The total lines of code may refer to the final code put into production or it may be all code written, e.g. for documentation, test programs, discarded modules, etc. The total personnel time may refer only to actual coding time or may include all time spent in design, training, travel etc. This ratio can be used for predicting the effort which will be required to produce future similar systems.

This ratio has limited usefulness because it indicates only the rate of code production and tells us nothing of the total time or cost of developing a system. For example, a system which could be developed in 24 months using 20,000 lines of assembler code has the same productivity rate as if it were developed in 12 months using 10,000 lines of COBOL code. While the rates are the same, the total time and costs are doubled.

In recent literature, less importance is being given to the quantity of software produced and more consideration is being given to the quality of software [2,3]. The aspects of software reliability, correctness and efficiency are being explored. These aspects are as important as reducing programming effort, and in fact play an important part in reducing the maintenance effort. Quality can also be measured for the purpose of modelling or estimating as

$$\frac{\text{\# bugs found}}{\text{\# lines of code}} \qquad \frac{\text{actual efficiency}}{\text{expected efficiency}}$$

The first ratio measures software reliability, which may be required to fall within a certain tolerance to be considered usable software. The second ratio measures machine resources used versus the allowable or avaliable resources, and a minimum tolerance may be specified to indicate whether the software has been successfully implemented. For example, the daily production must run within a 24-hour period.

It has become apparent that too much energy has been spent on increasing productivity rates and not enough on maintaining or improving program quality. One of the main reasons for this is the dramatic cost reduction in the development phase as a result of increased productivity. However, the savings are often cancelled when the cost of maintaining the error-prone code is considered. Therefore, productivity tools must treat the issue of quality with equal importance as quantity.

In the following discussions, we will be concerned with the issue of quality as well as quantity. The relationship between increased quality and reduced effort will be covered.

III. WHAT IS REX/3000?

REX/3000 is a high-level language and compiling system useful for report writing and general data processing. It was designed to boost productivity significantly through use of a combination of special-purpose and general purpose language constructs. Special-purpose constructs, also called non-procedural constructs, are the heart of the language, allowing programs to be written quickly. The general-purpose constructs, also called procedural constructs, build onto the special-purpose program and increase the flexibility of the language.

In the following discussion, we will show how the nature of the language promotes productivity. For a more detailed treatment of the language, see [4,5].

REX can be used to develop useful programs quickly. These same programs can be expanded as requirements grow. The following example illustrates this point.

```
   << WAREHOUSE PARTS SUMMARY >>
   DATABASE parts PASSWORD "ANY" ACCESS 5
     DATASET part-stock
   REPORT
     GET parts.part-stock
       LIST whse  AS "WAREHOUSE",         &
            part# AS "PART#",             &
            qty  AS "QUANTITY"            &
         SORTED BY whse, part#            &
         SUMMARIZING qty ON whse, part#
     LOOP  << end of GET ... LOOP >>
   END.
```

This is a complete REX program, which when compiled and run will produce the report shown in the Appendix.

The sections of the program are as follows:

1) DATABASE declaration. This special-purpose construct performs the following functions:
   a) it specifies the database name, password and access mode to be used.
   b) at compile time, all attributes of the database, the datasets indicated, and the items within each of the datasets are known - the programmer need not redeclare the database layout, provide buffers, etc..
   c) at execution time, the database is opened using the parameters from (a).
   d) access to the database is available through the GET construct.
   e) at the end of the program the database is closed.

   This is a non-procedural construct, that is, it performs all of the logic necessary to access the database.
   The programmer is insulated from all mechanics of database use.

2) The REPORT block. This special-purpose construct performs all of the steps required to produce a sorted, formatted report:
   a) The items indicated in the LIST statement are read from the database and written to an extract file. The extract file is formatted and maintained by REX.
   b) At the end of the input phase, the extract file is sorted in the given sequence (SORTED BY ...).
   c) The report is now printed with the column headers (AS "...") and control breaks (ON ...) indicated.

   This is a non-procedural construct, since the mechanics of formatting the extract file, sorting it, setting up column headers, testing for control breaks, etc. are part of the REX system.

3) The GET statement. This is a special-purpose construct which reads data from the dataset and performs the following function:
   a) The dataset mentioned is read entry by entry (serially in this case, although chained access mode is also available).
   b) As each entry is read, the statements between the GET... and LOOP are executed (in this case, the LIST statement).
   c) After the last entry is read, transfer is passed to the statement following the LOOP.

   GET is a non-procedural construct in the sense that the mechanics of access are hidden from the programmer. The programmer does have to place the LOOP in the right place; if the LOOP is omitted, the compiler assumes the loop includes all statements up to the END.

This code could be written and running correctly in a matter of minutes. The user would be pleased with the results for at most two days, and then, of course, would want to expand the function to include the following:

1) Print the unit price and total value in stock for each part;
2) Place an asterisk in the column next to part #'s for which the quantity is zero;

Typically, this is beyond the scope of a non-procedural report writer. To perform the first requirement, the price will have to be extracted from a second dataset (PART-MSTR) and multiplied by the quantity. Some logic will have to be implemented to allow the quantity to be checked for zero and an asterisk inserted.

These requirements are not beyond the scope of REX, and in fact the original program may be modified to include the enhancements. The following is the REX program which will satisfy the above requirements.

```
   << WAREHOUSE PARTS SUMMARY >>
   << enhanced to print price and value  >>
   << will print an asterisk if qty = 0 >>
   DATABASE parts PASSWORD "ANY" ACCESS 5
     DATASET part-stock
   PROGVAR star A1
           value P5.2
   REPORT
     GET parts.part-stock
       IF qty = 0 THEN star = "*" ELSE star = " "
       GET parts.part-master                &
         WITH part# = parts.part-stock.part#
       LIST whse  AS "WAREHOUSE",           &
            star,                           &
            part# AS "PART#",               &
            qty  AS "QUANTITY",             &
            price AS "PRICE",               &
            value = price * qty             &
                   AS "VALUE"               &
         SORTED BY whse, part#              &
         SUMMARIZING qty ON part#,whse
     LOOP  << end of GET ... LOOP >>
END.
```

The following parts were added to the program:

1) PROGVAR declaration. A program controlled variable, star, was declared which can contain one alphaumeric character (A1). The variable value is a five-digit packed-decimal number.
2) IF THEN ELSE statement. This statement checks the qty for zero and sets the variable star accordingly.
3) GET parts.part-master WITH... . This statement accesses the master set (PART-MSTR) keyed by part# to locate the price.
4) value = price * qty. This calculation is performed to compute the total value of parts in stock.

The PROGVAR declaration, the IF THEN ELSE and the calculation are procedural constructs, that is, the programmer has to specify the mechanics of the function.

Notice that the enhancement was made by adding procedural (general-purpose) constructs into the original non-procedural (special-purpose) program. With most non-procedural report writers, the enhancement could not be made, and the application would have to be recoded using a fully procedural language (e.g. COBOL).

In summary, REX allows the creation of non-procedural programs which can be coded quickly and by less experienced staff members. In addition, enhancements and more complex programs can use the rich set of procedural constructs. The special-purpose (non-procedural) constructs and the general-purpose (procedural) constructs can be combined in the same application.

IV. REDUCING THE PROGRAMMING EFFORT.

The major emphasis of any productivity tool is to reduce the effort to produce software. That is, reduce the number of lines of code, and therefore the time, which would have been required to implement the system using a general-purpose programming language.

The use of productivity tools has proven effective [1]. The time and costs for software development have been significantly reduced using such tools, by as much as 50% to 90%.

In practice, productivity tools generally are not versatile enough to be used exclusively. This is the chief drawback to such tools making a significant impact on the software development process. Typically they are designed for a limited scope of applications and work well within these limits. Too often, the limits of the tool have the following negative effects:

1) Enhancement requests which exceed the limits of the tool are not done, denying the user timely access to useful information.
2) The corresponding general-purpose program which includes the enhancements costs so much to develop that the user will rationalize that the data is not important enough to justify the cost.

For example, consider the following application written in QUERY, a useful but limited tool:

```
DATA-BASE = PARTS
PASSWORD =>> ANY
MODE =>> 5
FIND ALL PART-STOCK.PART#
REPORT
H1,"WAREHOUSE PARTS REPORT",30
H2,"WAREHOUSE  PART#  QUANTITY",32
D,WHSE,15
D,PART#,22
D,QTY,30
S1,PART#
S2,WHSE
END
```

This code could be put into production in a short time and would provide useful information. However any enhancement requests must be looked at with the limitations of QUERY in mind.

For example, if the requests were the same as those in the example in the previous section, QUERY could not be used:

1) Print the unit price and total value in stock for each part (QUERY can access only one dataset at a time and

cannot perform multiplications);

2) Place an asterisk in the column next to part #'s for which the quantity is zero (QUERY does not have alphanumeric variables or conditional statements).

The application would have to be coded in a general-purpose language.

The COBOL program which includes the enhancements is given in the Appendix; it is in excess of 230 source lines.

The main point here is the great disparity in the sizes of the programs. QUERY has 13 lines where the same application with two minor enhancements takes nearly twenty times the number of source lines in COBOL. The cost of enhancements in this case is much greater than would be imagined, especially by the user.

REX, however, provides a reasonable solution. The enhancements mentioned require only seven additional lines of code and a few minutes of time. Furthermore, the same source code may be built upon, avoiding a rewrite in a more general-purpose language.

In summary, REX combines the features of QUERY and COBOL. The programmer can produce simple programs in a short time, and simple or complex enhancements can be made by building onto the original source.

Two additional benefits result from using productivity tools to reduce programming effort:

1) Throw-away programs become feasible.
   Code can be written for a "what if" inquiry and then discarded. This would not be possible with high development costs.
2) Maintenance effort is reduced. The effort, and therefore the cost, of correcting bugs and making enhancements is reduced. The maintenance duties can be performed by a less experienced programmer. The savings are dramatic when considering the cost of supporting several systems over an extended period of time.

V. QUALITY - MAINTAINING OR IMPROVING IT.

In the previous section we noted that a frequent problem with using productivity tools is their lack of flexibility. Two other problems are often identified:

1) While it is easy to write code, it is difficult to use structured programming disciplines or other techniques which encourage error-free, reliable code.
2) The run-time modules are inefficient, consuming far more machine resources than the equivalent program written in a general-purpose language.

These issues arise when dealing with general-purpose languages as well. The first point concerns the reliability of programs, i.e. how bug-free the programs are. The second point concerns the efficiency of the program, i.e., the amount of machine required to execute the program.

Specialized productivity tools are generally reliable. They do not have the capability of performing complicated sequences, making it difficult to introduce bugs. The reliability will be lower, however, when the tool is pressed to its limits - programmers often code 'clever' but difficult to understand programs, or use side-effects of the system to circumvent the limitations of the language. Where the language does have some procedural constructs, they are often prone to the usual logic errors found when using non-structured languages.

Reliability can be increased by 1) training the programming staff in one of the structured programming techniques and/or 2) using a programming language which encourages error-free code. The first is a common technique when a software department {_is committed to using FORTRAN or COBOL; it is usually necessary to set up careful coding guidelines and review all code produced. The second is less common, though increasing with the availability of structured languages, e.g. Pascal, JOVIAL, Ada; these languages, however, are not suited for commercial applications or report writing.

REX was designed to encourage reliable coding. The non-procedural constructs perform reliably due to the fact that their function is well-defined and not alterable by the programmer (e.g. REPORT ... LIST). The procedural constructs in REX are borrowed from PASCAL, a structured, high-level language [6]. Coding is done using constructs such as PROCEDURE and REPORT blocks, IF THEN ELSE, WHILE DO, REPEAT UNTIL and GET LOOP, etc. REX has no GOTO. In short, the programmer must work with constructs which encourage reliable coding; those constructs known to be error-prone (e.g. GOTO) are not available.

Efficiency is an important issue, since all programs must eventually run in production and produce their results in an acceptable amount of time. A program which is inefficient will not be used and must be designed and implemented again. A program which is marginally efficient, i.e. runs slowly but within an acceptable range, will be subject to many costly attempts to speed it up. A program which was easy to develop but must be tuned constantly once in production has produced no real savings.

While many specialized tools are inefficient at run-time, REX is actually as efficient or more efficient than general-purpose language systems. The main difference is that most tools are interpretive, whereas REX is a compiling system. An interpreter is a general-purpose system which has heavy demands on the machine: it is a large program which has many code segments and uses large data areas. In contrast, a compiler produces an efficient runtime module: the program and data area requirements are only a fraction of those needed by an interpretive system. Reducing code and data memory requirements can greatly improve performance [7].

REX produces efficient run-time modules, similar to those resulting from a general-purpose compiling system. Segmentation is done automatically to speed the operation of REPORT blocks - the input phase code is in one segment while the print phase code is in another. Segment switching is minimized by generating as much code inline and avoiding PCALs whenever possible. Data segment usage is kept to a minimum through efficient code generation and the use of local variables, i.e. avoid global variables [7]. Since the programs run efficiently, there is seldom a need to optimize, saving maintenance effort.

Using REX allows high quality code to be generated with little additional effort or expense. The resulting programs are easier and less costly to maintain. The benefits are efficient production programs without the effort of extensive tuning. Overall, user and programmer satisfaction will be high.

VI. HOW ARE COSTS CUT?

Whenever there is a reduction of effort, increased program reliability and dependable machine efficiency, there is a corresponding cost savings. These savings may be immediately noticable, e.g. when reducing development costs. Or they may occur over an extended period of time, e.g. in the maintenance phase of the software life cycle. In addition to the savings from reducing effort, costs can be cut through use of less experienced personnel.

These are some of the ways costs are cut using productivity tools such as REX/3000 which not only reduce programming effort but encourage high quality:

1) Higher coding productivity results in fewer person-months of effort with a direct cost savings.
2) Higher reliability and efficiency reduce the number of person-hours required for maintenance over the life of the software system.
3) Less experienced and therefore lower cost personnel can implement and maintain software systems. The more experienced staff members can devote more time to designing current

and future software systems without worrying about whether there will be time enough for implementation.

VII. SUMMARY AND CONCLUSIONS.

Productivity tools do exactly what they claim - reduce the time and cost to produce software. Those tools which also increase the quality of produced code have the additional benefits of reducing maintenance time and effort. Overall, using a productivity tool allows more careful design and planning and better personnel allocation, since the pressure of the great amount of programming effort is relieved.

The quantity of code is reduced through the use of special-purpose constructs. Where these constructs typically reduce the scope and flexibility of the language, REX/3000 has met this shortcoming by allowing general-purpose constructs to be built onto the special-purpose core of the program.

The quality of code produced by productivity tools typically is not so high as that produced by general-purpose languages. REX/3000 allows high-quality coding through the use of structured programming techniques and efficiently compiled program modules.

The features of these tools are attractive and the wise programming manager will use them to produce economical, timely systems. However those projects implemented using tools in any capacity are few in number. The overwhelming majority of software systems produced use general-purpose languages, and overall show low productivity.

The reasons for not using tools are varied: some are legitimate, e.g. machine portability requirements; most, however, are the result of the fear of using something "new", or something which appears simplistic. There is a streak of the old-time wizard in every programmer, and the fact that the non-data processing user cannot comprehend the nature of the business is comforting and even protective. Some see the use of productivity tools as a threat to this mystique. Another common reason for not using tools is the reluctance to try something other than the standard methods, unproductive as these are. With the cost of person-power increasing, the obvious move is towards increased productivity.

One observer noting the lack of use of productivity tools drew the following analogy:

> [They] are so busy digging ditches with pick and shovel that they haven't the time to go watch the bulldozer demonstration [8].

With the cost of manpower increasing, it is imperative that tools be used in the near future. Those managers who cannot control costs and time schedules because of low programmer productivity will have to compete with managers who can make a difference. Productivity tools, like REX/3000, will play a major role in making that difference.

ACKNOWLEDGEMENTS.

Many thanks to Grace Gentry and Jean Danver for taking the time to read this paper and make useful suggestions.

REFERENCES.

1. Government Accounting Office report on data processing costs, GAO report #FGMSD-80-38, Washington, D.C., 1980.

2. DACS, A Bibliography of Software Engineering Terms, IIT Research Institute, October 1979.

3. DACS, Quantitative Software Models, IIT Research Institute, March 1979.

4. REX/3000 USERS MANUAL, Gentry Inc., 1980.

5. Carnes, Lance, "Design and implementation of REX/3000", HPGSUG Meeting Proceedings, Lyon 1979.

6. Jensen and Wirth, Pascal User Manual and Report, Springer-Verlag, 1974.

7. Green, Robert, "HP3000 / Optimizing On-line Programs", HPGSUG, Denver, 1978.

8. McClure, Bob in a speech to the Software Underground, San Francisco, CA, April 1980.

APPENDIX.

This section contains the database schema and program source code and output mentioned in the paper:

Listing of the schema for the PARTS database, and the contents of each dataset.

REX example report.

QUERY example report.

COBOL example report.

```
 HP32216A.04.01 QUERY/3000  MON, JUL 28, 1980,  3:59 PM
 QUERY/3000 READY
B=PARTS
PASSWORD =
ANY
MODE =
1
FORM

DATA BASE: PARTS                          MON, JUL 28, 1980,  4:00 PM

SET NAME:
   PART-MSTR,MANUAL

      ITEMS:
         PART#,                Z4           <<KEY ITEM>>
         PART-NAME,            U16
         PRICE,                P8

      CAPACITY: 101             ENTRIES: 3

SET NAME:
   PART-STOCK,DETAIL

      ITEMS:
         PART#,                Z4           <<SEARCH ITEM>>
         WHSE,                 U6
         QTY,                  Z4

      CAPACITY: 414             ENTRIES: 7


LIST PART-MSTR

PART# PART-NAME         PRICE
 3122 MANUAL #177         275
 2142 BRACKET              75
 1785 BOLT 1 X 1/4          5

LIST PART-STOCK

PART# WHSE     QTY
 1785 101     2000
 2142 100      750
 3122 100      100
 2142 102      250
 2142 101      100
 1785 100     1000
 3122 102        0
```

Listing of the schema for the PARTS database, and the contents of each dataset.

```
                              REX/3000  VERSION A.1.0623
                                (C) GENTRY, INC. 1980

 1   1   1  << WAREHOUSE PARTS SUMMARY >>
 2   1   1  DATABASE PARTS PASSWORD "READER" ACCESS 5
 3   1   1    DATASET PART-STOCK
 4   1   2  REPORT
 5   2   2  GET PARTS.PART-STOCK
 6   2   3    LIST WHSE AS "WAREHOUSE",                        &
 7   2   3         PART# AS "PART#",                           &
 8   2   3         QTY AS "QUANTITY"                           &
 9   2   3      SORTED BY WHSE, PART#                          &
10   2   3      SUMMARIZING QTY ON WHSE
11   2   3  LOOP
12   2   2  END  << REPORT BLOCK >>
13   2   2  END.
```

```
WAREHOUSE PART# QUANTITY

100        1785     1000
100        2142      750
100        3122      100

                    1850

101        1785     2000
101        2142      100

                    2100

102        2142      250
102        3122        0

                     250
```

REX example report

```
REX/3000  VERSION A.1.0623
 (C) GENTRY, INC. 1980

 1   1   DATABASE PARTS PASSWORD "READER" ACCESS 5
 1   1     DATASET PART-MSTR
 1   2       PRICE P6.2
 1   3     DATASET PART-STOCK
 1   2
 1   2   PROGVAR VALUE P7.2
 1   1           STAR A1
 1   1
 1   1   REPORT
 2   2   GET PARTS.PART-STOCK
 2   3     IF QTY = 0 THEN STAR = "*" ELSE STAR = " "
 2   3     GET PARTS.PART-MSTR WITH PART# = PARTS.PART-STOCK.PART#
 2   3     LIST WHSE AS "WAREHOUSE",                         &
 2   3          STAR,                                        &
 2   3          PART# AS "PART#",                            &
 2   3          QTY AS "QUANTITY",                           &
 2   3          PARTS.PART-MSTR.PRICE AS "  PRICE",          &
 2   3          VALUE = QTY * PARTS.PART-MSTR.PRICE          &
 2   3                  AS "   VALUE"                        &
 2   3       SORTED BY WHSE, PART#                           &
 2   3       SUMMARIZING "SUMMARY ",QTY,VALUE                &
 2   4                     ON WHSE                           &
 2   4       TOTALING "GRAND TOTAL",QTY,VALUE
 2   3   LOOP
 2   2   END  << REPORT BLOCK >>
 2   2   END.
```

```
WAREHOUSE    PART# QUANTITY   PRICE     VALUE

100           1785     1000    0.05     50.00
100           2142      750    0.75    562.50
100           3122      100    2.75    275.00

SUMMARY                1850            887.50

101           1785     2000    0.05    100.00
101           2142      100    0.75     75.00

SUMMARY                2100            175.00

102           2142      250    0.75    187.50
102         * 3122        0    2.75      0.00

SUMMARY                 250            187.50


GRAND TOTAL            4200           1250.00
```

REX example report.

```
 HP32216A.04.01 QUERY/3000  TUE, JUL 29, 1980,  2:10 PM
 QUERY/3000 READY
DATA-BASE = PARTS
PASSWORD =
ANY
MODE =
5
FIND ALL PART-STOCK.PART#
7  ENTRIES QUALIFIED
REPORT
H1,"WAREHOUSE PARTS REPORT",30
H2,"WAREHOUSE  PART#  QUANTITY",32
D,WHSE,15
D,PART#,22
D,QTY,30
S1,PART#
S2,WHSE
END


         WAREHOUSE PARTS REPORT
       WAREHOUSE  PART#  QUANTITY
          100      1785    1000
          100      2142     750
          100      3122     100
          101      1785    2000
          101      2142     100
          102      2142     250
          102      3122       0
 exit
```

QUERY example report.

```
PAGE 0001   HEWLETT-PACKARD 32213C.02.03  COBOL/3000 MON, JUL 28, 1980,  3:57 PM (C

       001000$CONTROL USLINIT
       001100 IDENTIFICATION DIVISION.
       001200 PROGRAM-ID.    PARTCOB.
       001300 DATE-COMPILED.
                     MON, JUL 28, 1980,  3:57 PM.
       001400 REMARKS.
       001500           THIS PROGRAM READS THE 'PARTS' DATA BASE
       001600           LOOPS THRU MASTERS, GETS ASSOCIATED DETAILS
       001700           AND SORTS THEM; IT THEN READS THE SORT FILE,
       001800           OUTPUTING THE SORTED RECORDS, GIVING A SUMMARY
       001900           OF TOTAL QUANTITY & COST AT EACH CHANGE IN
       002000           'WAREHOUSE'
       002100           *
       002200           PRIMARY SORT KEY    -  WAREHOUSE
       002300           SECONDARY SORT KEY  -  PART
       002400           *
       002500           NO PAGE CONTROL PRESENT
       002600           *
       002700           SET FILE EQUATION :FILE LP=$STDLIST;CCTL
       002800           BEFORE EXECUTING
       002900           *
       003000 ENVIRONMENT DIVISION.
       003100 CONFIGURATION SECTION.
       003200 SOURCE-COMPUTER. HP3000.
       003300 OBJECT-COMPUTER. HP3000.
       003400 INPUT-OUTPUT SECTION.
       003500 FILE-CONTROL.
       003600     SELECT REPORT-FILE    ASSIGN TO "LP  ".
       003700     SELECT SORT-FILE      ASSIGN TO "SORT,DA".
       003800 DATA DIVISION.
       003900 FILE SECTION.
       004000 FD  REPORT-FILE
       004100     RECORD CONTAINS 72 CHARACTERS
       004200     LABEL RECORD IS OMMITTED.
       004300 01  REPORT-FILE-REC.
       004400     05  REPORT-FILE-REC-LINE     PIC X(72).
       004500 SD  SORT-FILE
       004600     RECORD CONTAINS 24 CHARACTERS.
       004700 01  SORT-FILE-REC.
       004800     05  SORT-FILE-REC-KEY.
       004900         10  SORT-FILE-REC-WHSE   PIC X(08).
       005000         10  SORT-FILE-REC-PART   PIC 9(04).
       005100         10  FILLER               PIC X(12).
       005200 WORKING-STORAGE SECTION.
       005300 01  SORT-RCD.
       005400     05  SORT-KEY.
       005500         10  SR-WHSE              PIC X(08) VALUE SPACE.
       005600         10  SR-PART              PIC 9(04) VALUE ZERO.
       005700     05  SORT-DATA.
       005800         10  SR-QTY               PIC 9(04)  VALUE ZERO.
       005900         10  SR-PRICE             PIC S9(05)V9(02)
       006000                                       VALUE ZERO.
       006100
       006200 01  CONTROLS-AND-SUMS.
       006300     05  SUM-QTY                  PIC S9(09) VALUE ZERO.
       006400     05  SUM-COST                 PIC S9(12)V9(02) COMP-3
       006500                                  VALUE ZERO.
```

```
006600         05  TOTAL-QTY                  PIC S9(09) VALUE ZERO.
006700         05  TOTAL-COST                 PIC S9(12)V9(02) COMP-3
006800                                        VALUE ZERO.
006900 01  HDR-LINE.
007000         05  HL-CC                      PIC X(01) VALUE SPACE.
007100         05  FILLER                     PIC X(52) VALUE
007200             "WAREHOUSE   PART#  QUANTITY    PRICE         VALUE  ".
007300 01  DTL-LINE.
007400         05  DL-CC                      PIC X(01) VALUE SPACE.
007500         05  DL-WHSE                    PIC X(08) VALUE SPACE.
007600         05  DL-FILLER                  PIC X     VALUE SPACE.
007700         05  DL-STAR                    PIC X(02) VALUE SPACE.
007800         05  DL-PART                    PIC Z(06) VALUE ZERO.
007900         05  DL-QTY                     PIC Z(09) VALUE ZERO.
008000         05  DL-PRICE                   PIC Z(07).9(02) VALUE 0.0.
008100         05  DL-COST                    PIC Z(12).9(02) VALUE 0.0.
008200 01  SUM-LINE.
008300         05  SL-CC                      PIC X(01) VALUE SPACE.
008400         05  TEXT-LINE                  PIC X(17) VALUE "SUMMARY       ".
008500         05  SL-QTY                     PIC Z(09) VALUE ZERO.
008600         05  FILLER                     PIC X(10) VALUE SPACE.
008700         05  SL-COST                    PIC Z(12).9(02) VALUE 0.0.
008800 01  BLANK-LINE                         PIC X(72) VALUE SPACE.
008900 01  MISC.
009000         05  COST                       PIC S9(09)V9(02) COMP-3
009100                                               VALUE ZERO.
009200         05  LINE-COUNT                 PIC S9(04) USAGE COMP SYNC
009300                                                   VALUE ZERO.
009400         05  AT-END-FILE                PIC S9(04) USAGE COMP SYNC.
009500         05  IMAGE-MODE                 PIC S9(04) USAGE COMP SYNC.
009600 01 IMAGE-DATASET-NAMES.
009700         05  IDN-PART-MSTR              PIC X(16) VALUE "PART-MSTR ".
009800         05  IDN-PART-STOCK             PIC X(16) VALUE "PART-STOCK ".
009900 01  IMAGE-STATUS-AREA.
010000         05  ISA-COND-WORD              PIC S9(04) USAGE COMP SYNC.
010100         05  ISA-DATA-LENGTH            PIC S9(04).
010200         05  ISA-RECORD                 PIC S9(09) USAGE COMP SYNC.
010300         05  ISA-CHAIN-LENGTH           PIC S9(09).
010400         05  ISA-ADDRESS-BACK           PIC S9(09).
010500         05  ISA-ADDRESS-FORWARD        PIC S9(09).
010600 01 IMAGE-CONTROL-WORDS.
010700         05  ICW-TEMP                   PIC S9(04) USAGE COMP SYNC.
010800         05  ICW-DBNAME                 PIC X(16) VALUE "  PARTS; ".
010900         05  ICW-DATASET                PIC X(16) VALUE SPACES.
011000         05  ICW-PASSWORD               PIC X(08) VALUE "READER  ".
011100         05  ICW-MODE                   PIC S9(04) USAGE COMP SYNC.
011200         05  ICW-DATALIST               PIC X(04) VALUE "@   ".
011300         05  ICW-SEARCH-ARG             PIC X(16) VALUE SPACES.
011400 01 IDB-PART-MSTR.
011500         05  IDB-PM-PART            PIC 9(04) VALUE ZERO.
011600         05  IDB-PM-NAME            PIC X(16).
011700         05  IDB-PM-PRICE           PIC S9(05)V9(02) COMP-3.
011800 01 IDB-PART-STOCK.
011900         05  IDB-PS-PART            PIC 9(04) VALUE ZERO.
012000         05  IDB-PS-WHSE            PIC X(06) VALUE SPACES.
012100         05  IDB-PS-QTY             PIC 9(04) VALUE ZERO.
012200 01 IMAGE-FIND-ITEM                 PIC X(08) VALUE "PART#  ".
```

```
PROCEDURE DIVISION.
MAIN-PROCESS-CONTROL SECTION.
PAR-1.
    PERFORM OPEN-DB-E.
    PERFORM DO-THE-REPORT.
    STOP RUN.
DO-THE-REPORT.
    SORT SORT-FILE ON ASCENDING KEY SORT-FILE-REC-WHSE,
                                    SORT-FILE-REC-PART
        INPUT PROCEDURE IS GET-ENTRIES-LOOP
        OUTPUT PROCEDURE IS REPORT-ENTRIES.
GET-ENTRIES-LOOP SECTION 60.
PAR-A.
    MOVE "PART-MSTR"     TO ICW-DATASET.
    MOVE  2              TO ICW-MODE.
    CALL "DBGET" USING ICW-DBNAME,
                       ICW-DATASET,
                       ICW-MODE,
                       IMAGE-STATUS-AREA,
                       ICW-DATALIST,
                       IDB-PART-MSTR,
                       ICW-SEARCH-ARG.
    IF ISA-COND-WORD = ZERO
       PERFORM GET-NEXT-MASTER UNTIL AT-END-FILE = +11.
    GO TO END-OF-INPUT.

GET-NEXT-MASTER.
    PERFORM GET-THE-DETAILS.
    MOVE "PART-MSTR"     TO ICW-DATASET.
    MOVE  2              TO ICW-MODE.
    CALL "DBGET" USING ICW-DBNAME,
                       ICW-DATASET,
                       ICW-MODE,
                       IMAGE-STATUS-AREA,
                       ICW-DATALIST,
                       IDB-PART-MSTR,
                       ICW-SEARCH-ARG.
    IF ISA-COND-WORD NOT = ZERO
        MOVE  +11 TO AT-END-FILE.

GET-THE-DETAILS.
    MOVE "PART-STOCK"    TO ICW-DATASET.
    MOVE  1              TO ICW-MODE.
    MOVE IDB-PM-PART     TO ICW-SEARCH-ARG.
    CALL "DBFIND" USING ICW-DBNAME,
                        ICW-DATASET,
                        ICW-MODE,
                        IMAGE-STATUS-AREA,
                        IMAGE-FIND-ITEM,
                        ICW-SEARCH-ARG.
    IF ISA-COND-WORD = ZERO
       MOVE +5 TO ICW-MODE
       PERFORM PART-STOCK-LOOP
               UNTIL ISA-COND-WORD NOT = ZERO.
PART-STOCK-LOOP.
    CALL "DBGET" USING ICW-DBNAME,
                       ICW-DATASET,
                       ICW-MODE,
                       IMAGE-STATUS-AREA,
                       ICW-DATALIST,
                       IDB-PART-STOCK,
                       ICW-SEARCH-ARG.
    IF ISA-COND-WORD = ZERO THEN
            MOVE IDB-PS-WHSE  TO SR-WHSE
            MOVE IDB-PM-PART  TO SR-PART
            MOVE IDB-PS-QTY   TO SR-QTY
            MOVE IDB-PM-PRICE TO SR-PRICE
            RELEASE SORT-FILE-REC FROM SORT-RCD.
END-OF-INPUT. EXIT.
REPORT-ENTRIES SECTION 70.
PAR-C.
    PERFORM CLOSE-DB-E.
    OPEN OUTPUT REPORT-FILE.
    MOVE ZERO TO AT-END-FILE.
    RETURN SORT-FILE INTO SORT-RCD
           AT END  DISPLAY " NO SORT RECORDS"
                   STOP RUN.
    WRITE REPORT-FILE-REC FROM HDR-LINE
             AFTER  ADVANCING 1 LINES.
    WRITE REPORT-FILE-REC FROM BLANK-LINE
             AFTER ADVANCING 1 LINES.
    MOVE SR-WHSE TO DL-WHSE.
    PERFORM WRITE-THE-REPORT UNTIL AT-END-FILE = +99.
    MOVE SUM-QTY    TO SL-QTY.
    MOVE SUM-COST   TO SL-COST.
    WRITE REPORT-FILE-REC FROM SUM-LINE
           AFTER ADVANCING 2 LINES.
    WRITE REPORT-FILE-REC FROM BLANK-LINE
           AFTER ADVANCING 1 LINES.
    ADD SUM-QTY TO TOTAL-QTY.
    ADD SUM-COST TO TOTAL-COST.
    MOVE "GRAND TOTAL" TO TEXT-LINE.
    MOVE TOTAL-QTY TO SL-QTY.
    MOVE TOTAL-COST TO SL-COST.
    WRITE REPORT-FILE-REC FROM SUM-LINE
           AFTER ADVANCING 2 LINES.
    WRITE REPORT-FILE-REC FROM BLANK-LINE
           AFTER ADVANCING 2 LINES.
    GO TO END-OF-REPORT.

WRITE-THE-REPORT.
    IF SR-WHSE NOT = DL-WHSE
       THEN MOVE SUM-QTY    TO SL-QTY
            MOVE SUM-COST   TO SL-COST
            WRITE REPORT-FILE-REC FROM SUM-LINE
                  AFTER ADVANCING 2 LINES
            WRITE REPORT-FILE-REC FROM BLANK-LINE
                  AFTER ADVANCING 1 LINES
            ADD SUM-QTY TO TOTAL-QTY
            ADD SUM-COST TO TOTAL-COST
            MOVE ZERO TO SUM-QTY, SUM-COST
            ADD 3 TO LINE-COUNT.
    IF SR-QTY = ZERO
       THEN MOVE "* " TO DL-STAR
```

```
PAGE 0005   PARTCOB
    ELSE    MOVE " " TO DL-STAR.
    MOVE SR-WHSE   TO DL-WHSE.
    MOVE SR-PART   TO DL-PART.
    MOVE SR-QTY    TO DL-QTY.
    MOVE SR-PRICE  TO DL-PRICE.
    MULTIPLY SR-PRICE BY SR-QTY
               GIVING COST.
    MOVE COST      TO DL-COST.
    ADD  COST      TO SUM-COST.
    ADD  SR-QTY    TO SUM-QTY.
    WRITE REPORT-FILE-REC FROM DTL-LINE
          AFTER ADVANCING 1 LINES.
    ADD 1 TO LINE-COUNT.
    RETURN SORT-FILE INTO SORT-RCD
          AT END MOVE +99 TO AT-END-FILE..

END-OF-REPORT.  EXIT.

SUPPORT-ROUTINES SECTION 80.
OPEN-DB-E.
    MOVE 5 TO ICW-MODE.
    CALL "DBOPEN" USING ICW-DBNAME,
                        ICW-PASSWORD,
                        ICW-MODE,
                        IMAGE-STATUS-AREA.
    IF ISA-COND-WORD NOT = ZERO
       THEN DISPLAY "ERROR IN DBOPEN",
                    ISA-COND-WORD
            STOP RUN.
CLOSE-DB-E.
    MOVE 1 TO ICW-MODE.
    CALL "DBCLOSE" USING ICW-DBNAME,
                         ICW-DATASET,
                         ICW-MODE,
                         IMAGE-STATUS-AREA.
    IF ISA-COND-WORD NOT = ZERO
       THEN DISPLAY "ERROR IN DBCLOSE",
                    ISA-COND-WORD.
```

```
WAREHOUSE    PART#  QUANTITY    PRICE          VALUE

100           1785     1000       .05          50.00
100           2142      750       .75         562.50
100           3122      100      2.75         275.00
SUMMARY                1850                   887.50

101           1785     2000       .05         100.00
101           2142      100       .75          75.00
SUMMARY                2100                   175.00

102           2142      250       .75         187.50
102         * 3122               2.75            .00
SUMMARY                 250                   187.50

GRAND TOTAL            4200                  1250.00
```

COBOL example report.

APPLICATIONS PROGRAM TRANSFORMATIONS

Presentor

Larry Van Sickle, Cole & Van Sickle

The files used by applications programs often need to be changed to meet changing requirements and to tune the performance of the applications system. Changes to the file systems - data bases, KSAM files, MPE files - are usually easy. Changing existing applications programs to work with the changed file systems can be very difficult. I present principles and specific methods for organizing applications programs so they are easier to modify and maintain. I will also discuss the use of data dictionaries and other automated system development tools in simplifying applications programs transformations.

THE ALL PURPOSE COMPUTER

Presentor

Roger Hoff, President, HP

Integral Systems, Inc. (ISI) is a proprietary software vendor specializing in the development and marketing of comprehensive Payroll/Personnel Systems. In June of 1980, a study was initiated to determine the feasibility of acquiring a general purpose computer to partially replace time-sharing services. The study pointed out that not only could an installed computer yield substantial cost savings in the major production effort--system development--but could also provide ISI the opportunity to further automate additional functions.

The selection and installation of an HP3000 series III has served to automate numerous functions in corporate headquarters. By simultaneously acquiring RJE/3000, Text and Document Processer 3000, and Interactive Mainframe Link software, significant capabilities over stand-alone systems, expedited marketing mailings, improved system demonstrations for prospective clients, and provided responsive administrative systems to satisfy management information needs. With the success realized to date, further extensions of the system are planned to improve communications with regional offices and provide field demonstration capabilities.

QCALC/3000

QUICK CALCULATION

by:

Earl E. Colmer, Jr.

Automated Sciences Group, Inc.

Automated Sciences Group, Inc.
700 Roeder Road
Silver Spring, Maryland 20910
301/587-8750

QCALC/3000 - QUICK CALCULATION

QCALC/3000 is a multi-function ASCII string calculator that displays your answer in ASCII scientific notation. QCALC consists of two programs: QCALCR, and QCALCL. QCALCR is accurate to 6.9 decimal places while QCALCL is accurate to 16.5 decimal places.

QCALCR and QCALCL allow the user 26 registers, known as equate variables (A-Z), 7 operators (+, -, *, /, ¯, !, ?), hierarchy (), 11 functions (Cosc, Cosh, Sinc, Sinh, Tanc, Tanh, Logc, Logn, Atan, Absv, Rand), and equation storage and recall. Command interpreter includes: Redo, Debug (Program Debug), :Debug (MPE Debug), MPE Commands, Exit. Both programs include an interactive 'FOR' loop which can take a function from the starting value/variable, increment it by the increment value/variable, and check it against the limit, which can also be a value or a variable.

Both QCALCR and QCALCL contain an 'ENTRY POINT' which allows the user to store QCALC commands into a file and have the results written to another file (default = $STDINX, $STDLIST). This allows either calculator to be created as a 'SON PROCESS', which can perform calculation after which it reactivates the father process. This technique is very useful in plotting programs. Your plot program never needs to be recompiled because your function can be accepted at run time, and passed to the son calculator.

Source programs may be purchased upon written request, but A.S.G., Inc. will retain all sales rights!

Example of Equate:	A=B=C;D=2;E=3;F=3.14E+00
Example of String:	((A+B/2.3)-(F¯2)*8E-2)
Example of For Loop:	A*COSC(X=1/B/.5/) [;SHOW]
Example of Equation=:	EQUATION=((A+B/2.3)-(F¯2)*8E-2)
Example of Equation:	EQUATION