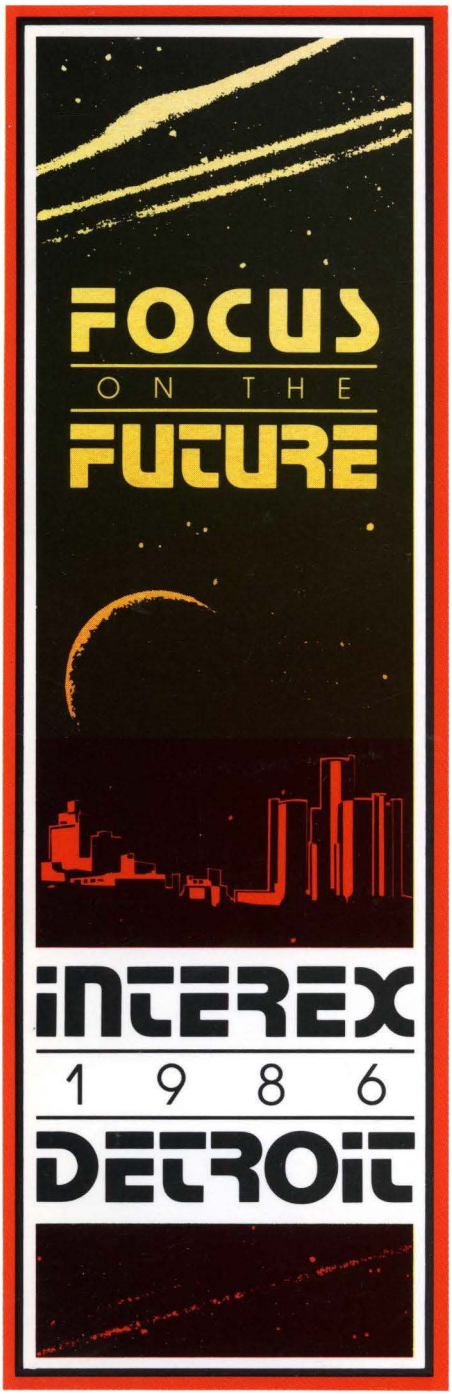


BUSINESS PROCEEDINGS

HP 3000/SERIES 100

VOLUME II

INTEREX
DETROIT CONFERENCE
SEPTEMBER 28 - OCTOBER 3, 1986

4GL & The Changing Role Of The Programmer

Ian Farquharson
Infocentre Ltd.
6303 Airport Road
Suite 300
Mississauga, Ontario
L4V 1R8

Application development using Fourth Generation Programming Languages is now a reality.

The existing data processing professionals are competent, educated and comfortable in the use of traditional 3rd generation programming tools and methodologies.

The systems analyst will continue to perform systems analysis regardless of the implementation tool. But what about the task of programming and the programmer himself? The programmer has spent years refining his or her skills and expertise in traditional languages and their associated development methodologies. The analyst on the other hand has been refining the art of needs assesment, user interviewing and systems analysis.

With Fourth Generation Software, the task of implementing application software systems becomes one of telling the HP3000 what to do but not necessarily, how to do it. The job entails the intricate involvement of the various end users.

This paper will focus on the changing role and expectations of the programmer in HP3000 data processing environments as a result of the implementation of 4th and higher generation programming tools. It will address the changes in the required skills set and the considerations necessary to ensure the smooth transition to programming of the future for todays existing data processing professionals.

Drive screens -- don't let them drive you crazy

by Michel Kohon
Tymlabs Corporation

Once upon a time, the computer world was nice and cozy. Users didn't exist, and programmers were heroes. That was the time of the 80-column punch card and the endless listings. Cards were generally used to write memos, sometimes to enter data in the big 64K IBM/360. Listings were used to decorate the top of office cabinets, though we know of some zealous employees who actually read them.

This ideal period ended suddenly when computer manufacturers such as HP introduced interactive systems. They tried, sometimes with success, to have users and programmers talk to each other, using the CRT console as an interface. Suddenly, programmers had to write systems that would be used directly by users. And there the chaos began!

No one knew how to communicate with the users. What shall we display on the console? Where? When? Instead of letting users enter data in columns, as with punch cards, someone decided it was better to present data in rows. The controversy heightened.

Some programmers were in favor of columns while others were in favor of lines. Then a smart programmer suggested that instead of displaying a prompt at every transaction, it could be just as easy to leave the prompt on the screen, erase the old data, and let the user enter new stuff. Then another one dared to put several columns on the screen so that users could see the whole form during the transaction.

Things could have settled down a bit, but instead they got worse. A wise programmer realized that sending the data field by field to the computer was not efficient for the machine. Instead, it would be more productive to send the whole block of data for processing at each transaction.

The block mode was born, creating a much bigger hoopla. Was block mode user-friendly? (The term then was ergonomic.) Some programmers wondered. Since they were doing software for users, they had to deal with them, and although you don't want to mix the two, these things happen. Some programmers and users became friends to the point where they actually talked to each other; hence, the concept of user-friendliness was created.

Block mode and character mode advocates continued to war. "Anyway, in both cases the whole form is displayed on the screen," said a user.

"That's true," said the two groups of programmers. "Is it not what you wanted?"

"No," said the user, "you never asked us before."

Most of the current applications have adapted themselves to the strengths and weaknesses of one system or the other. In the case of block mode, a form is displayed, data is collected without controls (except some editing), and the data is sent to the processor. In the case of character mode, a form is displayed, and data is collected with controls on a field-by-field basis.

In both systems, the screen drives the user instead of the user driving the screen, and users don't want that anymore. Did they ever want it?

Let's take an example of what users want in real life. The screen displays "Name." User Joe enters his name. Then the prompt "Sex" appears. Joe enters "M" for male.

Following the old approach, the next prompt would have been "Maiden Name." Joe doesn't need that - he is a male! Why not skip to the next prompt? On the other hand, and to follow the same example, the fact that Joe is a male might trigger a new set of questions related to his military duties.

New screen-driving techniques can display a new form for these military-related questions. The new form will overlay part of the old and disappear as soon as it is completed. The windowing concept is here!

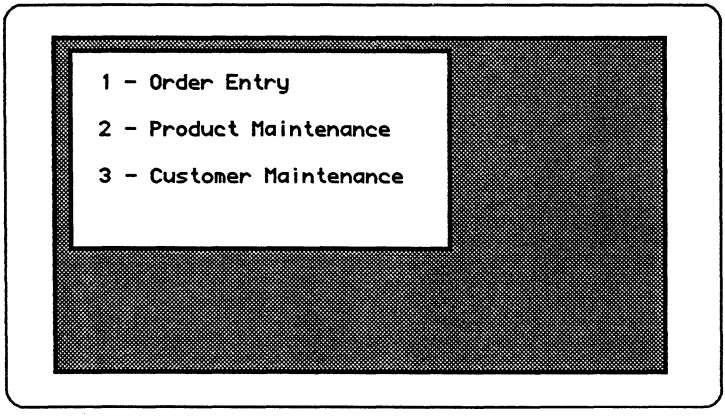
A windowing system enables you to pop, hide, and overlay windows on the screen so that your program follows the user's train of thinking instead of the user following the program's rigid process. Programming windows is easy as long as you put your feet in the user's shoes. Programming a so-called intelligent system becomes a day-to-day habit. With windows, your program comes alive and your users become more productive.

A correct windowing system will at least provide the capabilities to create and display windows, overlay windows, fill windows, and delete windows. In addition to those basic functions, a good windowing package will offer some "window dressing" (it was too easy), features such as changing colors, changing borders, and moving the title's position in the border.

Since you are now familiar with the idea of windows, brought to us by the Xerox STAR system and effectively used by Apple and now IBM, let us redesign the classic-order entry system that most packages use as a tutorial. But first, let's walk through the order program as it was. The dialogue between the user and the program assumes that VPLUS is used.

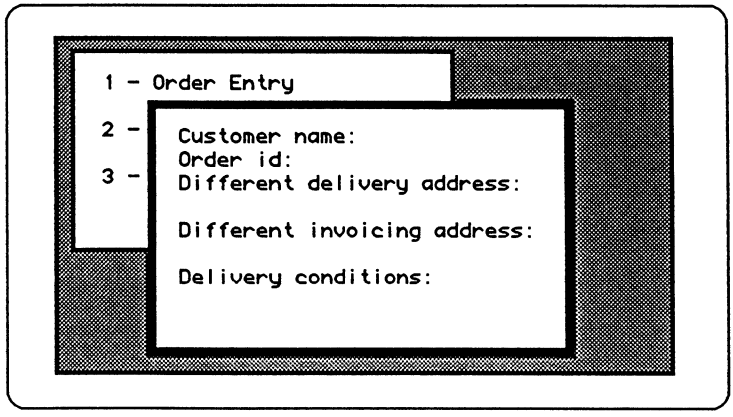
The user enters a customer name, an order ID, and answers "No" at the prompt "Different delivery address?" At this point, the user tabs 14 times to reach the delivery-condition field, a product number, the quantity, and a discount. The user presses ENTER to ship the data to the processor.

The same application using windows will look like this. The program opens a window with three options:



This window is in the top left corner of the screen with blue background. Although colors are optional, they really make sense with a windowing system as a visual guide.

The user enters a 3, and a second window comes on the screen with a green background and a large border. This window sightly overlays the first one, leaving the first line and the column with 1-2-3 still visible. It is a good practice to display the new window under the selected line. Our new screen will look like this:



Let's now assume a different delivery address. The program reacts immediately by opening a third window with the name and address prompts overlaying the second window. Once the delivery address is entered, the foreground window disappears and the user is back to the previous window.

At this point, the user might request some online help. The old system would either have to erase the whole screen or have one line ready for a help message. Both solutions are really inconvenient for users. A windowing system can simply pop up a help window at a convenient location that the environment is not disturbed.

As this short example shows, a good windowing system helps build a more productive user interface and can help an application create the same excitement as Symphony, Topview or Sidekick.

A windowing system in itself is not a multitasking operating system but rather a set of tools callable from a program. The programmer can use these tools to create a myriad of "intelligent" user interface for any environment, ranging from a small payroll application to the most complex multi-tasking operating system. Many designers will see, as IBM and HP have, that this is the way to go.

An Expert Financial Planning System

Don MacKenzie, Ross Hopmans, Shawn Brayman

Brant Computer Services Limited
Burlington, Ontario

Contents

1.0 Problem Description and Overview

This paper discusses a financial planning package developed by Brant Computer Services that combines 4th and 5th Generation Languages in a financial service oriented expert system.

The mandate from our client, an experienced financial planner, was to design a micro-computer based system that provides two distinct yet integrated functions:

1) A package that stores the client's personal and financial data. Once the data is in the system, it must be formatted into the necessary financial statements, schedules, and accompanying calculations intrinsic to the financial planning model.

2) The capability to provide recommendations based on quantitative and qualitative information about the client being evaluated. The knowledge required to make these recommendations originates from the financial planner and this knowledge is imparted to the expert system.

The interesting implications of this project stem from the practicality and feasibility of expert systems in the financial services sector and the degree to which important business and financial decisions can be formalized into a computer based system. Related to this issue is the question as to whether the system can be economical, user friendly, and portable.

Secondly, the attempt to deliver a system for production use which is a hybrid of an artificial intelligence program and a conventional fourth-generation language program has proven to be a challenging and exciting process.

2.0 Design Approach and Analysis

Initially we defined the two distinct functions of the system. As mentioned in the introduction, one component of the package involves data capture, reporting functions, and a modelling capability. These tasks are typically straightforward and formalized and they must be carried out by the financial planner in order to make strategic decisions based on the financial and personal status of the client. To put it more simply, this is the 'front end' of the financial planning process: the accumulation and integration of the client's financial information. The

information used up to this point is quantitative.

The second and more elaborate function of the system is the 'expert' capability that involves the digestion of both the quantitative and the qualitative factors in the financial planning process (more about this later). This functionality implies a less tangible process than the one that occurs in the front end section. However, a similarily formalized procedure is required on the expert side although the decision criteria and considerations for the expert process are broader and involve factors that are not purely quantitative.

Our aim here is to let the financial planner provide the rules and reference points to the expert system so that the expert system can supply valid strategies for the client to follow. Herein lies the obvious challenge in this type of project: the formalization of the complex and seemingly non-procedural processes and the duplication of these processes on a computer based system.

The identification of the 'front end' and the 'expert end' of the system gave us a starting point in the design of the system. By determining the data requirements for the reporting functions and the higher level processes, we were able to design the 'front end' section of the system.

2.1 System Specifications

The system front end was designed to handle data entry, reporting and modelling functions. The primary functional requirements of this system were defined from a number of sources. Primarily, our client provided us with samples of the input documents for his clients as well as sample plan presentations. The client was generating the bulk of his reports and schedules through a LOTUS 123 spreadsheet.

On top of this, we received a number of plan presentations from other financial planning services. The exposure to the various planning approaches and presentations enabled us to extract the common requirements of the different plan philosophies.

We attended a financial planning conference and paid particular attention to the different planning software packages that were being promoted at the conference. Our emphasis here was on the capabilities and user friendliness of these packages.

Critical to our analysis was the extensive amount of time spent

with our client on the financial planning process in general. We did have access to a broad range of financial planning material so that we were able to get an idea of the amount and nature of the information involved in the financial planning process itself.

The formalization of the planning process was achieved by sitting down with our client, extracting his knowledge, and formulating a set of rules to guide the expert system. This 'knowledge engineering' involves refining the expert's knowledge into a series of relationships and conditions that can be utilized by the expert system. It is this information that guides it's decision process.

3.0 The Financial Planning Process

Before describing our particular model, I will briefly discuss the financial planning process in general. Essentially, this process involves the assimilation of all the aspects of an individual or family's financial and personal status. With this information, the planner devises a methodical strategy whereby the client can achieve his/her specified level of financial independence. This sounds relatively straightforward, although it will become apparent later in this paper that the process is complex and involves many variables.

Currently, the financial planning industry is unregulated and fragmented. Financial planning, unlike other financial services such as accounting, does not enjoy a uniform, standard methodology. Depending upon the financial planner involved, the quality, method, and philosophy behind the planning does vary. On top of this, financial planning services may be industry driven (the service may be an arm of a financial institution such as an insurance company or investment dealer) or product driven (the planner favours a particular investment vehicle).

The purpose of this paper is not to perform a critical analysis of the industry but rather to highlight the fact that there is a need to standardize and define some of the universal requirements of financial planning. One objective in designing a package such as this is to provide a front end to the expert side of the system that is relatively devoid of one particular approach. In other words, the facts and information that are supplied to the expert system should be unencumbered by a particular planning philosophy and the expert system should not be constrained by a limited amount of information about the client.

3.1 The ROGI Model

The philosophy behind the Rate Of Growth on Investments (ROGI) model is simple and easy to understand. The aims of this model are as follows:

1) determine client's current financial status.

2) determine future capital requirements and objectives.

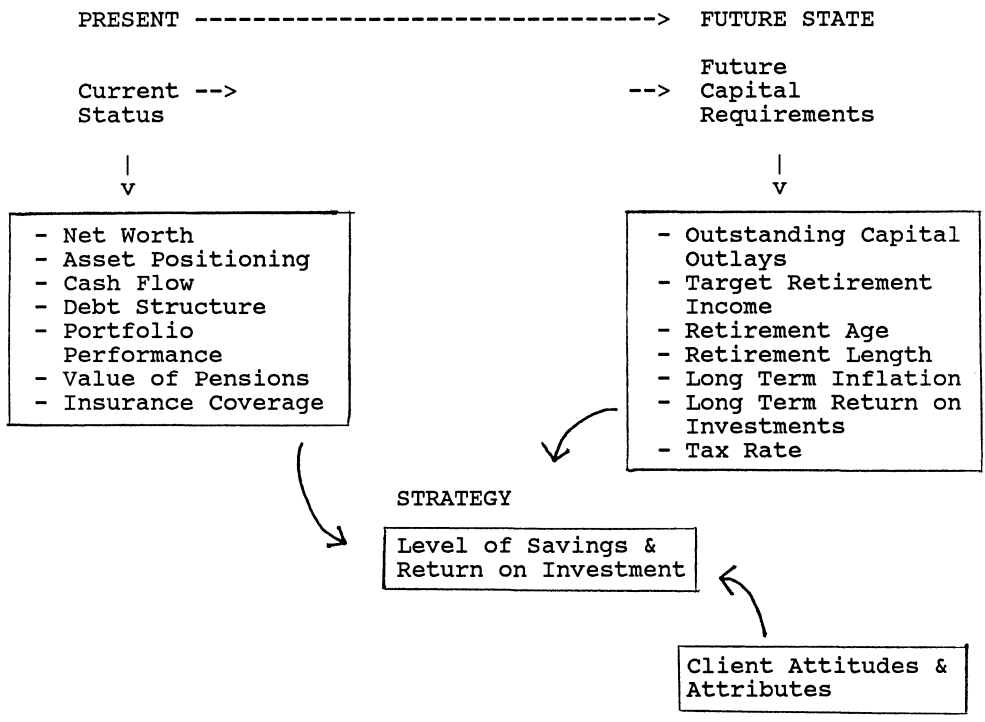
3) using a strategy of savings and optimal return on investment, enable the client to achieve the targeted financial goal within a time frame that is both suitable and realistic for the client.

The task here is not dissimilar to a typical optimization problem where a function is to be maximized subject to a constraint. In this type of problem, both the maximizing function and the constraints are non linear since many of the variables involved are not quantifiable. To put it simply, it is hard to build information like "I hate gold, love real estate, am indifferent to stocks as long as I maintain enough money in GIC's", etc. into a formalized equation.

The key philosophy behind the ROGI model is that the achievement of the client's financial objectives take into consideration all of the aspects of the client's financial and personal status. This means that the optimal strategy involves accounting for asset management, cash flow, investment planning, estate conservation and distribution, and the client's attitudes toward financial independence.

3.2 ROGI Strategies

In the ROGI model, the client's future capital requirements are laid out in such a way as to enable the planner and the client to relate the client's specified financial target to the client's current financial status. If the client is already on a path that will allow him or her to achieve the desired financial goals, it is unlikely that the individual is in need of a financial planning service. If, on the other hand, the client must alter current savings, portfolio performance or retirement expectations, then a strategy must be devised to enable the client to achieve the target capital pool.



Looking at the diagram above, we can get an idea of the types of considerations that come into play in the planning process. The front end of the system will provide the planner with the storage of the data, reports required from the hard data and the impact of the predictive data on future capital requirements.

The ability to assess the impact of the predictive data is what I have referred to as the modelling function of the front end of the system. By modelling, I mean that by manipulating certain variables (they are quantitative) the planner can see the resulting impact on the capital requirements of the client's financial goals. This is important since there is not necessarily one optimal savings and investment rate, and the planner must be able to determine the impact of alternative scenarios on the client's future capital requirements.

With the client's current status and projected requirements in hand, the planner and the client can get an initial impression as to the viability of the client's goals. The planner can begin to see how the client's current cash flow and portfolio fit in with the level of savings and investment that will lead to the desired retirement income.

OPTIMIZE SAVINGS	OPTIMIZE RETURN ON INVESTMENTS
1. Earn more	1. Increase Risk
2. Spend Less	2. Investment Education
-budget	3. Professional Management
-restructure debt	
-alter living standards	
3. Taxes	
-shelters	
-deferral strategies	

Given that a client has specified a retirement income that is translated into a target capital pool, an optimal savings and investment return are required. How are these levels determined? Obviously, the client's projected cash flows and portfolio performance are determinants. On top of this, the client has certain attitudes, opinions and biases that prevent particular combinations of savings and returns on investments from being viable planning strategies for the planner's client.

What are these constraints? These are considerations in the planning strategy that are not purely economic or quantitative. For example, the planner may recommend a target savings level for the client to follow that is within the realm of the clients cash flow. The client, however, may find this savings level unacceptable in that it reduces his or her immediate standard of living to a level that does not justify the purpose of the savings plan. In the same manner, a suggested portfolio performance may be rejected by the client on the grounds that it implies a risk tolerance above that of the client. Or the planner may recommend a restructuring of assets to increase liquidity that the client cannot accept because the asset in question has a 'sentimental value' to the client.

4.0 The 4GL Component

The previous discussion has given an indication of the

considerations involved in the financial planning process. Despite the repeated warnings about the intangibility of much of the data, there is obviously a need for a variety of financial statements and reports based on quantitative information. An objective of the 4GL component of this package is to integrate information and give a profile of the client's financial status.

On the asset management side, the key reports to be produced are statements of net worth, asset positioning, and the portfolio performance of the the client. Specifically, the net worth information relates the debt/equity aspect of the client's holdings. Asset positioning gives a breakdown of the types of assets held by the client (personal, invested capital, tax shelter etc.), as well as the liquidity of the assets. The portfolio profile is useful in determining if the individual is adequately diversified and has a portfolio that is in line with his or her comfort level with regards to safety of capital. The portfolio profile can also indicate how each component of the portfolio is performing relative to the portfolio as a whole.

Cash flow reports indicate to the planner the ability of the client to reduce current debt and to increase the client's net investable capital. When determining an optimal savings level for the client, the planner must have a good feel for the client's discretionary cash flow from the present time to retirement or financial independence.

Referring to the diagram on page 4, the asset and cash flow reports profile the current status of the client. The future capital requirements of the client are calculations that are based on a combination of the client's needs in the future as well as certain predictions about long range economic conditions such as the rate of inflation, the client's nominal return on investments, tax rate on retirement income etc... Other factors to be determined include years to retirement and the length of retirement.

These financial and future requirement reports are the basis of the planning process. The client's attitudes and characteristics may have a large impact on the particular strategy devised for the client, yet these qualitative characteristics are meaningless if the economic circumstances of the client preclude a realistic chance at attaining the client's objectives. In other words, before the planner can be concerned with all of the considerations

of the planning process, he or she must have the basic financial profile of the client in order to determine a realistic financial objective for the client. This relates back to the statement that while the 4GL is providing the 'less glamorous' component of this package, the overall efficacy of the expert system will be proportional to the quality of the information it is accessing, as well as the rules that make it up.

4.1 Modelling

The planner and the expert system require the capability to determine the impact of various strategies and manipulations on the client's future cash flows, capital requirements, and portfolio performance. Given that a particular savings level and return on investment are not feasible for the client, the planner will investigate the viability of various savings levels, returns on investment, retirement parameters, investment vehicles etc... The impact of a proposed scenario or strategy must be immediately reflected in the system. Of particular importance in the ROGI model is the ability to see how various assumptions and scenarios impact on the target capital pool required by the client at retirement.

For example, if a particular financial objective is not realistic given the client's current desires and financial profile, the planner may want to determine whether a 5 year deferment of retirement will enable the client to amass the desired capital pool at retirement. Alternatively, the planner may test the impact of an increased investment return on the client's ability to achieve the required level of invested capital. Below are some examples of the modelling capabilities that the 4GL should provide:

1) The effect of a different tax rate on retirement cash flow

2) The impact of the purchase of a real estate investment on capital requirments at retirement

3) The impact of a deferral of retirment on the required level of savings and return on investment to achieve the targeted financial goal

4) The impact of a restructuring of the client's portfolio on the overall portfolio performance

4.2 Pros and Cons of the 4GL

The choice of a 4GL for the front end functions is rooted in the requirments of the front end:

1) strong screen handling/menu capabilities
2) data base access
3) user friendly
4) good reporting capability
5) linkup to 5GL
6) prototyping function
7) portability between HP3000 and IBM compatible
8) modular design capability

The prototyping capability of the particular 4GL that we used (Speedware) is an important feature in designing a package such as this one. Our client had a manual process of collecting the client's financial data and we had to streamline the data entry process to be easy to understand yet at the same time be able to capture all of the information required by a comprehensive set of financial documents. We were able to streamline the data capture process by entering the test data on the system and producing the various financial statements. In this manner we were able to play with the data entry screens and let the design stage evolve to the point where the system was capturing the necessary data while keeping the data entry process relatively clear and simple to use.

A very strong feature of the 4GL we used was a module called DESIGNER, which allows the programmer to create an application using an online menu, screen, and report writer. Changes in the input/inquiry screens of the system are reflected by a corresponding data base modification generated by the DESIGNER. Restructuring the data base as the development phase is ongoing is simply a matter of defining a new data definition at the screen or user level. The data base and code for the menu, screen and report handling is generated by DESIGNER. The approach in this type of system design is to define the system requirements from the end user viewpoint without having to be too concerned with various file structures or coding strategies.

The modularity of the system is another important consideration because if the system is designed to handle a complex scenario, it is convenient to be able to take out the more detailed functions if they are not required for that particular planning situation.

If the client only requires an asset and cash flow profile, the data entry process will entail entering asset and cash information

without any prompting for future expectations or requiring specific investment information regarding tax planning, risk attitudes, desired retirement cash flows etc...

The ROGI model has one input screen for the client's personal information, one screen to handle family data, one screen for all asset types and liabilities (personal, investment, tax shelter), one screen for cash flow, one screen for future capital requirements, one screen for supplementary income at retirement (pensions, annuities etc.), one screen for capital requirements at death, and one screen for insurance profiles.

The portability of the 4GL application between the mini and the micro computer enables development to occur on the HP3000 with a port down to an IBM compatible. All that is required from the 3000 environment is the download of the data base schema file and the 4GL code - and, of course, MicroSpeedware on the PC. Once on the micro, the data base is generated via a data base generating utility and the 4GL code is interpreted by the SPEEDWARE intepreter. The requirments of the micro are a minimum 512 meg storage and a hard disk to handle the 4GL interpreters and utilities.

There is a certain amount of functionality that cannot be duplicated on the IBM compatible micro. Given that the micro does not have more than one page of terminal memory, certain screen jumping functions are not available. Secondly, the use of function keys is diminished on the micro - this is an attractive feature in the 3000 environment. Thirdly, unlike the 3000 environment, external language subroutines cannot be called. There are also certain data types that do not perform well on the micro such as data items defined as floating point integers (ie: J1, J2 fields).

The most significant shortcoming of the 4GL is the fact that it does not handle exponentiation in it's calculation routines. This posed an inconvenience in the development phase as many of the calculations performed were either present value or annuity functions. To handle this, it was necessary to write looping routines. To complicate matters further there is a calculation involving determining the Nth root of a number. While the calculations could be performed in the report language of the 4GL, the lack of the exponentiation function added overhead and processing time to the reports that require present value or compounding rate calculations.

Regardless of the difficulties, the ability to design and deveiop

a system on the HP3000 and run it without modification on the micro with everything including an Image compatible database, is a tremendous time saver.

5.0 The Expert Component

With the client's information in hand, the task of the expert system is to use the same considerations in defining an optimum savings and investment strategy that the human expert uses. The mandate in the system design phase is to convert the expert's decision process into a set of rules and relationships that can be incorporated into the expert system.

All expert systems are goal oriented in that they are trying to solve a problem. In this case the problem faced by both our human expert and our expert system is "How do I structure a plan for this individual that best allows him to meet his financial goals with a minimum of pain?" It is not good enough to propose just a solution that works, we must select a strategy that takes into account a client's fears, goals, loves, hates, desires and more.

5.1 Knowledge Engineering

Knowledge engineering is the process of determining how it is our financial planner makes his decisions when he is preparing a strategy or plan. At this stage two Brant knowledge engineers have met with the financial planners about a dozen times in trying to distill only the rules of thumb that the expert is using.

Before getting into specific examples of the process and the rules it is worth reviewing a few basic "rules" about knowledge engineering.

The first thing that Brant's knowledge engineers discovered was that, unlike a conventional systems analyst role where the end user has too many suggestions and you must weed out the deliverables, our expert had the opposite problem. He appeared to have nothing to say.

Our financial planner explained up front that he didn't really have rules, but rather made his decisions intuitively based on years of experience in "reading" clients personalities. We were informed that there was not a list of rules or procedures that he followed to make his decisions.

This response is actually a well documented "pseudo-truth" that

has been addressed in many Expert Systems text books. The problem has been termed the "Expert Paradox", where psychologists have discovered that the better your expert, the less capable he or she will be in describing how they make a decision. It has been determined the experts use what has been termed "compiled knowledge", where over the years the individual rules or conditions in a circumstance become compiled so that the expert actually does not use basic rules but can jump straight to the intuitively correct answer. The knowledge engineer's job is to decompile the knowledge into its composite rules.

Because the expert does not even recognize that he is using compiled knowledge, it is useless to ask him or her "What route did you use in this instance?" The expert honestly may not know what he had to do. What has become fairly standard procedure in this type of process was to provide the financial planning expert with example after example, scenario after scenario, and ask what he would do and why. From these actions or responses our knowledge engineers distilled the necessary rules for the expert system.

5.2 Rules from the Expert

An example of the process and rules from the knowledge engineering stage can be provided based upon one aspect of the decisions that our expert would make; specifically, determining the risk tolerance of the individual or family in respect to their investment portfolio.

In our discussions with the expert on how risk tolerance played into his decisions about portfolio structuring, we asked how he picked a factor. In the "Fact Finder" on the client, the client had been asked to rate his own risk tolerance from 1 to 9, where 1 was extremely conservative (ie: he would only invest in guaranteed investment certificates in major banks that were insured) and 9 was extremely daring (ie: all of his money was in penny stocks of the Vancouver Stock Exchange or with bookies across the coutry).

The first response from the expert was that he used the number provided by the client to determine risk tolerance. When asked what he did with a similar number provided by the spouse of the client, he informed us that he took that into account as well. We discovered that he looked at such factors as who made the investment decisions, who appeared more knowledgeable about investments, who was the primary breadwinner and several other

factors before he selected a risk tolerance that may not relate to either the spouse's or the client's selection.

A similar process occured when we discussed how the selection was made for the family's after-tax income after retirement. It turned out that the client was asked this question directly, but in almost all cases this number was ignored. The expert asked a wide variety of lifestyle questions that were each broken down into composite numbers. The planner always selected the income requirement figure that was largest, regardless of how it was arrived at. In almost every aspect of the knowledge engineering process we discovered that a myriad of "invisible rules" became apparent once the questioning got underway.

There may be dozens and dozens of rules and relationships that affect the selection of a risk tolerance. From these relationships that were derived from examples or case studies, generalized rules were determined.

```
risk tolerance    (RT)if
client tolerance  (RT)and
spouse's tolerance (RT).
```

In other words, if the client and spouse express similar tolerances, go with it. If the client is also the primary breadwinner, we go with his or her specified tolerance.

6.0 Conclusion

At this stage we have completed the 4GL system and are in the process of refining the expert system. We are pleased with the implementation and excited by the success of our first hybrid system of 4GL and 5GL systems.

HOW TO DEVELOP NEW APPLICATIONS: A STRATEGY

MARK WALLACE
ROBINSON, WALLACE & COMPANY
11693 SAN VICENTE BOULEVARD
SUITE 168
LOS ANGELES, CA 90049

I would like to discuss a strategy for developing new applications. First of all, I'd like to distinguish between two different classes of software application that you might you want to develop in your shop. I'd like to make that distinction based on the size of the application. The kind of application that I'm going to be talking mostly about is of a size that I call "industrial strength." For example, like industrial strength Drano cleaner.

These applications are relatively large. Let me give you an example. AC Sparkplug in the Detroit area is the manufacturer of sparkplugs for General Motors and also sells to the after-market. They have a data base schema which by itself is over 80,000 lines. That is to say, when you run the DB schema to compile their data base description, the listing if taken to the printer would be over 1,500 pages long. That's an industrial strength application.

Another example would be airline reservations for American Airlines or Eastern or United; clearly industrial strength. At the other end of the scale, we have what I call the toy or tinkertoy-sized application.

The example that I will use of a tinkertoy-sized application is the company bowling league report. Once a week, in the evening, some of us go down to the bowling lanes and we bowl in a team league. Every week a report gets drawn up showing the scores of each individual and the handicap and so on and so forth. If we want to automate that, that would be tinkertoy-sized application.

Now, why is it important to illustrate the difference between these two? The reason is that a very different approach is required if you're going to be building an industrial strength application. The same strategy that works for the tinkertoy size may very well lead to a disaster if you try to apply it to the industrial strength size. For the tinkertoy size you may be able to simply go ahead and write something down and see how it works. If it doesn't work out, throw it away and start over again.

But that's not going to be too good an approach if you're building an industrial strength application. The next question is, why is that? Why do we need a strategy? Let me draw some analogies from other areas. How would you buy a computer? How would you select a 4th generation language to use in your shop? How would you choose a school which to attend? How would you build a house that met your needs? I'm going to come back to each of these in a minute and attempt to show that some planning is definitely called for.

Last year, when I discussed the problems with prototyping, I made a general statement to the effect that vendors or vendor representatives sometimes encourage customers to use their 4th generation languages as a substitute for planning. I made that statement in general because, first of all, I believe it to be the case, but secondly, there really wasn't anything in writing for me to point to. That situation has changed.

I have a piece of advertising literature that I picked up from one of the 4th generation language vendors and I'd like to quote from that document for you. It suggests that the application development cycle now has been pretty substantially changed due to prototyping with their 4gl.

First of all they talk about the old way of doing things. The old way started off with a feasibility study and that's where you decide whether or not the system is feasible, that is to say, cost effective. Whether the benefits it will bring are such as to justify the cost of building it. That's the first step in what they call the traditional approach. The next step is a detailed systems analysis. Next step is a data base design. Then come the program specifications including the screen and report layouts

Then the coding, then the testing of the program, and finally documentation. That's what they call the traditional approach. That's not too bad. What I'm going to be recommending that you use is pretty similar to that. I don't defer the documentation until the end, but by and large what they've outlined is a pretty reasonable statement of the traditional approach. On the other hand, what do they recommend.

Here's what they say: prototyping with their 4th generation language, I won't mention it by name. First step, build the system. That is where we start. Okay. We've left out a few things, like the feasibility study. The feasibility study was where we decided whether it was cost effective to even do it or not. All right. Now how are we supposed to dispense with that just because we've got a 4GL? That is a mystery to me. But anyway, that's their first step.

Second step: show the system to the user. Third step: modify the system according to user's comments and then, finally, repeat from step one above until the users are happy. Now they claim that the prototyping approach is only feasible because of the reduced development time and I agree with that, but I would still say that for an industrial strength application on the 3000 they have not reduced it anywhere near enough.

Let's go back and look at that approach or something similar to it as applied to the four cases that I cited previously. How would you buy a computer? Well, if you take their approach, you just order one. Don't do any planning, don't even decide whether you need a computer or not, whether it's cost justifiable, just get one in and see if people are happy with it. If they're not, throw it out and get another one, trade in your HP for an IBM or a Vax or trade in your Vax for an HP. Just keep going through them until you get one that everybody's happy with.

How would you select a 4th generation language? Same approach. Get one in, write a bunch of programs, if people don't like the language, throw it out and get another one.

What about choosing a school? Suppose you're from Minnesota and you want to study marine biology. Well, go to a school. Go to the University of Minnesota. If it doesn't work out try the University of Nevada in Las Vegas in the middle of the desert. If that doesn't work out, give Kansas a shot. If that doesn't work out, try another one. Finally, you enroll in the University of Hawaii or University of Miami and you find a good program.

How would you build a house or a building to meet your needs? Well, don't do any drawings or plans. Suppose I was an architect and you hired me to supervise the construction of your new dream house and after a 5 minute conversation I said, "Well look, you've got the lot Let's don't waste any time, meet me there tomorrow morning." And you do that and I show up with a work crew and a steamshovel and we start digging a hole in the ground.

I think you would recognize that that is a little bit premature, that building the house and if it doesn't work out let's knock it down and start over again, that's really not the best way to go. It might work if you're building a dollhouse with tinkertoys but if you're building the Empire State Building and you propose to erect it and then if the users are not happy, then we'll knock it down and try version two, that really is a bit of a problem.

You know, if you want the extreme example, you've got the case of the space shuttle where that was launched without adequate preparation and clearly the users were very unhappy with the most recent results. So what are we going to do, just go back and put another one up and see if that works out? The point I'm trying to make is, that if the project is of industrial strength size you're going to need a more disciplined approach, otherwise many things can go wrong.

The general idea as to what's going to go wrong if you don't have a strategy is that you're simply groping for what it is the user actually wants. And the computer industry case histories suggest that prototyping is an especially bad way to grope for user needs in a situation where they are not previously well understood, and where the project is of industrial strength size. It's going to be very ineffecient because the minute you have to redo the user needs the entire prototype can be thrown out, and, potentially you have to start over from scratch.

Well, this idea of prototyping is simply one specific example of implementing the software too soon. That is not something that's new to 4th generation languages. There's a phenomenon known as WISCA, stands for Why Isn't Sam Coding Anyway?, and that phenomenon was named by a man named Gerald Weinberg, author of the "Psychology of Computer Programming."

He named that because it occurs so often in the field. It occurs because the project manager believes that the only productive work is the actual writing of the program and that anything prior to that--design, analysis, interviewing the user, is simply some kind of ritualistic waste of time that we go through before we are ready to get down to the real nitty-gritty.

Another reason that we have implemented to soon, historically, is because management indulges in an activity that I called backward "estimating" and I say in quotes because it's really not estimating. What they say is something like: We'd like this software to be ready on February 1st of next year and we feel that it's going to take 4 months to program and therefore you had better start programming on November 1 regardless of where you are in analysis or design. So even though you don't know what the user wants or have any strategy for solving the situation, just start programming, because we think it's a 4 month programming job.

That is not the best way to build a system that satisfies the user's needs. But temptation to start implementing too soon is worse with the 4th generation languages. Ideally, you would think that management would recognize more time should be available for upfront planning because the implementation will go so much faster. But that seems not to be the case. First of all, if managers themselves did not use 4th generation languages, if they come from a 3rd generation language background, they won't be familiar with how fast the programming can go.

Number two, they may still have fallen into the trap by the vendors that you can just go for broke and build the system without doing any advance planning. So, in summary, why do developers rush into the implementation phase? I feel that most of the time it's one of three reasons. The first one is because we are more afraid of being late than we are of being wrong.

The second reason is what a friend of mine named Tom DeMarco called hysterical optimism. That's where we think "it's can't happen to me." We can launch the space shuttle, nothing's going to go wrong. Hysterical optimism is the idea that you can pull something off without having put in the work that normally would be required to ensure success.

The third reason is an area that is not the fault of the data processing department, and that is lack of user cooperation. Sometimes the data processing team will be given an assignment and will not be allowed to communicate with the end users. Either the users themselves will refuse to provide data on their needs or top management will forbid the computer people to go talk to the users because it would distract the users from their day-to-day work.

Now that's a shortcut to disaster. I feel that developing a requirements specification has to be a 50/50 proposition between the systems analyst and the end user. If the user is not willing to play an equal role then the resulting product is almost guaranteed not to be satisfactory unless the application is so simple, like the bowling league report, that it can be knocked out in an afternoon.

Let me give you some historical support for the problems people have run into by prototyping without the proper background. First of all from Bankers Trust Company in New York, as reported in ComputerWorld of June 4, 1984, some quotes: "prototypes were being misused, acrobats (an acrobat is someone who can make your 4GL stand up and jump through a hoop) attack requests without proper analysis, and that led to poor design and documentation and to data integrity problems.

In this gentleman's view, a prototype is a full working representation and not something that is thrown together in an afternoon. The worst danger of all he refers to is "users were losing interest in confirming systems specifications since they always assumed the prototype could be changed." That's a disaster. It's hard enough to get the user involved as it is. If you give them the impression that they don't need to be involved because you can just magically change the entire system because of the speed of your 4GLs, you're going to be bringing down a lot of grief on your head if you're trying to build something that's industrial strength.

Here's another case study from Hughes Aircraft as reported at the Structured Development Forum Conference in February, 1985. Prototyping, the author concludes, is not a substitute for analysis and design, number one. Number two, there is too much emphasis on details of form and screen layouts. This distracts attention from essential policy requirements. Number 3, prototyping is dependent on a preexisting stored data model. That's a critical point. If you do not have an existing data base containing the data, if you're trying to determine what a new data base or set of files should look like, prototyping is a terribly inefficient way to do that.

Number 4, there is always the temptation to "enhance" the prototype into the final system if either the project is late or politics rears it's ugly head. Number 5, prototyping does provide valuable assistances in defining the man/machine interface. What should the screens and forms look like? How should we navigate through them? These are important issues when we get to the design and implementation stage, but not before that.

So the problem is that with these 4th generation languages now we have to deal with both WISCA: Why isn't Sam Coding Anyway?, and WISPA: Why isn't Sally Prototyping Anyway? Both of those situations are dangerous.

Now having called your attention to the problem, what do I propose for a solution? The solution is an application development strategy. That strategy starts with a phase called the Blitz. The Blitz is a very fast pass over user requirements. It should last less than a week for any system that you're building on an HP3000. The Blitz is not a complete model of the requirements but simply a preliminary version that's used for management purposes to plan the rest of the effort.

Following the Blitz we do a detailed requirements specification and simultaneously we do a data or information modeling of the data structures that are going to be required. The results of those two activities feed into a design step where we do a detailed data design and, if we're going to implement in COBOL, a detailed software design. If we're using a 4GL, we can bypass the software design because the functionality will be built into the 4GL.

Following that, we can implement the system and subject it to testing. If it passes the test, it then is acceptable for installation. I want to look at some of these steps in more detail. First of all, the specification or requirements definition phase. That is also known as systems analysis. The purpose is to establish what the system must do. How to do it is deferred until design time.

Again from Bankers Trust, their conclusion, "What is required is a functional specification, in fact a project cannot proceed to the next stage without it." Another quote from May '84 ComputerWorld, "The lack of complete and correct specifications is the problem." Which brings us to an observation known as Gordon's Law. Mr. Gordon stated that "If a thing is not worth doing at all, it's not worth doing well."

Another way to phrase that is, "Pay me now or pay me later," from the television commercial. Meaning that if you don't pay up front to have your engine maintained by doing things like changing the oil, you're going to have an expensive rebuilding job on your hands. You can pay for a lot of oil changes with what it would cost you to have an engine rebuilt.

It's the same thing in data processing. The cost of fixing an error in the specification, if that hypothetically would be $100 for your system, the cost of fixing it in the design would be $1000. And the cost to fix it once the programs are written would be $10,000. So correcting problems up front is a lot more cost effective way to go.

We're going to suggest that, in order to do this, you build a model of the user's requirements. A model has been defined as "a description or analogy used to help visualize something that cannot be directly observed." That's from Webster. I would suggest that your new system cannot be directly observed yet because you haven't built it.

So we're going to build this model as a way of capturing the user requirements. Now, a model necessarily must omit some aspects of the new system, otherwise it would be the new system itself. The structured analysis model that we are proposing omits things like screen formats, report layouts, physical storage details (e.g., packed or zoned) and the flow of control from one screen to the next or when do we trigger a report and so on. Unfortunately the prototyping model that is typically built with a 4GL focuses on exactly those details. That is very premature when you're simply defining requirements.

Now I'd like to talk a little bit more about the idea of the essential requirements for a system and the concept of the Blitz. First of all, let's take a look at how defining the requirements and building a system has been done in the past several years. The diagram that I'm referring to is very similar to the statement that was made by the 4GL vendor in their advertising as to the traditional approach.

We start with the survey, then we do a detailed analysis, then we do a design which would include both the data and a function design in the 3rd generation environment or simply a data design in the 4th generation environment, then go on to implementation which would include programming, testing and so on.

Now, let's take a more detailed look at the way that the structured analysis phase has been performed. We start by modeling the existing system. We study the current implementation. Then the next step is to derive the logical or essential requirements. We look into the current system and we try to strip out any signs of that system that are due to historical technology and not fundamental policy requirements.

Let me mention a little war story here. IBM last year announced that it was closing its final plant that produced 80 column punchcards. This is in 1985. They're finally getting around to closing this plant. It doesn't mean that you can't get punchcards still, simply that there was not enough demand for IBM to continue producing. Why is it that systems using the latest state of the art microchips and circuitry still operate with punchcards or data bases where every record is 80 characters long? The reason is because the developers of the systems never took the intermediate step of deriving the logical or essential requirement policy. So we're going to do that.

Then we will add in the new requirements for the next version of the system and finally we will select the automation boundary. That is to say, we will decide which activities are going to be done by computer and which ones will be done by human beings because, remember, we need to focus on both of those pieces of the system.

Now, when people actually applied these traditional ideas of structured analysis, they came up with the following evaluations. First of all as far as the analyst goes, they found improved communication between themselves and the users and also between themselves and the developers but there were some problems. Problems include, on the technical side, finding the true requirements. Number one, What are the logical or essential requirements? Number two, how do we make the transition to a software design.

There was also a managerial problem. How do we efficiently apply this structured analysis technique? How do we allocate our manpower and how do we plan or estimate or control this work when we've never done it ourselves? The reasons behind these problems stem from the fact that classical structured analysis provides very strong documentation conventions, but it does not provide as detailed a set of application strategies, or how to apply the documentation conventions that it generates.

I've been calling this classical structured analysis and that may be somewhat of a misnomer, especially in the HP3000 world, because it really is only about 10 years old and many people in the 3000 area, because they have not been building industrial strength applications, have no exposure to the techniques. The point is they've been out there long enough to have been refined and re-engineered in a second pass and that pass is known as essential systems analysis.

Essential systems analysis was developed by Steve McMenamin and John Palmer. It consists of a conceptual framework upon which are built those technical and managerial strategies. The technical strategy is a set of procedures for deriving the current physical, current logical, new logical and new physical models. It is based on the assumption that there are no managerial constraints on the project. That is to say that we have as much time as we want and as many personnel as we would ever want and no constraint on money that we spend during the analysis stage. So, in other words, it's an idealized approach.

Then they have the managerial strategy, which is the technical strategy optimized by the assumption that the project is constrained. Therefore we bring in assumptions that we don't have an unlimited amount of time. We don't have an unlimited amount of people or money to spend building this requirements model so how should we proceed.

The first step is what I refer to as the Blitz. And when we Blitz a system we could basically start off in one of two ways. We could Blitz the existing system to determine the requirements that it currently is providing a solution to, or we could directly Blitz the new system, ignoring the current one, to try and build the model from scratch.

Now, which approach you take depends on which strategy will yield the most appropriate essence. For example, you need to consider how forgiving is the system's environment. If it's an air traffic control application, you're going to need to be more careful than if it's the bowling league. Which way will minimize mistakes? How different is the new essence going to be from the existing essence?

How accessible is the existing essence? If the current system documentation consists only of uncommented assembler computer program listings, it may not be the most cost effective appoach to dig into those. However, typically, on most development efforts, you will choose to study the essence of an existing system to provide the basis for defining the essence of the new one.

The reasons why are that there is an existing essence which is close to what the new one will be. Normally when we build a new application, we don't change the underlying policies to any great extent. We may change the implementation technology considerably but the business problem that it solves generally will not change very much. We have to have some policy today in order to be in business even though we're using pencil and paper or some other approach.

Also, the essence of the current system normally is reasonably accessible and we have sufficient time to study it. Finally, there is sufficient danger in trying to ignore the current system and specify the new essence from scratch. We may miss something. It would be very embarrassing to deliver a brand new system after a year of work only to hawe the user say, "you forgot function X and our old system has been doing that for 10 years." And your answer is, "Mr. User, you never told us that X was a requirement." And the user comes back and says, "but I assumed that everybody knew that." Okay. To avoid that phenomenon, amongst others that can get us into trouble, we generally study the current system first.

Now, the Blitz is a high level look at that system in order to optimize the use of project resources. We teach a 3 day class, the central point of which is blitzing and in this discussion I'm only going to have time to give you the briefest summary of that material. Basically in the blitz we establish the purpose of the system in the first step, second, we determine the context of the system. That involves specifying which functions are included versus which ones are in the outside world.

Third step is to identify the events or transactions that the system has to respond to. The fourth step is to develop the data or information model, whichever term you want to give it. The fifth step is to build models for each essential activity. There'll be a separate essential activity reponding to each event. And then the sixth step is to integrate those essential activity models into one overall essential model of the current system.

Now once we've done that, we have a reasonable understanding of the essence of the new system and of the resources to fully define those essential features. The Blitz is your crystal ball. It helps you see the future of the development effort, to know where the project is headed. However, you are not done with analysis at this time. You still need to go back and do a detailed model wherein you will provide things that the Blitz model is missing

Such things as a rigorous definition of context, detailed activity specifications, and detailed data group and data element definitions. The Blitz model is only a high level or abstract specification of the essential requirements. A detailed specification of those requirements must be completed during the rest of the development effort. Now what happens after you have specified the essential requirements.

The next step is to identify in the new physical/design stage what are the candidate processors. Are we going to use a mainframe, are we going to use a 3000, what about micros, PCs; are we going to use a canned package that we can buy from a software center, are we going to use a 4GL or do we have to use a 3rd generation language, what humans are involved? Don't forget the human processors. They are equally important a part of your new system as the computer.

You have to take the essential activities and then allocate them to the candidate processors. This piece is going to go on the 3000, this will be done by a package, this goes to the mainframe, this one is going to be done by human, and so on. Then you can do your data base or file design from the stored data model that was identified in the specification, and you can proceed to the implementation.

At this point, prototyping can be very useful, because given our stored data model and our data base for file design, we can bring a family of screens up to implement the online system almost immediately. So, that would be an excellent use of the 4th generation language. If you don't have it of course, you're going to have to go to a 3rd generation approach.

Now I mentioned that one candidate processor would be a package. Why would you bother to go through this detailed requirements definition if you're going to solve your problem by buying a package? Well, how do you know you're buying the right one? There have been several tragic case histories of companies that spent hundreds of thousands of dollars on a manufacturing package in either a mainframe environment, or I know of even some examples on HP3000s, only to throw it out and switch to another software vendor.

If you don't know your requirements before you buy the package, you are at the mercy of the software vendors. If you have a structured analysis model of the requirements, you are in control. You can say to them, here's what I need, how does your program measure up?

So, the conclusion is that 4th generation languages are great tools but like all tools they need to be used in areas where they are appropriate. Furthermore, prototyping is not a substitute for systems analysis. You need a methodology or strategy. Structured analysis has proved to be an effective tool for requirements definition and it is the cornerstone of our methodology or strategy. It should be used even when packages are considered for implementation.

The final example I want to give is from one particular 4GL on the Hewlett-Packard and it bears out our claim that a structured analysis model in effect is a prototype on paper. We want to go through our blitzing phase, then come back, do the information modeling and the detailed specifications, and then take these stored data models from those two specs and merge them and then, finally, now that we have our data base or file design, we can deliver a prototype.

What I now want to show you is a way to deliver that prototype that requires almost no human intervention. We start by doing the structured analysis on a PC or a Hewlett-Packard Vectra. From that we can extract a DeMarco-style data dictionary. We can then upload that to the 3000, and feed it into a package called LL'Spirit. LL'Spirit is a Powerhouse generator. It generates Powerhouse programs.

Those get fed into Powerhouse and up comes the running prototype automatically, consisting of all the screens for every file as well as a top level menu. So my final conclusion is one that I hope will be a happier result for systems analysts. That is that we now have time to do systems analysis because automatic prototyping will put a running system in front of the user as soon as requirements have been specified.

The Mini and the Micro
Distributed Application Development and Processing

Patrick Fioravanti
InfoCentre Ltd.
6303 Airport Road
Suite 300
Mississauga, Ontario
Canada L4V 1R8

Introduction.

The role of the Personal Computer in the HP3000 data processing installation is due for some changes. In many cases the PC is underutilized, serving as a standalone workstation for word processing, graphics, and spreadsheet analysis. These are excellent uses for the PC, and remove the need for the aforementioned services to be offered on the HP3000, however we are now in the position to fully exploit the capabilities of our PC's and further reduce the load on the HP3000 associated with application system development and execution.

With new 4TH Generation development software and data communication technology, our Personal Computers can play an integral role in system development and distributed system processing. An associated challenge concerns maintaining the security of our distributed corporate data.

We will be describing in this paper, opportunities available for integrating the use of your PC's into your everyday application system processing, and we will be considering the impact these opportunities may have on the continued security of your corporate data.

Before launching into this discussion let's look at some justifications for expanding the use of our micros.

1) Cost. The cost of PC hardware continues to fall. It is not difficult to acquire an MS-DOS based machine, with a generous configuration for a purchase price less than or equal to that of a HP video display terminal. The cost of software is another consideration. There can be no question that PC software is available for a fraction of the price of minicomputer software having similar functionality.

2) Redundancy. What happens when your HP3000 is unavailable for use? Few HP3000 installations have a spare machine that can be pressed into service when catastrophe strikes. On the other hand, it is far more likely that an installation would have a spare PC on hand, to keep the micro based applications running when a particular machine is down.

3) Performance. There is a lot of computing power in our Personal Computers waiting to be harnessed effectively. This computing power can be put to work doing some of the tasks currently undertaken by our HP3000s. If the workload is distributed wisely, then optimal use can be made of both resources. Our overworked HP3000 can shed some of its burden, possibly stalling an impending upgrade. As a result, we can improve the performance of our mini, which in turn can make our user community more productive, and make better use of our available resources.

OK, so we know we should be making better use of our PCs, removing some processing burden from our HP3000's, but we need to equip our Data Processing Department with the appropriate tools in order to accomplish this integration.

Firstly we require an application development environment that is common between the two machines. Specifically, we require the same programming language on the micro as we have on the HP3000, and the same DBMS. It is advantageous for this common programming language to be a Fourth Generation Language (4GL). A 4GL offers several benefits critical to the successful

integration of minis with micros:

- Portability. If the micro version of the 4GL is a true implementation of its HP3000 based counterpart, then applications developed using that language will be easily ported from one environment to the other. Furthermore, there is no need for the conversion of source to executable code (compile and link process) required by third generation languages, which again simplifies the porting process.

- Standardization. 4GL's standardize the appearance and behaviour of menus and screens, enabling users to work with different applications within the organization, with minimal application specific training. You can take this standardization one step further, making the micro screens look and operate like their HP3000 counterparts. Additionally, if the 4GL includes a module which generates user documentation from the source code, then this standardization benefit will extend to your system documentation.

- Speed of development. The same benefits being realized in HP3000 system development efforts using 4GL, directly apply to the micro environment.

A common application development environment provides two very important benefits:

- our system development staff can develop applications for the PC's with no new learning. All existing knowledge of the programming language and experience with creating, manipulating, and accessing Database files is transferable.

- application development activities can be undertaken either on the PC or the HP3000, regardless of where the finished product will ultimately run.

Secondly we require a mini to micro communication facility. This facility must enable communication between the machines in both directions. It will be used to transfer text files during the system development phase, and to transfer data when our users are running the application.

Consider as an example of these tools, InfoCentre's successful 4GL Speedware and associated micro based product microSpeedware. The combination of Speedware and microSpeedware provides a common programming language - REACTOR, while microSpeedware's Speedbase is an IMAGE clone for the MS-DOS environment. We will discuss later in this paper the communication facility that is available to the Speedware installation.

With these tools in place, let's turn our attention to integrating our micros with our minis. Like anything else, a good sytems integration tool will provide you with choices, rather than locking you into one fixed "solution".

For discussion purposes let's identify three approaches available to the Speedware installation for implementing micro-mini integrated solutions:

1) Standalone applications.

2) Batch Integrated.

3) OLRT Integrated.

The first approach entails identifying the machine for which a particular application is best suited, then developing the application to run only on that machine. This is what most of us are doing now. Our serious applications are developed to run on our HP3000's, frequently with little thought given to the role that our Personal Computers can play in the corporate systems strategy.

Consider however that some applications are ideally suited to run in a standalone fashion on a Personal Computer. Many shops have several candidate applications hiding in their applications backlog. These may be those system requests for a small standalone system, benefitting one department within the organization, that may not justify the allocation of HP3000 computing and development resources. Complicating matters further, the application may have several twists to it, making it unsuitable for the popular PC Database packages.

Given the toolset described earlier, the following scenario becomes possible:

Using an IMAGE application generator, such as the DESIGNER module of Speedware, develop the application on your HP3000.

Your system development staff are already familiar with the tools - Speedware and IMAGE, and the tools get the job done very quickly. An experienced Speedware user will produce results in a fraction of the time required by his COBOL oriented counterpart.

DESIGNER will generate the application, which consists basically of two components:

- an IMAGE Database

- a REACTOR specifications file containing the Speedware code for the Menus, Screens, Online HELP, Reports, and Transaction Processing programs required by the application.

The developed application can be implemented on the Personal Computer by downloading the IMAGE schema, and the Speedware code (Specifications file). This text file transfer can be accomplished with the use of the file transfer utility of your choice. The IMAGE schema becomes a Speedbase Database by compiling the schema text with the Speedbase Schema Processor. With the Database created on the PC, the user accesses the application by running microREACTOR against the downloaded specifications file.

Developing standalone PC applications as outlined above yields several benefits:

- the user gets the application he needs.

- you didn't add another application on to the load of your HP3000.

- your staff developed the application without embarking on yet another learning curve.

Ongoing maintenance to this application can be undertaken in the same fashion. Use DESIGNER to make programming or Database structure changes to the application, then download the new version (as described above). Alternatively, the application can be maintained locally (on the PC) using text editing software to implement programming changes to the Specification file, or structural changes to the schema text file.

To summarize on this standalone approach, the tools described above enable one to develop an application on either the Personal Computer or the HP3000, and then implement that application on either machine. This is made possible by the system development environment which provides a common DBMS and programming language on both machines.

In the example outlined above, an application was developed on the HP3000 for use on a Personal Computer. Once implemented, this application will run standalone on the PC, and that is where the data resides. Prudent computer system operation procedures dictate that the data should be protected and secured. The data can be protected by the implementation of rigorous backup procedures. This will be the responsibility of the PC user who should be encouraged to develop and practise these procedures. Data security poses a much bigger problem. In the absence of the familiar MPE/IMAGE umbrella, how do we prevent unauthorized access to the data resident on the Personal Computer? This problem introduces a major stumbling block which possibly limits the usefulness of this mini - micro integration approach to casual applications processing insensitive, non-critical data.

The second choice was labelled Batch Integrated. This approach enables an application system to be designed where the processing, and the data, is shared between the HP3000 and any number of Personal Computers. With this approach, the communication between the HP3000 and the micros is batched. For example, at the end of the day, or the end of the week, all of the transactions processed by the PC workstations are uploaded to the HP3000 and posted to the central IMAGE Database. At the same time, new versions of the "master" or reference type Datasets are downloaded to the PC workstations, enabling them to carry on with the next batch of transaction entry.

Capitalizing on the common development tools in the two machine environments, this type of application can be developed and implemented quite easily. The micro to mini communication (the batch transfer of files) could be undertaken with a file transfer utility such as HP's AdvanceLink.

The advantages to this approach can be:

- the processing involved in data editing and general transaction entry is offloaded from the HP3000. The PC earns its keep.

- for remote workstations data communications costs can be reduced. The workstation is not connected all day, and

the data transferred has been pre-edited by the application programs.

There are several drawbacks to this approach:

- information is not shared in a timely fashion. Depending on the application, this drawback will vary in severity.

- the duplication of data is costly, time consuming, and depending on file sizes possibly impractical.

- data security is compromised since corporate data resides outside of the realm of protection offered by MPE and IMAGE. This problem is compounded by the duplication factor mentioned above.

With the Batch Integrated approach to mini - micro integration we are still faced with the problem of protecting and securing the data which resides on the Personal Computer. The problem is aggravated with this second approach since we would be using this approach to tackle larger, more complex applications (and hence more critical data) than with the Standalone approach, and because there are potentially many copies of this unsecured data. We can partially offset these concerns with the consolation that the data left unsecured on a Personal Computer at any point in time is but a snapshot of the entire Database, and that the central IMAGE Database which contains the whole picture can be secured and protected.

The third approach we called OLRT (On-Line Real Time) Integrated. This solution is similar to the Batch Integrated solution with one very important difference. By introducing a transparent networking mechanism called Remote Dataset Capability, we can eliminate the need for data duplication, and do the mini to micro communications in real time.

Remote Dataset Capability is defined as: The ability within a Database to define a Dataset which is physically resident on a different CPU, and to access this Dataset in a fashion that is transparent to the application program.

This approach involves designing an application that will be distributed across any number of Personal Computers connected to an HP3000. As the application designer you choose where the processing will be done (all on the PC's, or shared between the PC's and the HP3000), and where the data will reside (all on the HP3000 or shared between the PC's and the HP3000).

Those Datasets that are to reside on the HP3000 in the central IMAGE Database are identified to the Personal Computer as a "Remote" Dataset. When the Personal Computer user is running the application, the micro and the mini communicate in a fashion that is transparent to the user. Any Database transactions involving the remote Dataset (reads, writes, deletes or updates) are passed to the HP3000 where the appropriate IMAGE intrinsics are executed and the results returned to the micro.

Using this arrangement, the Personal Computers are connected to the HP3000 via a terminal port. The HP3000 treats the port as an I/O device as opposed to a Job/Session device. A number of PC workstations can share the same port.

To illustrate with an example, consider an Order Processing application where the actual entry of customers' orders is to be distributed across a number of PC workstations. An IMAGE Database could be developed to support this application consisting mainly of the reference or master Datasets such as Customer and Product. The PC workstations would each have their own local Database, perhaps matching the structure of the IMAGE Database. The local Databases would define the Customer and Product Datasets as Remote Datasets. As orders are entered at the PC workstations, lookups and validations for Customers and Products are processed against the centralized copy of those files maintained in the IMAGE Database. The resultant order transaction records would be stored locally at each workstation. If desired, at any time the transactions could be uploaded from the PC workstations and consolidated within the IMAGE Database, where they would be available for centralized reporting.

This On-Line Integration approach offers these benefits:

- distributed processing, shifting some of the processing load from the HP3000 to the PC's.

- information that is physically resident on the HP3000 can be read and updated immediately.

- there is no data duplication.

- the critical corporate data can be left on the HP3000 where it is protected by the security provisions of MPE and IMAGE. Access to the data is controlled by the application software.

What is the appropriate mini - micro integration approach? All of the approaches offered in this paper support application development activities on either machine. Depending on the approach, the system processing will take place exclusively on the PC, exclusively on the HP3000, or will be shared between the two resources. Depending on the approach, the data will be physically resident on the PC, or the HP3000, or both and sometimes with duplication. Where the data resides can have serious impact on the security of the data. Obviously the choice of approach must be determined by your resource availability and your application needs. To add to the decision making complexity, consider also that the choices are not mutually exclusive. A mixture of approaches may be appropriate for the various components of a specific application.

Summary:

As this paper has pointed out, it is the case that we have at our disposal the technology to effectively integrate the use of our Personal Computers into our HP3000 based application system development and processing. In order to achieve this end of mini - micro integration we need to embrace the tools required: a 4GL programming language common to both the MPE and MS-DOS environments, a common DBMS, and transparent micro to mini communications. With these tools in place, we can design applications, distributing the processing and the data across a network of MPE and MS-DOS machines, with a tremendous amount of application design flexibility. The tools make it easy. The challenge is to maintain the high degree of data security that we have become accustomed to with the security provisions offered by MPE and IMAGE.

Is There Life Besides IMAGE?

May Kovalick

Hewlett-Packard
Information Technology Group
Cupertino, California, USA

Introduction

"Knowledge is of two kinds: we know a subject ourselves, or we know where we can find information upon it." - Samuel Johnson, 1775.

Samuel Johnson is the originator of the first English language database. We all have a copy of a similar database on our desk today - the dictionary. In 1775, to have that database at one's disposal was rare and privileged. Today, the possession of such is commonplace. The body of knowledge that existed then was minuscule compared to that of today. Nonetheless, the amount of knowledge or information that we know ourselves is becoming smaller compared to that we have access to. Fortunately, technology is on our side for managing the ever increasing amount of data.

For most of the HP3000 users, the IMAGE database management system has been the tool for storage and retrieval of data for many years. With the advent of more sophistication in the usage of databases, and the increased emphasis on productivity and flexibility, the offering of the relational technology on Hewlett-Packard's family of computers is a must.

The new ALLBASE product is Hewlett-Packard's advanced database management system for the Hewlett-Packard Precision Architecture systems for both the commercial and technical markets. It combines both relational and network model data access in a single product (See Figure 1). HPIMAGE is the enhanced version of the recently introduced TurboIMAGE database management system. HPSQL is the relational interface that uses the de facto industry standard SQL (Structured Query Language) for both data definition and data manipulation. The co-existence of both interfaces in one database management system allows the user the flexibility to choose the appropriate data model for each database application.

The rest of this paper will concentrate on the HPSQL interface. I will give an overview of the basic data definition and data manipulation functions of HPSQL, describe the major components of HPSQL, highlight some of the features that are unique to Hewlett-Packard's SQL, and give a preview of future directions of Hewlett-Packard's database product offerings.

HPSQL

HPSQL is a family of relational products available on the different Hewlett-Packard computers:

- HPSQL/V is available on the Series 70 and all previous MPE-V based HP3000 systems.
- HPSQL/XL (a component of ALLBASE/XL) will be available on the HP3000 900 series systems.

- HPSQL/HP-UX (a component of ALLBASE/HP-UX) will be available on the HP9000 800 series systems.

- HPSQL/300 will be available on the HP9000 series 300 systems.

HPSQL provides all the advantages of relational technology. It is easy to learn and use, and provides a set of powerful commands for data definition, data manipulation, security and authorization control, transaction management and database administration.

The implementations of all the HPSQL products are highly leveraged and are totally compatible with one another. Customers may develop applications on one system and be able to move those applications to another without source code modifications.

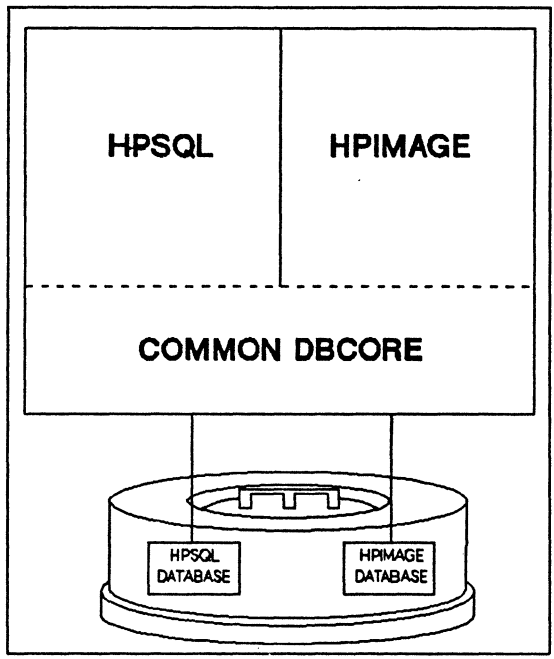


Figure 1. ALLBASE Product Structure

Data Definition

An HPSQL database is a collection of database objects consisting of tables, views and indexes. A *table* consists of columns and rows, and may be created with the CREATE TABLE command. The relationships among tables are determined, *not* by explicit pointers, but by the data values in the columns of the tables themselves.

A *view* is a virtual table derived by a data manipulation statement from one or more physical tables or views. Views provide some data independence from certain changes to the database. If the user wishes to condense multiple tables into one, or split one table into

many, views can protect the user from modifying programs that read these tables. Another use of views is for security. If the user wishes to restrict access to data based on content, a view may be defined to perform value based security checking.

A user may create *indexes* on a table to reduce the time it takes to retrieve data from it. Unlike HPIMAGE application programs where an access path has to be explicitly specified for each database retrieval operation, HPSQL application programs do not specify indexes to be used for the query commands. HPSQL automatically analyzes data access requests in terms of the indexes available and chooses to use the one that will optimize performance.

Tables, views and indexes may be added or deleted dynamically to the database while it is in use. An existing table may also be expanded by the addition of one or more new columns. These dynamic data definition capabilities of HPSQL allow the users to restructure the database easily to reflect changing needs.

Data Manipulation

HPSQL provides the SELECT, INSERT, UPDATE, and DELETE commands for data access and modification.

The SELECT command in HPSQL allows users to perform the three basic relational retrieval operations of selection, projection and join. Selections produce a horizontal subset (of rows) in a table that satisfies certain criteria. Projection produces a vertical subset (of columns) in a table. Join combines data from two or more tables by matching values in a column of one table with values in a comparable column in the other tables. In addition, the SELECT command supports arithmetic expressions, sorting, grouping operations and a set of built-in aggregate function such as MIN, MAX, AVG, etc.

The INSERT command allows users to add one or more rows to a table. The UPDATE command allows users to modify values in one or more rows of a table. The DELETE command allows users to delete one or more rows from a table.

Using these four basic data manipulation commands, the users can easily specify what data to access or modify without having to specify how to do it.

Components of HPSQL

Figure 2 shows the architecture of HPSQL. There are five major components:

- The interactive user interface (ISQL).
- The utility package for performing database administration tasks (SQLUtil).
- The preprocessors that provide programmatic access to the database (Preprocessors)
- The parser and query processor (SQLCore).
- The kernel database access module (DBCore).

I will describe these components from the bottom up.

DBCore

DBCore comprises the command executor and the low-level services. The command executor is a single, cleanly defined interface point that accepts commands from the interfaces above and calls the low-level services to perform the tasks. The low-level services include all the routines to store, access and update data, and provides transaction management, multi-user concurrency control, logging and recovery.

DBCore does not presume the relational, nor any, model of data. It handles data in a model independent manner. It is this feature that allows both the HPSQL and HPIMAGE interfaces to be built on top of it.

This layer of the software is most dependent on the operating system, especially the modules for accessing files. However, these OS-dependent routines are well encapsulated and insulated, thus making the operating system transparent to the HPSQL and HPIMAGE interfaces above. This modularity allows ALLBASE to be easily portable to the different operating systems.

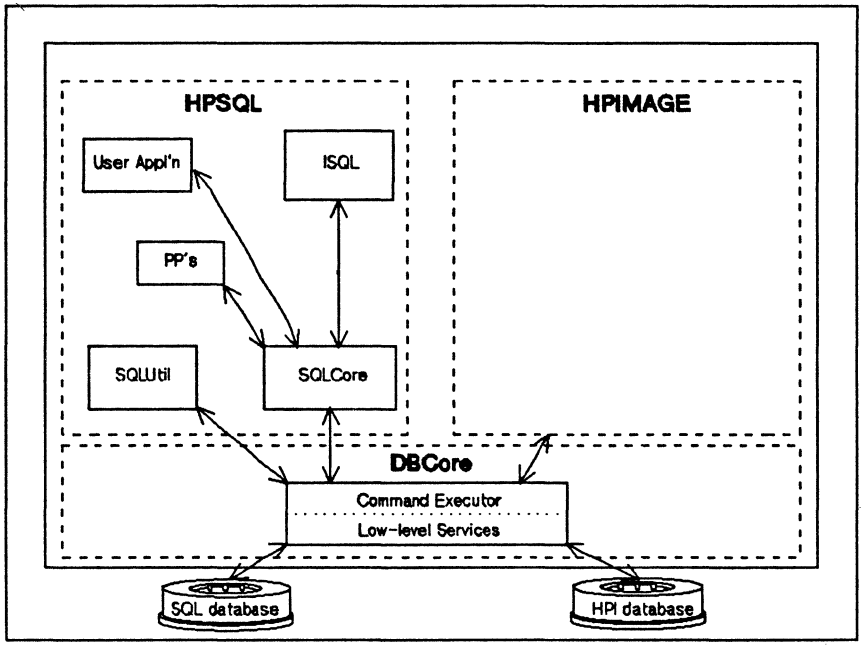


Figure 2. HPSQL Components

EW

SQLCore

SQLCore comprises the parser and the query processor. The parser parses SQL commands and generates command trees which are "flattened", i.e. all the internal machine-dependent pointers are replaced by a machine-independent linear representation. The linearized command trees are then passed on to the query processor.

The query processor performs protection validation, query optimization and access path selection. It also makes sure that the tables and columns referenced in the query are valid, and generates the appropriate DBCore commands to execute the query.

The set of powerful data manipulation commands allows the user to specify what data is to be accessed or modified, but not necessarily how to access them. A query optimizer is contained in the query processor to look at the query and evaluate the current physical structure of the database to determine the most optimal path to access the data. This, however, does not mean that the user has no control over the performance of his queries. Because of the powerful data definition capabilities provided by HPSQL, the user can tune the performance of his/her applications by creating and dropping appropriate indexes for the tables, or by changing the physical configuration of the database.

One such example is the use of a clustering index for a table. When a row is inserted or updated into a table that is defined with a clustered index, HPSQL will attempt to place that row on the same or consecutive data page with other rows with similar key values. Because the rows are physically close, I/O overhead is reduced and performance may be improved whenever the rows are retrieved in key order.

Preprocessors

HPSQL provides users programmatic access to HPSQL databases via preprocessors. The preprocessors are programs that read the source code of user application programs which have SQL commands embedded in them. The preprocessor looks for the predefined directives (EXEC SQL) in the source programs that define access to HPSQL and replaces them with language specific calls to HPSQL. It also performs optimization for the queries and stores the predefined database commands in the database so that, when the program is executed, the preprocessed commands are executed.

There are two ways that database management systems can allow application access to a database: preprocessors and intrinsics (e.g. DBGET or DBPUT for IMAGE). There are a number of advantages in using the preprocessor approach. Firstly, a preprocessor is usually more friendly than intrinsics. It can take care of data type conversions, language-dependent and OS-dependent calling conventions, and error handling in a manner that is transparent to the user. Secondly, the preprocessor approach improves performance by allowing query optimization to be performed when the application is preprocessed instead of at run-time. At run-time, HPSQL will detect if a change in the database structure has invalidated the access strategy for any of the queries and will automatically re-process those queries for the new structure. Thirdly, there is the advantage of being compatible with other industry implementations and thus provides portability for SQL applications.

The Pascal and COBOL preprocessors are available with HPSQL/V and HPSQL/XL. The Pascal, FORTRAN and C preprocessors are available with HPSQL/HP-UX and HPSQL/300.

SQLUtil

SQLUtil provides a set of commands for the database administrator to perform various administrative tasks, such as altering the configuration of the database environment, managing database files, and backing up and restoring the database environments, etc. It can be invoked either from ISQL or from the operating system.

ISQL

ISQL is the interactive user interface that provides the user with functionally complete interactive SQL access to the data. It accepts user SQL commands, sends them to the parser, then passes the "flattened" command trees to the query processor for processing.

ISQL accepts commands from three sources: the terminal, the command buffer, or a command file. The *command buffer* is an area for holding one or more commands for the duration of an ISQL session. The contents of the command buffer may be changed, kept in a file or executed. A *command file* is a system file that contains one or more SQL commands. It may be created outside the ISQL environment, using an editor or a program written by the user.

ISQL also provides a *command history buffer* for holding the 10 most recently submitted commands. Any of the commands from the command history buffer may be listed, recalled for re-execution, or edited and re-executed.

ISQL is a useful facility for different types of users. Frequent users may use it for ad hoc retrieval and modification of data with the usual data manipulation commands. It can be a program development tool for application programmers to build test databases, and to try out queries to be embedded in applications. It is also a tool for database administrators to create and maintain databases, to load/unload data from/into external files, and to define and control physical storage for the databases.

HPSQL Unique Features

Most of you may already be familiar with HPSQL/V or the industry standard implementation of SQL and the basic functions provided by SQL. For more information, you may refer to the SQL Reference Manual.

In this section, I would like to highlight a few of the SQL features that are unique to the HPSQL product. These features include the following:

- Authorization Groups for security management
- Savepoints for transaction management
- Logging and Recovery
- Bulk Table Processing through the Programmatic Interface

Security Management and Authorization Groups

A major function of a database administrator is security management, i.e. controlling access to a database and its objects such as tables, views, etc. Since every user must have appropriate authorization in order to access the database and perform operations, the DBA can use HPSQL authorization to maintain security for a multi-user database environment.

An authority is a privilege given to a user or a group of users to access the database environment, create database objects, perform a specific operation, preprocess and run application programs containing SQL commands, or maintain the database environment. Authorization is provided for the tables, views, and resources of the database. The owner

(initially the creator) of an object can perform all operations on the object and can grant authorization to another user to operate on the object. Authorities on an object may be revoked, and ownership of an object may be transferred to another user or group of users.

For most business organizations, a database is shared by multiple departments having different types of access and operations performed on the database. For example, a personnel database may be accessed by different departments such as accounting, payroll, personnel clerks, personnel managers, etc. Each of these departments may have one or more users, and each of the departments may require different access rights to the database. In order to make it easier for the DBA to establish authorization for the database for groups of users according to their database requirements, HPSQL provides a unique feature called *authorization group*.

An authorization group is a group of users that possesses the same set of authorities. A user can be a member of any number of groups, and groups can also be members of other groups. Authorization groups may be created or dropped dynamically. Once a group is created, individual users or groups can be added to an authorization group or removed from the group. When a user is added, he or she automatically acquires the authorities belonging to the group.

The GRANT and REVOKE commands allow the user to specify a group as the receiver of the authority. These authorities then belong to the group and not to the individual members of the group. That is, as long as a user is a member of a group, the user possesses the authorities belonging to the group. If the user is removed, he or she no longer possesses the authorities of the group.

Authorization group is therefore a valuable security management feature for the database administrator to easily control database access for groups of users.

Transaction Management and Savepoints

A transaction is a unit of work specified by a sequence of SQL commands. A transaction is started by the command BEGIN WORK and ended with the command COMMIT WORK, in which case all changes made by the transaction become permanent. The transaction may be aborted with the command ROLLBACK WORK, in which case none of the changes are made to the database. Most of the commercial SQL implementations provide the above facilities for managing user transactions.

In addition to the above, HPSQL supports savepoints within transactions to allow users to rollback some of the changes in a transaction. A *savepoint* defines a set of commands within a transaction that can be aborted without aborting the entire transaction. A savepoint is defined using the SAVEPOINT command. The user can then undo the changes within a transaction since the savepoint was defined by using the ROLLBACK WORK command. Multiple savepoints can be defined within a transaction and are referred to by a number returned by the query processor in the SAVEPOINT command.

A savepoint may be used in a long transaction that does several operations, some of which might have to be rolled back. Savepoints can greatly reduce the number of transactions that have to be resubmitted because part of the transaction was unsuccessful.

Logging and Recovery

In order to support concurrency and still provide data integrity and reliability, HPSQL provides extensive logging and recovery features.

The most common form of damage to a database occurs when a user or system process fails during the execution of a transaction and is unable to complete it, thus rendering the data inconsistent. This is called a soft failure since the database is not seriously corrupted and can potentially be reparied without requiring complete restoration of the data.

HPSQL logs all changes to the data of a database in its log file. Should a soft failure occur, HPSQL will automatically attempt to bring all data back to a consistent state with the information recorded in the log file. All transactions which successfully committed prior to the crash will be recovered. Transactions which failed to complete prior to the crash will be rolled back, or undone. This is called *rollback* recovery. Rollback recovery is automatic and is always available.

A second form of failure results from a hard failure which renders the database unreadable or completely corrupted. Such failures may be due to hardware problems, such as a disc head crash or an operating system error that allows random data to be written on a table in the database.

Should a hard failure occur, it is necessary to restore a stored copy of the damaged database(s) and then roll-forward, or redo, all transactions that were committed before the hard crash and since the stored copy was created. This is called *roll-forward* recovery.

To support roll-forward recovery in HPSQL, an *archive* mode of logging is provided. When HPSQL is run in archive mode, all changes to the database are logged and the log space is never reused. To perform roll-forward recovery, an old copy of the database is restored from a backup or archive copy. The current log contains all the changes since the last backup. Using the START DBE ... RECOVER command, the database is recovered to a consistent state by incorporating all work done by transactions committed before the failure, and excluding any changes made by transactions that did not commit by the time of the failure. A date and time may also be specified in this command for recovering the database to the desired date and time.

A *dual logging* option is available in HPSQL to further enhance integrity. Two separate logs on separate media are maintained. Both logs are written for all operations. Normally only one log is read during recovery, but if an error is encountered, HPSQL switches to the other log automatically. Data integrity is maintained, provided that there is at least one good copy of each log record on either of the logs.

Programmatic Interface and Bulk Table Processing

HPSQL provides the full set of data definition, data manipulation and transaction management commands through the programmatic interface. This includes the use of host variables, indicator variables for handling null values, run-time error checking and handling, the use of cursors, and dynamic query processing for executing SQL commands that cannot be defined until run-time.

The data manipulation commands in HPSQL allow the user to insert, delete, update and select rows from a database. A single row or multiple rows can be operated upon with one data manipulation command. A cursor may be used to operate on a multiple-row query result, one row at a time. Like the cursor on a terminal screen, an HPSQL cursor is a position indicator. It allows the user to move through the multiple-row query result,

retrieving a row at a time into host variables and optionally updating or deleting the row. Reporting applications may find this technique useful.

In addition to the above, HPSQL provides a unique feature in the programmatic interface for *bulk table processing*. The user may specify an application program to retrieve or insert multiple rows with the execution of a *single* SQL command. Three bulk commands are available:

- The BULK SELECT command can be used when you know in advance the maximum number of rows in a multiple-row query result, or when the query result is not too large. For example, an application that retrieves a query result containing a row for each month of the year might find this command useful.

- The BULK FETCH command can be used to handle large query results or multiple-row query results whose maximum size is unpredictable. If a single execution of the BULK FETCH command does not retrieve the entire set of query result, it may be re-executed to retrieve subsequent rows in the query result. This use of a cursor is most suitable for display-only applications, such as programs that allow a user to browse through a query result, so many rows at a time.

- The BULK INSERT command can be used to insert multiple rows into a table. Rows are inserted from a host variable declared as an array.

In the bulk retrieval commands, the user may specify the maximum number of rows to be retrieved and where to put the data. Rows are retrieved into a host variable declared as an array. HPSQL fetches as many rows as will fit in the retrieval area (or the specified maximum number of rows, or the number of rows remaining in the query result, whichever is less). A value is returned telling the user the actual number of rows fetched.

The BULK SELECT command minimizes the time a table is locked for the retrieval operation, because the program can execute the BULK SELECT command, then immediately terminate the transaction, even before displaying any rows. Similarly, the BULK INSERT command is efficient for concurrency, because any exclusive lock acquired to insert rows need be held only until the BULK INSERT command is executed.

The set of bulk table processing commands provided by HPSQL is very valuable for application builders. It provides a set of features complementary to the row-at-a-time cursor operation commands. It allows application programmers easy access and handling of multiple-row data and query results. It also provides good performance for applications that need to handle large amounts of data efficiently.

Future Directions

Many of Hewlett-Packard's customers have invested heavily in developing database applications using IMAGE. With the introduction of Hewlett-Packard's Precision Architecture systems, many of these customers may choose to migrate their existing database applications to ALLBASE. HPIMAGE certainly provides an easy and logical migration path. However, users may want to take advantage of the relational technology to increase their productivity and to ease their programming effort.

In view of this, Hewlett-Packard plans to provide users the capability of accessing HPIMAGE data using HPSQL in future releases of ALLBASE (Figure 3).

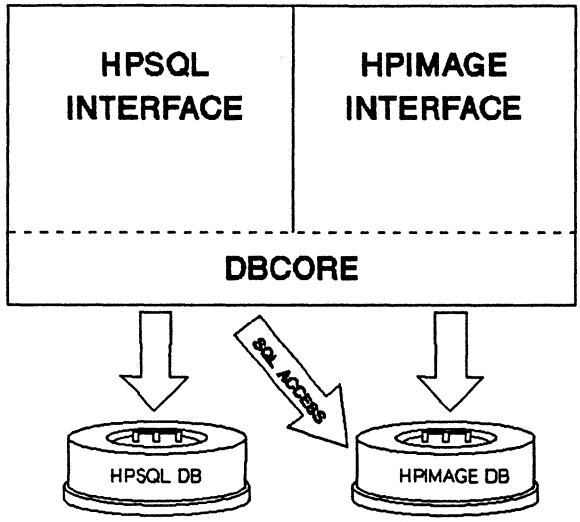


Figure 3. SQL Access to HPIMAGE Data

Users may be able to develop new applications against existing HPIMAGE databases using the HPSQL programmatic interfaces. The same data may also be accessed on an ad hoc basis using the powerful and flexible interactive query capability provided by ISQL.

This integration will thus allow users to gain the optimal benefits of both technologies without introducing data redundancy or inefficiencies into the MIS environment.

Another Hewlett-Packard long term goal is to provide its customers with the hardware and software needed to support distributed database management systems. One important step toward that goal is to provide the same powerful database management software on all computers so that data can easily be shared. By offering its customers SQL, the de facto industry standard for relational technology, Hewlett-Packard is moving toward compatibility not only between its own computer systems, but also with those of other suppliers.

Conclusions

Yes, with ALLBASE, there is life besides IMAGE. The HPSQL and HPIMAGE interfaces serve as complements for each other in ALLBASE. The architecture of ALLBASE is modular, allowing for changing architecture (at the bottom) and for additional user interfaces (on the top). The system architecture of ALLBASE provides a solid foundation to carry Hewlett-Packard's database management plans well into the next decade.

Performance Programming With HPSQL/V

Paul E. Dembry
Hewlett-Packard
19447 Pruneridge Avenue
Cupertino, CA 95014

Hewlett-Packard's newest database management product, HPSQL/V, is designed to reduce the amount of DBMS internals knowledge needed by the average user in order to achieve good application performance. For example, HPSQL/V provides a query optimizer which automatically determines the most efficient access plan for user queries. However, in order to fully realize the extraordinary power of HPSQL/V, programmers must understand some of the internal strategies used in such areas as index management, concurrency control, and transaction management.

In this paper, the basic concepts behind HPSQL/V's internals are examined along with their impact on application performance. The focus is on how to exploit HPSQL/V's inherent strengths for maximum performance. The reader is assumed to have read the HPSQL/V DBA and SQL Reference Manuals.

Product History and Structure

HPSQL/V evolved from the same base as the recently introduced ALLBASE product. Therefore, the concepts described here also apply to the ALLBASE product. Application developers can use HPSQL/V not only to develop production applications on their current HP3000 systems but also as a learning and debugging tool for their new HP3000/930 relational applications.

HPSQL/V consists of several major modules, two of which will be addressed herein: DBCORE/V and SQLCORE/V. DBCORE/V performs the actual record retrieval and storage, index management, logging and recovery, space management, concurrency and locking control, and transaction management functions. SQLCORE/V checks, parses, stores and retrieves the SQL commands, performs the authorization checking, and optimizes the access path.

DBCORE/V will be examined first with special emphasis on the locking and concurrency control systems. A firm understanding of these is essential to achieving acceptable application performance. This section will be followed by a discussion of SQLCORE/V and how it controls the effectiveness of DBCORE/V. Some observations on database table

definition and a few general overall suggestions will wrap up this effort.

DBCORE/V

A key factor in application throughput is the multi-programming level supported by a particular subsystem. A subsystem that allows only one user to use it at a time can be much simpler to design but will produce a bottleneck in the system. DBCORE/V allows up to 192 concurrent users, or "threads", at any one time. In order to support this high level of concurrency without compromising data integrity and reliability, DBCORE/V provides extensive locking and recovery features.

DBCORE/V stores user data in relations which consist of records called tuples. Each tuple is partitioned into fields called columns consisting of varying length byte strings. Relations correspond to HPSQL/V tables and tuples correspond to rows. The unit of storage used within DBCORE is called a page. It is a 4096 byte record which contains the relation's tuples along with some header information.

Locks

In any system that allows a high number of concurrent users, there is always the possibility of two or more users trying to update the same data simultaneously or of one user's partial data changes being seen by other users before they are complete. The latter is called a "dirty" read since the data may or may not be consistent with respect to the other changes.

DBCORE/V deals with this issue by using locks to restrict a user's access to data. The use of these locks is fundamental to DBCORE/V's ability to ensure data integrity and therefore the user is not allowed to bypass the locking facility, but does have some control over the types of locks used.

DBCORE/V locks at two levels, page and table, and provides three types of locks: exclusive, shared, and share-subexclusive. The first type of lock is an exclusive lock which prevents any other users from accessing a relation or data page. Exclusive locks are automatically placed on any page that has been altered by a transaction.

The second type of lock is a share lock. This lock allows other users to read but not alter the data in a relation or page. Share locks allow many users to read the same pages concurrently.

Finally, the third type of lock is a share-subexclusive lock, which allows other users to read parts of a relation but allows only one transaction to modify the relation at a time. This more complex lock is intended for users who intend to update part of the relation but want to allow other users to read the relation until they are ready to do the update.

It should be clear that an exclusive lock cannot be granted on a relation when there are outstanding share locks on the same relation and vice-versa. If this situation arises, DBCORE/V will suspend the requesting transaction until its lock request can be satisfied. Before doing this, however, it checks to see if suspending the user will cause a deadlock, a situation where a ring of users are all waiting on the user ahead of them. Consider the example shown below:

User 1:	User 2:
-Read page A	-Read page B
Is it locked?	Is it locked?
No, acquire share lock on page A.	No, acquire share lock on page B.
-Modify page B	-Modify page A
Is it locked?	Is it locked?
Yes, deadlock?	Yes, deadlock
No, wait for exclusive lock.	Yes, deadlock detected!

User 1 needs an exclusive lock on page B in order to modify it, but user 2 has a share lock on page B so user 1 is suspended behind user 2. Unfortunately, user 2 needs an exclusive lock on page A in order to modify it, but user 1 has a share lock on page A so user 2 would be suspended behind user 1. This would create a deadlock situation where neither transaction could advance. In this case, DBCORE/V will abort the lower priority transaction, undo all the changes made by it (see the section on transactions), and thereby resolve the deadlock situation.

There are two ways that an application can affect the types of locks used on its behalf. First, at table creation time, the user can specify the implicit type of locking to be used with the table. The types are PUBLIC, PUBLICREAD, and PRIVATE with the default being PRIVATE. These are related to the locks described above as shown in this table:

Table Type	Lock			
	READ	LEVEL	WRITE	LEVEL
PRIVATE	Exclusive	Table	Exclusive	Table
PUBLICREAD	Share	Page	Exclusive	Table
PUBLIC	Share	Page	Exclusive	Page

PRIVATE corresponds to an exclusive lock on the relation. As soon as a user accesses the relation for either a read or a write, DBCORE/V gets an exclusive lock on the entire relation. This eliminates the possibility of a deadlock within this table, eliminates the locking overhead for this table, and eliminates any hope of allowing users to share the table, thereby limiting the concurrency to 1. This will not be very good for multi-user throughput.

PUBLICREAD corresponds to a share-subexclusive lock on the relation while PUBLIC implies locking on a page level instead of a relation level. In terms of overall system throughput, PUBLIC tables are the best choice in most, but not all, situations. It is important to note that, at the present time, HPSQL/V does not allow the user to change the implicit table locking strategy after the table has been created.

In addition to the locking strategy defined at table creation time, the application can try to override this with the "LOCK TABLE" SQL command. This command allows the application to tell DBCORE/V to apply either a share or exclusive mode lock on the entire relation.

Share mode allows multiple users to read the relation while allowing only one user to update it. This function is useful for PUBLIC tables if the application wants to prevent multiple users from updating the table at the same time. It has no effect on PUBLICREAD or PRIVATE tables since they already have an equally or more restrictive locking strategy.

Exclusive mode tells DBCORE/V to treat PUBLICREAD and PUBLIC tables like PRIVATE tables with respect to locking.

Which locking should you use in your application? Unfortunately, this is not a simple question to answer. The figure below gives a graphical description of some of the issues involved in this decision:

```
Low                Concurrency                  High
  |-----------------------------------------|
PRIVATE          PUBLICREAD                  PUBLIC

        Low      Locking overhead             High
        Low      Deadlock potential           High
                 over time
```

As you can see, locking strategies and achievable concurrency are tightly coupled. In addition, the overhead associated with locking is also related to the locking used. Tables also have a continuum of sorts as shown here:

```
Read               Predominant Access          Write
  |-----------------------------------------|
```

Therefore, an application's locking strategy is a compromise between concurrency and overhead, in addition to deadlock probability. In general, if an application is going to read or update a large portion of a table, it is advisable to acquire a SHARE or EXCLUSIVE lock, respectively. This keeps DBCORE/V from having to manage a large number of locks. It also reduces the chance of deadlocks. One example of this is during table loading(EXCLUSIVE lock) or unloading(SHARE lock). However, it will probably reduce the achievable concurrency. On the other hand, applications which only read or update a small part of a table at a time should create their tables as PUBLIC tables. The additional locking overhead will be overshadowed by the increase in concurrency.

Transaction Support

A unit of work is called a transaction and consists of any number of SQL statements enclosed within a BEGIN WORK and either a COMMIT WORK or ROLLBACK WORK statement. Once a transaction starts, it either commits(COMMIT WORK), in which all changes made by the transaction are made permanent and visible to other users, or it aborts(ROLLBACK WORK or deadlock), in which case none of the changes are made. For this reason, the transaction is considered the atomic unit of recovery. This property of transactions is guaranteed for both user sessions and system failures. It is important to note that all locks acquired during a transaction accumulate throughout the transaction and are released only when the transaction is committed or aborted.

This last point is perhaps the most important thing to remember about transactions. As noted in the above discussion about locks, anything a transaction does with respect to data in a table requires a lock.

Reading through a table, for example, will apply share locks to the data pages and will prevent any other transaction from updating that page.

The concept of a transaction can be applied very effectively within an application. For example, any data that is accessed during a transaction is guaranteed not to change during that transaction. There is never any need to refetch a record to make sure that it has not changed during the transaction. Until the transaction commits or rolls back, any data that it touches is guaranteed to remain consistent!

Also, DBCORE/V supports SAVEPOINTS which are markers within a transaction that allow the user to undo part of the transaction without going back to the beginning. This allows an application to explore different "paths" without having to worry about explicitly undoing its changes if it needs to back up and try a different approach. For example, a bank debits a customer's checking account for $100 which overdrafts the account. The customer has overdraft protection so the bank sets a savepoint and tries to transfer $100 from MasterCard to the checking account. This hits the customer's credit limit on MC and so the application rolls back to the savepoint, thereby undoing the MC debit without undoing the original checking account debit. Declaring a new savepoint, the application checks Visa-it is below the credit limit so the transfer occurs and the transaction is committed. With savepoints, the transaction did not have to start all over again, it could undo just part of its work.

It is good practice to keep your transactions as small as possible in order to minimize lock conflicts between users. Also, in the case of a deadlock, the transaction may be aborted. The cost of redoing the transaction from the beginning needs to be considered when designing transactions. DBCORE/V supports the idea of a transaction priority from 0 to 255, with 0 being the highest priority. The application specifies this in the BEGIN WORK statement; the default is 127. DBCORE/V first checks the priority of the two transactions causing the deadlock and aborts the one with the lower priority. If the cost of a deadlock abortion is high for a particular transaction, then the application should specify a low transaction number.

Terminal reads should be avoided within transactions since the user may go to lunch at that point. If they are necessary to the application, then a timeout should be set before the read. On the other hand, if the transaction uses only PUBLIC or PUBLICREAD tables which are never updated by other users, then there is no possibility of one transaction locking out another.

Indexes

In order to avoid serially searching through relations on every access, HPSQL/V allows users to create indexes. DBCORE/V supports B-tree indexes on relations. It also has two options for indexes: clustering and unique. Indexes can be created or dropped at any time and can have up to 15 column keys. Clustering indexes can improve the efficiency of sequential processing for queries that use the index. This is because DBCORE/V will place new tuples physically near other tuples with similar key values whenever possible. Creating a clustering index has no effect on the existing data in the relation, only on new tuples. Also, there can be only one clustering index on a relation.

Unique indexes are a special type of B-tree index which prohibit tuples with duplicate keys. These can be used very effectively by applications that deal with unique fields such as Social Security numbers.

An index can be defined as unique, clustering, unique and clustering, or none of these. In this last case, duplicate keys are allowed and tuples are inserted into the relation wherever there is room. In addition, a relation can have any number of indexes defined on it.

The advantages of indexes are not free, however. Every time an index key column is updated, deleted, or inserted, DBCORE/V must adjust the index to reflect the new value. The more indexes that contain the altered key column, the more overhead will be generated. Also, DBCORE/V needs space in which to store the index. Each index entry contains the key value(s) along with the tuple identifier of the actual data tuple. Thus, the greater the number of columns in the key, the more space that will be required to store it. For these reasons, users must be careful when developing their index strategy.

All large relations should generally have at least one index defined on them corresponding to the most frequently specified key(s). In addition, the table should be loaded in that index's key order. Applications which do a great deal of bulk reads based on that key will benefit from this ordering. Also, after many inserts and deletes, the index pages may become very sparsely populated even though the number of index levels remains high. This condition can be identified by comparing the "cluster count" with the "number of pages" and "number of rows" columns in the SYSTEM.TABLE relation. This procedure is described in the HPSQL/V DBA Guide.

Note: Even though there is an index on a relation, HPSQL/V may decide not to use it. This is further explained in the SQLCORE/V section on queries.

Storage management

DBCORE/V stores data and indexes in files called DBEFiles. These are privileged MPE files with a 4096 byte record size, the same size as the data page. A relation's data and index(es) are stored in a logical group of these DBEFiles called a DBEFileset. DBEFiles can be added or removed from the set at any time and can be designated as table, index, or mixed storage. Since relations are created within DBEFilesets, their data is spread throughout all the DBEFiles, of the correct type, in the set.

There is a tradeoff between potential application throughput and storage efficiency. MIXED DBEFiles contain both table and index data and can be very useful for small tables. This is because DBCORE/V will try to place table data and index keys on contiguous DBEFile pages. MIXED DBEFiles also offer better storage efficiency since the user does not have to allocate separate DBEFiles for tables and indexes.

If the table is large and spans multiple DBEFiles, it is more advantageous to define separate TABLE and INDEX DBEFiles and then move them to different spindles using SQLUTIL/V. This can reduce the disk workload by spreading index and table I/Os across different disks. For example, if an index is used during query execution, then the index I/Os will go to one spindle and table I/Os to another. This will reduce the number of disk reseeks. Otherwise, applications will queue up on the disk first to get the index and then again to get the data.

For maximum flexibility, users can define a DBEFileset for every relation. This way, the DBEFiles in the set will contain only the data and index(es) for that relation making it easy to know which DBEFiles should be moved to different spindles.

It is also very important that the system DBEFiles and the log file(s) be placed on the least accessed device(s) on the system since these DBEFiles are very heavily accessed for almost all queries.

SQLCORE/V

SQLCORE/V acts as the HPSQL interface to DBCORE/V. It provides all the system authorization checking, query handling, and, most importantly for this discussion, query optimization.

Queries

Query design is an important factor in overall application performance. Poorly designed queries will not only take longer to execute but may also generate many more locks than are really necessary, thereby affecting other users.

The information provided in the predicate, or WHERE clause, is used by SQLCORE/V to decide on its query execution strategy. It is important to specify enough selection criteria here in order to minimize the amount of sifting the application has to do on the result.

If there is an index defined on one of the source tables and it includes one or more of the columns from the WHERE clause, then the performance of the query may be significantly better than otherwise. However, just because such columns are specified does not imply that SQLCORE/V will decide to use the index. There are several criteria that must be satisfied for this to occur.

1. The first key in the index must match one of the columns in the WHERE clause. Multiple column indexes are wonderful for sorting and controlling uniqueness, but only the first key counts in the optimizer's choice of "useful" indexes.

2. SQLCORE/V does not yet fully optimize "OR" predicates. The indexed column must have an "AND" relationship to the query. This means that if the column does not satisfy the condition, then the row will not be part of the solution.

3. The indexed column must not be updated by the query being processed. This is to avoid getting into an infinite update loop.

4. The value against which the indexed column is being compared must be compatible with the column, or at least the value must be convertible by SQLCORE/V into a the index column data type.

After all the conditions are satisfied, SQLCORE/V may decide to use the index. The optimizer may decide that, since there are very few rows in the table anyway, it would be faster to simply do a serial search instead of looking up the index and then getting the rows!

Another area to watch is arithmetic operations. Arithmetic expressions should be evaluated outside of the query if possible. For example,

```
SELECT c1,c2 FROM t1,t2
WHERE c1 = :hostvariable * 3;
```

should be coded as

```
hostvariable := hostvariable * 3;
SELECT c1,c2 FROM t1,t2
WHERE c1 = :hostvariable;
```

As the number of tables being joined in a query increases, the response time of the application will most likely increase non-linearly. If the application's queries are joining five or more tables at once, then it would be advisable to either break up the query or "unnormalize" the tables a bit. This increases the amount of redundant data but speeds up access to that data. Table structure is discussed in more detail further in this paper.

Data conversions are not a great idea, as explained above in the index selection section. SQLCORE/V can provide data conversion between host variable types and HPSQL/V types but this will always create more overhead for the application. Wherever practical, host variables should be defined as the same type as their companion columns.

The application's queries should only return the actual data required instead of specifying '*' in the select-list. This minimizes the amount of data traffic between DBCORE/V and SQLCORE/V and can cut down on the number of DBCORE/V calls which will improve performance. Even better, the select-list order should be close to that of the tables being accessed. This way, HPSQL/V will not have to rearrange the order of the retrieved columns.

Table structure

The structure of the database in terms of its tables has a major effect on the performance of the applications which use it. We have already covered the ramifications of the implicit locking defined for a table in the DBCORE/V section.

Column definitions

HPSQL/V has a data value called "NULL". It is neither zero nor blank and can be used to signify that a column has no value(*something better than this!*). Like most very nice things columns which allow NULL values create extra overhead for HPSQL/V. If the NULL property is not required for the application, then the table columns should be defined as NOT NULL.

Another very useful feature is the VARCHAR data type. This corresponds to the PASCAL "string" type and can save a great deal of storage space for character data. For example, if the application is using data which has a maximum length of 80 bytes, but an average length of only 15 bytes, the VARCHAR type would save a great deal of space over a fixed CHAR type. Keeping track of VARCHAR lengths, however, creates work for DBCORE/V. Unless the average length of the data is less than 70% of the maximum length or disk space is at a premium, it is better to use a fixed length CHAR field.

Normalization

No discussion of relational DBMS application performance would be complete without a section on data normalization. As discussed in the section on query definition, the cost of joins increases non-linearly with the number of tables. If some of the tables in the system will be frequently joined, it may be advantageous to simply combine them. This will spoil the normalized beauty of the application but will do wonders for the response time and overall system throughput. After approximately five tables, joins will rapidly become a bit cumbersome. As we gain more experience with optimization strategy, this number may increase; until then, keep the number of join tables low!

General good ideas

HPSQL/V's optimizer is fairly bright and makes good decisions based on the data stored in the system catalogs. These catalogs contain table and index characteristics such as average column length, number of rows in the table, and number of indexes on the table. These values are updated only when the user issues an UPDATE STATISTICS command. If the table statistics become obsolete, HPSQL/V may not be able to generate the most optimal access plan for a query. It is a good idea to update the system catalogs for a table whenever major changes have occurred on that table, such as the creation of a new index or after large number of rows have been inserted/delete/updated.

HPSQL/V allows the user to dynamically restructure the database while other users are working. All actions such as creating new DBEFiles, new tables or indexes, updating table statistics, altering user capabilities with the GRANT command, make changes to the system catalogs which are very frequently accessed. Therefore, such changes should be done at periods of low usage and should be either committed or rolled back quickly.

Finally, the very rich authorization capabilities of HPSQL/V have not been examined herein. As might be assumed, the more complex the user authorization scheme, the slower the system will run. The key is to implement only the security checking that is absolutely required without going overboard.

Summary

HPSQL/V is a very new product that is gaining much customer attention. There is not yet a great deal of empirical performance data and much of the performance programming techniques have yet to be developed. This paper addresses the programming issues that I have experienced so far and will hopefully become a working document over the lifetime of the product.

RELATIONAL DATA BASE: HOW DO WE KNOW IF WE NEED ONE?

Orland Larson
Hewlett-Packard Company
19447 PRUNERIDGE AVE.
CUPERTINO,CA 95014

Summary

The field of relational technology is clearly misunderstood by a large number of people. One major obstacle to acceptance of the relational model is the unfamiliar terminology in which relational concepts are expressed. In addition, there are a number of misconceptions or "myths" that have grown up in the past few years concerning relational systems. The purpose of this paper is to define those terms, correct some of those misconceptions and to help you decide if your company can benefit from adding relational data base technology to your current capabilities.

This paper reports on the growing body of knowledge about relational technology. It begins by reviewing the challenges facing the MIS organization and the motivation for relational technology. It then briefly describes the history of relational technology and defines the basic terminology used in the relational approach. This will be followed by an examination of the productivity features of the relational approach and why it should be seen as a complement rather than a replacement for existing network databases such as the IMAGE data base management system. Typical application areas where the relational approach can be very effective will also be surveyed. Finally, a checklist will be reviewed that will help the audience determine if, indeed, they really can benefit from using a relational data base.

Introduction

The Challenges Facing MIS

The MIS manager is facing many challenges in today's modern information systems organization. The backlog of applications waiting to be developed is one of key challenges concerning MIS. In most medium to large installations the backlog of applications waiting to be developed is anywhere from two to five years. This estimate doesn't include the "invisible backlog," the needed applications which aren't even requested because of the current known backlog. Software costs are increasing because people costs are going up and because of the shortage of skilled EDP specialists. The data base administrator is typically using nonrelational data bases where a great deal of time is spent predefining data relationships only to find that the users data requirements are changing dynamically. These changes in user requirements cause modifications to the data base structure and, in many cases, the associated application programs.

The application programmer is spending a significant amount of time developing applications using these non-relational data bases, which require traversing or navigating the data base. This results in excessive application development time. Because the users requirements change dynamically, it also means a great deal of time spent maintaining applications. The programmer is also frequently restricted by the data structures in the data base, adding to the complexity of accessing data.

End users or business professionals are frustrated by the limited access to information that they know exists somewhere in the data base. Their business environment is changing dynamically, and they feel MIS should keep up with these changes. They find the applications are inflexible, due to the pre-defined relationships designed into the data base. They also lack powerful inquiry facilities to aid in the decision-making process, which would allow them to ask anything about anything residing in that data base.

The Motivation for Relational

Dr. Codd, considered to be the originator of the relational model for data bases , noted when presented the 1981 ACM Turing Award, that the most important motivation for the research work resulting in the relational model was the objective of providing a sharp and clear boundary between the logical and physical aspects of data base management (including data base design, data retrieval, and data manipulation). This is called the data independence objective.

A second objective was to make the model structurally simple, so that all kinds of users and programmers could have a common understanding of the data, and could therefore communicate with one another about the database. This is called the communicability objective.

A third objective was to introduce high level language concepts to enable users to express operations on large chunks of information at a time. This entailed providing a foundation for set oriented processing (i.e., the ability to express in a single statement the processing of multiple sets of records at a time). This is called the set-processing objective.

Another primary motivation for development of the relational model has been to make data access more flexible. Because there are no pointers embedded with the data, the relational programmer does not have to be concerned about following pre-defined access paths or navigating the database, which force him to think and code at a needlessly low level of structural detail.

The Relational Data Model: A Brief History

In 1970, Dr. E.F. Codd published an article in the Communications of the ACM entitled "A Relational Model of Data for Large Shared Data Banks." This classic paper marks the "birth" of the relational model. Dr. Codd was the first to inject mathematical principles and rigor into the study of database management.

By the mid 70's, there were two database prototypes being developed. IBM was behind a project called "System R," and there was another relational database being developed at the University of California, Berkeley called INGRES. It was late 1979 before the first commercially available relational database arrived in the marketplace called ORACLE, from ORACLE Corp., which was an implementation based on System R. In 1981 Relational Technology Inc. introduced INGRES which was a different implementation based on the research done at Berkeley. Today there are several additional advanced relational products available, such as SQL/DS and DB2 from IBM and Rdb from Digital Equipment Corporation. There are additional products sometimes referred to as "born again" relational databases such as IDMS/R from Cullinet, ADR's DATACOM/DB, and Software AG's ADABAS, to name a few.

Relational Database Defined

The relational database model is the easiest one to understand - at least at the most basic level. In this model, data are represented as a table, with each horizontal row representing a record and each vertical column representing one of the attributes, or fields, of the record. Users find it natural to organize and manipulate data stored in tables, having long familiarity with tables dating from elementary school.

The Table, or two dimensional array, in a "true" relational data base is subject to some special constraints. First, no row can exactly duplicate any other row. (If it did, one of the rows would be unnecessary). Second, there must be an entry in at least one column or combination of columns that is unique for each row; the column heading for this column, or group of columns, is the "key" that identifies the table and serves as a marker for search operations. Third, there must be one and only one entry in each row-column cell.

A fourth requirement, that the rows be in no particular order, is both a strength and a weakness of the relational model. Adding a new item can be thought of as adding a row at the bottom of the table; hence there is no need to squeeze a new item in between preexisting items as in other database structures. However, to find a particular item, the entire table may have to be searched.

There are three kinds of tables in the relational model: base tables, views, and result tables. A base table is named, defined in detail, filled with data, and is more or less a permanent structure in the database.

A view can be seen as a "window" into one or more tables. It consists of a row and/or column subset of one or more base tables. Data is not stored in a view, so a view is often referred to as a logical or virtual table. Only the definition of a view is stored in the database, and that view definition is then invoked whenever the view is referenced in a command. Views are convenient for limiting the picture a user or program has of the data, thereby simplifying both data security and data access.

A result table contains the data that results from a retrieval request. It has no name and generally has a brief existence. This kind of table is not stored in the database, but can be directed to an output device.

The Relational Language

The defacto industry standard language for relational data bases is SQL. SQL stands for Structured Query Language. This name is deceiving in that it only describes one facet of SQL's capabilities. In addition to the inquiry or data retrieval operations, SQL also includes all the commands needed for data manipulation. The user only needs to learn four commands to handle all data retrieval and manipulation of a relational database. These four commands are: SELECT, UPDATE, DELETE and INSERT.

The relational model uses three primary operations to retrieve records from one or more tables: select, project and join. These operations are based on the mathematical theories that underlie relational technology, and they all use the same command, SELECT. The select operation retrieves a subset of rows from a table that meet certain criteria. The project retrieves specific columns from a table. The join operation combines data from two or more tables by matching values in one table against values in the other tables. For all rows that contain matching values, a result row is created by combining the columns from the tables eliminating redundant columns.

The basic form of the SELECT command is:

```
SELECT      some data (field names)
FROM        some place (table names)
WHERE       certain conditions (if any) are to be met
```

In some instances WHERE may not be neccessary. Around this SELECT..FROM..WHERE structure, the user can place other SQL commands in order to express the many powerful operations of the language.

In all uses of SQL, the user does not have to be concerned with how the system should get the data. Rather, the user tells the system what he wants. This means that the user only needs to know the meaning of the data, not its physical representation, and this feature can relieve the user from many of the complexities of data access.

The data manipulation operations include UPDATE, DELETE and INSERT. The UPDATE command changes data values in all rows that meet the WHERE qualification. The DELETE command deletes all rows that meet the WHERE qualification and the INSERT command adds new rows to a table.

When retrieving data in application programs it is important to remember that SQL retrieves sets of data rather than individual records and consequently requires different programming techniques. There are two options for presenting selected data to programs. If an array is established in the program, a BULK SELECT can retrieve the entire set of qualifying rows, and store them in the array for programmatic processing. Alternatively, it is possible to activate a cursor that will present rows to programs one at a time.

SQL has a set of built-in, aggregate functions. The functions available are count, sum, average, minimum, and maximum. They operate on a collection of values and produce a single value.

In addition to commands for data retrieval and modification, SQL also includes commands for defining all database objects. The data definition commands are CREATE, ALTER and DROP. The CREATE command is used to create base tables and views. The ALTER provides for the expansion of existing tables and the DROP deletes a view. One of the most powerful features of SQL is its dynamic definition capability. This function allows the user to add columns, tables and views to the database without unloading and reloading existing data or changing any current programs. More importantly, these changes can be made while the databases are in use.

Productivity Features of Using Relational Technology

Relational technology is one very important tool that can contribute to making data processing professionals more productive. The programmer can benefit from a facility called interactive program development, which allows the development and debugging of SQL commands and then permits the moving of those same commands into the application programs. It is convenient and easy to set up test databases interactively and then to confirm the effect of a program on the database. All of these characteristics make SQL a powerful prototyping tool. The on-line facilities of SQL can be used to create prototype tables loaded with sample or production data. On-line queries can easily be written to demonstrate application usage. End users can see the proposed scheme in operation prior to formal application development. In this prototype approach, people-time and computer-time are saved while design flaws are easily corrected early in development.

The data base administrator profits from the productivity features already described for programmers. The database administrator has a great deal of freedom in structuring the database, since it is unneccessary to predict all future access paths at design time. Instead, the DBA can concentrate on specific data requirements of the user. Nonrelational models, on the other hand, require all relationships be pre-defined, which adds to the complexity of the application and lengthens development time.

Additional productivity features for the database administrator include the capability to modify tables without affecting existing programs and the capability to dynamically allocate additional space while the database is still in use. SQL goes far beyond many database management systems in the degree of protection that it provides for data. Views make it possible to narrow access privileges down to a single field. Users can even be limited to summary data. Protection can be specified for database, system catalog, tables, views, columns, rows and fields. It is also possible to restrict access to a subset of commands. These access privileges can be changed dynamically, as the need arises.

In many installations, the key to overall productivity is the ability of DP too offload the appropriate portions of the development and maintenance to the end user. The flexible design approach of relational databases allows an application to be designed with the end user's requirements in mind. This could enable the DP professional to

implement an application up to the point where the end user could create and execute his own queries, thereby expanding the application on his own and reducing his dependence on the data processing department. Through SQL, the end user is provided with extremely flexible access and simple but powerful commands.

Relational and Nonrelational: Complementary Technologies

Within a data processing department already using a well-established nonrelational DBMS, what role can relational technology be expected to play? We know that DP will not automatically drop everything and go to relational. Rather, relational technology should be seen as a complement rather than a replacement for nonrelational database systems. Both approaches offer a host of benefits, and most applications can be implemented with either of the two.

The relational approach is preferred when the application has a large number of data relationships or when the data relationships are unknown or changing dynamically. The relational approach provides the needed flexibility to establish relationships at the time of inquiry, not when the database is designed. If the application has unknown of incomplete data specifications, which is usually the case in a prototyping environment, then a relational system may be preferable. If the application requires a quick turnaround, the quick design and implementation capabilities of a relational database can be important. The ability to handle ad hoc requests is a definite strength of the relational model as is the ability to extract data for use in a modeling, forecasting, or analytical framework.

The nonrelational approach is preferred for high-volume on-line transaction processing applications where performance is the most critical requirement.

Choosing the Right Technology

The choice of the "correct" database management system must be based on the environment in which the database will be used and on the needs of the particular application. The key feature of relational technology is that it allows for maximum flexibility, and will probably be the choice for many new applications. On the other hand, nonrelational systems may continue to be preferrable for very stable or structured applications in which data manipulation requirements are highly predictable, and high transaction throughput is important.

The optimum approach for many MIS departments will be to use the relational system concurrently with the existing nonrelational system, matching the appropriate technology to the application. The only problem with such an approach is that the data for an application developed in one technology may sometimes be needed by applications developed in the other technology. Data may be "locked out" from an application that needs it, or users might be tempted to duplicate the data, maintaining both copies. The most desirable solution would obviously be to provide both relational and nonrelational access to a single database. This capability will be available with HP's ALLBASE.

Relational Applications

There are many application areas - particularly those involving user analysis, reporting, and planning - where the very nature of the application is constantly changing. Some typical application areas are:

* Financial
  - Budget analysis
  - Profit and Loss
  - Risk assessment

* Inventory
  - Vendor performance
  - Buyer performance

* Marketing and sales
  - Tracking and analysis
  - Forecasting

* Personnel
  - Compliance
  - Skills and job tracking

* Project management
  - Checkpoint/milestone progress
  - Development and test status

* EDP auditing
  - Data verification
  - Installation configuration

* Government/education/health
  - Crime and traffic analysis
  - Admissions/recruiting/research
  - Medical data analysis

These applications typify instances where it is of primary importance to establish interrelationships within the database and to define new tables.

Checklist for Deciding Whether or Not You Need A Relational Database

Note: If you answer yes to any of the following questions, you should seriously consider taking advantage of relational technology.

1. Does your company have an excessive backlog of applications to be developed, including an invisible backlog?

2. Is your company spending too much money developing applications due to the complexities of using nonrelational systems?

3. Do your users' requirements for information change dynamically?

4. Are your programmers spending too much time maintaining applications caused by changing data requirements or relationships?

5. Do your users feel restricted by a nonrelational database?

6. Are your programmers spending an excessive amount of time writing code to navigate through nonrelational databases?

7. Is the nature of your applications such that it is constantly changing?

8. Would your users find it natural to organize and manipulate data in tables?

9. Do your users currently use LOTUS 1-2-3 or spreadsheets?

10. Is your company moving towards a true distributed database environment?

SUMMARY

A fully implemented relational system presents a user with a data description that is irreducibly simple, a language that allows him to ask questions instead of writing programs, implementation transparency that ensures survival of programs in the face of change, system-optimized efficiency that adapts, and controls and views that are at once specific and efficient. For the database administrator it means unprecedented power and ease in controlling and restructuring a database. For the application programmer, it increases productivity many fold and lifts programming to the level of problem solving. For the casual user, it makes direct access to data at last possible. The bottom line is that relational technology is here to stay!

Bibliography

Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," CACM, 13 6,(June 1970),pp. 377-387.

Codd, E.F.,"Relational Database: A Practical Foundation for Productivity," CACM, 25 2,(February 1982,pp. 109-117.

Date, C.J., An Introduction to Database Systems. Addison-Wesley, 1977.

_____,Relational Technology: A Productivity Solution, Hewlett-Packard Co., Computer Systems Division, Cupertino, Ca., 5954-6676, January 1986.

Biography

Orland Larson is currently Information Resource Management Specialist for Hewlett-Packard. As the database and application development specialist for the Information Systems Tactical Marketing Center he develops and presents seminars worldwide on database management, information systems prototyping and productivity tools for information resource management. He is a regular speaker at Hewlett-Packard's Productivity Shows and Users Group Meetings and also participates in various National Data Base and 4th Generation Language symposiums. Previously he was the Product Manager for IMAGE/3000, Hewlett-Packard's award winning database management system.

Before joining HP he worked as a Senior Analyst in the MIS Department of a large California-based insurance company and prior to that as a Programmer/Analyst for various software companies. Mr. Larson has been with Hewlett-Packard since 1972.

RELATIONAL DATABASES VS. IMAGE: WHAT THE FUSS IS ALL ABOUT

by Eugene Volokh,
VESOFT, Inc.
7174 Melrose Ave.
Los Angeles, CA 90046 USA
(213) 937-6620

Presented at Greater Los Angeles Users Group meeting, Apr 1986.

ABSTRACT

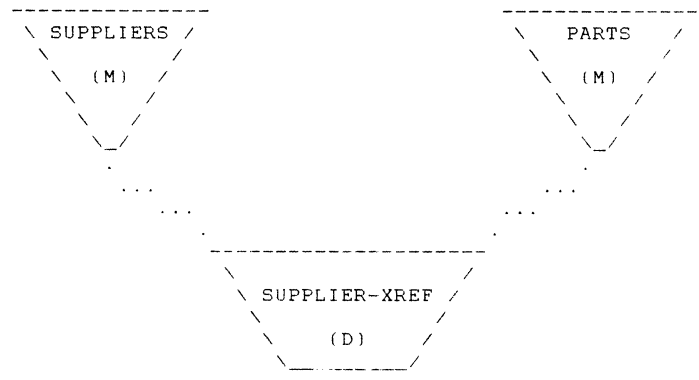
What are "relational databases" anyway? Are they more powerful than IMAGE? Less powerful? Faster? Slower? Slogans abound, but facts are hard to come by. It seems like HP will finally have its own relational system out for Spectrum (or whatever they call it these days). I hope that this paper will clear up some of the confusion that surrounds relational databases, and will point out the substantive advantages and disadvantages that relational databases have over network systems like IMAGE.

WHAT IS A RELATIONAL DATABASE?

Let's think for a while about a database design problem.

We want to build a parts requisition system. We have many possible suppliers, and many different parts. Each supplier can sell us several kinds of parts, and each part can be bought from one of several suppliers.

Easy, right? We just have a supplier master, a parts master, and a supplier/parts cross-reference detail:



Relational Databases

Every supplier has a record in the SUPPLIERS master, every part has a record in the PARTS master, and each (supplier, part-supplied) pair has a record in the SUPPLIER-XREF dataset.

Now, why did we set things up this way? We could have, for instance, made the SUPPLIER-XREF dataset a master, with a key of SUPPLIERS#+PART#. Or, we could have made all three datasets stand-alone details, with no masters at all. The point is that the proof of a database is in the using. The design we showed -- two masters and a detail -- allows us to very efficiently do the following things:

* Look up supplier information by the unique supplier #.

* Look up parts information by the unique part #.

* For each part, look up all its suppliers (by using the cross-reference detail dataset).

* For each supplier, look up all the parts it sells (by using the cross-reference detail dataset).

This is what IMAGE is good at -- allowing quick retrieval from a master using the master's unique key and allowing quick retrieval from a detail chain using one of the detail's search items.

However, lets take a closer look at the parts dataset. It actually looks kind of like this:

```
PART#          <-- unique key item
DESCRIPTION
SHAPE
COLOR
...
```

What if we want to find all the suppliers that can sell us "framastat"? A "framastat", you see, is not a part number -- it's a part description. We want to be able to look up parts not only by their part number, but also by their descriptions. The functions supported by our design are:

* Look up PART by PART#.

* Look up SUPPLIERS by SUPPLIERS#.

* Look up PARTs by SUPPLIERS#.

* Look up SUPPLIERs by PART#.

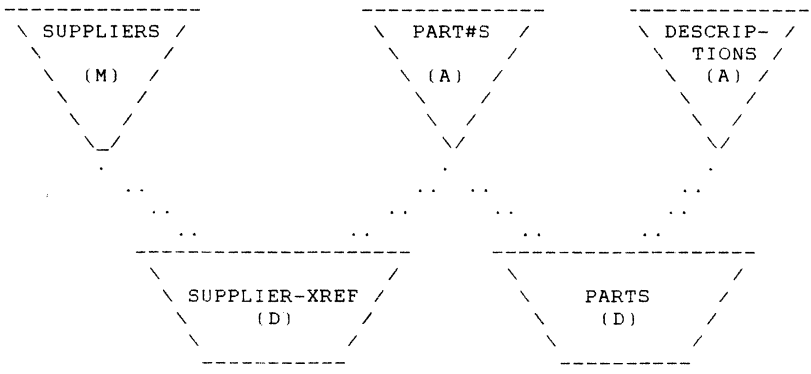
What we want is the ability to

* Look up PART by DESCRIPTION.

The sad thing is that the PARTS dataset is a master, and a master dataset supports lookup by ONLY ONE FIELD (the key). We can't make DESCRIPTION the key item, since we want PART# to be the key item; we can't make DESCRIPTION a search item, since PARTS isn't a detail. By making PARTS a master, we got fast lookup by PART# (on the order of 1 or 2 I/Os to do the DBGET), but we forfeited any power to look things up quickly by any other item.

And so, dispirited and dejected, we get drunk and go to bed. And, deep in the night, a dream comes. "Make it a detail!" the voice shouts. "Make it a detail, and then you can have as many paths as you want to."

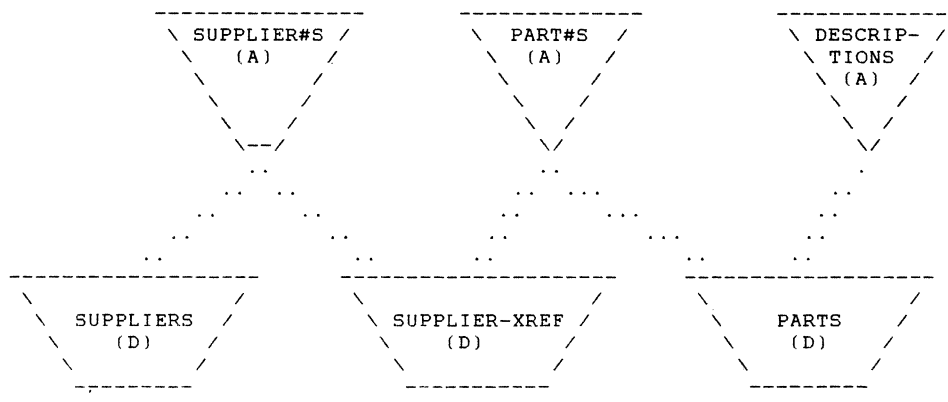
We awaken elated! This is it! Make PARTS a detail dataset, and then have two search items, PART# and DESCRIPTION. Each search item can have an automatic master dataset hanging off of it, to wit:



What's more, if we ever, say, want to find all the parts of a certain color or shape, we can easily add a new search item to the PARTS dataset. Sure, it may be a bit slower (to get a part we need to first find it in PART#S and then follow the chain to PARTS, 2 I/Os instead of 1), and also the uniqueness of part numbers isn't enforced; still, the flexibility advantages are pretty nice.

So, now we can put any number of search items in PARTS. What about SUPPLIERS? What if we want to find a supplier by his name, or city, or any other field? Again, if we use master datasets, we're locked into having only one key item per dataset. Just like we restructured PARTS, we can restructure SUPPLIES, and come up with:

Relational Databases



Note what we have done in our quest for flexibility. All the real data has been put into detail datasets; every data item which we're likely to retrieve on has an automatic master attached to it.

Believe it or not, this is a relational database.

IF THIS IS A RELATIONAL DATABASE, I'M A HOTTENTOT

Surely, you say, there is more to a relational database than just an IMAGE database without any master datasets. Isn't there? Of course, there is. But all the wonderful things you've been hearing about relational databases may have more to do with the features of a specific system that happens to be relational than with the virtues of relational as a whole.

Consider for a moment NETWORK databases. IMAGE is one example, in fact an example of a rather restricted kind of network database (having only two levels, master and detail). Let's look at some of the major features of IMAGE:

* IMAGE supports unique-key MASTERS and non-unique-key DETAILS.

* IMAGE does HASHING on master dataset records.

* IMAGE has QUERY, an interactive query language.

Which of these features are actually NETWORK DATABASE features? In other words, which features would be present in any network database, and which are specific to the IMAGE implementation? Of the three listed above, only the first -- masters and details -- must actually be present in all databases that want to call themselves "network". On the other hand, a network database might very well use B-trees or ISAM as its access method instead of hashing; or, it might not provide an interactive query language. It would still be a network database -- it just wouldn't be IMAGE.

Why is all this relevant? Well, let's say that somebody said "Network databases are bad because they use hashing instead of B-trees." This statement is wrong because the network database model is silent on the question of B-trees vs. hashing. It is incorrect to generalize from the fact that IMAGE happens to use hashing to the theory that all network databases use hashing. If we get into the habit of making such generalizations, we are liable to get very inaccurate ideas about network databases in general or other network implementations in particular.

The same goes for relational databases. The reason that so many people are so keen on relational databases isn't because they have any particularly novel form of data representation (actually, it's much like a bunch of old-fashioned KSAM/ISAM-like files with the possibility of multiple keys); nor is it because of some fancy new access methods (hashing, B-trees, and ISAM are all that relational databases support). Rather, it's because the designers of many of the modern relational databases did a good job in providing people with lots of useful features (ones that might have been just as handy in network databases).

WHAT ARE RELATIONAL DATABASES -- FUNCTIONALITY

The major reason for many of the differences between relational databases and network databases is simple: age. Remember the good old days when people hacked FORTRAN code, spending days or weeks on optimizing out an instruction or two, or saving 1000 bytes of memory (they had only 8K back then) ? Well, those are the days in which many of today's network databases were first designed; maximum effort was placed on making slow hardware run as fast as possible and getting the most out of every byte of disk.

Relational databases, children of the late '70s and early '80s had the benefit of perspective. Their designers saw that much desirable functionality and flexibility was missing in the older systems, and they were willing to include it in relational databases even if it meant some wasted storage and performance slow-down. The bad part of this is that, to some extent, modern relational databases are still hurting from slightly decreased performance; however, this seems to be at most a temporary problem, and the functionality and flexibility advantages are quite great.

Relational Databases

THE USER INTERFACE -- THE RELATIONAL QUERY LANGUAGE

If you look at the theoretical definition of relational databases -- the one given by Codd in his original paper that first introduced the subject -- you'll find that nowhere does it talk about B-trees or hashing or internal dataset format or anything like that. In fact, what defines a relational database is the format of each "relation" (or dataset) and the Relational Query Language that lets a user retrieve from and update these datasets. Since IMAGE detail datasets very closely fit the requirements of relational dataset format, we won't talk much about this; the concept of a Relational Query Language, however is another story.

Everyone understands the "Query Language" part -- it's an interface that allows interactive access to the database, just like QUERY/3000 (note that "query" means any database retrieval, insertion, update, or deletion). What makes the query language RELATIONAL?

Imagine for a moment the following QUERY/3000 statement -- let's call it "SELECT":

* It has the ability of a >LIST command to select certain records based on selection conditions (like PRICE < 1000 AND COLOR = "RED").

* Just like a >LIST, it can output only those fields you want (not just the entire record).

* Instead of just saying "list the fields PRICE and COLOR" you can also specify expressions such as PRICE * NUM-ITEMS, so that the statement might look like:

```
SELECT PRICE, COLOR, PRICE * NUM-ITEMS, NUM-ITEMS + 100
```

* Finally -- the most important difference of all, you can RETRIEVE ITEMS FROM SEVERAL DATASETS AT ONCE! This is what QUERY's >JOIN and >MULTIFIND commands let you do, but it's a lot less clumsy.

The capability of retrieving stuff from several datasets (called "joining", from which QUERY's >JOIN command gets its name) is the most important difference between Relational Query Languages and orthodox query languages (such as QUERY-A, which didn't have even the > JOIN command). If there's one thing the advocates of relational databases can be proud of, it's their efforts for widespread implementation of joins in various query languages.

With the >SELECT statement, we could say something like:

```
SELECT SUPPLIER.NAME, SUPPLIER.NUMBER,
       PART.NAME, PART.NUMBER
FROM   SUPPLIER, PART, SUPPLIER-XREF
WHERE  SUPPLIER.NUMBER = SUPPLIER-XREF.SUPPLIER# AND
       PART.NUMBER = SUPPLIER-XREF.PART# AND
```

```
PART.COLOR = "RED" AND
SUPPLIER.STATE = "CA"
```

This finds every SUPPLIER in the state of CAlifornia who supplies red parts, and prints the name and number of that supplier together with the name and number of every red part that he supplies.

The first thing you should notice about the >SELECT command is that THERE'S NOTHING IN IT THAT QUERY/3000 DOESN'T HAVE. Functionally speaking, it's merely a combination of QUERY's >LIST, >JOIN, and >MULTIFIND. The big thing is that you don't have to specify an explicit >JOIN command that tells QUERY how to navigate the pathways between SUPPLIER, SUPPLIER-XREF, and PART. It figures all this out automatically.

For instance, the >SELECT command shown above figures out that it should find all the CAlifornian SUPPLIERs (maybe there's even a search item on STATE that'll make the search faster), for each of those finds the appropriate items in SUPPLIER-XREF, and for each SUPPLIER-XREF item it checks to see if the cross-reference record refers to a red part. Alternatively, it might look up all the red parts first, and then for each of those parts go through SUPPLIER-XREF and find all the CAlifornian SUPPLIERs.

It's a great convenience to have the query language figure out the way the join should be done, but functionally this doesn't give you any capability you don't already have with QUERY/3000. In a way, therefore (with a few technical exceptions), QUERY/3000 IS a relational query language, although not easy to use as many such languages that relational database systems support.

A BRIEF, BUT IMPORTANT DIGRESSION

RELATION = TABLE = DATASET.

ATTRIBUTE = COLUMN = FIELD (or ITEM -- it's close enough).

TUPLE = ROW = RECORD.

The relational database community is fond of calling things "relations", "attributes", and "tuples". This is just needless confusion. A "relation" is a dataset, and "attribute" is a field/item, and a "tuple" is a record. Table, column, and row are more names for the same thing.

I try to always use the words dataset, item, and record. If anybody throws any of the relational buzzwords at you, just do a quick mental translation.

FLEXIBILITY -- BUILDING AUTOMATIC MASTERS ON THE FLY

Relational Databases

If you look carefully at the theoretical definition of relational databases, you'll find that it doesn't actually mention any analog of an automatic master dataset. It describes "relations", which are just detail datasets, and some broad parameters of the Relational Query Language, but it says nothing about "indexes", which is what most relational systems use in the same way that IMAGE uses automatic masters.

Thus, a database system may well have all its selections and joins implemented using serial reads and still be relational! Of course, such a system wouldn't sell very well, so all relational systems allow you to build "indexes" just like IMAGE lets you set up automation masters. The thing that the architects of these systems did right, however, was that they let you build indexes whenever you liked, not just when you were initially building the database. Similarly, you could delete them whenever you found that the overhead of constantly updating the index exceeded the benefit that the index gave you for searching.

In some ways, this feature -- although, as I said, theoretically not a "relational" feature -- is one of the most important advantages of relational systems. It's this that gave rise to the often-heard but quite misleading statement that "in relational systems, you can find things by any item, not just a search item". In both IMAGE and relational databases, you can find records using non-search items, but it'll require a serial search; in both IMAGE and relational databases, you can set up any item as a search item, but then you have to pay a penalty every time you add or delete a record. The big advantage of relational is that you can make ordinary items into search items (and vice versa) very easily -- even easier than with ADAGER.

MORE FLEXIBILITY -- CHANGING DATABASE STRUCTURE ON THE FLY

The ability that a relational database user has to easily build and destroy indexes is actually just a special case of the ability the user has to easily create and destroy any dataset in the database. He can add datasets, delete datasets, restructure them, and so on.

This can be done not just after the system is built, but even while other parts of the database are in use. Again, any ADAGER user will understand how useful this kind of ability is (although ADAGER is actually rather more powerful than most relational databases' restructuring tools).

Again, this is not part of the "theoretical" definition of databases, since the theoretical definition doesn't worry about "unimportant" things like performance or database maintenance; still, it's pretty universal among database systems.

MORE USEFUL FUNCTIONALITY

Relational Query Languages and Ease of Restructuring are two of the three most major features that relational databases provide (the third one I'll get to later). What I've taken pains to point out is that these features aren't unique with relational systems, and to a large extent are present in IMAGE, especially if you also throw in ADAGER. However, relational systems did pioneer these features, and should be given much credit for the spread of these features to non-relational systems like IMAGE.

Some other useful features -- also not part of the theoretical definition of relational databases, but implemented in most relational systems -- are:

* B-tree and ISAM indexes to the addition (or sometimes to the exclusion) of hashing indexes. With these, you can do KSAM-like accesses by which you can quickly find all the customers whose names start with "SMI" or retrieve all the parts in sorted order by part number.

* Indexes on multiple items, just as if you were able to build a single IMAGE automatic master on not just the VENDOR# field of the INVOICES dataset, but both the VENDOR# and the INVOICE# fields.

* Views, which look to the user just like a dataset, but which are actually the results of FINDs (or MULTIFINDs) on other dataset (s). In other words, a user (or a database administrator) can define a view called RED-CALIFORNIAN-PARTS, which corresponds to the

```
SELECT SUPPLIER.SNAME, SUPPLIER.SNUMBER,
       PART.PNAME, PART.PNUMBER, PART.PSHAPE
FROM   SUPPLIER, PART, SUPPLIER-XREF
WHERE  SUPPLIER.NUMBER = SUPPLIER-XREF.SUPPLIER# AND
       PART.NUMBER = SUPPLIER-XREF.PART# AND
       PART.COLOR = "RED" AND
       SUPPLIER.STATE = "CA"
```

command that we discussed above. This view will look just like a dataset which contains the data retrieved by this >SELECT -- the only difference is that the dataset won't actually take up any disc space, but rather any command that refers to it will automatically be translated into one that extracts all the requested data from the SUPPLIER, SUPPLIER-XREF, and PARTS datasets.

Thus, if we define the RED-CALIFORNIAN-PARTS view to be equivalent to a SELECT like the one above, then

```
SELECT RED-CALIFORNIAN-PARTS.SNAME,
       RED-CALIFORNIAN-PARTS.PNAME
WHERE  RED-CALIFORNIAN-PARTS.PSHAPE="CUBE"
```

will get all the red parts that are made by CAlifornian SUPPLIERs AND are cubes (the parts, not the suppliers).

* Integrity constraints, by which you can say something like "all employee salaries must be positive" or "the sum of HOURS-WORKED and HOURS-VACATION may not be more than 48". Any time a user -- through a program or the interactive query environment -- tries to add a record to the database that violates the integrity constraints, he'll get an error. On the other hand, the built-in integrity constraints of IMAGE, namely that all detail records must have corresponding appropriate manual master records, are not supported.

* Virtually all relational database systems have built-in "transaction-level recovery". Without going to deeply into this -- users of IMAGE roll-forward or roll-back recovery already know what this is -- this is a good way of making sure that the database is kept safe and sound (even in case of disk errors or such). It also allows you to automatically prevent half-completed transactions -- for instance, the addition of half the line items of an invoice - from being left in the database in case of system crash.

FUNCTIONAL ADVANTAGES OF IMAGE OVER RELATIONAL

As a rule, most relational systems are more powerful than IMAGE. Still, the very fact that IMAGE is a network system gives it some advantages over relational systems.

* The most major one is the fact that in IMAGE, all detail records must have corresponding records in any appropriate manual masters. This is a kind of "integrity constraint" (see above) that most relational systems don't support. It's often quite useful for the database itself to make sure that a user can't enter an invoice, say, belonging to a non-existent vendor.

* Another advantage of IMAGE (or, to be precise, of QUERY) is that unlike many (but not all) relational database systems, QUERY can handle several databases open at once.

BUT WHAT ABOUT RELATIONAL DATABASE PERFORMANCE?

One thing that's often been said about relational databases is that they're slow. The reason why this rumor is so prevalent is that it's often been true. Relational databases are young creatures, not yet as well-optimized as older, more mature systems. Still, the recent (past 3 years) 'coup' of relational systems has shown great improvement, and there's no reason why they can't - now or in the near future - match and exceed the performance of older systems like IMAGE/3000.

What exactly was it that has made many older relational systems slow? It wasn't the data structures - they can use hashing and B-trees just as

well as IMAGE or KSAM (the fact of the index and the dataset being stored in two different places isn't really a problem). It wasn't the recovery logging - a smart relational system might actually do less I/Os to support database reliability than IMAGE does (with each one of its writes having to be posted immediately to disk). The greatest speed problem of relational systems actually happens to be their greatest performance advantage - embedded query languages.

As I mentioned before, relational databases were invented almost backwards (some of their proponents claim that the non-relational databases were the ones invented backwards). Instead of describing the physical structure first - hashed masters, details, search items, sort items, double-word pointers, etc. - and then adding the query language as an afterthought, relational query capabilities were designed first and then some reasonable physical structure was built around them. That's why relational query languages are so nifty.

The problem the relational architects got was this:

In a relational query system, to print out the names of all the employees who made more than $50,000 per year, the names of their departments, and the addresses of their buildings, sorted by department name, we could say

```
SELECT EMPLOYEE.NAME, DEPARTMENT.NAME, BUILDING.ADDRESS,
       BUILDING.CITY, BUILDING.STATE
WHERE  EMPLOYEE.SALARY > 50000   AND
       EMPLOYEE.DEPT# = DEPARTMENT.DEPT#   AND
       EMPLOYEE.BLDG# = BUILDING.BLDG#
SORT BY DEPARTMENT.NAME
```

But, this is all in QUERY - how can we do this from our program? Do we now have to go back to the old way of doing all those DBGETs and DBFINDs?

What the designers of several relational systems decided was that

* YOU CAN EMBED RELATIONAL QUERY COMMANDS INTO YOUR OWN PROGRAMS!

Think about it for a moment. Your program might look like this:

```
   ACCEPT "WHAT SALARY THRESHOLD DO YOU WANT?", SALARY
#     SELECT  :ENAME = EMPLOYEE.NAME, :DNAME = DEPARTMENT.NAME,
#             :ADDRESS = BUILDING.ADDRESS, :CITY = BUILDING.CITY,
#             :STATE = BUILDING.STATE
#     WHERE   EMPLOYEE.SALARY > :SALARY   AND
#             EMPLOYEE.DEPT# = DEPARTMENT.DEPT#   AND
#             EMPLOYEE.BLDG# = BUILDING.BLDG#
#     SORT BY DEPARTMENT.NAME
#     DO
C    This code will be executed once for each employee-department-
C    building triplet found, with the ENAME, DNAME, ADDRESS, CITY,
```

```
C    and STATE variables set to the proper values.
#      DOEND
```

The query you want to execute is just embedded right in your program (with each line preceded by, say, a "#" to indicate that it's part of a query). Any variables - like the threshold SALARY or ENAME, DNAME, ADDRESS, CITY, or STATE - can be passed to and from the query. Finally, in the case of SELECT queries (which only retrieve data), a bunch of code will be executed for each thing retrieved; other queries, like APPEND, DELETE, or UPDATE can also be conveniently embeded.

What an idea! This is almost like having a fourth-generation language - all the power of a procedural language with all the ease of use of the relational query facility. However, truly heroic measures have to be undertaken to be sure that this isn't the slowest database system known to man.

The simplest way of allowing embedded queries is to have the compiler (or, more commonly, a preprocessor program that converts the embedded code into something readable to the compiler) compile calls to the relational query language. Think of it like QUERY/3000 wasn't a standalone program, but rather a procedure, and you'd say

```
CALL QUERY ("FIND EMPLOYEE.SALARY > 10000")
```

This'll work, but imagine the overhead QUERY will have to go through to parse and interpretively execute this operation! It has to recognize the FIND, find the dataset EMPLOYEE, find the data item SALARY, figure out what indexes there are on the EMPLOYEE dataset - you'll be lucky if it's done by Christmas. And, believe it or not, this is how some early relational database systems worked.

Now, there is a more intelligent approach, but it's harder to implement. What it involves is that the preprocessor should do much of the query parsing when it preprocesses the user's program. It sees our SELECT, for instance, and looks up all the datasets and items that it refers to. Then, the pointers to all these things are kept around, so at run time, no parsing or dataset/item lookup needs to be done. In the best possible case, the preprocessor might actually compile the SELECT into the appropriate DBGETs, much like the interactive QUERY facility would translate the SELECT into a bunch of DBGETs that it has to do.

It is on the success of this "preprocessor-time precompilation" that the performance of a commercial relational system - including, in particular, HP's new relational offering - rests.

HP might decide to forbid embedded queries altogether and have a DBGET/DBPUT/etc.-like interface. This won't be all that bad, but it won't be very nice (believe me, embedded queries are VERY useful). Or, it could give us slow embedded queries and a fast DBxxx-like interface, telling us to "make our choice" - believe me, this will be no choice at all, considering just how slow non-precompiled embedded queries are.

The worst thing HP can do is to give us slow (non-precompiled) embedded queries WITHOUT a procedure-level interface to the system. This means that their entire relational product will be a total, unmitigated disaster.

HP might do some simple precompilation - say, parsing of the embedded query into individual tokens ("SELECT", "EMPLOYEE", etc.). WRONG. All it saves is just a little text scanning at run-time.

HP might have the preprocessor parse out the entire query and also look up all the various datasets and items mentioned in it, thus saving the extra overhead at run-time. This may make for a viable, tolerably fast system.

Finally, if HP is feeling really audacious - and smart - it can compile the embedded query into some kind of pseudo-code that's as close to the actual DBGET/DBPUT/etc. call level as possible. That way, all that will be needed at run-time is for the program to step through this pseudo-code and do a DBxxx call or something like that for each pseudo-instruction. If this is done, there's no reason I can see why the relational database can't be as fast or faster than IMAGE.

CONCLUSION

My conclusions are simple:

* Relational databases are nothing revolutionarily new. Their main advantage lies not in any radically different data representation structure, but rather in a lot of useful functionality/flexibility features.

* All those functionality/flexibility features are really GREAT. If HP doesn't degrade performance too badly, you ought to like the relational system a lot better than you do IMAGE.

* The performance of the system depends primarily on how good a job HP does of optimizing the preprocessing of embedded queries.

* You'll love embedded queries, if they're fast. If they're slow, you'll hate them.

Relational Databases

WHAT I HAVEN'T SAID AND WHY I HAVEN'T SAID IT

The purpose of this paper is to clarify for the average user the differences between IMAGE/3000 and common relational database systems - like HP's imminent relational offering. To do this as tersely as possible, I've omitted many details (some of them pretty substantial ones) that I think are insignificant to the broad picture I'm trying to present.

Still, I think that it's appropriate for me to point out these details and explain why I didn't think it necessary to discuss them in greater detail.

* THE MATHEMATICAL THEORY OF RELATIONAL DATABASES - RELATIONAL CALCULUS AND RELATIONAL ALGEBRA. I believe that these are completely irrelevant to practical uses of relational database systems. I get particularly irritated when people say that "relational databases are Mathematically Sound" - something I've heard many a time - this implies that network databases are Mathematically Unsound. I've worked on them for six years, and believe me, they're sound enough.

* OPTIMIZING LARGE, COMPLICATED JOINS AND QUERIES. In my discussion of performance, I emphasized the importance of good preprocessing and precompilation. This is a big deal for small, simple queries that take only a small amount of processing time, and thus any parsing overhead really slows them down. However, a query that does, say, a join of three 100,000-record datasets won't be slowed down much by parsing - what will speed it up most is a good Join Optimization algorithm, one which knows, say, which dataset to read through first, and so on. The reason why I didn't talk much about it is that the big three-way 100,000-tuple joins aren't all that frequent, and we can live with them being slow. The small queries have to run like greased lightning.

* FIRST, SECOND, THIRD, BOYCE-CODD, AND OTHER NORMAL FORMS. Interesting for a paper on how to design a database to minimize programming problems, but irrelevant to this discussion. In any case, anything you can represent in IMAGE is normalized enough to be directly translated to relational, although some database designs may be prettified by further normalization.

THE FUTURE OF DATA BASE TECHNOLOGY

By Mark S. Trasko, President, Dynamic Information Systems Corporation

The Spectrum machines recently announced by HP have generated much enthusiasm in the user community. The new 930 and 950 models will provide major increases in CPU power along with a powerful new DBMS with both network and relational interfaces. Many sites with large data base applications have been anxiously awaiting these machines as an alternative to switching to another vendor's supermini or mainframe. With the wide array of CPU's available in the 5 to 20 MIP range instead of the Series 68's meager 1 MIP speed, a hardware upgrade is an obvious solution to their performance problems.

Or is it?

CPU POWER ALONE MAY NOT BE THE ANSWER

Unfortunately, most data base applications can be improved only slightly by increases in CPU speed. Why? Because they are disc I/O bound rather than CPU bound. Most inquiries and reports require lengthy serial or chained reads of the data base even when only a small subset of records is of interest. The time required to accomplish the report depends on disc throughput rather than CPU speed. And while CPU speeds continue to increase as semiconductor technology evolves, disc drives, because they are mechanical devices, are still limited to about 30 I/O's per second.

Sophisticated disc controllers and operating systems can increase the aggregate I/O throughput of multiple disc systems to 100 - 150 I/O's per second by issuing I/O requests to several drives concurrently. Still, most data base applications are I/O bound. Today's CPUs are capable of executing 100,000 or more instructions for each disc access, so they spend much of their time waiting for the next I/O to complete.

CAN DISC CACHING HELP?

Disc caching can help significantly when read locality is high (serial reads for example). Caching is simple in concept. When a block of data is requested from the disc, the next several blocks are also read into memory in anticipation that those blocks will be requested next. If that happens, the requests can be satisfied directly from the memory cache instead of additional disc I/O's.

Unfortunately while caching increases effective I/O throughput in some cases, it reduces I/O throughput in others. When read locality is poor or write locality is high, caching decreases the effective I/O throughput because few I/O's are eliminated and the length of each I/O is increased. (See Note 1.)

In addition, caching can place high demands on CPU and memory resources. CPU's and semiconductor RAM are rapidly becoming cheaper and faster, so conserving the machine's most precious resource, disc I/O, is well worth the additional CPU & memory expense. The problem is that effective disc caching in a data base environment is difficult to achieve, and caching can actually degrade performance as the Guttman study shows. (See Note 1.)

For example, IMAGE master entries are accessed using both calculated reads (in mode 7 DBGETs or in DBFINDs on details) and serial reads. Optimum performance would be achieved if we could disable caching on calculated reads because reading several unneeded blocks of data in addition to the entry we want slows disc I/O and wastes CPU and memory resources. Conversely, we would like to do huge cached reads from the disc when serially reading a master, because the total amount of disc I/O required could be reduced by a factor of 10 or more.

Cached access to detail records suffers from the same problem. In a serial read, large "fetch quantums" (cached reads) are obviously desirable. But in chained reads, which are much more common, the optimum cache length varies from call to call. A long chained read on the primary path of a freshly reloaded detail could benefit greatly from a large fetch quantum. Conversely, chained reads on any other path, short (1 or 2 entry) chained reads, and the chained reads implicitly done by IMAGE when detail records are added would all benefit if caching were disabled. Why? Because several blocks of data are being read when only one is needed.

In general the more data sets that need to be accessed in an inquiry, the less benefit disc caching provides. For example highly normalized data bases, while more flexible to access and easier to maintain, can degrade report and transaction performance markedly because more data sets need to be accessed to retrieve the desired information. Hence the number of I/O's required is increased and the potential benefits of disc caching or large block reads are eliminated due to poor locality. Normalization is an important consideration in all data bases, but the extreme degree of normalization used in most relational data bases has contributed greatly to their poor performance reputation.

Note 1: Paula Guttman of Bell Communications Research analyzed the performance impact of caching on large IBM mainframes. Her study reveals that caching is a two-edged sword, helping performance in some situations and hurting in others. The paper, "Methods for the Deployment of IBM 3880 Model 13 Cached Storage Controllers" was presented at the CMG XV International Conference on the Management and Performance Evaluation of Computer Systems, December 1984.

I'm probably going to get in trouble for saying this, but effective disc caching on data base applications can only be accomplished by the DBMS. The DBMS has the best idea what data is likely to be accessed next. Ideally the DBMS would even accept occasional help from the application program, since in some cases the program might have a better idea what data will be requested. At present, the operating system is responsible for controlling caching, which is unfortunate since it is totally in the dark regarding what the user, application program, and DBMS will do next.

There is a common misconception at many sites who think they are CPU-bound when in fact they are not. This results from the high CPU and memory utilization seen when disc caching is being used. Lengthy serial reads using disc caching can benefit somewhat from a faster CPU because records in the cache can be accessed faster. Even with a 20 MIP CPU, however, those serial reads could only achieve a speed matching that of currently available utilities such as Suprtool and High Performance Quiz. These utilities do large blocked (MR/Nobuf) reads directly from disc to the program's stack. No matter how fast the CPU, reading from disc to a cache buffer and then from the cache buffer to the program's stack will be slower than going directly from disc to stack. So when fast serial access to one data set is required, software solutions are the best answer.

Most reports and nearly all transactions require many more random (calculated, chained, directed) reads than serial reads resulting in poor disc locality. In these situations, caching helps very little. The caching software simply has too little knowledge of what data will be requested next to be effective. A faster CPU can do nothing to address this fundamental problem. It's a bit like running a program with an infinite loop: a faster CPU would make the loop execute faster but the program won't finish any sooner.

WHAT IS THE SOLUTION?

There is only one. The amount of disc I/O required to "get the job done" must be dramatically reduced. In data base applications, this means that the amount of I/O needed to select and retrieve a desired subset of records for display or reporting must be cut drastically and the amount of I/O required to complete a transaction must be minimized.

How can that be done?

There are 5 major ways:

- Sophisticated indexing
- Optimized data base design
- Advanced reporting techniques
- Smart buffering (disc caching) by the DBMS
- Routine maintenance (physical re-ordering of the data)

SOPHISTICATED INDEXING

Efficient, flexible, and powerful indexing of the information in a data base is the only way to dramatically decrease the disc I/O needed to retrieve that information. Indexing allows us to locate a desired subset of records without scanning the data files. With advanced indexing techniques, record selections by virtually any criteria can be accomplished using relatively few disc reads. The disc I/O required for a retrieval can be reduced by a factor of 1,000 or 10,000 or more, the time reduced from hours to seconds.

OMNIDEX, an enhancement to HP IMAGE, provides such state of the art indexing capabilities. Information can be accessed in seconds by any combination of words and values across multiple fields, regardless of data base size. Features include:

* Record selection by multiple fields without serial reads

* Generic (partial key) retrieval and sorted sequential access.

* Multiple keys in masters

* Keyword retrieval on textual data

In an OMNIDEX retrieval, records are qualified at the rate of 10,000 records per second per keyword (selection value), regardless of the size of the data set. Suppose a data set of one million companies contained 10,000 company contacts with the name MARK, 30,000 companies in DENVER, and 8,000 companies whose name contained the word SYSTEMS. OMNIDEX would qualify all MARK's in DENVER who worked for a SYSTEMS company in about 5 seconds. In comparison, a serial read of the data set on a lightly loaded system would take about an hour, a chained read along the City path (the only practical IMAGE or ALLBASE key in this example since they do not provide keyword retrieval) would take 10-20 minutes. (See Figure 1.) No DBMS available on any machine today approaches the retrieval speed and flexibility of OMNIDEX.

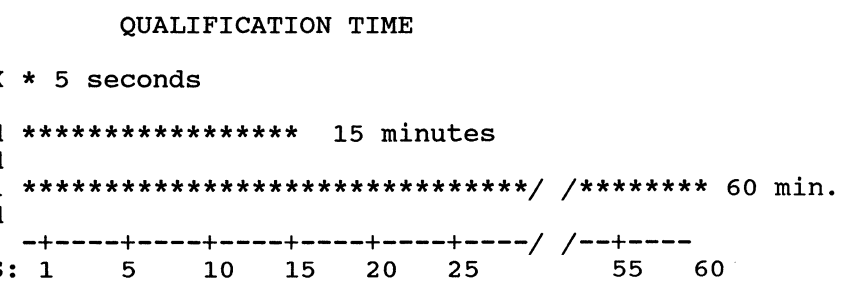


Figure 1. Multiple Field Selection on a 1,000,000 Record Data Set

OMNIDEX can provide almost instant access to data that would otherwise be inaccessible in on-line applications and can dramatically reduce the time it takes to select and report a subset of records in a data base. But two issues come to mind: What effect does the indexing overhead have on transaction speed? And how can OMNIDEX help when a report must include all or nearly all records rather than a subset?

The second question is easy to answer: OMNIDEX won't do anything for you, but data base design, maintenance, and advanced reporting techniques can do a great deal. More on this later.

What about the effect of indexing overhead on transaction volume? That depends. OMNIDEX uses very sophisticated indexing techniques that are low in overhead, but all indexing costs disc I/O's. The goal is to minimize the total I/O's required by a transaction, which has two parts: validation and updating. Validating a transaction requires retrievals on data sets that should be heavily indexed to permit the fastest, most flexible access possible. Master files, for example, are frequently read and rarely updated. They should be heavily indexed by keys and keywords to allow instant access by any words or values in any combination of fields. High transaction detail sets should be lightly indexed to minimize update overhead.

If a high transaction data set also needs heavy indexing for reporting there are two solutions. One is to defer indexing to a batch run during off-hours, a feature provided by OMNIDEX. The other is to maintain a heavily indexed informational data base which is updated periodically from the operational data base. This is an almost ideal solution. The operational data base can be optimized for on-line updating, while the informational data base is optimized for reporting and ad hoc inquiry.

Typically an operational data base uses extensive OMNIDEX indexing on master files and non-volatile details, but minimum IMAGE or IMSAM (the Btree sorted sequential access component of OMNIDEX) indexing on high transaction details. An informational data base uses extensive OMNIDEX indexing on all data sets that require fast record selection.

OMNIDEX includes a batch utility that indexes data about 10 times faster than on-line updates, which makes the indexing of large data bases very practical. For example, a one million record data set with 8 keyword fields can be indexed in about 8 hours. This is more than 10 times faster than an IMAGE detail with 8 paths or a KSAM or relational data base file with 8 key fields. No DBMS available on any machine today approaches the indexing speed of OMNIDEX.

DATA BASE NORMALIZATION

Normalization seems to be a very popular concept these days, especially with relational data base advocates. I'm oversimplifying the concept, but for this discussion I will define normalization as the division of data into several files (data sets) to eliminate data duplication and conserve space. The files can be linked logically or physically or both, depending on the underlying DBMS.

For example an IMAGE data base contains masters and details which are linked by physical pointers, and it typically also contains master sets used as "code tables" which are logically linked to other sets. A state code table that contains the state name and time zone is an example. Given the state code we could display the state's name & time zone simply by referencing the code table.

Maintaining variable length data (for example invoice line items) in details and using code tables conserves space, eliminates data redundancy and simplifies maintenance ("change it in one place instead of a thousand"). These normalization techniques are in common use. They offer numerous benefits, but they have one serious drawback: they can destroy report performance.

The reason is simple. A report on a heavily normalized data base has to link to numerous sets, requiring far more disc I/O than a report on a single data set. Disc locality is lost, rendering caching or large block reads by the program useless. If a small subset of records is of interest and they can be quickly isolated by indexing (they always can with OMNIDEX) the disc I/O is manageable. But if a large number of records must be retrieved for the report the amount of disc I/O will be enormous.

The fault lies with the data base design, not IMAGE or the HP3000. If the application were ported to an IBM mainframe, the performance would still be bad. A little faster because IBM disc drives are a little faster (40 I/O's per second rather than 30), but still bad.

OPTIMIZED DATA BASE DESIGN

How can we improve the performance? If any of the code tables are small and not being updated by other processes, programs can keep a copy of them in memory. That eliminates the disc I/O associated with accessing them. Beyond that, there is only one option to improve performance: de-normalize the data base.

Consider some rough numbers. Suppose a 100,000 entry master has an 800,000 entry detail linked to it. Assume that we generate a report by serially reading the master, linking to the detail, and linking from the detail to another master. For example the linkages might be Customer master to Invoice detail to Parts Master. Over 800,000 I/O's would be required just to access the second master once for each detail record, regardless of caching or optimized ordering of those detail records.

800,000 I/O's at 30/second takes nearly 8 hours, which would have to be added to the time it took to access the detail entries, sort and output the data. This report would take a long time.

Now suppose we de-normalized this data base by taking all the data needed by the report from the second master and duplicating it in the detail. Obviously this can seldom be done on operational data bases, but can easily be done on an informational data base that was updated weekly or monthly. The 800,000 I/O's that were required to access the second master would be eliminated shaving 8 hours off of the report time! How long would the report take now?

That depends. This is a situation where disc caching, CPU speed, and careful data ordering can really pay off. If the detail was freshly reloaded in primary path sequence and caching was enabled with a fetch quantum of 32 sectors, all master and detail records could be read with less than 35,000 I/O's. This assumes a master record size of 512 bytes (16 per cache buffer) and a detail record size of 256 bytes (32 per cache buffer). That is a relatively large detail record but denormalization would have increased its size. Assuming 15 I/O's per second for the large (but very effective!) cached reads, the total retrieval time for the report would be only 40 minutes.

To illustrate the value of optimum data ordering and caching in this example, the same report without caching would require about 110,000 I/O's (assuming Blockmax=1024). The same report with a worst case data ordering, which some production data bases approach, and the original level of normalization could take 1,600,000 I/O's -- 2 per detail record, regardless of whether caching was enabled or not. This is a performance difference of 40 to 1 based solely on optimized data base design and data ordering!

Yet you could throw hundreds of thousands of dollars of hardware at the problem instead, and see NO improvement. That's why I always get a glazed look in my eyes when people tell me they need more horsepower to solve the data base performance problems on their 68. It makes me wish I sold hardware.

Some of you are wondering what assumptions I made on chain lengths in the above example. After all the purpose of ordering detail records in path sequence is so that a chained read is localized to one portion of the file. If so, and if the chains are long (many records with the same key), we accrue all the benefits of IMAGE buffering and disc caching if it's enabled.

For example, a chained read of 32 256-byte records could be accomplished in one 32-sector cached read from the disc if all 32 records were contiguous. But if a chain contained only 1 entry, the benefits of caching and buffering would be lost because 32 records were read to get the 1 needed. Right? Not in this case.

If the records in the master and its details are in the same physical order by key value, the length of the chains is totally insignificant. As we serially read the master and link to its details, the chained reads against the details will proceed in their physical order. Hence every cached read will be a "perfect" one giving us precisely the entries we need next, regardless of chain lengths. At the end of a detail chain, the next master entry in the cache will match the next entry in the detail cache. (See Figure 2.)

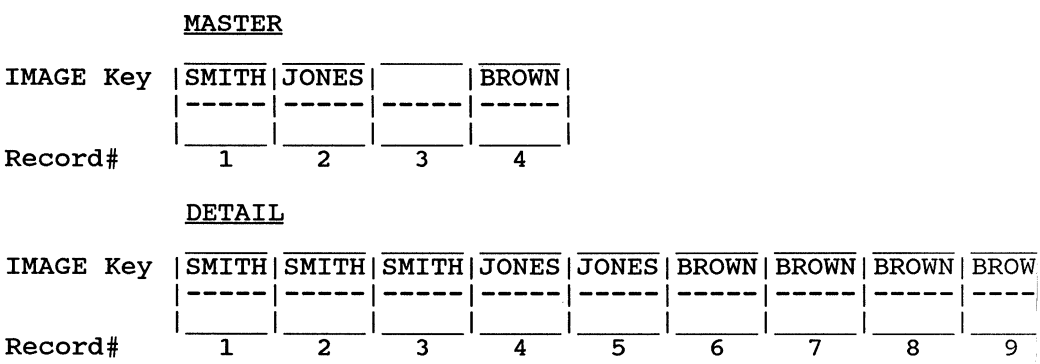


Figure 2. Optimum Data Ordering for Hashed Master

The next question is: How do you load the detail records in the same physical order (by key value) as the master? The answer is a little embarassing. You cannot do it with DBMGR, a product I wrote a few years ago. DBMGR sorts the detail records by primary key value before reloading them. This benefits chained reads (especially long ones), but not as much as it would in this situation if DBMGR sorted the records by the key's hash value instead. The hash value is the target record number of the corresponding master entry. Ignoring synonyms, the hash value is the physical order of the master entries. We'll enhance DBMGR to optionally sort by hash value soon, probably by the time you read this, so that details can be reloaded in the same physical order as their corresponding master.

A chained DBUNLOAD/LOAD would order the details optimally as long as you didn't change the master's capacity in the new data base. Rego still hasn't sent me my ADAGER tape (where is it Alfredo?), but I assume DETPACK will do it for you. Finally, if you're loading an informational data base, you can guarantee perfect detail ordering regardless of synonyms by loading the master first, then loading the detail by serially reading the informational data base master and linking back to the operational data base to unload its detail entries with chained reads.

OPTIMIZED REPORT PERFORMANCE

Because this data base is so optimized for fast retrieval, report performance will be heavily influenced by several other factors. If a large number of records are extracted, the I/O overhead incurred writing the records to a file and the time required to sort those records become important. Large block (MR/Nobuf) writes to the unload file should be used. Otherwise, writing to that file will create a bottleneck given the high speed at which data can be unloaded from the data base. Using large block writes, very high (1000 sectors/second) transfer speeds can be attained.

Since nearly perfect disc caching will occur during extracts from this data base, CPU speed will be a major factor influencing performance. Virtually every data base application on the HP3000 is disc I/O bound, so CPU speed has little impact. This application, however, would have such high I/O efficiency under caching that it would be CPU bound. Caching is quite CPU intensive, so perfect caching (always reading the right records into the cache) makes if difficult for the CPU to keep up with the disc drives.

The next step to improving performance on this application is to place the master, detail, and unload file on three separate drives. Because the data is physically ordered in exactly the way it will be extracted, the extract and write can be accomplished by positioning the disc heads at the beginning of each of the three files and slowly spiraling in. A full track or cylinder of data would be transferred, then the head would index in one track and continue. If no other users were competing for disc I/O's during the report, the data throughput would be very high assuming the CPU could keep up.

The MPE sort is fast, but given the speeds at which data can be extracted from this data base, sort times could become a major portion of the report time. One simple solution would be to use IMSAM. IMSAM provides sorted sequential access and partial key retrieval by any fields in an IMAGE master or detail. The fields need not be IMAGE keys. Records can be extracted in sorted sequence of any IMSAM key, eliminating the need for a separate sort operation.

In this application, however, there is a caveat. The physical order of the records in the data base will not match the sorted sequence desired. The records are physically ordered by hash value rather than by sorted sequence of any IMSAM key field. Hence disc locality will be lost and caching will provide no benefit, returning us to a 1 record per I/O situation. When a subset of records is of interest, IMSAM retrieval is the optimum choice. 1000 I/O's to read 1000 master entries in sorted sequence is much faster than 10,000 I/O's to serially read a 100,000 entry master and sort the 1000 entries extracted. But if all the records need to be retrieved from a data base with a hashed master, a serial read and sort is much faster than using an IMSAM sequential read.

Most data bases require a wide mix of inquiry and reporting. On-line inquiries must quickly select a few records for display. Fast selection is the critical issue, which can be accomplished using IMAGE, IMSAM, and OMNIDEX indexing options. Reports cover both subsets and full data sets of records. In small subsets, selection speed determines performance, so indexing power is the controlling factor. As the subset increases in size, retrieval speed starts to dominate, so data ordering and organization become crucial. Can a data base be optimized for all sizes of reports?

THE LOWLY J2 KEY

A data base can be optimized for any size report by using a J2 field for the master's key. J2 keys are much maligned in literature, to the point where most data base designers refuse to use them. Because they can cause terrible synonym problems, J2 keys are unsuitable in most situations unless they meet one simple criteria. If the value of a J2 key does not exceed the capacity of the master set, the record number of an entry will equal its key value. Several wonderful things happen as a result:

- The master set will contain no synonyms.
- No pad space is required in the set (no disc space is wasted)
- Master entries with J2 keys can be batch loaded many times faster than hashed masters.
- Since the physical position of an entry is its key value you can physically order the entries in any sequence desired.

The first two benefits are minor. The third is significant and can be easily accomplished in batch loads. If entries are added in ascending key sequence (1, 2, 3, ...) with the "output deferred" option enabled (DBCONTROL mode 1), a full block of entries can be added before IMAGE posts the block to disc. For example if a master set had a blocking factor of 4, 4 entries could be added for each disc I/O. The larger the block size (IMAGE Blockmax), the higher the blocking factor and the faster the load.

The major benefit of J2 keys is the ability to physically order master entries in any sequence desired. This is done by assigning key values starting at 1 to the entries in sorted sequence of another field, such as an IMSAM key field. (See Figure 3.) This yields optimum performance for all reports that retrieve data in that sequence. Since the data is physically ordered in the same sequence that we're retrieving it, the full benefits of caching are available during the retrieval stage of the report. In addition, the need to sort the output is eliminated. This design permits the ultimate inquiry and report performance on any computer system.

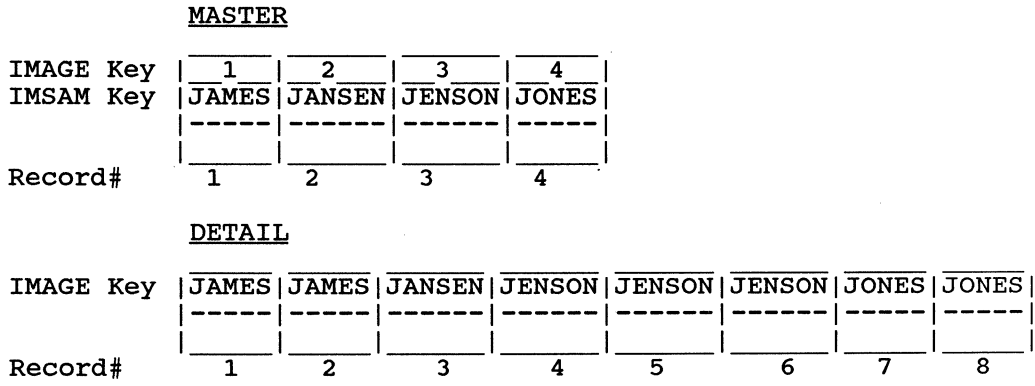


Figure 3. Optimum Data Ordering for Master with J2 Key

Consider an accounting data base with a 20 byte account code as the key. Most likely, the account code is comprised of several components concatenated together to form a hierarchical key. Sorted sequential access allows us to select a subrange of accounts for review, frequently necessary in accounting applications.

If we redesign the data base to use a J2 field as the IMAGE key instead, and make the account code an IMSAM key, we can map this data base into the ultimate performance design just described. IMSAM sequential retrievals by the account code will access records in the same sequence as their physical order, so any size retrieval will be optimized. Records can be added at will since IMSAM indexing is updated automatically. Report performance will gradually decline as more records are added because the new records are not in sorted sequence. Periodically the data base can be reloaded to reorder the data in account sequence for maximum performance.

If OMNIDEX indexing is added to this data base, it becomes the ultimate informational data base. Users can instantly access records by selection criteria on multiple fields, perform keyword retrievals, access records in sorted sequence of any field or combination of fields, and generate reports up to a hundred times faster than on most data bases. Giving users such fast, flexible access to information will dramatically increase the computer's value to any organization.

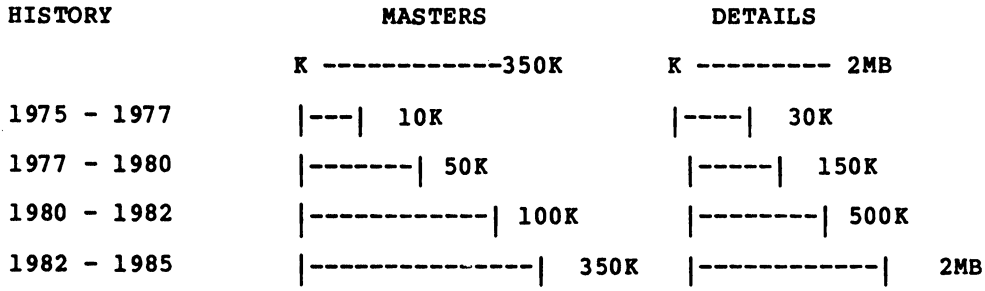
All the tools you need to design data bases with incredible performance on the HP3000 are available. The rest is up to you. Just remember, make every I/O count. It's that simple, and that difficult. And it's the only way on the HP3000 or any other computer system.

TRENDS IN "IMAGE"

BRADMARK COMPUTER SYSTEMS

YEAR	SYSTEM	MEMORY CAPACITY	TERMINAL CAPACITY
1975	CX	64K	5
1976	Series II	512K	16
1978	Series III	2MB	64
1979	Series 30/33	1MB	32
1981	Series 40	2MB	56
1982	Series 44	4MB	96
1982	Series 64	8MB	144
1983	Series 48	4MB	104
1983	Series 39	3MB	32
1984	Series 68	8MB	336
1984	Series 42	3MB	60
1985	Series 37	2MB	7

IMAGE DATABASE SIZES



RESULT

- PERFORMANCE CONSCIOUSNESS
- THROUGHOUT AWARENESS
- LOAD SCHEDULING
- VISIBILITY INTO "IMAGE" DATASETS

IMAGE "B" CONTROL BLOCKS

SDBCB - SYSTEM DATA BASE CONTROL BLOCK

RDBCB - REMOVE DATA BASE CONTROL BLOCK

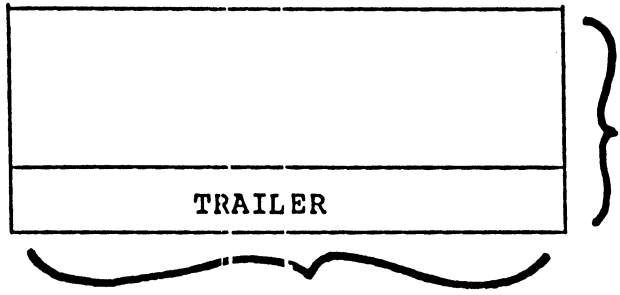
DBCB - GLOBAL DATA BASE CONTROL BLOCK

ULCB - USER LOCAL CONTROL BLOCK

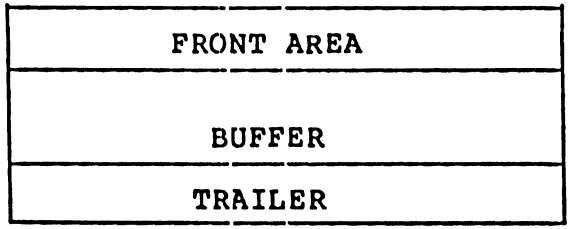
ILCB - INTRINSIC LEVEL RECOVERY CONTROL BLOCK

IMAGE "B"

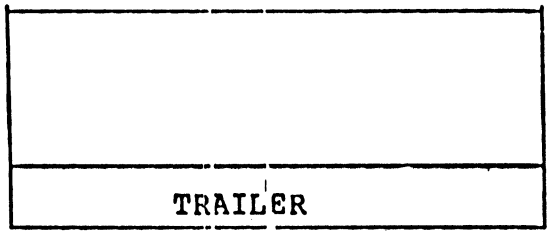
SYSTEM DATABASE CONTROL BLOCK
(SDBCB)



DATABASE CONTROL BLOCK
(DBCB)



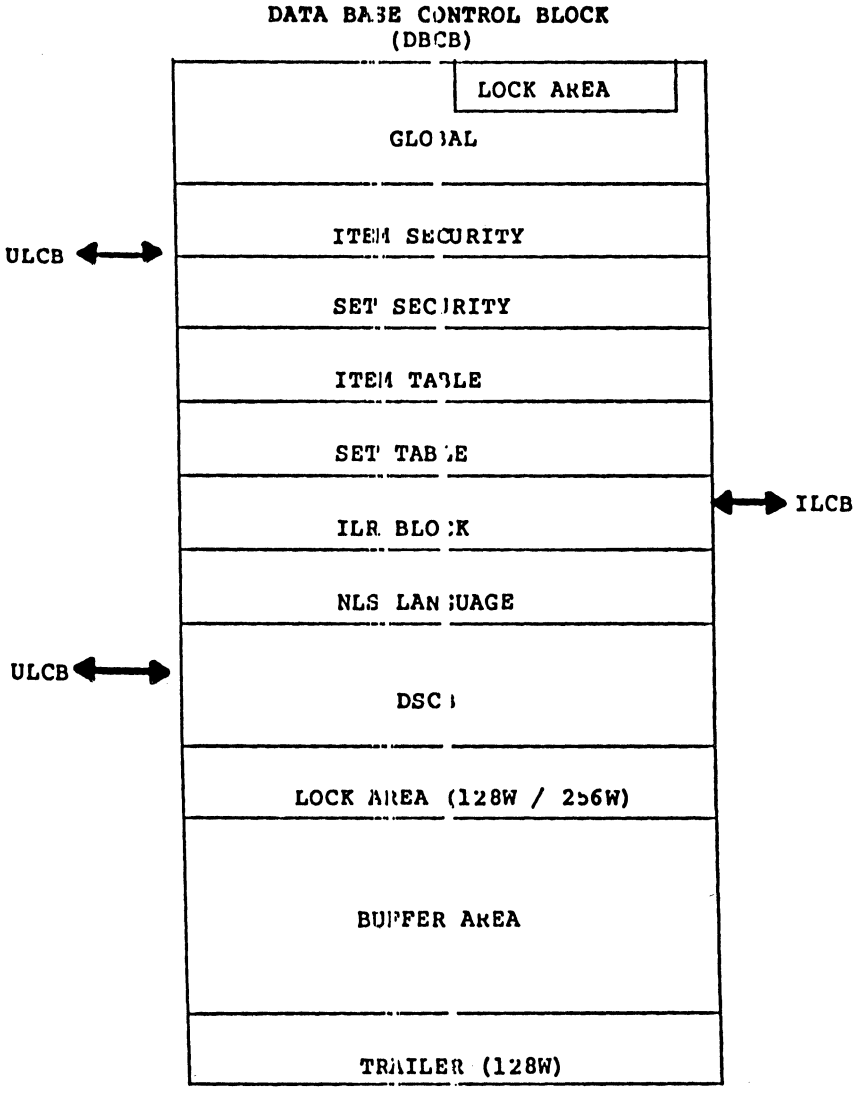
USER LOCAL CONTROL BLOCK
(ULCB)



GLOBAL DBCB

CONTAINS:

- DATABASE STRUCTURE
- DATABASE USERS
- LOCKING AREA
- TRANSACTION STAGING


DATA BASE CONTROL BLOCK
(DBCB)
LOCK AREA
GLOBAL
ITEM SECURITY
ULCB
SET SECURITY
ITEM TABLE
SET TABLE
ILCB
ILR BLOCK
NLS LANGUAGE
ULCB
DSCB
LOCK AREA (128W / 256W)
BUFFER AREA
TRAILER (128W)

DBCB BUFFER SIZE

TOTAL SPACE	32,600
MAX LOCKING AREA	-6,144
FIXED AREA	-150
	26,206

INTERNAL TABLES:

- ROOT FILE LENGTH (5000)
- TRAILER LENGTH
- NO. OF ITEMS
- NO. OF SETS SECURITY
- 12 X NO. OF USERS

SPACE AVAILABLE FOR BUFFERS

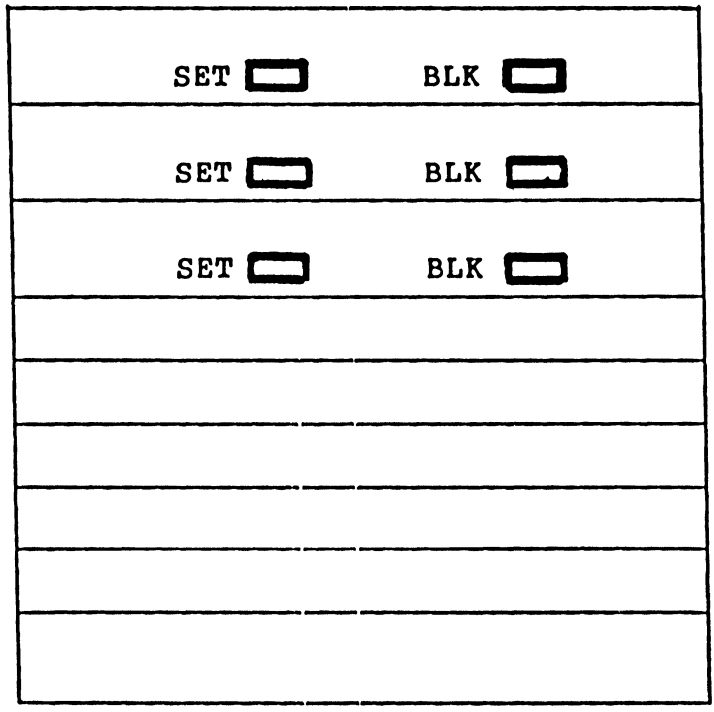
INTERNAL BUFFERS

TYPICAL SPACE (20,000 WORDS)

BLK SIZE	BUFFERS
512	40
1024	20
2048	10

OVERRIDE WITH BUFFSPECS

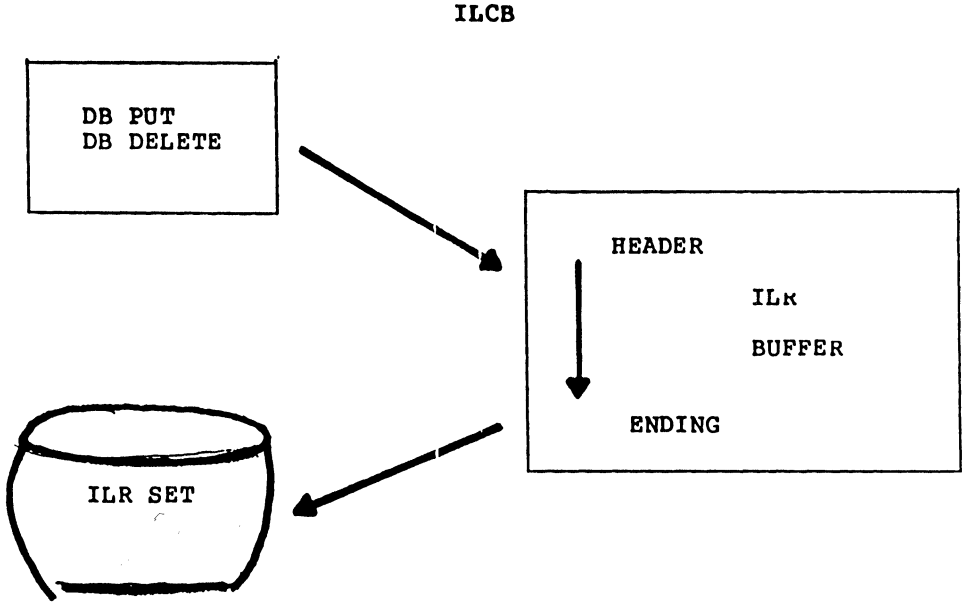
BUFFERS



ULCB

CONTAINS:

- USER CAPABILITY STRUCTURE
- TRANSACTION STAGING
- USER ITEM LIST


ILCB
DB PUT
DB DELETE
HEADER
ILR
BUFFER
ENDING
ILR SET

TREND CHANGES IN IMAGE DUE TO CURRENT LIMITS

LIMITS	"B" IMAGE	TURBO IMAGE
DATA ITEMS	255	1023
DATA SETS	99	199
DATA ITEMS/SET	127	255
DATA ENTRIES/SET	8M	2B
DATA ENTRIES/DETAIL CHAIN	65K	2B
DETAILS/MASTERS	16	16
MASTERS/DETAILS	16	16
MAX ENTRY SIZE	4096	4096
LEVEL WORDS SUPPORTOR	YES	NO
GLOBAL DBCB	1	1
LOCAL DBCB	-	+/USER

TURBO MODIFICATION

- ROOT FILE
  - ITEM & SET TABLE MAP
  - ITEM TABLE FORMAT
  - EXTENDED SET TABLE FORMAT

- SETS
  - POINTER FORMAT
  - ENTRY COUNT WITHIN MASTER

ACTUAL FILE LIMITATIONS

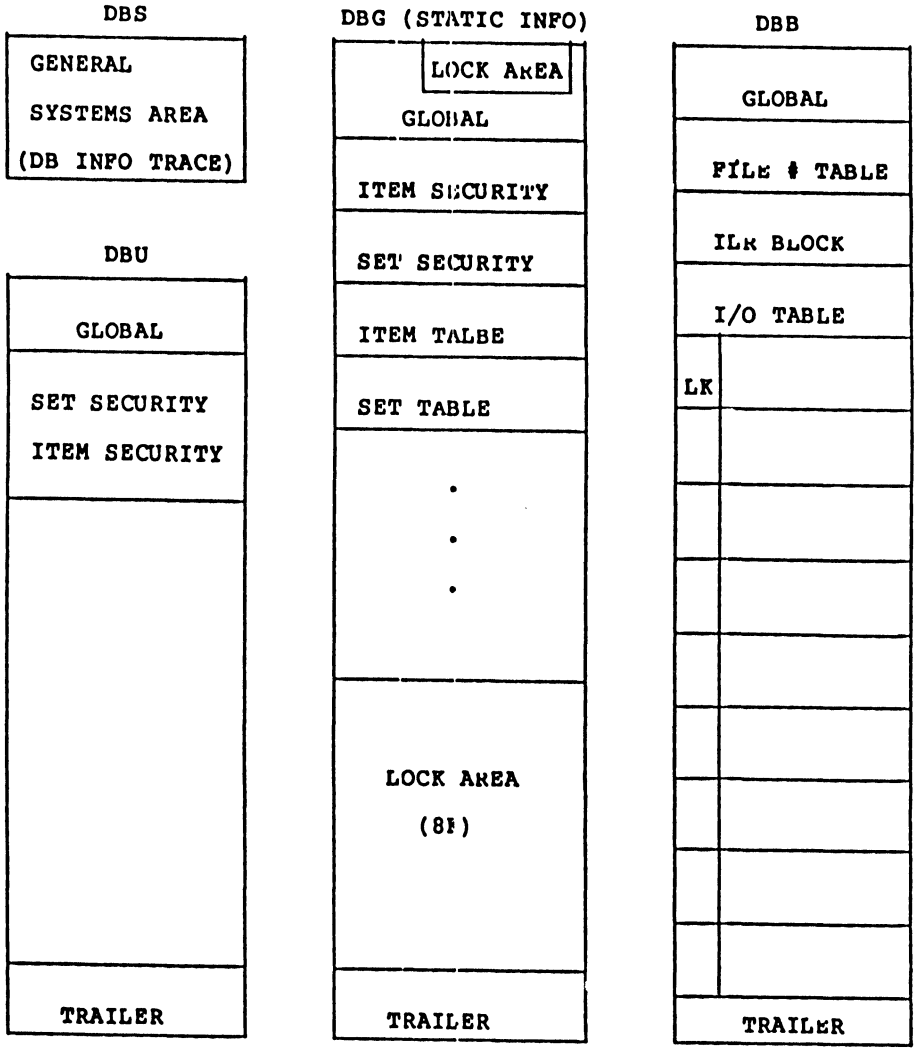
MPE RESTRICTIONS:

1 EXTENT	65K	sectors
1 FILE	32	EXTENTS

IMAGE'S SIZES

DEFAULT BLOCK SIZE	BLOCKING FACTOR	ACTUAL FILE LIMIT
512	1	500
512	2	1 M
512	4	2 M
512	8	4 M

TURBO IMAGE


DBS
GENERAL
SYSTEMS AREA
(DB INFO TRACE)
DBU
GLOBAL
SET SECURITY
ITEM SECURITY
TRAILER
DBG (STATIC INFO)
LOCK AREA
GLOBAL
ITEM SECURITY
SET SECURITY
ITEM TALBE
SET TABLE
LOCK AREA
(8I)
TRAILER
DBB
GLOBAL
FILE # TABLE
ILK BLOCK
I/O TABLE
LK
TRAILER

MAJOR TURBO IMPACTS

- BREAKING UP THE GLOBAL DBCB into
  - 1 GLOBAL DBCB
  - MULTIPLE LOCAL DBCBs
- SINGLE THREADING
  - DBPUT
  - DBUPDATE
  - DBDELETE
- MULTI - THREADING
  - DBGET
  - DBFIND

THREADING CONCEPT

SINGLE I/O s ARE MULTI THREADED

MULTIPLE I/O s ARE SINGLE THREADED

'THREADING

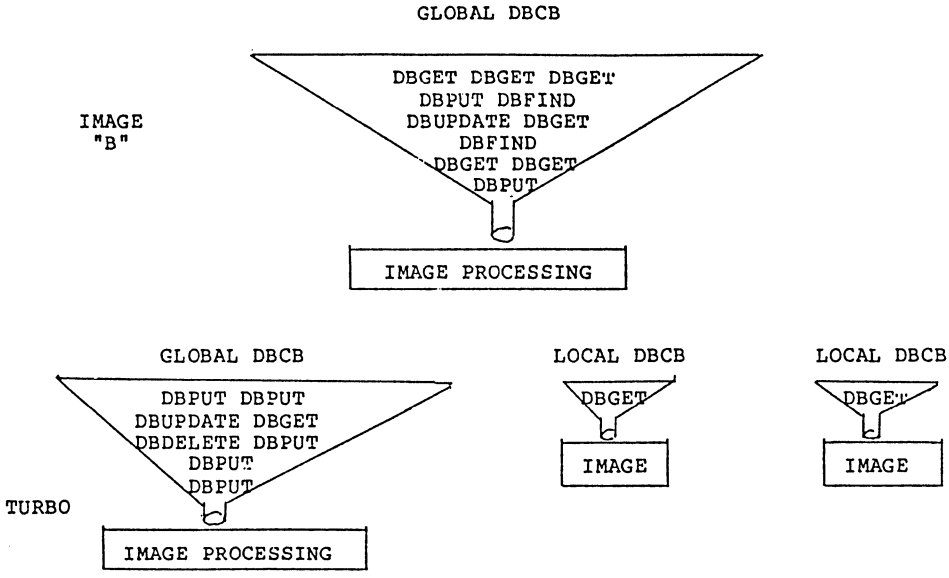
SINGLE:

DBPUT, DBDELETE, DBFIND,
DBGET (2,3,7)

MULTI:

DBUPDATE,
DBGET (1,4,5,6,8,)

POTENTIAL PERFORMANCE IMPROVEMENT



QUEING UP THE SERVICE

A - GET (5)

B - UPDATE

C - GET

D - GET

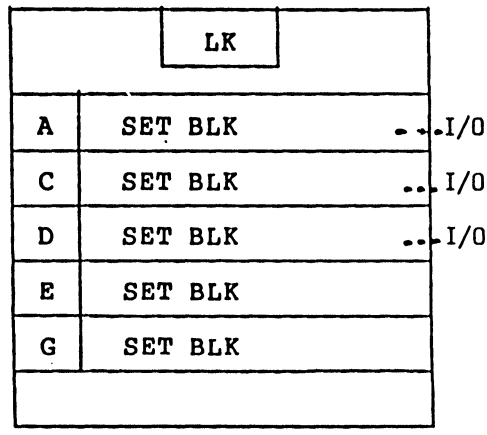
E - PUT

F - UPDATE

G - GET

B

F



WHAT DOES THIS ALL MEAN?

THE TREND IS TOWARDS LARGER DATABASES

THE PRODUCT RANGE HAS BROADENED SUBSTANTIALLY

THE NUMBER OF USERS PER SYSTEM HAS EXPONENTIALY INCREASED

NEED FOR "IMAGE" DIAGNOSTICS

TO DETERMINE AND EXPEDITOUSLY CORRECT CURRENT PROBLEM

TO KEEP TABS ON POTENTIAL PROBLEMS

TO HAVE BETTER CONTROL OVER YOUR ENTIRE "IMAGE" ENVIRONMENT

DB.GENERAL (3.A.00)
Licensed to: ACME MANUFACTURING INC.

MASTER MENU OF COMPREHENSIVE FEATURES

1.0 INTERNAL STRUCTURES	2.0 DIAGNOSTICS W/RECVRY	3.0 DATASET MAINTENANCE
[1.1] USER CLASSES	[2.1] MST SET ANALYSIS	[3.1] CAP TABLE ENTRY
[1.2] ITEM LISTS	[2.2] SYNONYM CHAIN TEST	[3.2] CAP MANAGEMENT
[1.3] SET LIST [W/RELATIONS]	[2.3] DTL SET ANALYSIS	[3.3] MST CAP CHANGE
	[2.4] PATH CHAIN ANALYSIS [OF MST-DTL LINKAGE]	[3.4] MST CAP SAMPLER
[1.4] SET-ITEM LISTS		[3.5] DTL CAP CHANGE
[1.5] SCHEMA GENERATOR	[2.5] CHAIN RESTORATION	[3.6] DTL SET REORGANIZE

4.0 GENERAL UTILITIES	5.0 ROOT FILE MAINTENANCE	6.0 COPYING FEATURES
[4.1] RESTORE	[5.1] PASSWORD CHANGES	[6.1] BASE GENERATOR
[4.2] RENAME	[5.2] ITEM CHANGES	[6.2] UNLOAD/RELOAD
[4.3] PURGE	[5.3] DATASET CHANGES	[6.3] TO/FROM "MPE" FILE
[4.4] ERASE	[5.4] SET-ITEM CHANGES	[6.4] BY SELECTED KEYS
[4.5] DEVICE TRANSFER	[5.5] DATAPATH CHANGES	[6.5] GENERAL FEATURES
[4.6] FAST COPY	[5.6] ACTIVATE CHANGES	

Option selected (option number, HELP or END):

POTENTIAL "IMAGE" PROBLEM

- SET CAPACITY OVERRUN
- EXCESSIVE SECONDARIES IN MASTERS
- POOR ENTRY DISTRIBUTION
- BROKEN SECONDARY CHAINS IN MASTER
- BROKEN CHAIN DIAGNOSTIC IN DETAIL
- DEFECTIVE LABEL IN SET
- INCOMPATIBLE CAPACITY IN ROOT FILE
- DEFECTIVE DELETE CHAIN

CAPACITY MANAGEMENT

CAPACITY RANGE PARAMETER SETTINGS:

(Please enter a percentage value for each request)

MAXIMUM THRESHOLD FOR EACH MASTER DATASET:

MAXIMUM THRESHOLD FOR EACH MASTER DATASET: 70

EXPANSION FACTOR: 15

MINIMUM THRESHOLD FOR EACH MASTER DATASET: 30

COMPRESSION FACTOR: 40

MAXIMUM THRESHOLD FOR EACH DETAIL DATASET: 90

EXPANSION FACTOR: 20

MINIMUM THRESHOLD FOR EACH DETAIL DATASET: 50

COMPRESSION FACTOR: 25

CAPACITY MANAGEMENT TABLE

DATA BASE: OEDB FRI, DEC 7, 1984 3:52 PM

SETS:	TYPE	CAPACITY	ENTRY COUNT	CAP THRESHOLD BELOW MIN(%)	CAP THRESHOLD ABOVE MAX(%)	CAPACITY CHNG COMPRS BY(%)	CAPACITY CHNG EXPAND BY(%)
PRFX	A	2511	1801	30	70	40	15
PRFD	D	4637	4489	50	90	25	20
CUST	M	5273	1437	30	70	40	15
SHIP	D	1282	486	50	90	25	20
ORDL	D	2593	1421	50	90	25	20
ORDM	M	2593	1421	30	70	40	15
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

CAPACITY MANAGEMENT USAGE

```
--- CAPACITY MANAGEMENT PROCESS

SOURCE DATABASE: OEDB
       PASSWORD:

DATA BASE: OEDB                          MON, DEC 17, 1984  6:27 PM

                                ENTRY  --- MINIMUM ---  -- MAXIMUM --
SETS:    TYPE  CAPACITY         COUNT  LEVEL  COMPRESS  LEVEL  EXPAND

 SHIP     D        1282          486    641       961    1154

***** previous set requires compression ---

    CAPACITY CHANGE IN PROGRESS ---

                486 RECORDS COPIED --

    DETAIL SET CHANGE SUCESSFULLY COMPLETED:

 ORDL     D        2593         1421   1037              2074
 ORDM     M        2593         1421    778              1556
   .      .          .            .      .        .        .        .
   .      .          .            .      .        .        .        .
   .      .          .            .      .        .        .        .
```

4.0 GENERAL UTILITIES

[4.1] RESTORE
[4.2] RENAME
[4.3] PURGE
[4.4] ERASE
[4.5] DEVICE TRANSFER
[4.6] FAST COPY

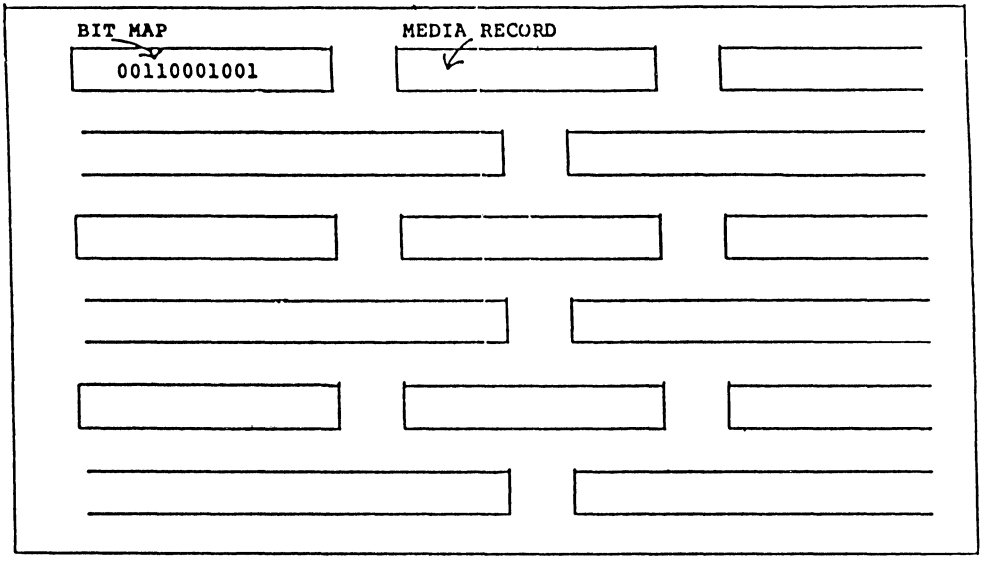
5.0 ROOT FILE MAINTENANCE

[5.1] PASSWORD CHANGES
[5.2] ITEM CHANGES
[5.3] DATASET CHANGES
[5.4] SET-ITEM CHANGES
[5.5] DATAPATH CHANGES
[5.6] ACTIVATE CHANGES

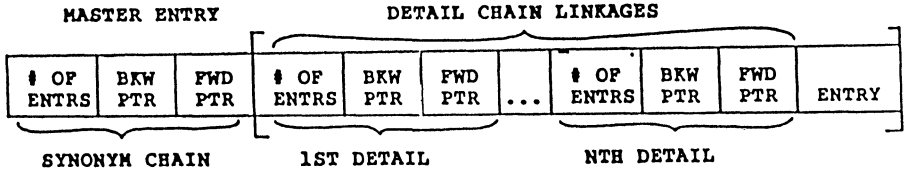
6.0 COPYING FEATURES

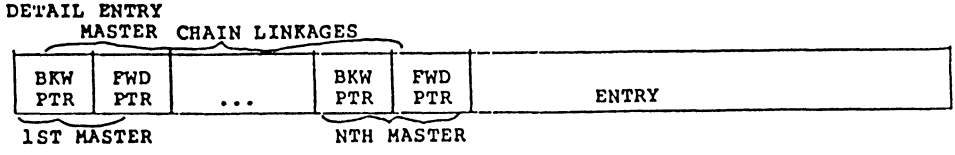
[6.1] BASE GENERATOR
[6.2] UNLOAD/RELOAD
[6.3] TO/FROM "MPE" FILE
[6.4] BY SELECTED KEYS
[6.5] GENERAL FEATURES

REVIEW OF "IMAGE" BLOCKING LAYOUT

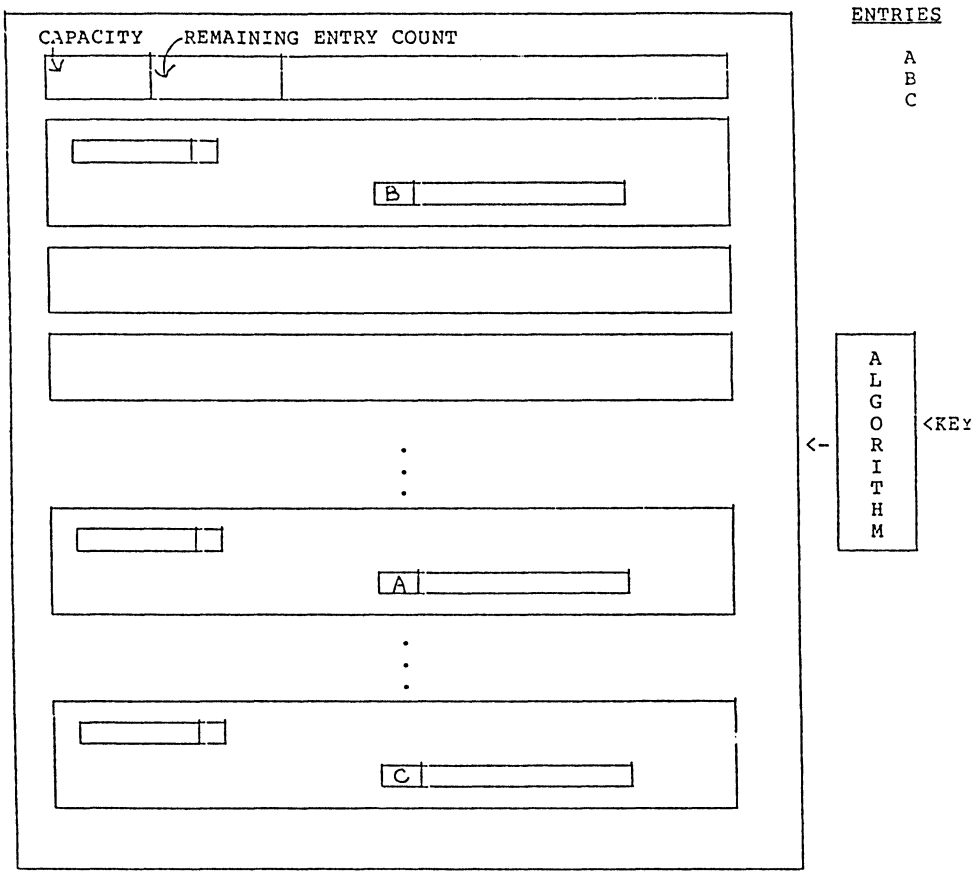


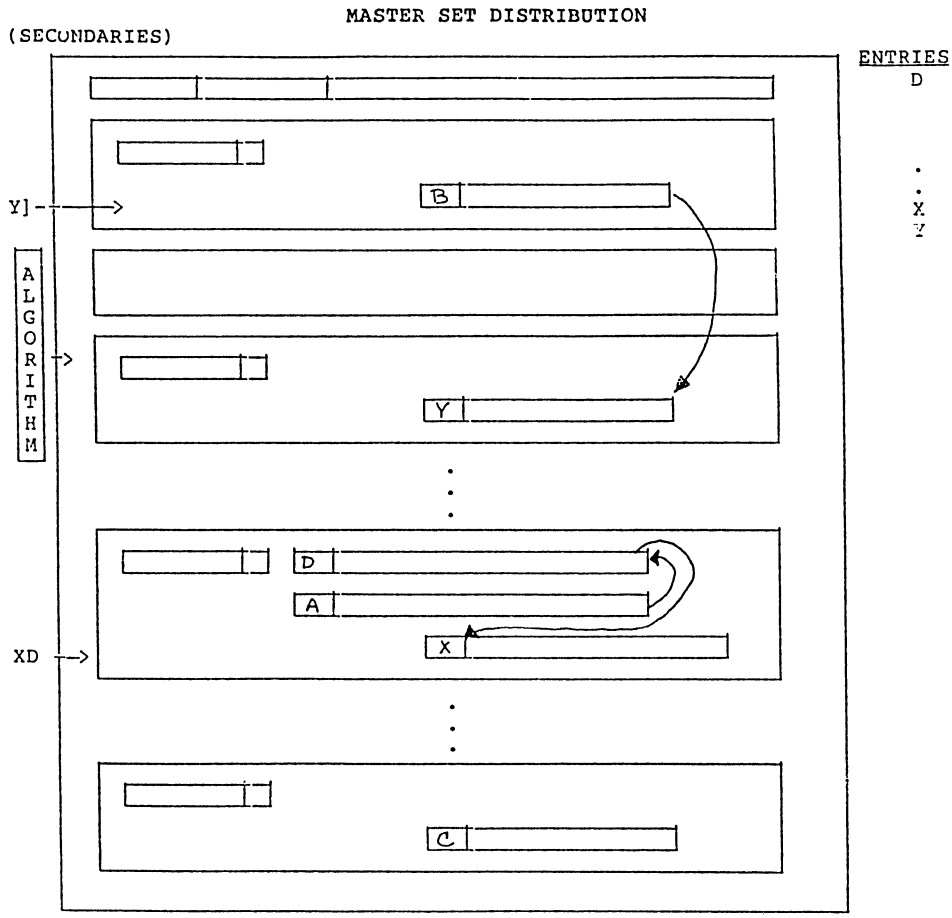
BIT MAP - BIT FOR EACH ENTRY IN BLOCK
MEDIA RECORD IMAGE ENTRY WITH ALL POINTERS





MASTER SET ORGANIZATION
(PRIMARIES)




MASTER SET DISTRIBUTION
(SECONDARIES)
ENTRIES
D
X
Y
Y]
ALGORITHM
B
Y
D
A
X
XD
C

MASTER SET DISTRIBUTION
(MIGRATING SECONDARIES)

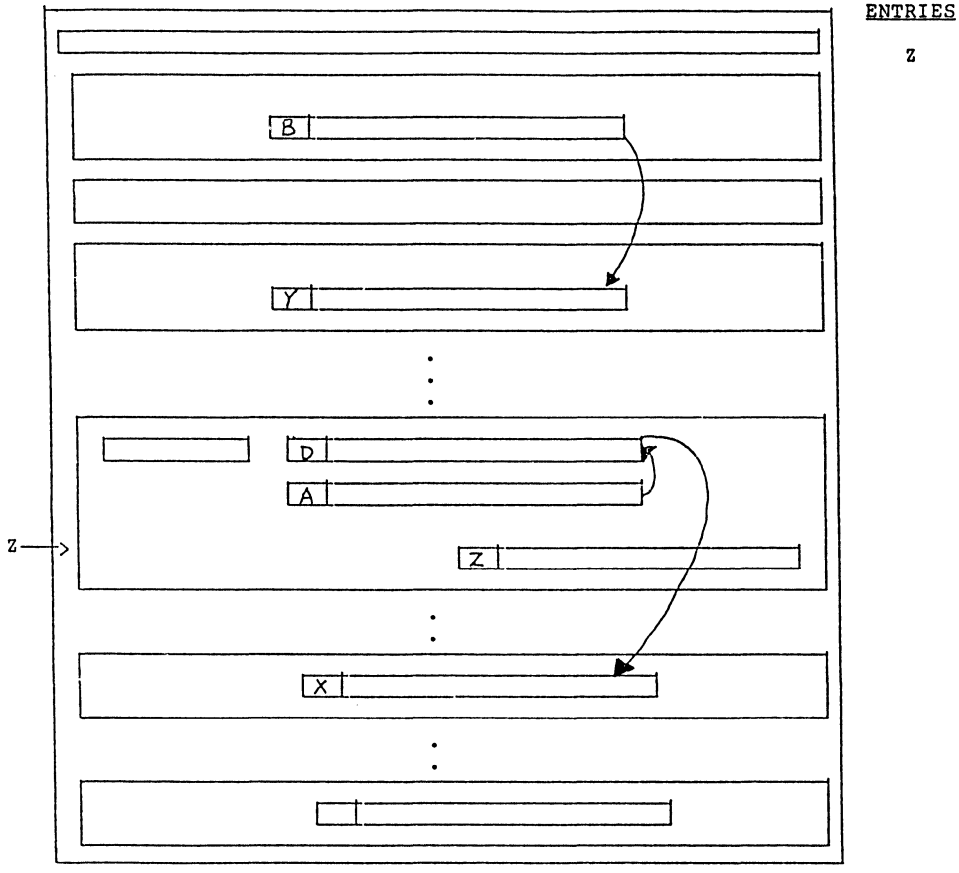


IMAGE ALGORITHMS

KEY: NUMERIC DATA TYPE

MODULE (I,J,K,R)

PRIME NUMBER - BEST CAPACITY

KEY: NON-NUMERIC DATA TYPE

HASH (U,X,F,P)

NON-PRIME - BEST CAPACITY

MASTER SET ANALYSIS

DATASET: CUST

GENERAL INFORMATION

DATASET NAME	CUST
SET NUMBER	03
SET TYPE	M
ENTRY SIZE	24
RECORD SIZE	106
BLOCKING FACTOR	04
NO OF POINTERS	6
SET CAPACITY	53276
ACTIVE ENTRIES	41296

SET CAPACITY	53276	HIGHEST ACTIVE ENTRY	53271
ACTIVE ENTRIES:		PRIMARIES W/ SECONDARIES	10945
PRIMARIES	20774	LONGEST SECONDARY	5
SECONDARIES	20522	AVERAGE SECONDARY	2
TOTAL ...	41296	POSSIBLE ENTRIES IN ERROR	0
REMAINING ENTRIES	11980	NO OF TOTAL BLOCKS	13,319
% SPACE LEFT	22%		

SECONDARY ANALYSIS OF MASTER SET

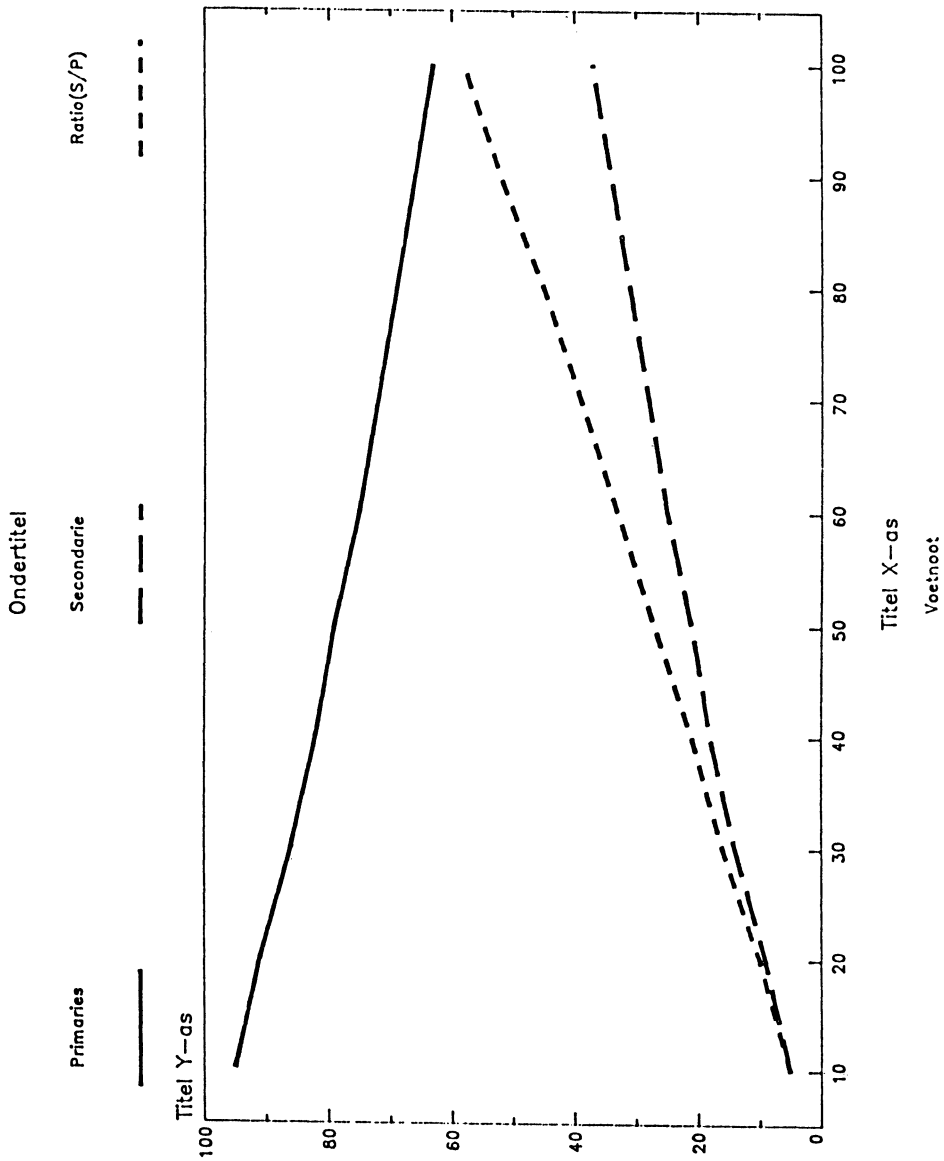
PRIMARIES	SECONDARIES							
	1	2	3	5	10	50	100	100+
20774	5070	4100	2043	1020	0	0	0	0

NO. PRIMARIES IN MASTER SET

$$\text{OPTIMAL PRIMARIES} = \frac{1 - e^{-u}}{u}$$

Where u = entries / capacity

% FULL	PRIMARIES	SECONDARIES	RATIO
10%	.951	.048	.050
20%	.906	.093	.103
30%	.863	.136	.157
40%	.824	.175	.213
50%	.786	.213	.270
60%	.751	.248	.329
70%	.719	.280	.390
80%	.688	.311	.452
90%	.659	.340	.516
100%	.632	.367	.581


Ondertitel
Primaries
Secondarie
Ratio(S/P)
Titel Y-as
Titel X-as
Voetnoot
100
80
60
40
20
0
10
20
30
40
50
60
70
80
90
100

CAPACITY SENSITIVITY

ENTRIES: 172

CAPACITY	% FULL	PRIMARIES	SECONDARIES 1	2	3	5	10	50
301	57%	114	23	12	9	4		
300	57%	36	6	8	6	5		111
293	58%	140	28	4				

FINAL RESULTS OF CAPACITY SAMPLING:

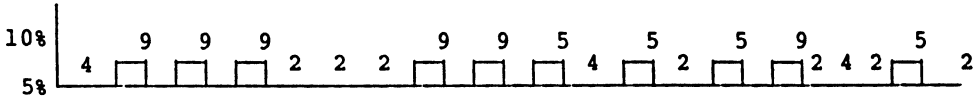
Cap.	Prim.	Sec.	Rat.	Av/se Ch.	Prim Dist.	Tot. Di.
60011	691	6	0.00	1.0	10.6	10.9
60012	690	7	0.01	1.0	27.0	27.2
60013	531	166	0.31	1.2	50.4	66.1
60014	694	3	0.00	1.0	24.8	24.9
60015	697	0	0.00	0.0	10.2	10.2

The capacity with the best primary to secondary ratio & set distribution is:

60015

Option selected
(£, MPE, HELP, or END): H

(MASTER DATASET DISTRIBUTION)



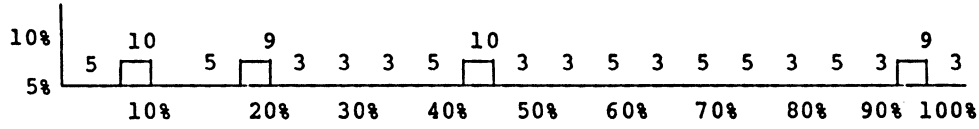
distribution of entries across the set ---->

ENTRIES PER DISTRIBUTION CATAGORY:

5%	1652	25%	825	45%	3717	65%	825	85%	1652
10%	3720	30%	820	50%	2064	70%	2064	90%	825
15%	3717	35%	830	55%	1652	75%	3717	95%	2064
20%	3714	40%	3717	60%	2064	80%	825	100%	825

The following distribution is for PRIMARIES only!

(MASTER DATASET DISTRIBUTION)



distribution of entries across set ---->

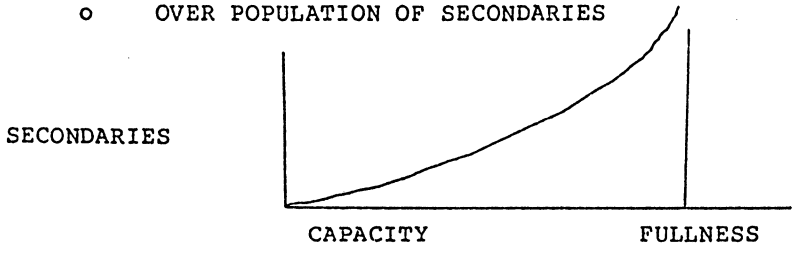
ENTRIES PER DISTRIBUTION CATAGORY:

5%	1039	25%	623	45%	2077	65%	623	85%	1039
10%	2077	30%	620	50%	623	70%	1039	90%	623
15%	1039	30%	626	55%	623	75%	1039	95%	1870
20%	1870	40%	1039	60%	1039	80%	623	00%	623

MASTER SET ANALYSIS

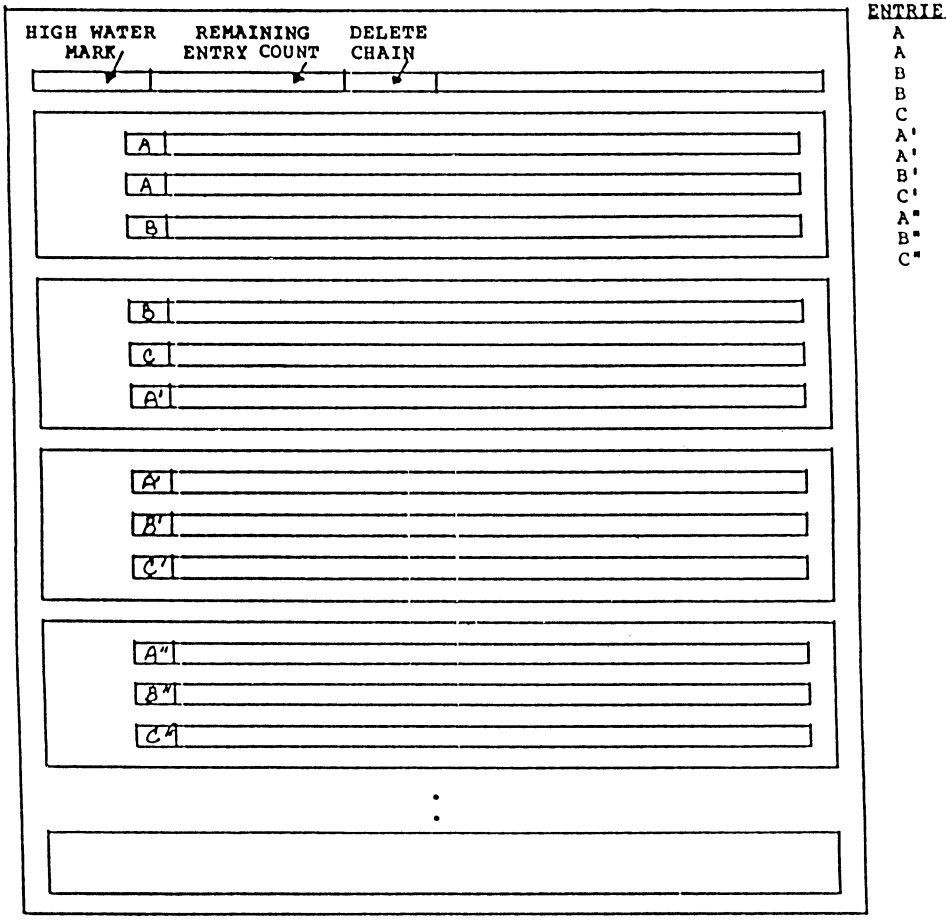
POTENTIAL PROBLEMS

- EXCESSIVE I/O PROCESSING DUE TO TOO MUCH RESIDUAL SPACE
- OVER POPULATION OF SECONDARIES

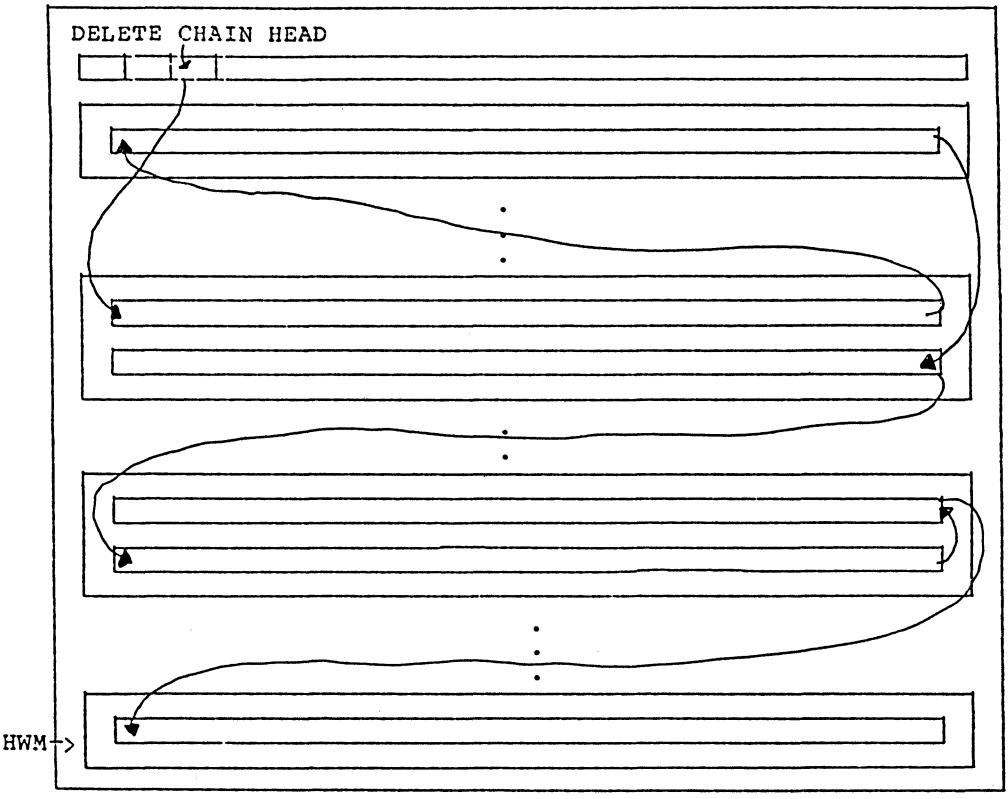


- POOR DISTRIBUTION OF PRIMARIES
- BROKEN SECONDARY CHAINS
- DEFECTIVE LABEL

DETAIL SET DISTRIBUTION



DETAIL SET DISTRIBUTION
(SIGNIFICANCE OF DELETE CHAIN)



DETAIL SET ANALYSIS

DATASET: SHIP

GENERAL INFORMATION:

DATASET NAME	SHIP
SET NUMBER	04
SET TYPE	D
ENTRY SIZE	64.
RECORD SIZE	68.
BLOCKING FACTOR	07.
NO OF POINTERS	1.
SET CAPACITY	111354
ACTIVE ENTRIES	93576

SUMMARY INFORMATION:

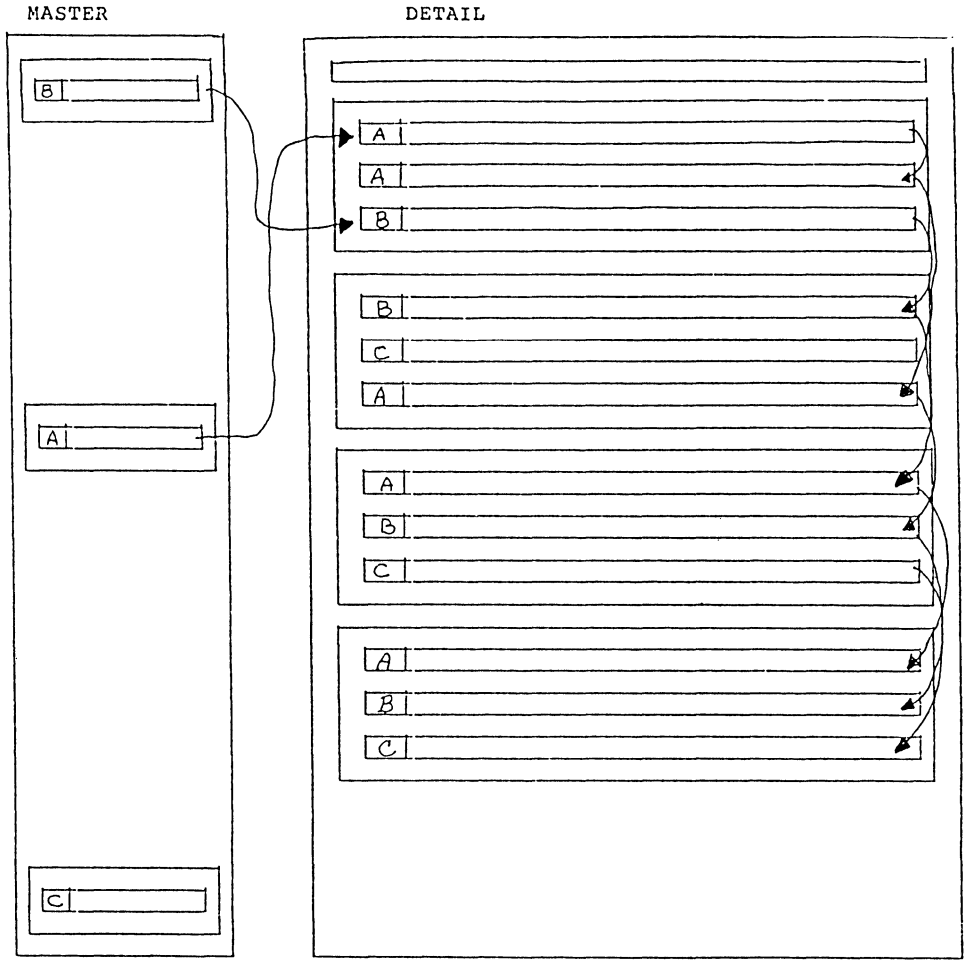
SET CAPACITY	111354	HIGHEST ACTIVE ENTRY	111207
ACTIVE ENTRIES	93576	SECOND HIGHEST ENTRY	111022
HIGH WATER MARK	111350	POSSIBLE ENTRIES IN ERROR	0
ENTRS ON DELETE CHAIN	17774	NO OF TOTAL BLOCKS	15907
INACT. ENTRIES COUNTED	17774	ENTRIES ON OTHER SIDE	
DEVIATION	0	OF HIGH WATER MARK	0
REMAINING ENTRIES	17778	TRUE REMAINING ENTRIES	17778
% SPACE LEFT	15	% SPACE LEFT	15

DETAIL SET ANALYSIS

POTENTIAL PROBLEMS

- CAPACITY MAY NOT BE CONSISTENT WITH ROOT FILE
- DEFECTIVE LABEL
  - BAD HIGH WATER MARK
  - BAD ENTRY COUNT
  - INACCURATE DELETE CHAIN HEAD
- BROKEN DELETE CHAIN

PATHING DISTRIBUTION



PATHING ANALYSIS

DATASET: SHIP

CHAIN TESTING PHASE ---

PROCESSING CHAIN THROUGH MASTER: CUST CHAIN 1 OF 1
AND DETAIL CHAIN 1 OF 1

MASTER COUNTED: 41296
MASTERS W/DETAILS: 12277

LARGEST DETAIL CHAIN: 85
SECOND LARGEST CHAIN: 72
AVERAGE DETAIL CHAIN: 7

DETAIL DISTRIBUTION SUMMARY:

PATH-SIZE	DETAILS	MASTERS	AVG	BLK READS	OPT READS	% INEFF
1-50	66180	11856	6	33549	11856	181
51-100	27396	421	65	14987	3914	282
101-500	0	0	0	0	0	
501-1000	0	0	0	0	0	
1001-5000	0	0	0	0	0	
5001-12500	0	0	0	0	0	
12501-25000	0	0	0	0	0	
25001-65000	0	0	0	0	0	

TOTAL DETAILS COUNTED: 93576
ENTRIES IN DETAIL SET: 93576

NO BROKEN CHAINS FOUND ALONG THIS PATH!

NO BROKEN CHAINS FOUND IN THIS DATASET!

PATH ANALYSIS

POTENTIAL PROBLEMS

- INEFFICIENT DISTRIBUTION OF ENTRIES ACROSS SET
- UNNECCESSARY I/O's DUE TO EXCESSIVE DELETED ENTRIES
- UNNECCESSARY CHAIN PATHS F
  - ALL DETAILS CHAINED TO ONE OR TWO MASTERS
- BROKEN CHAIN CONDITIONS
  - IMPROPER BIT MAP
  - DEFECTIVE KEY
  - BAD LEVEL
  - BAD CHAIN POINTER
- WRONG PRIMARY PATH ASSIGNMENT

SUMMARY

- "IMAGE" DATABASES SINCE 1976 HAVE GROWN EXPONENTIALLY
- FEASIBILITY OF A UNLOAD/RELOAD OR SOLVE QUESTIONABLE PROBLEMS IS IMPRACTICAL
- "IMAGE" USER NEED VISIBLITY INTO THEIR DATABASES
- FREQUENT DIAGNOSTICS ARE NO LONGER A NICETY THEY'RE ESSENTIAL
- UTILITY DIAGNOSTIC AVAILABLE

  DBCHECK
  DBSORT
  HOWMESSY
  DBLOADING
  DBTEST
  DB-GENERAL

- THERE IS A TREMENDOUS NEED FOR DYNAMIC RECOVERY TOOLS
- TURBO ENCOURAGES LARGER PROBLEMS!

```
 DB.GENERAL  (3.A.00)                                  Copyright BRADMARK 1982
Licensed to:          ACME MANUFACTURING INC.

                    MASTER MENU OF COMPREHENSIVE FEATURES

 1.0 INTERNAL STRUCTURES  2.0 DIAGNOSTICS W/RECVRY   3.0 DATASET MAINTENANCE

[1.1] USER CLASSES        [2.1] MST SET ANALYSIS     [3.1] CAP TABLE ENTRY
[1.2] ITEM LISTS          [2.2] SYNONYM CHAIN TEST   [3.2] CAP MANAGEMENT
[1.3] SET LIST            [2.3] DTL SET ANALYSIS     [3.3] MST CAP CHANGE
      [W/RELATIONS]       [2.4] PATH CHAIN ANALYSIS  [3.4] MST CAP SAMPLER
[1.4] SET-ITEM LISTS            [OF MST-DTL LINKAGE] [3.5] DTL CAP CHANGE
[1.5] SCHEMA GENERATOR    [2.5] CHAIN RESTORATION    [3.6] DTL SET REORGANIZE

 4.0 GENERAL UTILITIES    5.0 ROOT FILE MAINTENANCE  6.0 COPYING FEATURES

[4.1] RESTORE             [5.1] PASSWORD CHANGES     [6.1] BASE GENERATOR
[4.2] RENAME              [5.2] ITEM CHANGES         [6.2] UNLOAD/RELOAD
[4.3] PURGE               [5.3] DATASET CHANGES      [6.3] TO/FROM "MPE" FILE
[4.4] ERASE               [5.4] SET-ITEM CHANGES     [6.4] BY SELECTED KEYS
[4.5] DEVICE TRANSFER     [5.5] DATAPATH CHANGES     [6.5] GENERAL FEATURES
[4.6] FAST COPY           [5.6] ACTIVATE CHANGES

        Option selected (option number, HELP or END):
```

COMPUTER ASSISTED QUALITY ASSURANCE FOR SOFTWARE DEVELOPMENT

JONATHAN ROSENBERG
OPERATIONS CONTROL SYSTEMS
560 San Antonio Road
Palo Alto, CA 94306

INTRODUCTION

At OCS we have been exposed to a wide variety of software development environments and have found that the most successful ones have established a systematic, well-controlled approach to Quality Assurance.

Although it is best to determine procedures and controls before starting any software development project, most mature DP organizations have a number of projects already in progress before they realize the impact that formal standards and procedures could have on Quality Assurance.

As the workload in our shop grew, a number of problems generated the need for an automated system. Few formal standards and procedures existed for the development and maintenance of our various systems. Monitoring the movement of files into and out of the testing and production environments was performed informally with no ability to generate audit trails. During testing, the QA function could not identify specifically those changes which had been made to each file since the last testing. Recovery of prior versions of a file was difficult and in some cases impossible. We also needed to prevent unauthorized or unanticipated modifications and deletions. Finally, from a management perspective, we needed to track the progress of problem resolution and determine when and why changes were made.

This paper describes the approach we adopted to solve these problems.

WHY A COMPUTERIZED SOFTWARE TEST ENVIRONMENT

Although all of the concepts discussed in this paper can be addressed by a comprehensive manual system, it has been our experience that any system that is not easier to use than to circumvent will fail. Because of the number of errors associated with a manual system, a computerized system becomes essential. In addition, for any reasonably-sized development environment, the cost of administering a manual system far exceeds the cost of an automated one.

TAKING STOCK

The first step in implementing the system is to choose a high-payoff project to bring under computerized QA control.

Next, establish a fixed target or Baseline which freezes the system. To continue testing without establishing a Baseline would create a number of problems:

It would allow the system to change while still under test.

It would allow the system to become fragmented, with each element tested at different times in different environments.

It would allow changes in the background environment to invalidate the segregated test results.

To avoid these problems, we must establish a Baseline governed by the following rules:

The Baseline must be tested.

The Baseline must contain all of the final production software, including all programs, jobstreams, UDCs, data bases, files and documentation.

The tests must be reproducible.

The testing must be automated.

If necessary, go back to the latest existing complete version of the system and then work forward, or assemble all of the source files, recompile them and begin testing from that point forward.

Assign an identifier to each deliverable. In our shop we use the file name due to its easy link with our automated system. Since we keep most of our user documentation on PCs, we use Filename, Group, Account and System as the complete identifier.

It is crucial that elements be tested in a known, static environment which duplicates as closely as possible the production environment in which the system will eventually run.

THE SOFTWARE LIBRARY

The Software Library is an effective medium for managing and controlling the elements of a software project. The Library should consist of a centralized data base containing the names of all the various versions of software and data files. It should contain at least the following three logical areas:

The Master Library - where protected files exist. It may be changed only after formal approval, such as by structured walkthroughs, a review board or QA approval.

The Working Area - where development files reside. It is controlled by technical personnel on the project.

The Quality Assurance Area - where complete Baselines are staged for testing. Prior to releasing a Baseline into production, all source files must be re-compiled by the Quality Assurance function, the executable files tested as a complete system and all files checked back into the Master Library as a complete unit.

The file structure of the Library should reflect the hierarchy of the completed product.

TAKING CONTROL

To ensure that the system under test be protected from ad hoc changes, the Quality Assurance area must be securely protected from change and controlled by a test group which acts as a filter.

At this point, the shop needs to categorize system users into distinct groups such as programmers, librarians, operators and quality assurance personnel. Once these groups have been identified, the most common types of permissible file movements for each group should be defined.

SERVICE REQUEST AND CHANGE TRACKING

Having established an appropriate system for controlling your Software Library, setting up an effective Service Request and Change Tracking System is essential for controlling the change process:

In our shop we use the following system:

Our Service Request and Change Tracking System provides the information necessary for informed decision-making and allows for rapid feedback to maximize responsiveness and efficiency. The need for an automated system is clear when one considers the number of groups impacted by customer service requests, such as Customer Service, R&D, Quality Assurance, Sales and Senior Management.

A typical Customer Service request cycle originates with a central data base administrator who collects all customer request forms, verifies them for completeness and enters them into the data base. A report is generated periodically which summarizes new requests. This report is reviewed by R&D and QA managers who attach time estimates to each Request.

With these time estimates as well as the information provided on each customer request form, management can allocate resources with full knowledge of the magnitude of each problem, the impact on each customer, the projected sales impact, and the resources required to complete the request. This establishes work priorities for the QA and R&D departments. With accurate work schedules defined, customer service representatives can

supply reliable and prompt responses to their customers.

At this point, an overall priority has been determined, and all customer service requests are updated in the central data base and assigned their appropriate status using the following classifications:

1) PENDING: The initial status of all requests before prioritization.

2) OPEN: The request has been accepted for action and has a time estimate, an assigned programmer and a release number.

3) DESIGN: The request exceeds original design specifications and will be retained on file for input to new product design.

4) REVIEW: Additional time is required to investigate the request.

5) QA TEST: The request is being tested by the QA department.

6) FIXED: The request has been accepted, fixed, tested and is ready for release.

CONCLUSION

Initially a shop should establish a fixed Baseline to freeze the status of a development project. Next it should define all of the files contained in the Software Library as well as the rules governing their movement and change. In this way it can ensure that the Quality Assurance function can work within a stable, controlled environment. Once this set of standards and procedures is defined, the shop must integrate its approach with a Service Request and Change Tracking System. The result is a predictable, efficient, well-controlled software development life cycle that delivers software of a known quality level.

Linking To HP System Dictionary

By Ron Harnar
Hewlett-Packard
Information Networks Division
Cupertino, CA 95014

System Dictionary is a new data dictionary product which, for the first time on the HP3000, allows programmatic access to a fully extensible dictionary architecture. Prior to its release on MPE V G.01.04 (T-delta-4), one of System Dictionary's largest "customers" was Hewlett-Packard itself. HP used the System Dictionary intrinsics to develop the definition loaders SDDBD, SDVPD, and SDCONV, a dictionary maintenance and reporting utility SDMAIN, and the definition extractors SDDBC and SDCDE.

These utilities by no means exhaust the possible applications of the System Dictionary intrinsics. But they do embody some dictionary access techniques that will almost certainly be used in every System Dictionary application.

The purpose of this paper is to describe these techniques so that application developers can spend less time reinventing them and more time developing System Dictionary-linked solutions for their target users. This paper starts with a brief overview of the System Dictionary architecture and intrinsics, but a general familiarity with System Dictionary and its utilities is assumed. For details on syntax and usage, refer to the *HP System Dictionary Intrinsics Reference Manual (32254-90002).*

Entity-Relationship Model

System Dictionary uses a simple but powerful entity-relationship model in which the central components are entities, relationships between entities, and attributes that define and characterize entities and relationships. You can use the entity-relationship model to describe the real-world objects of an information network, and to define logical connections between these objects.

ENTITIES AND RELATIONSHIPS

Entities and relationships are the definitions that you store and retrieve in the dictionary. *Entities* are dictionary definitions that refer to objects in the real world of an information network. An entity has attribute values that further define the real-world object or that describe the entity itself (when it was created in the dictionary, who "owns" it, and so on). A *relationship* is an ordered list of entities. Relationships express logical connections between real-world objects. Most relationships involve two entities and are called *binary* relationships. System Dictionary also supports *N-ary* relationships involving up to six entities. Relationships have attribute values that in some cases describe the dictionary definition itself and in other cases describe one of the *entities* in the relationship. Relationship attribute values can thus be used to document an entity in a particular context or usage.

STRUCTURES

The System Dictionary entity-relationship model provides a set of structures that support the creation and retrieval of entities and relationships. *Entity types* are structures used to define, categorize, and qualify entities. IMAGE-DATABASE, for example, is a System Dictionary entity type used to define entities that describe IMAGE data bases. *Relationship types* are structures for defining, categorizing, and qualifying relationships. IMAGE-DATABASE contains IMAGE-DATASET is a System Dictionary relationship type used to define relationships between IMAGE data bases and data sets.

Attributes are structures for defining the attribute values of entities and relationships. IMAGE-DATASET-TYPE, for example, is a System Dictionary attribute associated with the IMAGE-DATASET entity type. CAPACITY is an attribute associated with the IMAGE-DATABASE contains IMAGE-DATASET relationship type. System Dictionary supports six attribute types. Four of the types are fixed-length: *boolean*, a true-or-false value stored in one byte; *character*, an alphanumeric value with a maximum length of 255 bytes; *integer*, a 16-bit or 32-bit binary value; and *floating*, a 32-bit or 64-bit floating-point format. The remaining two attribute types--*alias* and *variable*--are "free-floating" in that they are never assigned to entity types or relationship types, but values for them can be assigned to any entity or relationship. An alias attribute is a 32-byte character field that contains an alias value for an entity or relationship. An attribute of type variable (sometimes called a variable-length attribute) has no maximum length, and is typically used to store descriptions, edit masks, default values, long names and other variable-length values. Since not every entity or relationship needs alias and variable-length attributes, System Dictionary sets aside storage space for them only if they are explicitly assigned. When an alias or variable-length attribute value is deleted, the storage space is released.

A *relationship class* is a name, like contains or uses, that describes the action or connection of a relationship. In System Dictionary, a relationship type belongs to a relationship class. The relationship types RECORD contains ELEMENT and FORM contains ELEMENT, for example, belong to the contains relationship class. In some cases, a relationship class serves as a qualifier for a relationship type. For example, System Dictionary has three relationship types that involve element pairs:

```
ELEMENT contains ELEMENT
ELEMENT redefines ELEMENT
ELEMENT references ELEMENT
```

The first type is used to create relationships between parent and child entities. The second type indicates that two elements share common storage in a program. The third type documents an element that references another element, as in a Pascal type reference. These ELEMENT ELEMENT relationship types are *qualified* by relationship class.

In System Dictionary, each entity type and relationship type has an attribute list. Attributes are associated with an entity type or relationship type by means of a structural component called a *type-attribute association*. When an attribute is associated with an entity type or relationship type, each entity or relationship of that type must have a value for the attribute.

Extensibility

System Dictionary provides a built-in set of structures called the *core set*. The core set includes entity types, relationship types, relationship classes, attributes, and type-attribute associations. Users can extend their dictionaries by creating new structures or by making limited changes to the core set such as changing the names of structures. The extensibility feature allows the dictionary to be customized by the user. It also allows Hewlett-Packard to localize System Dictionary to a user's native language, and to update the core set as new subsystems are added to MPE.

Domains and Versions

In System Dictionary, a single physical dictionary can be divided into multiple, logically separate domains. A domain is a "name space" that contains entities and relationships. Domains are typically used to keep the entity and relationship definitions of one application group separate from others. They can also be used to avoid naming conflicts. Every System Dictionary has a built-in domain called the *common domain* that is always present and always public. Users can create their own domains, called *local* domains, and assign them a sensitivity level of public or private.

Domains can be further subdivided into named partitions called versions. Versions allow you to designate one set of entities and relationships in a domain as the current *production* version, and other sets as *test* or *archival* versions. Unlike domains, which are considered separate name spaces, versions should be thought of as more-or-less complete *copies* of the entities and relationships in a domain. In a version of test status, you can add or delete entities and relationships, or change their attribute values. An archival version cannot be modified, and is usually kept for historical purposes. A production version also cannot be modified, and only one production version is allowed in a given domain. The version feature thus allows you to maintain a stable production version while you experiment with other versions. It also allows you to re-activate an archival version if it becomes necessary to return to a previously used set of definitions.

An important function of a data dictionary is to ensure the standardization and integrity of the definitions used in an information system. The domain and version features allow you to experiment with new definitions and maintain separate sets of definitions. If used carelessly, however, the System Dictionary domain and version features can defeat the standardization value of the dictionary. System Dictionary has security features that prevent unauthorized users from creating domains and versions, and that prevent one user from creating new versions of an entity or relationship owned by another user.

To help overcome the potential integrity problem of having duplicate definitions in several domains, System Dictionary allows an entity or relationship definition in the common domain to be *linked* with one or more definitions in one or more local domains. When definitions are linked they automatically share attribute values except for special attributes such as SCOPE-OWNER and DATE-CREATED. The sharing of attribute values has two advantages. It allows multiple entities or relationships to share the same attribute value storage space, and it ensures that the shared values can only be modified in the common domain, not in the local domains. This gives the owner of the common domain entity or relationship some control over the integrity of the shared attribute values.

Dictionary Security

System Dictionary provides a comprehensive security scheme consisting of scopes, sensitivity levels, and scope associations.

A scope is a dictionary user name with a password and a set of capabilities called scope rights. The System Dictionary scope rights determine whether a user can create or merely read entities and relationships (the *create* and *read* scope rights), extend the dictionary structure (the *extend* scope right), create scopes and obtain information about dictionary security (the *secure* scope right), and create domains and versions (the *domain* and *version* scope rights).

Scopes "own" entities, relationships, structures, domains, versions, and even other scopes. Some intrinsics require you to be the owner of the dictionary object on which the requested operation (such as deletion) is to be performed. Every dictionary has a built-in scope named CORESET that owns the entity types, relationship types, and other structures of the core set. When you create a dictionary, you specify a name and password for the Dictionary Administrator (DA) scope. The DA scope automatically has all scope rights plus other capabilities not available to other scopes.

Each entity and relationship has a *sensitivity level* that determines whether other scopes can read and/or modify it. Domains also have sensitivity levels that designate them as public or private. If an entity or relationship has a sensitivity level of *read* or *private*, its scope-owner can grant a higher level of access to a selected scope by creating a *scope association.* The scope-owner of a private domain can similarly create a scope assocation that gives a particular scope access to the domain.

Intrinsic Groups

System Dictionary has a large number of intrinsics (92). The number is large because the features built into the dictionary architecture--domains, versions, scopes, and an extensible structure--give it tremendous complexity, and because the intrinsics provide comprehensive access to the dictionary. However, since not every System Dictionary application needs every intrinsic, it may help to divide the intrinsics into the task-oriented groups mentioned below.

The *control* intrinsics are used to perform operations such as opening and closing the dictionary and checking for errors. The *entity* and *relationship* intrinsics are used to create, delete, modify, or retrieve entities and relationships. For most applications that access System Dictionary, these will be the most commonly used intrinsics.

The *structure* intrinsics have two general uses. One use is to retrieve information about entity types, relationship types, attributes, and other components of the dictionary structure. This information is typically used to support the creation or retrieval of entities and relationships. A more specialized use (requiring a scope with the extend scope right) is to change the dictionary structure by adding, deleting, or modifying entity types, relationship types, relationship classes, attributes, or type-attribute associations.

The domain and version features of System Dictionary are supported by the *domain* and *version* intrinsics. These intrinsics are used to create, delete, modify, and retrieve

information about domains and versions. They also provide the ability to switch from version or domain to another.

The *security* intrinsics can be used to retrieve information about scopes and scope associations, or to create and maintain a security scheme. The security intrinsics generally require the user to have a special capability (the secure scope right) or owner access to the desired information. This helps prevent unauthorized users from cracking the dictionary security. The scope intrinsics also allow you to switch from one scope to another while the dictionary is open.

System Dictionary Applications

Although there are many applications for the System Dictionary intrinsics, they generally fall into three categories:

- *Loading* definitions into the dictionary.
- *Extracting* information from the dictionary for use in a subsystem or application.
- *Maintaining* dictionary definitions, structures, domains, versions, and security.

These application types are discussed below.

Loader Applications

A definition *loader* is a program or routine that uses the dictionary intrinsics to ensure that an entity or relationship is correctly defined in the dictionary. In some cases, this is done by *creating* a new definition with SDCreateEnt or SDCreateRel. In other cases, the loader may *use* an existing compatible definition, or it may *modify* an existing definition if its attribute values are incompatible and the dictionary security allows the modification.

HANDLING LOADING CONFLICTS

The simplest case of loading a dictionary definition is to create a new entity or relationship where none exists. Often, however, a utility or application program will encounter conflicts because the desired definition already exists, but its attribute values are incompatible with the definition to be loaded. For example, if a program wants to create an entity named ZIP-CODE of entity type ELEMENT with a BYTE-LENGTH attribute value of 9, and the ZIP-CODE entity already exists in the current version with a BYTE-LENGTH value of 5, the program must decide whether to keep the existing definition or replace it with the new one. (Of course, the decision can be postponed by creating the new definition in another version, but someone must eventually decide whether ZIP-CODE has 9 digits or 5.)

One loading problem occurs when an entity or relationship must be modified but the current scope lacks modify access to the definition. The SDModifyEnt intrinsic requires a scope with modify access to the target entity (or SDERR 412 is returned). A scope that lacks modify access to a relationship cannot use SDModifyRel against that relationship (SDERR 612). If the loader "modifies" an entity or relationship by deleting and re-creating it, the security

restriction is even greater: to delete an entity or relationship, you must be its scope-owner (see SDERR 406 and 617). Also, when you delete an entity or relationship, you automatically delete its attribute values and scope associations. When you delete an entity, any relationships involving the entity are automatically deleted also. SDModifyEnt and SDModifyRel leave this network of information intact. For these reasons, you should use "modify" intrinsics instead of the delete-create approach whenever possible.

Another loading problem can arise when the current scope is not the scope-owner of an entity or relationship to be *created*. You may wonder how you can be the owner of something that does not exist yet. The answer is that it may exist in other versions of the current domain. System Dictionary requires that all versions of an entity or relationship in a given domain have the same SCOPE-OWNER attribute value. You therefore cannot create an entity or relationship in one version if it already exists in another version of the same domain and has a different scope-owner. A loader utility should check for this error (SDERR 406 or 617) when creating an entity or relationship.

HANDLING LINKED DEFINITIONS

When you create or modify an entity in a local domain, you can optionally link it to an entity in the common domain. This is done by passing the common domain entity's name in the *CommonEntity* parameter of SDCreateEnt or SDModifyEnt. From then on, until the local entity is deleted or the link is broken by passing a slash (/) in the *CommonEntity* parameter, the linked entities share all attribute values (except for special attributes such as SCOPE-OWNER and DATE-CREATED), and the values can only be changed by modifying the entity in the common domain. Relationships can also be linked or unlinked by using the *CommonEntityList* parameter in SDCreateRel or SDModifyRel. Because the attribute values of linked definitions can only be modified in the common domain, a loader utility accessing a local domain needs to be aware of this potential conflict (SDERR 421 and 638).

How can you tell if two entities are linked? The answer depends on whether you are in the common domain or a local domain. In a local domain, you can determine the link status by calling SDGetEnt and checking the *CommonEntity* parameter. After a call to SDGetEnt, the *CommonEntity* parameter contains either the name of the linked entity or, if the entity is not linked, blanks. In the common domain, checking for a link is more complicated because a single entity may be linked to one or more local domain entities, and the linked entities may be in any version of any local domain. A common domain entity may therefore have a *list* of linked entities. To retrieve this list, you call the SDGetLocalEntList intrinsic. Similar techniques can be used to determine the link status of a relationship by calling SDGetRel or SDGetLocalRelList.

If you are loading definitions into a local domain, and you want to modify an existing definition that is linked to the common domain, you can use either of two methods. The first method is to remove the link by passing a slash in the *CommonEntity* parameter of SDModifyEnt or the *CommonEntityList* parameter of SDModifyRel. The second method is to keep the link intact, switch to the common domain, and modify the linked definition. To use this approach, you would first call SDGetEnt or SDGetRel and save the *CommonEntity* or *CommonEntityList* parameter so that you know which definition to modify in the common domain. You would then call SDGetVersion to find the name of the common domain version that contains the linked definition. Next, you would switch to this version in the common domain by calling SDSwitchDomain. After switching to the common domain, you would use SDModifyEnt or SDModifyRel to change the desired attributes of the

linked entity or relationship. These changes would automatically be reflected in the linked definition in the local domain. Finally, you would switch back to the local domain and continue the loading process.

Note that there is an important philosophical difference between the two methods of updating linked definitions. When two or more definitions are linked across domains, they are *equivalent* definitions that share a common set of attribute values. When the common domain definition is modified, the local definitions are automatically changed as well. The common domain definition should therefore be modified only if the change is intended to be global for all the definitions linked to it. If the change is intended only for one local domain definition, the common domain link should be broken.

To handle potential conflicts gracefully, then, a loading routine should:

- Check for conflicts after any SDCreateEnt or SDCreateRel call. It may also help to call SDGetEnt or SDGetRel first, to see if the desired entity or relationship already exists before trying to create it.
- Use SDModifyEnt or SDModifyRel to modify an existing definition rather than deleting and re-creating it.
- Be prepared to handle scope conflicts that arise when the current scope lacks modify access or is not the definition's scope-owner.
- Decide whether to handle common domain links by breaking the link or using domain switches.

The loader routine should also have a general error procedure for handling unexpected errors from IMAGE (System Dictionary is built on an IMAGE data base) or the file system.

Extractor Applications

Now that you have loaded your entity and relationship definitions into the dictionary, along comes an application that looks up the definitions and uses them in a subsystem or application. This type of program is sometimes called an *extractor*, because it extracts information from entity and relationship definitions in the dictionary. Some typical extractor operations include:

- *Generating* data bases, source code, or other subsystem objects according to definitions in the dictionary.
- *Resolving* data definitions at compile or run time by looking them up in the dictionary.
- *Navigating* through a computer system or network guided by dictionary definitions.
- *Defaulting* according to dictionary values rather than hard-coded values.

Dictionary access for an extractor utility is simpler than for a loader. The extractor does not need to be concerned about the scope, domain, and version conflicts that can arise when creating new definitions or modifying existing ones. Read access to already existing definitions is sufficient.

GENERATING SUBSYSTEM OBJECTS

The main concern of an extractor utility is to provide a correct and consistent translation of System Dictionary definitions into subsystem definitions, objects, or procedures. A COBOL copy library utility, for example, must understand the System Dictionary entities, relationships, and attributes used to define data elements, records, and files. It must also recognize the data definition syntax, reserved words, and restrictions of the COBOL language.

The designer of a System Dictionary-linked extractor utility should examine the core set to see if it supports the definition of objects in the target subsystem. If additional dictionary structures are needed, the utility designer must consider how the structure changes will be distributed to other dictionaries. If Hewlett-Packard agrees to implement a proposed structure change as an enhancement to the core set, it will be distributed to all System Dictionary users worldwide. Otherwise, it becomes the application developer's responsibility to distribute the change.

The designer of an extractor utility is often interested in a specific set of entities that can be identified by the relationships that link them together. An IMAGE data base creation utility, for example, may wish to find all the IMAGE data sets defined in the dictionary that belong to a data base named ORDERS. In System Dictionary terms, this task translates into finding all the relationships of the type `IMAGE-DATABASE contains IMAGE-DATASET` in which `ORDERS` is the `IMAGE-DATABASE` entity. This type of retrieval is done with the SDFindRelList intrinsic. SDFindRelList has an *EntitySearchList* parameter in which you specify a list of entity names to be matched during the search. Unlike SDGetRelList, which returns all the relationships of a specified type, SDFindRelList returns only those relationships that meet the entity list criteria.

RESOLVING DATA DEFINITIONS

The term *fourth generation language* (4GL) is used to describe a programming language that looks up data definitions in the dictionary while the program is being compiled or run. A 4GL saves the programmer the effort of having to write a detailed working-storage section. The programmer supplies the names of the variables to be used, and the 4GL looks up their data types, lengths, and other information in the dictionary. Also, when a data definition must be changed, the change can be made once in the dictionary and it will automatically be reflected in all the programs that use the definition. In the case of a compiled 4GL, the change is transmitted when the programs are re-compiled. In the case of a run-time 4GL, the changes take effect immediately.

NETWORK NAVIGATION

A data dictionary can be a central repository of information about a network. Networking applications can rely on the data dictionary to provide current information about network devices, connections, and routing. System Dictionary is designed to provide the functionality of a network dictionary. It can play a central role in the design of networking applications.

DEFAULT VALUES FROM THE DICTIONARY

A data dictionary can be thought of as a storage facility that allows default values to exist independently of the applications that use them. For example, suppose a company is developing a series of programs to manage inventory information in an IMAGE data base. The data base has an item named AMT-ON-ORDER that specifies the number of units to be ordered for the current month. In each program, the default value for this item is set at 10.

After several weeks of using the new programs, the users decide that they want the default value of AMT-ON-ORDER changed to 5. To implement this change, the programmers must find the programs that assign default values to AMT-ON-ORDER, make the requested change, and recompile the programs.

If, on the other hand, the programs had been designed to check the dictionary for the default value of this item, the change could be made in all the programs by changing a single dictionary definition. Furthermore, the change could be made on-line without having to recompile the programs.

One way to store a default value in System Dictionary is to use the core set attribute DEFAULT. DEFAULT is a variable-length attribute, which means that it can contain a default value of any length or format, and it can be assigned to any entity or relationship in the dictionary.

In the AMT-ON-ORDER example, you might create an entity named AMT-ON-ORDER of type ELEMENT, and assign it a DEFAULT attribute value of 10 (and later change this value to 5). It is then up to the programmers to use SDGetEnt to retrieve the AMT-ON-ORDER entity and check its DEFAULT attribute value.

Maintenance Applications

Maintenance operations involve the ad hoc creation, deletion, modification, or retrieval of definitions. In a broader sense, maintaining the dictionary also includes creating and maintaining a security scheme, customizing the dictionary structure, and handling domains and versions. It might also involve *merging* the definitions and structures of one dictionary into another.

Unlike a loader utility, which can restrict its operations to a narrowly defined set of entity types, relationship types, and attributes, a maintenance utility usually takes a generic, extensible view of the dictionary. The designer of a dictionary maintenance utility should therefore use the various "list" intrinsics rather than hard-coded lists of definitions or structures. Attributes and their values should also be handled with an extensible routine.

USING STRUCTURE LISTS

The "list" intrinsics SDGetEntTypeList, SDGetRelTypeList, SDGetRelClassList, and SDGetAttrList allow the programmer to retrieve lists of dictionary structures. The lists can be presented to the end user for selection of a structure. Also, since System Dictionary allows the dictionary structure to be extended, programmers can use the list intrinsics to

keep up with these changes. The maintenance and reporting utility SDMAIN makes extensive use of the list intrinsics.

One example of list intrinsic usage is the task of finding all the relationship types in the dictionary. Since every relationship type belongs to a relationship class, the first step is to call SDGetRelClassList to get the list of relationship classes available in the dictionary. Then, for each relationship class, SDGetRelTypeList would be called to retrieve all the relationship types of that class. SDMAIN uses this two-stage retrieval method when you give the command:

```
>DISPLAY RELATIONSHIP-TYPE.
```

Because you omitted the RELATIONSHIP-CLASS parameter in this command, SDMAIN must call SDGetRelClassList to get the list of classes before it can get any relationship types. If you specify a relationship class, as shown here,

```
>DISPLAY RELATIONSHIP-TYPE; RELATIONSHIP-CLASS=CONTAINS.
```

SDMAIN would skip the SDGetRelClassList intrinsic and go directly to SDGetRelTypeList for the list of relationship types belonging to the contains relationship class.

Another example of the list intrinsics is found in the attribute list prompting feature of SDMAIN. When you create, modify, or retrieve an entity in SDMAIN, you are given a list of attributes to choose from. The attribute list is assembled by calling SDGetEntTypeAttrList to get the list of attributes associated with the specified entity type, and SDGetAttrList to get the list of alias and variable-length attributes available in the dictionary. The complete attribute list presented to the user thus includes not only the entity type attributes but also the "free-floating" alias and variable-length attributes that can be assigned to any entity.

EXTENSIBLE ATTRIBUTE ROUTINES

A loader or extractor utility is usually interested in a fixed list of attributes, so it is a simple matter to devise a data structure for their attribute values. For maintenance utilities in which the user is allowed to *specify* an attribute list, however, a more generic and extensible method of handling attribute values is needed. The following discussion focuses on the retrieval of *entity* attribute values, but the same techniques can be used for relationships. Consider this example of entity retrieval in SDMAIN:

```
>REPORT ENTITY LAST-NAME; ENTITY-TYPE=ELEMENT;
>>LIST=SCOPE-OWNER, ELEMENT-TYPE, BYTE-LENGTH.
```

With this command, the user is requesting the SCOPE-OWNER, ELEMENT-TYPE, and BYTE-LENGTH attribute values of the LAST-NAME entity. One way to handle this request would be to retrieve each attribute value separately. First you would parse the list parameter to get the attribute names. Then, for each attribute, you would call SDGetAttr to determine its data type and length. Finally, you would call SDGetEnt supplying the attribute name in the *AttributeList* parameter and a variable of the appropriate type and length in the *AttributeValues* parameter. From a performance perspective the use of multiple SDGetEnt calls is expensive, but it makes for a simple looping algorithm that can

handle any number of attributes with a fairly small *AttributeValues* parameter (the maximum length of any single fixed-length attribute value is 255 bytes).

Another method is to retrieve all the requested attribute values in a single SDGetEnt call. In this example, you would simply build an *AttributeList* parameter like the user's list parameter. The difficulty of this method is in building the *AttributeValues* parameter. Since the user can specify any number of attributes, and the attributes can have a variety of data types and lengths, you would need to build a table of attribute information containing each attribute's data type, length, and position in the *AttributeValues* array. Retrieving a list of attribute values in a single SDGetEnt call requires more programming effort, but it is more efficient than the looping method described earlier.

You can assign arbitrary maximum lengths to the *AttributeValues* array and to your attribute information table, but a truly extensible entity retrieval routine should rely on System Dictionary maximums. The *AttributeValues* array should have enough room for all special attributes, all alias attributes (since these can be assigned to any entity), and all other attributes that could be assigned to the entity type via a type-attribute association. The special attributes require 102 bytes. The maximum number of alias attributes allowed in the dictionary (128) times their length (32 bytes) would add 4096 bytes to the array. The maximum length of other fixed-length, non-alias attribute values is 2048 bytes. The maximum length of the *AttributeValues* array should therefore be:

102 + (128*32) + 2048 = 6246 bytes

The length of the *AttributeList* parameter should also reflect the dictionary maximums. There are six special attributes assigned to each entity type. Each entity type can have up to 128 additional type-attribute associations. As already mentioned, there can be up to 128 alias attributes in the dictionary, and any number of these can appear in the attribute list. The *AttributeList* array should therefore have room for:

6 + 128 + 128 = 262 attributes

The *AttributeList* parameter can be defined as an array of 32-byte attribute names. It can also be defined as an array of 32-bit integers representing the *internal numbers* of the attributes. Internal numbers are retrieved from the attribute intrinsics (e.g., SGetAttr and SDGetAttrList). When you pass an array of internal numbers in the *AttributeList* parameter, the first 32-bit integer is used as a count item that indicates the number of attributes in the list. This format would therefore require a maximum of 263 array elements. Besides saving space in the *AttributeList* parameter, the use of internal numbers also improves performance by saving System Dictionary the overhead of attribute name lookups.

System Dictionary provides a flexible entity-relationship model and a set of environmental features--domains, versions, and scopes--that allow the dictionary to be tailored to application requirements. The System Dictionary intrinsics provide comprehensive access to entities, relationships, domains, versions, scopes, and the dictionary structure. You can use the intrinsics to design a wide a variety of applications that load, extract, or maintain dictionary definitions. This paper discussed some guidelines and techniques for implementing a System Dictionary-linked application.

Prototyping And Systems Development Using 4GL

Raymond Ouellette
Infocentre Ltd.
6303 Airport Road
Suite 300
Mississauga, Ontario
L4V 1R8

As Fourth Generation Software becomes more prevalent throughout all facets of data processing, it seems that the 3GL/4GL debate has subsided. Organizations have recognized that Fourth Generation Software offers a viable solution to many of the challenges facing their data processing departments.

Many organizations using a 4GL have found that the actual gains don't quite measure up to the anticipated achievements and benefits. The inadequacy lies not with the 4GL itself but within the system development methodology in which the 4GL is used. Any data processing organization committed to using a 4GL must be prepared to examine its current application development methodology.

Traditional methodology requires the complete understanding of the user requirements before design and implementation begins. It assumes that users know what they want and can communicate it and their requirements remain static. There are a number of factors that contribute to errors in specifying these user requirements and later, in designing and implementing the application.

As a means of streamlining the development process, some data processing organizations have introduced prototyping. The purpose of prototyping is to test assumptions about user requirements. Using 4th Generation Software, a prototype is created quickly and inexpensively and requires the active participation of the end-user early in the development process.

This paper will define prototyping and elaborate on the prototyping model. It will address some uses of the prototype, the resources and skills needed for prototyping and also look at some of the associated problems.

RINs, RINs, RINs

Benedict G. Bruno
S.T.R. Software Company
P. O. Box 12506
Arlington, VA 22209

Introduction

Several articles have been written discussing the merits of the HP 3000 Computer System as a transaction processing oriented machine. Topics have included process handling, interactive on-line systems, etc. An excellent resource available to the standard MPE user/programmer for process handling control is the Resource Identification Number, better known as a RIN.

RINs come in two flavors. Local RINs are available only to the creating session or job. Global RINs are permanently saved on the system disc and are available to anyone on the system.

The concept of RINs has not been adequately described. This paper will discuss the benefits of local versus global RINs in a multi-processing environment, i.e., how may RINs be utilized for the control and monitoring of user applications executing on the HP 3000. RINs are extremely effective in the process handling environment. Several programming examples will be provided.

Although MPE provides several system intrinsics and commands for creation and access of local and global RINs, no software tool is provided for display of this information. Thus it is difficult, almost impossible, to display the RIN number, password, creating user/account, information of locking process, and information of waiting process. This information is maintained by MPE in the System RIN table.

The contents of the System RIN table have been formatted for the user in the supplied SHOWRIN program. The author has developed this program for both MPE IV and MPE V based systems. Depending upon capability, portions of the RIN table are formatted in displays similar to the HP On-line Performance Tool (OPT) program. Using this SHOWRIN program and the discussion of RIN usage, you can start integrating local and global RINs into your HP 3000 applications right away!

Local RINs

MPE provides local and global RINs for process control. In order to begin our discussion, let us define local and global RINs.

Local RINs can be described having the following attributes:

* Created, accessed, and released on a session or job basis only. (intra-job)

* Programmatic MPE intrinsic access only.

* User maintains control of RINs by specifying which task each local RIN number is associated.

* The MPE LOCRINOWNER intrinsic returns the process identification number (PIN) of the process currently locking the local rin.

A local RIN is a resource allocated on a job or session basis for use by any process executed during this job or session. Local RINs are created and released only using the GETLOCRIN and FREELOCRIN intrinsics programmatically. The GETLOCRIN intrinsic allocates the requested number of local RINs for the current job or session. The FREELOCRIN intrinsic releases all local RINs previously obtained with the GETLOCRIN intrinsic.

Local RINs are accessed using the LOCKLOCRIN and UNLOCKLOCRIN intrinsics. Local RINs are referenced with the integer value assigned within range of the number allocated with the GETLOCRIN intrinsic. Thus, if the user requests 24 local RINs with the GETLOCRIN intrinsic, then the LOCKLOCRIN intrinsic may lock RIN numbers one through 24. The UNLOCKLOCRIN intrinsic performs the same way.

The MPE LOCRINOWNER intrinsic provides the PIN of the locking process for the supplied local RIN. This capability prevents deadlocks and informs any process in a process tree of the currently locking process. Thus, there is no need for a display of the local RIN entries from the system RIN table.

Global RINs

Global RINs can be described having the following attributes:

* Created and released on a system wide (global) basis for any session or job on the system. (inter-job)

* User assigned RIN password is associated with a unique entry in the system RIN table using the GETRIN MPE command. eleased only by creating owner with the FREERIN MPE command.

* System RIN table is system disc resident on logical device number 1. Modified only during reload.

* Any job or session may access the global RIN using the LOCKGLORIN and UNLOCKGLORIN intrinsics by supplying the RIN password and RIN number.

* The HP supplied SYSDUMP program displays the global RIN number with the creating user and account. The user specified password is NOT displayed. Hence, the need for the SHOWRIN utility program.

Global RINs are available to any session or job on the system provided the number and password are known. Global RINs are assigned to a creating user and account with the GETRIN command. The RIN associated with this password is permanently maintained in the System RIN table which is system disc resident. An entry may be removed using the FREERIN command. Note that the RIN entry may only be removed if the creating user and account match the logged on user and account names!

The size of the global RIN table residing on the system disc is configured with the SYSDUMP dialogue. Changing this size can only occur during a RELOAD. The standard MPE configuration is delivered with a table size of 48 entries. In order to utilize the global RIN feature, you should increase the size of the table appropriately.

The following commands can be executed to create and free the global RIN assigned using the password of GRIN.

```
:HELLO user.account
:GETRIN GRIN
RIN: nnn

:FREERIN nnn
```

MPE responds with the integer value assigned to the RIN password with the GETRIN command. It is important that you always remember this association. Currently, no HP supplied utility displays the RIN entry number, password, and creating user and account names.

You should note that the FREELOCRIN intrinsic removes all created local RINs for the current session or job. The FREERIN MPE command releases ONE global RIN from the system RIN table if currently signed on as the creating user and account.

In order to continue, the reader should now note that the local RINs may be very effective during process control within an individual process tree, i.e., intra-job level processes. Global RINs are very effective during process control within several processes executing from different jobs, sessions, and accounts, i.e., inter-job level.

Global RIN Example

Global RINs are valuable in the monitor and control of several programs comprising an application. By assigning a global RIN to each program in the application and requiring each program in the application to lock its associated RIN, we are guaranteed that only one copy of the program is executing. Some programmers implement this feature by opening a file for exclusive access. Since the global RIN is locked during the entire execution of this program and additional data structures may need to be locked, i.e., IMAGE/3000 data bases, KSAM and MPE files, etc., the program file must be :PREPed with MR (multiple resource or better known as multiple RIN) capability in order to hold multiple locks. The user, group, and account capabilities must also be modified for MR capability.

Once each of the programs in the application is initiated and the lock is maintained on their RINs, another monitor program may control the execution of these processes even further. Suppose that this monitor program is responsible for the initiation, control, and termination of all programs within this application. Once each process is started, the RIN is held for the duration of this process. The monitor need only attempt a conditional lock in order to determine if the process is still executing. If the conditional lock succeeds, then the process is not executing since the monitor process was able to lock the RIN; otherwise, the lock fails and the process is executing since the RIN is unavailable.

This technique is further enhanced by utilizing the writers id feature with the open, close, and data record types of the MPE message file system. The requirement is to have each program in the application open the monitor message file for write access. If the monitor program opens this message file and enables the writers id feature, then as each process opens, writes, or closes this file, the monitor program receives a record containing the writers id, the file access, and data if it is a write. The writers id is an integer assigned by the file system in ascending order to each program opening the file. If a close record is detected unexpectedly, then the monitor program need only identify which program has failed and respond in a controlled manner. The monitor may restart the failing program or may request all other programs in the application to terminate immediately.

In addition to the initiation and termination of programs within the application, the monitor program may also control the execution of each of these programs. Specifically, the monitor program may suspend or resume execution of these programs. Utilizing the message file techniques above, the monitor program issues a conditional lock on a secondary RIN for each process requested to suspend. Once locked, the monitor requests that the process suspend itself by issuing an unconditional lock on this secondary RIN. The process is resumed

execution when the monitor releases the secondary RIN. Hence, an excellent method of global process control!

The SHOWRIN program was developed in order to determine which RINs were currently locked and by which process. Another useful feature is that the password could be associated with the RIN entry itself. This feature saved documenting the RIN entries and their passwords into a journal for later reference.

Now let's begin with an example to better explain these features. An application of one monitor and two programs will be used to demonstrate the global RIN concepts discussed earlier. Sample code will be provided in both SPL and COBOLII.

In Figure 1 below, the monitor program is diagrammed to use the parm value of the :RUN command as the number of the global RIN named 'MONITOR'. The monitor program issues a conditional lock on this global RIN. If successful, the program continues; otherwise, an error message is displayed since some other process currently has this RIN locked (this prevents two copies of the monitor program from executing because another copy is already executing). The monitor program opens three message files: the primary message file named 'M' is opened for read access enabling the writers id feature; the additional message files named 'A' and 'B' are opened for write access in order to communicate with process A and process B.

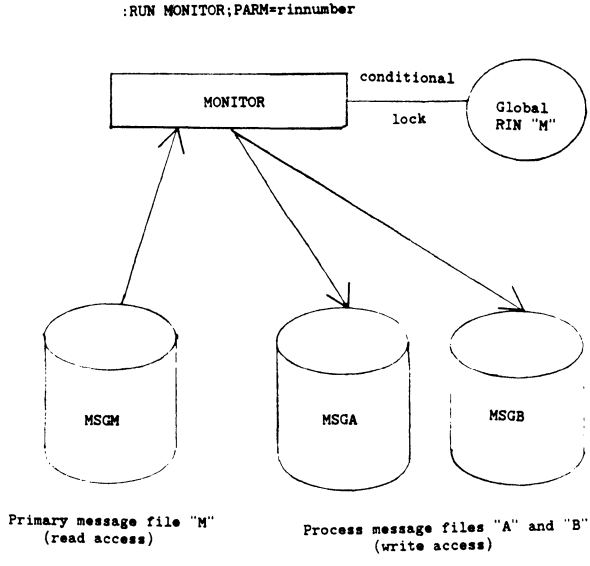


Figure 1: Monitor Program "M" Execution Flow Diagram

The example has been coded in SPL below in Figure 2. The program variable declarations are first displayed. Using the indentation and blank lines provide for easy reading.

Figure 2: Monitor program in SPL

```
1   $CONTROL USLINIT,MAP
2
3   <<*****************************************************************>>
4   <<** MONITOR Program Version Information.                          **>>
5   <<**                                                              **>>
6   <<** Version    Date    Who  Comments / Description               **>>
7   <<** -------  --------  ---  ---------------------------------- **>>
8   <<** B.01.00   5/31/86  BGB  Initial program release.            **>>
9   <<**                                                              **>>
10  <<** This program will lock the global RIN name of "MONITOR"     **>>
11  <<** with the RIN number passed in the ;PARM= parameter.         **>>
12  <<*****************************************************************>>
13  $PAGE "*****  MONITOR PROGRAM  *****"
14  BEGIN
15
16  COMMENT  EQUATES AND DEFINES
17  ****************************************************************;
18
19  DEFINE    IR'CONTROL'CODE          = LINPUT'RECORD#,
20            IR'PROGRAM'ID            = INPUT'RECORD(2)#;
21
22  EQUATE    RS                       = %036,
23            BEL                      = %007,
24            ESC                      = %033;
25  $PAGE
26  COMMENT   LOCAL VARIABLES
27  ****************************************************************;
28
29  LOGICAL ARRAY           LTERMLINE(0:79);
30  BYTE ARRAY              TERMLINE(*)=LTERMLINE;
31
32  LOGICAL ARRAY           LBA1(0:79);
33  BYTE ARRAY              BA1(*)=LBA1;
34
35  LOGICAL ARRAY           LINFO'STRING(0:127);
36  BYTE ARRAY              INFO'STRING(*)=LINFO'STRING;
37
38  LOGICAL ARRAY           LINPUT'RECORD(0:119);
39  BYTE ARRAY              INPUT'RECORD(*)=LINPUT'RECORD;
40
41  DOUBLE                  RUN'PARM;
42
43  LOGICAL                 DONE,
44                          LOCK'COND,
45                          OK,
46                          TIME'2'QUIT,
47                          TRUE'COND;
48
49  INTEGER                 CERROR,
50                          CPARM,
51                          FERROR,
52                          I,
53                          IN'LENGTH,
54                          INFO'LENGTH,
55                          LENGTH,
56                          MSGA'FILE,
57                          MSGB'FILE,
58                          MSGM'FILE,
59                          NUM'CHAR,
60                          PAUSE'RIN'A,
```

Continuing the program below, the intrinsic declarations are listed in alphabetical order. The GET'INFO procedure will locate the ;PARM= and ;INFO= parameters of the :RUN statement by locating the terminate stack marker.

Figure 2: Monitor program in SPL (Cont.)

```
                        PAUSE'RIN'B,
                        MONITOR'RIN;

INTRINSIC               ASCII,
                        BINARY,
                        COMMAND,
                        DATELINE,
                        FCHECK,
                        FCLOSE,
                        FCONTROL,
                        FERRMSG,
                        FOPEN,
                        FREAD,
                        FWRITE,
                        LOCKGLORIN,
                        PRINT,
                        PRINTFILEINFO,
                        QUIT,
                        TERMINATE,
                        UNLOCKGLORIN;
$PAGE
     PROCEDURE  GET'INFO (PARM, INFOL, INFOSTR);
        DOUBLE  PARM;
       INTEGER  INFOL;
 LOGICAL ARRAY  INFOSTR;

     BEGIN
     <<*****************************************************>>
     <<** The GET'INFO procedure will locate the ;PARM= and **>>
     <<** ;INFO= parameters from the :RUN statement.  This  **>>
     <<** procedure will correctly execute on any version of**>>
     <<** MPE IV and MPE V.  Note the Delta-P value change  **>>
     <<** from %0 of MPE IV to %40000 of MPE V.  By reading **>>
     <<** stack markers till these values, we eventually    **>>
     <<** locate the terminate marker and the parameters are**>>
     <<** then at SM-4, SM-5, and SM-6.                     **>>
     <<*****************************************************>>


     INTEGER          QREG = Q;

     INTEGER POINTER  SM;

     BYTE POINTER     INFO,
                      INFOT;

     <<************************>>
     <<** Start of main code **>>
     <<************************>>

     @SM:=@QREG;                 <<** Point to current marker **>>

     I:=0;
     DONE:=FALSE;

     DO
       BEGIN
       IF SM(-2) = 0 THEN        <<** MPE IV terminate marker **>>
         DONE:=TRUE
       ELSE
```

The FS'ERROR procedure formats a standard error message for any file system error detected while accessing any of the MPE files. This includes the text from FERRMSG and the tombstone from PRINTFILEINFO.

Figure 2: Monitor program in SPL (Cont.)

```
121             BEGIN
122             IF SM(-2) = %40000 THEN <<** MPE V terminate marker **>>
123               DONE:=TRUE
124             ELSE
125               BEGIN
126               @SM:=@SM-SM;            <<** Walk down stack marker  **>>
127               I:=I+1;
128               IF I = 100 THEN
129                 QUIT (-1);            <<** Cannot find terminate   **>>
130               END;                    <<** marker so abort!        **>>
131             END;
132           END
133         UNTIL (DONE);
134
135         PARM:=DOUBLE (SM(-4));        <<** SM-4 = value of ;PARM=  **>>
136         INFOL:=SM (-6);               <<** SM-6 = length of ;INFO= **>>
137         @INFO:=SM(-5);                <<** Byte ptr to info string **>>
138         @INFOT:=@INFOSTR&ASL(1);      <<** Byte ptr of target      **>>
139         MOVE INFOT:=INFO,(INFOL);     <<** Move to target          **>>
140         END;
141     $PAGE
142         PROCEDURE  FS'ERROR (FILE'INDEX, FILE'NUMBER, FS'INTRINSIC);
143             VALUE  FILE'INDEX, FS'INTRINSIC;
144           INTEGER  FILE'INDEX, FILE'NUMBER, FS'INTRINSIC;
145
146         BEGIN
147         FCHECK (FILE'NUMBER, FERROR);
148         MOVE TERMLINE:="** Unexpected FS error ",2;
149         NUM'CHAR:=ASCII (FERROR,10,BA1);
150         MOVE *:=BA1,(NUM'CHAR),2;
151         MOVE *:=" has occurred on ",2;
152         CASE FILE'INDEX OF
153           BEGIN
154       <<0>> ;
155       <<1>> MOVE *:="MSGM",2;
156       <<2>> MOVE *:="MSGA",2;
157       <<3>> MOVE *:="MSGB",2;
158           END;  <<** END OF CASE **>>
159         MOVE *:=" during ",2;
160         CASE FS'INTRINSIC OF
161           BEGIN
162       <<0>> ;
163       <<1>> MOVE *:="FOPEN",2;
164       <<2>> MOVE *:="FCLOSE",2;
165       <<3>> MOVE *:="FREAD",2;
166       <<4>> MOVE *:="FWRITE",2;
167       <<5>> MOVE *:="FCONTROL",2;
168           END;  <<** END OF CASE **>>
169         MOVE *:=". **",2;
170         LENGTH:=TOS-LOGICAL(@TERMLINE);
171         PRINT (LTERMLINE,-LENGTH,0);
172         FERRMSG (FERROR,LTERMLINE,LENGTH);
173         PRINT (LTERMLINE,0,0);
174         PRINT (LTERMLINE,-LENGTH,0);
175         PRINTFILEINFO (FILE'NUMBER);
176         TERMINATE;
177         END;
178     $PAGE
179         BEGIN
180
```

Execution of the program begins on this page. The program banner and system time are displayed. The global RINs for the MONITOR program, program A, and program B are retrieved from the ;INFO= parameter with the GET'INFO procedure. The MONITOR global RIN is locked to restrict only one execution of this program.

Figure 2: Monitor program in SPL (Cont.)

```
181         <<**********************************>>
182         <<** Start of main program code. **>>
183         <<**********************************>>
184
185         TRUE'COND:=TRUE;
186
187         MOVE TERMLINE:=("*** MONITOR *** B.01.00 Copyr. 1986, ",
188                         "Benge Bruno.  All rights reserved."),2;
189         LENGTH:=TOS-LOGICAL(@TERMLINE);
190         PRINT (LTERMLINE,-LENGTH,0);
191         DATELINE (TERMLINE);
192         PRINT (LTERMLINE,-27,0);
193         PRINT (LTERMLINE,0,0);
194
195         GET'INFO (RUN'PARM, INFO'LENGTH, LINFO'STRING);
196
197         MONITOR'RIN:=BINARY (INFO'STRING,3);
198         IF <> THEN
199           BEGIN
200           MOVE TERMLINE:="** Invalid monitor RIN passed. **",2;
201           LENGTH:=TOS-LOGICAL(@TERMLINE);
202           PRINT (LTERMLINE,-LENGTH,0);
203           TERMINATE;
204           END;
205
206         PAUSE'RIN'A:=BINARY (INFO'STRING(3),3);
207         IF <> THEN
208           BEGIN
209           MOVE TERMLINE:="** Invalid pause RIN for A passed. **",2;
210           LENGTH:=TOS-LOGICAL(@TERMLINE);
211           PRINT (LTERMLINE,-LENGTH,0);
212           TERMINATE;
213           END;
214
215         PAUSE'RIN'B:=BINARY (INFO'STRING(6),3);
216         IF <> THEN
217           BEGIN
218           MOVE TERMLINE:="** Invalid pause RIN for B passed. **",2;
219           LENGTH:=TOS-LOGICAL(@TERMLINE);
220           PRINT (LTERMLINE,-LENGTH,0);
221           TERMINATE;
222           END;
223     $PAGE
224         <<**********************************************>>
225         <<** Lock RIN 'MONITOR' to ensure that only one  **>>
226         <<** copy of this program is active.  Locate the **>>
227         <<** RIN number from the ;PARM= parameter.       **>>
228         <<**********************************************>>
229
230         LOCK'COND.(15:1):=0;
231         MOVE BA1:="MONITOR ";
232
233         LOCKGLORIN (MONITOR'RIN,LOCK'COND,BA1);
234         IF <> THEN
235           BEGIN
236           IF > THEN
237             BEGIN
238             MOVE TERMLINE:=("** Monitor program is currently ",
239                             "executing. **"),2;
240             LENGTH:=TOS-LOGICAL(@TERMLINE);
```

Now that the global RIN is locked, continue by opening the message files and enable the message file wait facility. Note the file and access options of the FOPEN intrinsic. This allows for multi-processing and inter-job access.

Figure 2: Monitor program in SPL (Cont.)

```
241             PRINT (LTERMLINE,-LENGTH,0);
242             END
243           ELSE
244             BEGIN
245             MOVE TERMLINE:=("** LOCKGLORIN intrinsic error: RIN# = "),2;
246             NUM'CHAR:=ASCII (MONITOR'RIN,10,BA1);
247             MOVE *:=BA1,(NUM'CHAR),2;
248             MOVE *:=".  RIN password = MONITOR. **",2;
249             LENGTH:=TOS-LOGICAL(@TERMLINE);
250             PRINT (LTERMLINE,-LENGTH,0);
251             END;
252           MOVE TERMLINE:="** This process cannot continue. **",2;
253           LENGTH:=TOS-LOGICAL(@TERMLINE);
254           PRINT (LTERMLINE,0,0);
255           PRINT (LTERMLINE,-LENGTH,0);
256           TERMINATE;
257           END;
258     $PAGE
259         <<**************************************************>>
260         <<** Now open the MONITOR, PROGRAMA, and PROGRAMB     **>>
261         <<** message files for read and write access.        **>>
262         <<**************************************************>>
263
264         MOVE TERMLINE:="MSGM ";
265         MSGM'FILE:=FOPEN (TERMLINE, %30105, %2300);
266         IF <> THEN
267           FS'ERROR (1,MSGM'FILE,1);
268
269         MOVE TERMLINE:="MSGA ";
270         MSGA'FILE:=FOPEN (TERMLINE, %30105, %2303);
271         IF <> THEN
272           FS'ERROR (2,MSGA'FILE,1);
273
274         MOVE TERMLINE:="MSGB ";
275         MSGB'FILE:=FOPEN (TERMLINE, %30105, %2303);
276         IF <> THEN
277           FS'ERROR (3,MSGB'FILE,1);
278
279         <<**************************************************>>
280         <<** Now enable extended waits on empty files for   **>>
281         <<** read access and full files for write access.   **>>
282         <<**************************************************>>
283
284         FCONTROL (MSGM'FILE, 45, TRUE'COND);
285         IF <> THEN
286           FS'ERROR (1,MSGM'FILE,5);
287
288         FCONTROL (MSGA'FILE, 45, TRUE'COND);
289         IF <> THEN
290           FS'ERROR (2,MSGA'FILE,5);
291
292         FCONTROL (MSGB'FILE, 45, TRUE'COND);
293         IF <> THEN
294           FS'ERROR (3,MSGB'FILE,5);
295     $PAGE
296         <<**************************************************>>
297         <<** Now execute the main body of the loop waiting **>>
298         <<** for the control codes to process.             **>>
299         <<**************************************************>>
300
```

The primary MSGM message file is read for incoming process requests. The external job streams for programs A and B may be started, stopped, suspended, or resumed.

Figure 2: Monitor program in SPL (Cont.)

```
301         OK:=TRUE;
302         TIME'2'QUIT:=FALSE;
303
304         DO
305           BEGIN
306           IN'LENGTH:=FREAD (MSGM'FILE, LINPUT'RECORD, -240);
307           IF <> THEN
308             FS'ERROR (1,MSGM'FILE,3);
309
310           IF IR'CONTROL'CODE < -4 LOR IR'CONTROL'CODE > -1 THEN
311             BEGIN
312             MOVE TERMLINE:="** Invalid control code of ",2;
313             NUM'CHAR:=ASCII (IR'CONTROL'CODE,10,BA1);
314             MOVE *:=BA1,(NUM'CHAR),2;
315             MOVE *:=" received.  Ignored. **",2;
316             LENGTH:=TOS-LOGICAL(@TERMLINE);
317             PRINT (LTERMLINE,-LENGTH,0);
318             END
319           ELSE
320             BEGIN
321             I:=IR'CONTROL'CODE * -1;
322             CASE I OF
323               BEGIN
324         <<0>> ;
325
326         <<1>> BEGIN  <<** Startup. **>>
327               MOVE TERMLINE:=("STREAM PROGRAMA.STREAMS",%15);
328               COMMAND (TERMLINE,CERROR,CPARM);
329               IF <> OR CERROR > 0 OR CPARM > 0 THEN
330                 BEGIN
331                 MOVE TERMLINE:=("** Unable to stream PROGRAMA. ",
332                                 "Cerror="),2;
333                 NUM'CHAR:=ASCII (CERROR,10,BA1);
334                 MOVE *:=BA1,(NUM'CHAR),2;
335                 MOVE *:=". Cparm=",2;
336                 NUM'CHAR:=ASCII (CPARM,10,BA1);
337                 MOVE *:=BA1,(NUM'CHAR),2;
338                 MOVE *:=". **",2;
339                 LENGTH:=TOS-LOGICAL(@TERMLINE);
340                 PRINT (LTERMLINE,-LENGTH,0);
341                 OK:=FALSE;
342                 END
343               ELSE
344                 BEGIN
345                 MOVE TERMLINE:=("STREAM PROGRAMB.STREAMS",%15);
346                 COMMAND (TERMLINE,CERROR,CPARM);
347                 IF <> OR CERROR > 0 OR CPARM > 0 THEN
348                   BEGIN
349                   MOVE TERMLINE:=("** Unable to stream PROGRAMA. ",
350                                   "Cerror="),2;
351                   NUM'CHAR:=ASCII (CERROR,10,BA1);
352                   MOVE *:=BA1,(NUM'CHAR),2;
353                   MOVE *:=". Cparm=",2;
354                   NUM'CHAR:=ASCII (CPARM,10,BA1);
355                   MOVE *:=BA1,(NUM'CHAR),2;
356                   MOVE *:=". **",2;
357                   LENGTH:=TOS-LOGICAL(@TERMLINE);
358                   PRINT (LTERMLINE,-LENGTH,0);
359                   OK:=FALSE;
360                   END
```

In order to suspend each process, the secondary global RIN is locked by the monitor program. Once locked, program A or B is informed to unconditionally lock this RIN. This will be demonstrated later in program A.

Figure 2: Monitor program in SPL (Cont.)

```
361                ELSE
362                  BEGIN
363                  MOVE TERMLINE:=("Programs A and B have been ",
364                                  "started."),2;
365                  LENGTH:=TOS-LOGICAL(@TERMLINE);
366                  PRINT (LTERMLINE,-LENGTH,0);
367                  END;
368                END;
369              END;
370
371        <<2>> BEGIN  <<** Shutdown. **>>
372              LTERMLINE:=-1;
373              FWRITE (MSGA'FILE,LTERMLINE,-2,0);
374              IF <> THEN
375                FS'ERROR (2,MSGA'FILE,4);
376
377              FWRITE (MSGB'FILE,LTERMLINE,-2,0);
378              IF <> THEN
379                FS'ERROR (3,MSGB'FILE,4);
380
381              MOVE TERMLINE:="Programs A and B have been stopped. ",2;
382              LENGTH:=TOS-LOGICAL(@TERMLINE);
383              PRINT (LTERMLINE,-LENGTH,0);
384
385              TIME'2'QUIT:=TRUE;
386              END;
387
388        <<3>> BEGIN  <<** Suspend A or B. **>>
389              IF IR'PROGRAM'ID = "A" OR IR'PROGRAM'ID = "a" THEN
390                I:=1
391              ELSE
392                IF IR'PROGRAM'ID = "B" OR IR'PROGRAM'ID = "b" THEN
393                  I:=2
394                ELSE
395                  I:=0;
396
397              CASE I OF
398                BEGIN
399
400          <<0>> BEGIN
401                MOVE TERMLINE:=("** Unknown process to suspend; must ",
402                                "be 'A' or 'B'. **"),2;
403                LENGTH:=TOS-LOGICAL(@TERMLINE);
404                PRINT (LTERMLINE,-LENGTH,0);
405                END;
406
407          <<1>> BEGIN
408                LOCK'COND.(15:1):=1;
409                MOVE BA1:="PROGRAMA ";
410
411                LOCKGLORIN (PAUSE'RIN'A,LOCK'COND,BA1);
412                IF < THEN
413                  BEGIN
414                  MOVE TERMLINE:=("** LOCKGLORIN intrinsic error: "
415                                  "RIN# = "),2;
416                  NUM'CHAR:=ASCII (PAUSE'RIN'A,10,BA1);
417                  MOVE *:=BA1,(NUM'CHAR),2;
418                  MOVE *:=". RIN password = PROGRAMA. **",2;
419                  LENGTH:=TOS-LOGICAL(@TERMLINE);
420                  PRINT (LTERMLINE,-LENGTH,0);
```

Program B is suspended by executing the code below. This completes the suspend code.

Figure 2: Monitor program in SPL (Cont.)

```
421                 OK:=FALSE;
422                 END;
423
424               LTERMLINE:=-2;
425               FWRITE (MSGA'FILE,LTERMLINE,-2,0);
426               IF <> THEN
427                 FS'ERROR (2,MSGA'FILE,4);
428
429               MOVE TERMLINE:="Program A is suspended.",2;
430               LENGTH:=TOS-LOGICAL(@TERMLINE);
431               PRINT (LTERMLINE,-LENGTH,0);
432               END;
433
434          <<2>> BEGIN
435               LOCK'COND.(15:1):=1;
436               MOVE BA1:="PROGRAMB ";
437
438               LOCKGLORIN (PAUSE'RIN'B,LOCK'COND,BA1);
439               IF < THEN
440                 BEGIN
441                 MOVE TERMLINE:=("** LOCKGLORIN intrinsic error: ",
442                                 "RIN# = "),2;
443                 NUM'CHAR:=ASCII (PAUSE'RIN'B,10,BA1);
444                 MOVE *:=BA1,(NUM'CHAR),2;
445                 MOVE *:=".  RIN password = PROGRAMB. **",2;
446                 LENGTH:=TOS-LOGICAL(@TERMLINE);
447                 PRINT (LTERMLINE,-LENGTH,0);
448                 OK:=FALSE;
449                 END;
450
451               LTERMLINE:=-2;
452               FWRITE (MSGB'FILE,LTERMLINE,-2,0);
453               IF <> THEN
454                 FS'ERROR (2,MSGB'FILE,4);
455
456               MOVE TERMLINE:="Program B is suspended.",2;
457               LENGTH:=TOS-LOGICAL(@TERMLINE);
458               PRINT (LTERMLINE,-LENGTH,0);
459               END;
460
461               END;  <<** END OF CASE **>>
462             END;
463
464        <<4>> BEGIN  <<** Resume A or B. **>>
465             IF IR'PROGRAM'ID = "A" OR IR'PROGRAM'ID = "a" THEN
466               I:=1
467             ELSE
468               IF IR'PROGRAM'ID = "B" OR IR'PROGRAM'ID = "b" THEN
469                 I:=2
470               ELSE
471                 I:=0;
472
473             CASE I OF
474               BEGIN
475
476          <<0>> BEGIN
477               MOVE TERMLINE:=("** Unknown process to resume; must ",
478                               "be 'A' or 'B'. **"),2;
479               LENGTH:=TOS-LOGICAL(@TERMLINE);
480               PRINT (LTERMLINE,-LENGTH,0);
```

Programs A and B are resumed by simply unlocking the secondary global RIN. This will allow programs A and B to successfully lock the RIN. The RIN is then immediately unlocked and the process continues as before. Once the monitor program completes, the MONITOR global RIN is unlocked and the message files are closed.

Figure 2: Monitor program in SPL (Cont.)

```
481                 END;
482
483           <<1>> BEGIN
484                 UNLOCKGLORIN (PAUSE'RIN'A);
485
486                 MOVE TERMLINE:="Program A has been resumed.",2;
487                 LENGTH:=TOS-LOGICAL(@TERMLINE);
488                 PRINT (LTERMLINE,-LENGTH,0);
489                 END;
490
491           <<2>> BEGIN
492                 UNLOCKGLORIN (PAUSE'RIN'B);
493
494                 MOVE TERMLINE:="Program B has been resumed.",2;
495                 LENGTH:=TOS-LOGICAL(@TERMLINE);
496                 PRINT (LTERMLINE,-LENGTH,0);
497                 END;
498
499               END;  <<** END OF CASE **>>
500             END;
501
502             END;  <<** END OF CASE **>>
503           END;
504         END
505       UNTIL (TIME'2'QUIT OR NOT OK);
506   $PAGE
507       <<*************************************************>>
508       <<** Now close the files and unlock the global RIN. **>>
509       <<*************************************************>>
510
511       UNLOCKGLORIN (INTEGER(RUN'PARM));
512
513       FCLOSE (MSGM'FILE,0,0);
514       IF <> THEN
515         FS'ERROR (1,MSGM'FILE,2);
516
517       FCLOSE (MSGA'FILE,0,0);
518       IF <> THEN
519         FS'ERROR (2,MSGA'FILE,2);
520
521       FCLOSE (MSGB'FILE,0,0);
522       IF <> THEN
523         FS'ERROR (3,MSGB'FILE,2);
524       END;
525   $PAGE
526   END.
```

In Figure 3 below, Process A is diagrammed to use the ;INFO= parameter of the :RUN command to locate the program suffix of A or B. It also contains the primary and secondary global RINs for the password of PROGRAMx, where x is the program suffix.

The program issues a conditional lock on the primary global RIN. If successful, the program continues; otherwise, an error message is displayed since some other process currently has this RIN locked (this prevents two copies of the program from executing because another copy is already executing).

Program A opens two message files: the primary message file named 'A' is opened for read access; the monitor message file named 'M' is opened for write access in order to communicate with the monitor process M.

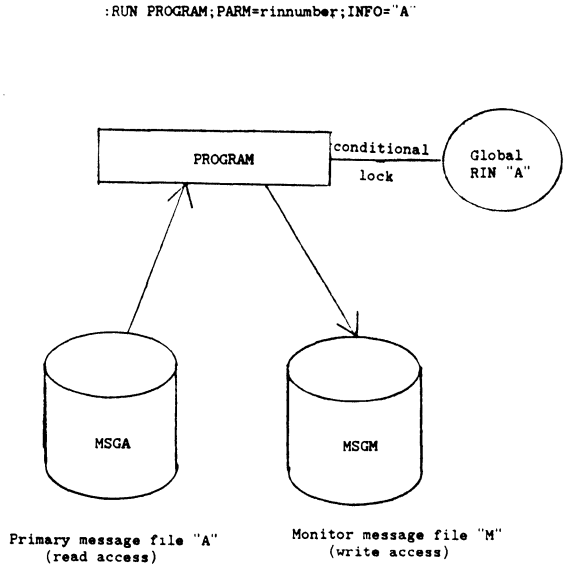


Figure 3: Program "A" Execution Flow Diagram

The second program example has been coded in COBOLII for comparison to that in SPL mentioned earlier. The variable names correspond closely to those used in the SPL version of the monitor program.

Figure 4: Program 'x' in COBOLII

```
1     $CONTROL SOURCE,MAP,CROSSREF,VERBS
1.1    IDENTIFICATION DIVISION.
1.2    PROGRAM-ID. PROGRAM.
1.3    AUTHOR. Benedict G. Bruno.
1.4    DATE-WRITTEN. June, 1986.
1.5    DATE-COMPILED.
1.6    REMARKS. The PROGRAM program will lock the global RIN name of
1.7             'PROGRAMx' with the RIN number passed in the ;INFO=
1.8             string and the 'x' replaced from the ;INFO=.
1.9
2      ENVIRONMENT DIVISION.
2.1    CONFIGURATION SECTION.
2.2
2.3    SOURCE-COMPUTER. HP3000.
2.4    OBJECT-COMPUTER. HP3000.
2.5
2.6    SPECIAL-NAMES. CONDITION-CODE IS COND-CODE.
2.7
2.8    DATA DIVISION.
2.9   $PAGE " "
3      WORKING-STORAGE SECTION.
3.1
3.2    01 TIME-2-QUIT          PIC X(1) VALUE "N".
3.3    01 OK                   PIC X(1) VALUE "Y".
3.4
3.5    01 INPUT-REC.
3.6       03 CONTROL-CODE       PIC S9(4) COMP.
3.7
3.8    01 DOUBLE-VARIABLES.
3.9       03 RUN-PARMD         PIC S9(9) COMP.
4         03 RUN-PARMI REDEFINES RUN-PARMD.
4.1          05 RUN-PARM1      PIC S9(4) COMP.
4.2          05 RUN-PARM2      PIC S9(4) COMP.
4.3
4.4    01 LOGICAL-VARIABLES.
4.5       03 DONE               PIC S9(4) COMP.
4.6       03 LOCK-COND          PIC S9(4) COMP.
4.7       03 TRUE-COND          PIC S9(4) COMP VALUE -1.
4.8
4.9    01 INTEGER-VARIABLES.
5         03 CERROR             PIC S9(4) COMP.
5.1       03 CPARM              PIC S9(4) COMP.
5.2       03 FERROR             PIC S9(4) COMP.
5.3       03 I                  PIC S9(4) COMP.
5.4       03 IN-LENGTH          PIC S9(4) COMP.
5.5       03 INFO-LENGTH        PIC S9(4) COMP.
5.6       03 LEN                PIC S9(4) COMP.
5.7       03 MSGM-FILE          PIC S9(4) COMP.
5.8       03 MSGX-FILE          PIC S9(4) COMP.
5.9       03 NUM-CHAR           PIC S9(4) COMP.
6         03 PAUSE-RIN          PIC S9(4) COMP.
6.1       03 PROCESS-RIN        PIC S9(4) COMP.
6.2
6.3    01 CHARACTER-VARIABLES.
6.4       03 TERMLINE           PIC X(79).
6.5       03 BA1                PIC X(160).
6.6       03 CERROR-A           PIC X(5).
6.7       03 CPARM-A            PIC X(5).
6.8
6.9    01 INFO-STRING.
```

The procedure division follows below with the main driver section. The START-UP paragraph performs the initialization tasks, namely the retrieval of the primary and secondary global RINs as below.

Figure 4: Program 'x' in COBOLII (Cont.)

```
7         03 PROGRAM-NAME       PIC X(1).
7.1       03 PROCESS-RIN-A      PIC X(3).
7.2       03 PAUSE-RIN-A        PIC X(3).
7.3    $PAGE " "
7.4     PROCEDURE DIVISION.
7.5
7.6    ****************************************************************
7.7    **** Start of main driver section.                          ****
7.8    ****************************************************************
7.9
8       MAIN SECTION.
8.1         PERFORM START-UP.
8.2         PERFORM PROCESS-INPUT UNTIL TIME-2-QUIT = "Y" OR OK = "N".
8.3         PERFORM FINISH-UP.
8.4         STOP RUN.
8.5    $PAGE
8.6    ****************************************************************
8.7    **** The initialization section displays the program banner, ****
8.8    **** locate the ;INFO= parameters, and open files.          ****
8.9    ****************************************************************
9
9.1     START-UP.
9.2
9.3         MOVE SPACES TO TERMLINE.
9.4         STRING "*** PROGRAM *** B.01.00 Copyr. 1986, "
9.5                DELIMITED BY SIZE
9.6                "Benge Bruno.  All rights reserved."
9.7                DELIMITED BY SIZE INTO TERMLINE.
9.8         DISPLAY TERMLINE.
9.9         MOVE SPACES TO TERMLINE.
10          CALL INTRINSIC "DATELINE" USING TERMLINE.
10.1        DISPLAY TERMLINE.
10.2        MOVE SPACES TO TERMLINE.
10.3        DISPLAY TERMLINE.
10.4
10.5        CALL "GET'INFO" USING RUN-PARMD INFO-LENGTH INFO-STRING.
10.6
10.7        IF PROGRAM-NAME NOT = "A" AND PROGRAM-NAME NOT = "a" AND
10.8           PROGRAM-NAME NOT = "B" AND PROGRAM-NAME NOT = "b" THEN
10.9            MOVE SPACES TO TERMLINE
11              MOVE "** Invalid program suffix passed. **" TO TERMLINE
11.1            DISPLAY TERMLINE
11.2            STOP RUN
11.3
11.4
11.5        IF PROCESS-RIN-A NOT NUMERIC THEN
11.6            MOVE SPACES TO TERMLINE
11.7            MOVE "** Invalid process RIN value. **" TO TERMLINE
11.8            DISPLAY TERMLINE
11.9            STOP RUN
12
12.1
12.2        IF PAUSE-RIN-A NOT NUMERIC THEN
12.3            MOVE SPACES TO TERMLINE
12.4            MOVE "** Invalid pause RIN value. **" TO TERMLINE
12.5            DISPLAY TERMLINE
12.6            STOP RUN
12.7
12.8   $PAGE
12.9   ****************************************************************
```

A conditional lock is executed for the primary global RIN as below. Once completed, the process and monitor message files are opened.

Figure 4: Program 'x' in COBOLII (Cont.)

```
13    **** Lock RIN 'PROGRAM' to ensure that only one copy of this ****
13.1  **** program is active.  Locate the RIN number from the      ****
13.2  **** ;INFO= parameter.                                       ****
13.3  ****************************************************************
13.4
13.5        MOVE SPACES TO BA1.
13.6        STRING "PROGRAM" DELIMITED BY SIZE
13.7               PROGRAM-NAME DELIMITED BY SIZE INTO BA1.
13.8        MOVE PROCESS-RIN-A TO PROCESS-RIN.
13.9        MOVE 0 TO LOCK-COND.
14
14.1        CALL INTRINSIC "LOCKGLORIN" USING PROCESS-RIN LOCK-COND BA1.
14.2
14.3        IF COND-CODE > 0 THEN
14.4            MOVE SPACES TO TERMLINE
14.5            MOVE "** Program is currently executing. **" TO TERMLINE
14.6            DISPLAY TERMLINE
14.7            MOVE "** This process cannot continue. **" TO TERMLINE
14.8            DISPLAY TERMLINE
14.9            STOP RUN
15          ELSE
15.1            IF COND-CODE < 0 THEN
15.2                MOVE SPACES TO TERMLINE
15.3                STRING "** LOCKGLORIN intrinsic error: RIN# = "
15.4                       DELIMITED BY SIZE
15.5                       PROCESS-RIN-A DELIMITED BY SIZE
15.6                       ".  RIN password = PROGRAM. **"
15.7                       DELIMITED BY SIZE INTO TERMLINE
15.8                DISPLAY TERMLINE
15.9                MOVE "** This process cannot continue. **" TO
16                     TERMLINE
16.1                STOP RUN
16.2
16.3  $PAGE
16.4  ****************************************************************
16.5  **** Now open the MONITOR and PROGRAMx message files for      ****
16.6  **** read and write access.                                   ****
16.7  ****************************************************************
16.8
16.9        MOVE "MSGM " TO TERMLINE.
17          CALL INTRINSIC "FOPEN" USING  TERMLINE %30105 %2303
17.1                                GIVING MSGM-FILE.
17.2        IF COND-CODE NOT = 0 THEN
17.3            CALL "FS'ERROR" USING 1 MSGM-FILE 1
17.4
17.5
17.6        MOVE SPACES TO TERMLINE.
17.7        STRING "MSG" DELIMITED BY SIZE
17.8               PROGRAM-NAME DELIMITED BY SIZE INTO TERMLINE.
17.9        CALL INTRINSIC "FOPEN" USING  TERMLINE %30105 %2300
18                                  GIVING MSGX-FILE.
18.1        IF COND-CODE NOT = 0 THEN
18.2            CALL "FS'ERROR" USING 2 MSGX-FILE 1
18.3
18.4
18.5  ****************************************************************
18.6  **** Now enable extended waits on empty files for read        ****
18.7  **** access and full files for write access.                  ****
18.8  ****************************************************************
18.9
```

The extended wait facility is enabled as below. This completes the START-UP paragraph and the initialization function. The PROCESS-INPUT paragraph is called from the main driver to process input requests until a shutdown is received. The SUSPEND-PROCESS paragraph is executed when the monitor program requests us to suspend. This is accomplished by unconditionally locking the secondary global RIN as below.

Figure 4: Program 'x' in COBOLII (Cont.)

```
19          CALL INTRINSIC "FCONTROL" USING MSGM-FILE 45 TRUE-COND.
19.1        IF COND-CODE NOT = 0 THEN
19.2            CALL "FS'ERROR" USING 1 MSGM-FILE 5
19.3
19.4
19.5        CALL INTRINSIC "FCONTROL" USING MSGX-FILE 45 TRUE-COND.
19.6        IF COND-CODE NOT = 0 THEN
19.7            CALL "FS'ERROR" USING 2 MSGX-FILE 5
19.8
19.9    $PAGE
20       PROCESS-INPUT.
20.1
20.2    ****************************************************************
20.3    **** Now execute the main body of the loop waiting for the   ****
20.4    **** control codes to process.                               ****
20.5    ****************************************************************
20.6
20.7        CALL INTRINSIC "FREAD" USING  MSGX-FILE INPUT-REC -240
20.8                                  GIVING IN-LENGTH.
20.9        IF COND-CODE NOT = 0 THEN
21              CALL "FS'ERROR" USING 1 MSGM-FILE 3
21.1
21.2
21.3        IF CONTROL-CODE < -2 OR CONTROL-CODE > -1 THEN
21.4            MOVE "** Invalid control code received.  Ignored. **" TO
21.5                TERMLINE
21.6            DISPLAY TERMLINE
21.7        ELSE
21.8            IF CONTROL-CODE = -1 THEN
21.9                MOVE "Y" TO TIME-2-QUIT
22              ELSE
22.1                IF CONTROL-CODE = -2 THEN
22.2                    PERFORM SUSPEND-PROCESS
22.3
22.4    $PAGE
22.5     SUSPEND-PROCESS.
22.6
22.7    ****************************************************************
22.8    **** Now suspend the program by unconditionally locking the  ****
22.9    **** pause RIN.  When the RIN lock completes, we release     ****
23      **** it immediately, thus resuming execution.                ****
23.1    ****************************************************************
23.2
23.3        MOVE 1 TO LOCK-COND.
23.4        MOVE PAUSE-RIN-A TO PAUSE-RIN.
23.5        MOVE SPACES TO BA1.
23.6        STRING "PROGRAM" DELIMITED BY SIZE
23.7               PROGRAM-NAME DELIMITED BY SIZE INTO BA1.
23.8
23.9        CALL INTRINSIC "LOCKGLORIN" USING PAUSE-RIN LOCK-COND BA1.
24          IF COND-CODE < 0 THEN
24.1            MOVE SPACES TO TERMLINE
24.2            MOVE "** LOCKGLORIN intrinsic error: RIN#=" TO TERMLINE
24.3            DISPLAY TERMLINE
24.4            MOVE "N" TO OK
24.5        ELSE
24.6            CALL INTRINSIC "UNLOCKGLORIN" USING PAUSE-RIN
24.7
24.8    $PAGE
24.9     FINISH-UP.
```

The FINISH-UP paragraph below completes program execution by unlocking the primary global RIN and closing the message files.

Figure 4: Program 'x' in COBOLII (Cont.)

```
25
25.1   ****************************************************************
25.2   **** Now close the files and unlock the global RIN.          ****
25.3   ****************************************************************
25.4
25.5        CALL INTRINSIC "UNLOCKGLORIN" USING PROCESS-RIN.
25.6
25.7        CALL INTRINSIC "FCLOSE" USING MSGM-FILE 0 0.
25.8        IF COND-CODE NOT = 0 THEN
25.9            CALL "FS'ERROR" USING 1 MSGM-FILE 2.
26
26.1        CALL INTRINSIC "FCLOSE" USING MSGX-FILE 0 0.
26.2        IF COND-CODE NOT = 0 THEN
26.3            CALL "FS'ERROR" USING 2 MSGX-FILE 2.
```

In order to execute the monitor, A, and B programs, the global RINs must be located. The GETRIN MPE command is executed for each of the program passwords. Figure 5 below displays the logon and GETRIN commands. The bold face type denotes input user input.

Figure 5: Executing the GETRIN command

```
:HELLO BENGE.BRUNO (RETURN)
HP3000 / MPE V G.01.01 (BASE G.01.01). SUN, MAY 25, 1986, 10:10 AM
:GETRIN MONITOR (RETURN)
RIN: 122
:GETRIN PROGRAMA (RETURN)
RIN: 87
:GETRIN PROGRAMA (RETURN)
RIN: 139
:GETRIN PROGRAMB (RETURN)
RIN: 126
:GETRIN PROGRAMB (RETURN)
RIN: 129
:BYE (RETURN)

CPU=1. CONNECT=3. SUN, MAY 25, 1986, 10:13 AM
```

The SHOWRIN program

The SHOWRIN program provides a formatted display of the system global RIN table. The program is written using privileged mode for execution on any MPE IV and MPE V based systems. The program may only execute in session mode and will immediately terminate in batch mode.

The group that SHOWRIN executes must have PM capability. For this reason, you may wish to place it in PUB.SYS or UTIL.SYS where PM is already in the group. The capability list of the user is checked when the SHOWRIN program is executed. If the user has system manager (SM) or privileged mode (PM) capability, then the SHOWRIN program may display the RIN table for all accounts; otherwise, only the user logon account may be used.

The SHOWRIN program may simply be executed using the :RUN statement. The alternate entry point of 'NOHELP' may be used to disable the initial help information as in Figure 6 below.

```
:HELLO BENGE,MANAGER.SYS (RETURN)
:RUN SHOWRIN (RETURN)

*** SHOWRIN ***  B.01.00 Copyr. 1986, Benge Bruno.  All rights reserved.
SUN, MAY 25, 1986,  7:30 PM

The SHOWRIN program provides a display of global RINs on your HP 3000.
Each of the commands are entered as single characters as in OPT/3000.
The following commands are available.

A -> Prompt for a specific account name; search on this account only.
B -> Display both locked and unlocked RIN entries found.
E -> Exit the program.
H -> Display this help information.
L -> Display locked RIN entries only.
P -> Enable/disable printing to RINLIST on device LP.
R -> Prompt for RIN numbers for additional information.
S -> Display RINs for all accounts; Requires PM or SM capability.
U -> Display unlocked RIN entries only.
W -> Display waiting processes and the holding processes (deadlock?).

To disable this initial help dialogue, use the entry point of 'NOHELP'.

Please press any key to continue.
```

Figure 6: Running the SHOWRIN program

After pressing the return key, the system RIN table will be displayed with format in Figure 7 below.

```
System RIN Table        LP    System-wide, LOCK ONLY                6:05 PM

RIN      RIN          Creating
 #     password   Username.Acctname     PIN   Jobnum          Job name
----   --------   -----------------    ----   ------   --------------------------
```

Figure 7: SHOWRIN header display

Note that the top line contains the current system time, LP, and RIN selection. Since the logged on user has PM or SM capability, the RIN table will be checked for all accounts. This is noted with the 'System-wide' literal. If PM or SM is not detected, then only the logged on account will be searched. This is noted with the 'Account: xxxxxxxx' literal.

Several dispositions of the global RINs may be selected. The 'LOCK ONLY' option displays only those RINs that are currently locked by a process. The 'UNLOCK ONLY' option displays only those RINs that are currently free. The 'LOCK w/WAIT' option displays only those RINs that are currently locked by a process and the processes that are waiting on these RINs. The 'LOCK, UNLOCK, WAIT' option displays all entries in the table.

When the print option is enabled, an asterisk (*) is placed to the left of the LP designator in the header. The asterisk is removed when the LP option is disabled.

Entries found in the RIN table are displayed in ascending sorted order. For each entry found, the four digit RIN value, password, and creating user and account are displayed. If the RIN is currently locked, then the PIN, job or session number, and the job/session, user, and account of the locking process is displayed. An additional line is displayed with the PIN, job or session number, and the job/session, user, and account of any waiting processes.

The SHOWRIN program has single character terminal reads enabled every 20 seconds. Should no data be received, then a timeout occurs and the table is displayed using the same selection criteria.

To continue with our simple example of the monitor program and two external programs, we must first acquire the global RINs. This requires that the GETRIN MPE command be executed for the values of

'MONITOR', 'PROGRAMA', and 'PROGRAMB' with the program values executed twice; once for the primary process RIN and second for the secondary pause RIN.

Since the RIN values assigned by MPE are system dependent, I do not want to even suggest what they may be. However, using the SHOWRIN program, we may locate their values by using the 'U' for unlock or 'B' for all entries within the account which you have created these RINs. Having performed this in my own account, I may use the SHOWRIN program as in Figure 8 below to locate the (currently) unlocked RIN values and the associated passwords.

```
System RIN Table        LP    Account: BRUNO, LOCK,UNLOCK,WAIT          6:56 PM

RIN      RIN          Creating
 #     password    Username.Acctname    PIN   Jobnum          Job name
----   --------   -----------------    ----   ------  --------------------------
  87   PROGRAMA    BENGE.BRUNO
 122   MONITOR     BENGE.BRUNO
 126   PROGRAMB    BENGE.BRUNO
 129   PROGRAMB    BENGE.BRUNO
 139   PROGRAMA    BENGE.BRUNO
```

Figure 8: Locating all RIN entries for a specific account

If the monitor, program A, and program B processes are initiated, then each program will lock its corresponding global RIN. Each process will await input by reading its associated message file. Figure 9 below shows this initial state, whereby each RIN value is displayed with the locking process.

```
System RIN Table        LP    Account: BRUNO, LOCK ONLY                 7:11 PM

RIN      RIN          Creating
 #     password    Username.Acctname    PIN   Jobnum          Job name
----   --------   -----------------    ----   ------  --------------------------
  87   PROGRAMA    BENGE.BRUNO          153   #J397   A,BENGE.BRUNO
 122   MONITOR     BENGE.BRUNO          251   #S445   BENGE.BRUNO
 126   PROGRAMB    BENGE.BRUNO          121   #J398   B,BENGE.BRUNO
```

Figure 9: Monitor, program A, program B normal execution

A useful feature of the global RINs discussed earlier is to use them for process control. If the monitor process were to lock the secondary pause RIN conditionally, programs A or B may be suspended if they attempt to lock the secondary pause RIN unconditionally. Figure 10 below displays this situation. You will note the previous entries as in Figure 9 for each of the processes primary RIN. The additional entries are showing that the secondary pause RINs are owned by the monitor process executed by session number 445. Programs A and B executing from job streams are then executing an unconditional lock for the pause RIN.

```
System RIN Table        LP    Account: BRUNO, LOCK ONLY                7:22 PM

RIN      RIN          Creating
 #     password    Username.Acctname     PIN   Jobnum        Job name
----   --------   -----------------     ----   ------   -------------------------
  87   PROGRAMA    BENGE.BRUNO           157   #J402    A,BENGE.BRUNO
 122   MONITOR     BENGE.BRUNO           159   #S445    BENGE.BRUNO
 126   PROGRAMB    BENGE.BRUNO           122   #J403    B,BENGE.BRUNO
 129   PROGRAMB    BENGE.BRUNO           159   #S445    BENGE.BRUNO
                                         122   #J403    B,BENGE.BRUNO
 139   PROGRAMA    BENGE.BRUNO           159   #S445    BENGE.BRUNO
                                         157   #J402    A,BENGE.BRUNO
```

Figure 10: Suspending programs A and B

Figure 11 below displays only the pertinent information from Figure 10 above in that only those RINs with processes waiting are displayed. This is effective in reducing the entries in the display to detect potential deadlocks. Looking closely at the entries in Figure 11 below, we see that the primary RIN values are not included.

```
System RIN Table        LP    Account: BRUNO, LOCK w/WAIT              7:22 PM

RIN      RIN          Creating
 #     password    Username.Acctname     PIN   Jobnum        Job name
----   --------   -----------------     ----   ------   -------------------------
 129   PROGRAMB    BENGE.BRUNO           159   #S445    BENGE.BRUNO
                                         122   #J403    B,BENGE.BRUNO
 139   PROGRAMA    BENGE.BRUNO           159   #S445    BENGE.BRUNO
                                         157   #J402    A,BENGE.BRUNO
```

Figure 11: Avoid deadlocks, display locked RINs w/waiting processes

Summary

Local and global RINs may be used effectively for the monitor and control of processes within an HP application. If RINs are used in conjunction with the MPE message file facility, complete process control of initiation, termination, and automated recovery techniques can be developed.

Global RINs can now be displayed quite easily with the SHOWRIN program. It is my hope that this program and/or a similar capability be incorporated into the HP Online Performance Tool (OPT/3000).

Biography

Benedict G. Bruno has been working with the HP 3000 family of computer systems for seven years. His experience includes working for Hewlett-Packard Company as a Systems Engineer in the Los Angeles Airport Sales Office, a Network Consultant for Information Networks Division in Cupertino, and a Senior Applications Systems Engineer in the Rockville Sales Office. The integration of local and global RINs, message files, SPL, and HP data communications products for several application systems in retail, electronic mail, and others have required his need to develop the SHOWRIN program. He is currently president and cofounder of S.T.R. Software Company where he has developed POS/3000 to provide unattended data communications application software.

Software Design for Long-Term Reliability & Maintainability

by: Toback, Bruce

We regret that this paper
was not received for
inclusion in these proceedings.

The Fourth Bear of IMAGE

Fred White
Adager
Apartado 248
Antigua
Guatemala

INTRODUCTION

The location of users, programs and databases relative to each other on the nodes of an HP3000 network can be critical to program and network performance.

When a user is remote from the program, additional memory and CPU resources are consumed at both nodes to support user I/O.

When a database is remote from the program, additional memory and CPU resources are consumed at both nodes to support database access.

Program performance is also negatively impacted by the presence of a remote user and/or a remote database.

Some of the questions we will attempt to answer are:

1. Is remote user access significantly slower than local user access?

2. Is remote database access significantly slower than local database access?

3. If the user and database are on different nodes is it better to run the program on the user's node or on the database's node?

All of these questions relate to the overhead associated with access to a remote user (or remote database) arising from the use of HP3000 network facilities (DS or NS) whose use requires each node to provide resources (in the form of memory, virtual memory, tables and CPU) that would otherwise not be needed.

TRANSACTIONS

Each transaction of a database application begins with the program prompting the user with a terminal write and then accepting data with a terminal read. There may be many such prompt/response pairs after which the program performs database access determined by the application and the user's responses to the prompts.

This process is repeated until the user input causes the program to perform a database close followed by program termination.

We are not interested in the performance of the program itself so the rest of this paper takes this performance as a given. However, we are interested in how much the performance of such a program would be impacted by either remote database access and/or remote user access.

The term "remote database" is used here to indicate that the node of an HP3000 network on which the database is resident differs from the node on which the program is running. A similar definition applies to the term "remote user".

A remote user (or database) will be said to be "R-remote" from a program if R network transfers occur whenever the program writes to the remote user (or database). In this definition, R=0 implies that the user (or database) is local.

1-REMOTE USER ACCESS

All user access is initiated by the program.

The sequence for 1-remote user access is:

1. The program calls a file management intrinsic.
2. File management packages the call information and invokes the network facility (DS or NS).
3. The network facility transmits the package to the user's node.
4. The network facility at the user's node passes the package to the user's command interpreter process.
5. The command interpreter process performs the same file management call as in step 1 above.
6. File management performs the terminal I/O.
7. The command interpreter process packages the result and invokes the network facility.
8. The network facility transmits the package to the program's node.
9. The network facility on the program's node awakens the program.
10. File management returns the result to the program.

Note that, for local user access, only steps 1, 6 and 10 are performed so that steps 2 thru 5 and 7 thru 9 constitute the additional overhead of 1-remote user access.

For character mode prompting, this overhead consumes about 8 times as much CPU as that required for local user access with about half of this consumed on the program's node and half on the user's node.

At first glance, one might conclude that 1-remote user access would be about 8 times as slow as local user access. In point of fact, it is typically less than 5 percent slower.

To understand why, first note that in order for a program to acquire input from a user it must perform a prompt/response pair so that the user access sequence shown must be traversed twice. Once for writing and once for reading.

For each of these it is necessary that we approximate the average wall time consumed by steps 2 thru 5 and 7 thru 9. Our estimate of 120 milliseconds was derived from Performance News Notes, published by HP, reflecting some LAN/3000 and NS/3000 performance tests run stand-alone on two HP3000/Series 48s and two HP3000/Series 68s.

Proceeding on the assumption that this estimate is quite reasonable, we can conclude that the wall time remote user overhead for a typical prompt/response pair would be on the order of 240 milliseconds.

The combined wall times for performing the terminal I/O of step 6, including think time and data entry time, is typically measured in seconds.

If the combined prompt/response time of step 6 is 5 seconds, then the remote user access overhead of 240 milliseconds would decrease user access performance by about 5 percent.

I have not investigated the degree to which this overhead would impact remote user access when the prompt/response pair is performed in page mode. My guess is that the remote user access overhead would increase by a factor of ten or more but that the combined prompt/response time would also increase by about the same factor so that similar conclusions would result.

1-REMOTE DATABASE ACCESS

All database access is initiated by the program.

The sequence for 1-remote database access is:

1. The program calls an IMAGE/3000 intrinsic.
2. IMAGE packages the call information and invokes the network facility (DS or NS).
3. The network facility transmits the package to the node of the database.
4. The network facility at the node of the database, passes the package to the user's command interpreter process.
5. The command interpreter process performs the same IMAGE intrinsic call as in step 1.
6. IMAGE processes this call. This may involve disc I/Os.
7. The command interpreter process packages the result and invokes the network facility.
8. The network facility transmits the package to the program's node.

9. The network facility on the program's node awakens the program.

10. IMAGE/3000 returns the result to the program.

Note that, for local database access, only steps 1, 6 and 10 are performed so that steps 2 thru 5 and 7 thru 9 constitute the additional overhead of remote database access.

If we denote the wall time of step i by Ti, then the performance of an intrinsic employing remote database access will be N times as slow as the same intrinsic using local database access where:

$$N = (T1+T2+ \ldots +T10)/(T1+T6+T10)$$

As with remote user access, the wall time overhead of steps 2 thru 5 and 7 thru 9 is about 120 milliseconds so that we have:

$$N = (T+120)/T = 1 + 120/T$$

where T is the elapsed time estimate, in milliseconds, of the intrinsic when performed locally. Although the impact on response time is the same for all IMAGE intrinsics, namely about 120 milliseconds, the relative impact varies with the intrinsic being called.

The DBINFO intrinsic, in some modes, is able to respond to the user locally, based on the contents of the remote database control block (RDBCB). In such cases, this equation does not apply and there is no remote database overhead.

In all other cases the intrinsic may perform an average of M disc I/Os where M is non-negative and may be a fraction.

If M=0, T is generally between 4 and 8 milliseconds so that N will be between 16 and 31.

For M=1/2, half of the calls would involve no I/O with a T value of about 5 and the other half would require 1 I/O with a T value of about 30. The time to complete these two would be 35 seconds locally and 275 milliseconds remotely so that N would equal 275/35.

Values of N for various values of M are:

M	N
0	~20
0.5	~8
1	5
2	3
4	2
8	1.5
16	1.25
32	1.125

Thus, database access for a given transaction will be at least twice as slow for remote access as for local access as long as the average number of disc I/Os of the IMAGE intrinsics (excluding those DBINFO calls which are responded to locally) involved in the transaction is less than 5.

REMOTE USER versus REMOTE DATABASE

So far, we have seen that 1-remote user access is generally less than 5 percent slower than local user access and that 1-remote database access may very easily be 5 or 10 times as slow as local database access.

We now ask the question: "In a 2-node network, with a user logged on node 1 and a database resident on node 2, is it better for a program which accesses the database to be run on node 1 or on node 2?".

The knee-jerk answer is "node 2" so that the program avoids the performance degradation of remote database access at the expense of the performance degradation of remote user access.

Fortunately, in the vast majority of cases (99%?), this is the correct answer.

To see this, let X denote the number of prompt/response pairs for the average transaction and Y denote the number of IMAGE intrinsic calls per average transaction.

The question becomes: "Is it better to expend 240X milliseconds in support of remote user access or 120Y milliseconds in support of remote database access?".

From this we see that remote user access will be better than remote database access whenever 240X is less than 120Y.

Our conclusion is then valid for those cases in which Y is greater than 2X.

It happens that, for an R-remote user, the additional wall time per network transfer is approximately linear so that the wall time overhead for each prompt response pair is 240R milliseconds.

The same linearity happens in remote database access yielding a wall time overhead of 120R milliseconds for an R-remote database.

Because of this linearity, the question of locality for best performance when the user and database are on distinct nodes of any network has the same answer, regardless of the number of intermediate nodes.

SUMMARY

If, in spite of everything, you find it necessary to use remote database access, you can estimate transaction degradation time due to remote database access as follows:

1. Let M denote the number of IMAGE calls in the transaction (excluding DBINFO calls satisfiable locally).

2. Let N denote the distance, in nodes, between the database and the program (N=0 for local database access).

3. Then, the transaction degradation time is approximately M x N x 120 milliseconds.

As a final caveat, note that large internode distances, low baud rates, network contention and CPU contention at any node can all increase this degradation time.

THE SPIRIT OF A NEWER SOFTWARE : LL'SPIRIT

LIM LIAT
26 AYER RAJAH CRESCENT #05-01/10
AYER RAJAH INDUSTRIAL ESTATE
SINGAPORE 0513

ABSTRACT

The author shares the spirit of "Automation of System Development" in this paper. Automation should cover the entire life cycle -- from analysis, to design, to programming, to testing, to documentation and even to system changes and enhancements. The insights and principles of developing an automated system development tools will be discussed. These include data vs procedure, development process vs program product, prototyping vs classical life cycle, testing by inference vs blank-box testing, automatic documentation, technical auditing of programs by knowledge based system. LL''SPIRIT, the intelligent automation tools developed by the author would be used to illustrate these principles. It is hoped that the audience could catch the same spirit and contribute towards greater productivity in system development.

INTRODUCTION

In the past, data processing personnel have been accused of not applying the computer to its own development work, even though they are very succesful in computerising other professions. Such accusation is no longer valid as can be seen from the large offering of Fourth, Fifth and even Sixth Generation Languages and Application Generators. The author wishes to share his experience and spirit in this pursuit of the automation of development.

THE SPIRIT OF AUTOMATION

The most obvious benefit of automation is productivity. The other important benefit is reliability or quality. We are all aware that human being is creative but inconsistent. We cannot rely on the human being to meticulously and consistently apply all the knowledge he has into any development work. Someone even suggested that "A machine that depends on the reliability of human beings is the most unreliable machine" !

The solution is to automate. The greater the degree of automation, the greater the benefits. This can be achieved by tranferring the knowledge from human beings into a machine either in the form of an algorithm or a knowledge base i.e. software. The software can then be relied upon to apply the accumulated knowledge consistently and thoroughly to the development work.

The spirit of LL'SPIRIT is then to "maximize automation by transfering knowledge into software".

BENEFITS

There are many benefits in transferring knowledge into software. Firstly, we can test whether the knowledge is actually useful by applying it. Secondly, there is no need to teach the users about the knowledge; just teach them how to use the software. Thirdly, there is no need to force them to apply the knowledge. Whenever the software is used, the knowledge in the software is automatically used. Translated into the data processing worlds, it means there is no need for standard manuals; there is no worry whether the standards are adhered to; there is no need for conducting classes. Why ? The standards and the methodology are in the software.

SOME BASIC CONCEPTS

The LL'SPIRIT software is an extension or more correctly put, encapsulation of the author's knowledge about MIS development. It is built upon some basic concepts and principles in MIS that are discussed below.

PROCESS VS PRODUCT

Programming has two parts. Firstly there is the process of writing the program. Secondly, there is the end product - the program. The algorithm used in the program and the properties of the program product are the interests of the computer scientists. The process of developing the program is the concern of the engineers. How much resources is spent in developing the program ? How could we do it better the next time ? As data processing personnel we should be concerned with these development issues. Strangely though, this process that took most of the programmer's time, sweat and tears is often neglected. One common misunderstanding is that the development process ends with the product. This is incorrect. The development process continues after the development of the product and must continuously monitor and improve the product. A program has life --- pregnancy, birth, growth and death. A system of programs has life --- some come, some go and some change. LL'SPIRIT recognises this endless life cycle and attempts to provide automation for this entire unending cycle.

DATA VS PROCEDURE

Data should be separated from the procedure modifying it so that data could be shared and managed with its own right. Decisions are based upon the values of data and not the procedure. Procedure is changing data from one state to another. There are many ways to change data from one state to another. Hence, data tends to out live the procedure modifying

them --- data is more stable than procedure. An information system should therefore be data centered. Information system building should start with developing the entity relationship (data) model first. LL'SPIRIT supports this data-centered methodology.

PROTOTYPING VS CONVENTIONAL LIFE CYCLE

The conventional life cycle approach of phased development has problems. Firstly, it is based on upon invalid assumptions such as

- users know what they want at the start

- users requirements do not change; they only need to be determined

In actual practice, we know that users change their mind. Their requirements change as they get better understanding of their systems through interaction with the analysts. Mistakes on the users' side may cause changes. Implementation of the desired system also triggers changes. Change is only natural.

Secondly, there are different langauages used in the various phases. The analysts speak structured English and Bubbles (DFD), designers speak structured charts, blue-print, green-print and programmers speak COBOL. Besides the translation problem, the language itself especially the analysts's languages are usually not machine- verifiable and suffers from ambiguity, omissions or inconsistency. One major problem is the delayed detection of error. Errors made in one phase are not discovered until later phases and by then it is more costly to fix.

Prototyping on the other hand enables a system to be built in steps with each step closer to the optimum. Prototyping however does not mean no specification and no control. With proper planning and control, the author believes it is a better approach to building an MIS system. LL'SPIRIT supports this prototyping approach. A prototyping approach requires two important things

- quick development

- ease of change

Another important factor is that the prototyping language used should also be the production language as well. There should not be another conversion process to a production language for performance unless this conversion process is automatic. Hence a careful selection of tools is needed for prototyping.

TYPES OF ERRORS

There are two types of errors - specifications and implementations. Specification error is when the analyst misunderstood the requirement. Implementations error is when the program does not do what it is supposed to. Consider a simple case of a tax formula:

```
taxable = gross-profit * tax-rate
```

A program that does not compute the above formula correctly commits an implementation error. However, if the actual tax formula is a step-function rather than a simple rate, then the system suffers from a specification error. It is no fault of the programmer. Implementation errors could be eliminated with bug-free software. But specification errors could only be eliminated by the users or perhaps a knowledge based system who understands that particular business and know what are the right and wrong things to do.

BLACK BOX TEST VS TESTING BY INFERENCE

Using test data and comparing the test results with desired results is probably the most rudimentary and commonly used method of testing. There are a few problems with this technique. Firstly, it is tedious. Secondly, it may not be repeatable. Consider the testing of online programs. Most DP shops rely on the programmer or the analyst to supply test data online. These test data are not captured and are quickly forgotten. Problem arises when another programmer wants to make changes to the on-line program. He could not repricate the testing made by the earlier programmers. After a modification is made, no one is confident whether the changed program will work when moved back into production.

Exhaustive testing does not guarantee 100% bug-free programs as well.

The alternatives are

1. Automation of Online Testing - Replayable Testing.

   All the test data tried out should be captured and replayable anytime by the programmer. This will ensure that whatever new changes added to a program, it would at least go through all the previous tests and be assured that it could not be worse than before.

2. Testing by Inference or Deduction

   A program could be written to deduce the specifications from the implementations. This derived specifications could be validated against the original specifications for errors.

3. Knowledge based Software to pick up the errors.

TECHNICAL AUDITING OF PROGRAMS

All of us, over the years of practicing, has accumulated a wealth of knowledge on many things. What makes a good program. What programming constructs to avoid. What not to do in IMAGE etc... All these knowledge should be in a program so that it can be called upon when needed, to validate, diagnose and solve problems. LL'SPIRIT has such knowledge built into it. It has some IMAGE knowledge and knowledge of Powerhouse. We can then let the software do the mundane tasks for reading lots of codes and we go on to the more interesting tasks of discovering new knowledge.

DOCUMENTATION

Documentation should be part of the development work and should come as a by-product of the developement work rather than an extra and usually post-implementation burden (that is frequently left undone !) A lot of documentation have already been done during development. It is in your dictionary, in the program sources etc. It is only natural that we derive the documentation from the work that we have already done. LL'SPIRIT does that. It brings much benefits. Besides productivity, it ensures quality. Because everytime you duplicate and make changes, there is the burden of keeping every duplicate and variation up-to-date when you change the original.

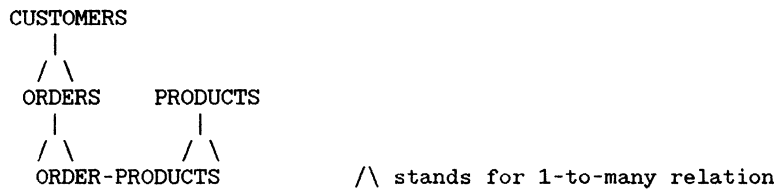
ILLUSTRATIONS OF ABOVE PRINCIPLES WITH LL'SPIRIT

Applying the most basic principle of software development "make use of existing software" or as James Martin puts it, "Standing on the shoulders of others", LL'SPIRIT is developed using Powerhouse and is designed for Powerhouse users, or users who are interested in getting the best development system.

DATA MODEL FIRST

The first step in any information system development is to derive the entity-relationship model or the data model.

Consider a simple order entry example below. The entities are CUSTOMERS, ORDERS and PRODUCTS. A CUSTOMER can have many ORDERS. Each ORDER can have many PRODUCTS. Each PRODUCT can belong to many ORDERS. The many-to-many relation between ORDERS and PRODUCTS can be resolved by introducing a relation ORDER-PRODUCTS such that an ORDER has many ORDER-PRODUCTS, an ORDER-PRODUCTS belong only to one ORDER. Similar relations exist between PRODUCTS and ORDER-PRODUCTS. The entity -relation model of the order entry system may be depicted as:



The data model corresponding to the entity model shown above can be coded as

CUSTOMERS[CUST-NO#,CUST-NAME,CUST-ADDR]

ORDERS[ORDER-NO#,CUST-NO,ORDER-DATE,ORDER-VALUE]

PRODUCTS[PROD-NO#,PROD-NAME,PRICE,QTY-ONHAND]

ORDER-PRODUCTS[ORDER-NO*,PROD-NO,QTY-ORDERED]

where # denotes unique key and * denotes duplicated key

VERIFYING THE DATA MODEL BY INFERENCE

LL'SPIRIT could verify the data model by deducing the following relationships amoung the entities.

File/Entity	Rel to	Entity
CUSTOMERS	1-<	ORDERS
ORDERS	1-<<	ORDER-PRODUCTS
PRODUCTS	1-<	ORDER-PRODUCTS
ORDER-PRODUCTS		

when 1-1 stands for one to one
1-< stands for weak one to many (not supported by key)
1-<< stands for strong one to many (supported by key)

You could see immediately that the deduced relationships is exactly what the user desired.

A SIDE TRACK - LL'SPIRIT DICTIONARY AND POWERHOUSE SCHEMA

The POWERHOUSE dictionary is rich in data model structures but is completely silent on procedures. LL'SPIRIT therefore has to have its own

dictionary. The LL'SPIRIT dictionary is a superset of POWERHOUSE QDD. Conversion from LL'SPIRIT dictionary to POWERHOUSE QDD schema is fully transparent both ways. Hence POWERHOUSE users need not be concerned about the duplicated dictionary. In fact, one should think of it as 2 versions of the dictionary.... the active QSCHEMAC and the enhanced LL'SPIRIT SPTDD dictionary where the user will be making changes.

AUTOMATIC DATA BASE DESIGN WITH IMAGE, KSAM KNOWLEDGE

LL'SPIRIT will load the data model into LL'SPIRIT dictionary. Element attributes like type and size can be specified upfront, or follow a element-usage name (eg NAME as X20, DATE as DATE etc.).

LL'SPIRIT by default will create MPE files for those entities without key and KSAM key files for those entities with keys. A KSAM file is chosen because logically, it is superior to IMAGE (KSAM can support compound keys and partial key access) and also restructuring is more convenient with KSAM (one-file at a time. Not everyone has ADAGER!). At the user's command, LL'SPIRIT will convert the KSAM files into the IMAGE data base. When LL'SPIRIT builds the IMAGE data base, it converts all entities into IMAGE detail datasets with IMAGE automatic masters created for keys. This is done because IMAGE detail datasets has the following advantages over manual masters:

1. Flexibility. New keys (paths) can be created anytime (with Adager) without affecting the existing program.

2. Performance. Serial read on a detail set ends with High-Water-Mark. Serial read on an IMAGE master physically reads the entire set! More key values could be packed in the master since only key values are kept for the automatic masters.

3. Record-level locking possible with detail dataset.

After the data model is loaded into the LL'SPIRIT dictionary, the user may change element attributes, etc through LL'SPIRIT screens written in QUICK. LL'SPIRIT provides a menu/form interface to the QDD schema. After changes, if any, the QDD schema can be created immediately.

BUILDING A COMPLETE HIGH QUALITY PROTOTYPE

LL'SPIRIT generates a complete set of QUICK source codes to maintain the entities defined in the data model.

LL'SPIRIT generates two kinds of QUICK programs. One is a menu type screen and the other is the entity screen. The menu type screen allows the operator to go to any of the entity programs for entry and

retrieval. For each entity, a program will be generated reflecting the data model.

1. The Menu Program:

The Menu Program is a QUICK menu screen allowing the users to go to any QUICK screen corresponding to the entity to add/get/update the entity. An example of the menu program is shown:

```
SCREEN ORD0001 MENU BLOCK
DEFINE SYS-TITLE STR*60 = CENTER(SYSNAME)
FIELD SYS-TITLE NOID NOLABEL DATA AT 3,10 PREDISPLAY
DRAW THICK 2,1 TO 4,80
SKIP 2
ALIGN (2,5,39) (41,44,79)
SUBSCREEN ORD0010  LABEL 'CUSTOMERS'
SUBSCREEN ORD0020  LABEL 'ORDERS'
SUBSCREEN ORD0030  LABEL 'PRODUCTS'
TITLE '-------------------------------------------------'
SUBSCREEN ORD0040  LABEL 'ORDER-PRODUCTS'
BUILD
```

```
MODE:x ACTION:xxxxxxxxx
*************************************************************************
*  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  *
*************************************************************************

 01 CUSTOMERS                              02 ORDERS
 03 PRODUCTS
 -------------------------------------------------
 04 ORDER-PRODUCTS
```

ORDER-PRODUCTS is separated out from the other entities like CUSTOMERS, ORDERS and PRODUCTS. This is because ORDER-PRODUCTS does not own any other entities and is owned by others.

2. The Entity Program:

This is the QUICK source program for add/get/update the entity. The generated program maintains the relations this entity has with the other entities. An example is shown:

The entity program to maintain ORDERS:

```
SCREEN ORD0020 BLOCK
FILE ORDERS PRI
FILE CUSTOMERS REF
FILE ORDER-PRODUCTS DELETE
```

```
TITLE 'O R D E R S ' AT 1,25
GENERATE
SUBSCREEN ORD0021 LABEL 'ORDER-PRODUCTS' MODE SAME AUTO  &
         PASSING ORDERS
BUILD
SCREEN ORD0021 BLOCK ON 25 &
      RECEIVING ORDERS
FILE ORDERS MASTER
FILE ORDER-PRODUCTS PRI
FILE PRODUCTS REF
TITLE 'O R D E R - P R O D U C T S ' AT 1,25
GENERATE
BUILD
```

The entity program for ORDERS shows how LL'SPIRIT honours the data model. Firstly, in the screen ORD0020, CUSTOMERS is declared as a REFERENCE file. This will ensure any value of CUSTOMER-NO entered must be an existing customer in the CUSTOMERS file preventing the capturing of an order belonging to a non-existent customer. Note also that ORDER-PRODUCTS is declared as a DELETE file thus ensuring every time an order is deleted, the corresponding records in the ORDER-PRODUCTS file will be deleted. This reflects the one-to-many relationship between ORDERS and ORDER-PRODUCTS. This one-to-many relationship is also reflected in the creation of a child-screen to allow addition of multiple ORDER-PRODUCTS records to an order.

Going to the screen ORD0021 for the ORDER-PRODUCTS, note the appearance of PRODUCTS as a REFERENCE file now. This will ensure that while entering the product for the order, the product must already exist in the PRODUCTS file.

This 'automatic construction' of programs from the data model will ensure that any entry, deletion, or update of data in the data base will be consistent with the data model.

AUTOMATIC DOCUMENTATION

QUICK documentor reads QUICK source codes and does the following:

- o Extracts the Files used in the program
- o Extracts the Screen to Screen relationships
- o Create a copy of the Screen Layout and put it in the group PHLAYOUT with the same screen name.
- o Extracts the descriptions of the screen and put in the group PHDESC with the same screen name.

- Extracts the help,descriptions and other attributes of the fields used in the screen from the dictionary and put in the group PHFIELD with the same screen name.

- Files in PHFIELD, PHLAYOUT, PHDESC will be used in automatically creating the end user documents.

QUIZ and QTP documentor reads QUIZ and QTP source codes respectively to extract the files used in the programs.

After the extraction, the information is loaded into the dictionary. This provides for Impact Analysis (which programs need to be re-compiled) when data base/file is changed.

LL'SPIRIT generates user documentation (show below) directly or the GALLEY (HP IUG Contributed text formatter like HP's TDP) command file for generating the documentation show below:

SAMPLE DOCUMENTATION FOR ORDER ENTRY SYSTEM

```
SCREEN :ORD0001

MODE:_ ACTION:_____
**********************************************************************
*                                                                    *
**********************************************************************

 01 CUSTOMERS                          02 ORDERS
 03 PRODUCTS
 ------------------------------------------------
 04 ORDER-PRODUCTS
```

The above is the menu screen. The _____ is where the schema title will be displayed.

```
SCREEN :ORD0010


MODE:_ ACTION:_____ C U S T O M E R S
01 CUST-NO       _____
02 CUST-NAME     ____________________
03 CUST-ADDR     ______________________________

<< if you have use DESC OF SCREEN in the QUICK source codes,
 then that descriptions will appear here >>
```

```
  ORD0010
  The screen processes CUSTOMERS.
          \--- this description is generated by LL'SPIRIT

Descriptions of Fields


  CUST-NO             X(8)         "C^^^^"

  CUST-NAME           X(20)

  CUST-ADDR           X(28)

The documentation for next screen is

SCREEN :ORD0020

MODE:_ ACTION:_________ O R D E R S
01 ORDER-NO          ________
02 CUST-NO           _________
03 ORDER-DATE        ________
04 ORDER-VALUE       ___________
05 SALEMAN-NO        ________
06 ORDER-PRODUCTS

  ORD0020
  The screen processes ORDERS.
  During processing, it refers to the following files:
       CUSTOMERS
  Upon deletion of ORDERS the following files will be deleted:
      ORDER-PRODUCTS

        \--- the above descriptions are generated by LL'SPIRIT

Descriptions of Fields

  ORDER-NO            X(8)

  CUST-NO             X(8)         "C^^^^"

  ORDER-DATE          DATE

  ORDER-VALUE         9(8)V9(2)

  SALEMAN-NO          X(8)
```

The documentation for screen ORD0021 is:

```
SCREEN :ORD0021


MODE:_ ACTION:_________ O R D E R - P R O D U C T S
01 PROD-NO      ________
02 QTY-ORDERED  ________


  ORD0021
  The screen processes ORDER-PRODUCTS.
  It is activated by receiving the following information from
  a higher level screen:
       ORDERS
  During processing, it refers to the following files:
       PRODUCTS


Descriptions of Fields

  PROD-NO           X(8)

  QTY-ORDERED       9(8)
```

This way of automatic documentation ensures your documents to be as up-to-date as the programs.

INCREMENTAL DEVELOPMENT - CONSOLIDATION WITH NEW MODEL

After some time, the designer may want to incorporate a new data model into the existing dictionary. Using another simplified example, a SALES ANALYSIS model as below:

```
SALEMEN [ SALEMAN-NO#,SALEMAN-NAME ]

ORDERS [ SALEMAN-NO*,ORDER-NO,ORDER-VALUE ]

COMMISSIONS [ SALEMAN-NO*,ORDER-NO,COMMISSION-VALUE ]
```

Note that this new model contains entities (eg ORDERS) that already exist. How will this impact the existing model ?

LL'SPIRIT allows the new model to be compared against the current model before consolidating it with the existing model. The following reports are produced by LL'SPIRIT:

```
File/Entity   Rel to  Entity

SALEMEN
              1-<<    ORDERS
              1-<<    COMMISSIONS

UNDEFINED ENTITIES REPORT
-------------------------

The following Entity/File are not defined

         SALEMEN
         COMMISSIONS

UNDEFINED ELEMENTS REPORT
-------------------------

The following Elements of Entity are not defined :

      ELEMENT      of    ENTITY

      COMMISSION-VALUE    COMMISSIONS
      SALEMAN-NAME        SALEMEN
      SALEMAN-NO          COMMISSIONS
                          ORDERS
                          SALEMEN
```

CONSOLIDATION REPORT

Note:
Items in data model but not in dictionary will be added
Item key type in dictionary will be set to values shown
Codes N: not in dictionary U: unique key D: duplicate key

Entity/File	In Dict	Item	In Dict	----Key Type---- Dict	Model	Final
COMMISSIONS	N	SALEMAN-NO	N		D	D
		ORDER-NO	N			
		COMMISSION-VALUE	N			
ORDERS	Y	SALEMAN-NO	N		D	D
		ORDER-NO	Y	U		U
		ORDER-VALUE	Y			
SALEMEN	N	SALEMAN-NO	N		U	U
		SALEMAN-NAME	N			

The consolidated model can be loaded into the dictionary.

In the above table, using ORDERS as for illustration. ORDERS is in the dictionary as shown by a 'Y' under the 'Dict' column. The SALEMAN-NO of ORDERS however is not as shown by 'N' in the Item-In-Dict column. The model asks for SALEMAN-NO to be a repeating key, the final consolidation will set it to be a repeating key as required. The item ORDER-NO of ORDERS exists in the dictionary and is a unique key. The model places no restriction on ORDER-NO so the consolidating process will preserve the original unique key.

If the model asks for repeating key when the dictionary has repeating key specification, then the consolidation will set it as repeating. However, it the model asks for unique key when the dictionary has it as repeating, then the consolidation will preserve the existing repeating key specification but print out "***" under the Notes column to highlight to the designer of possible conflicts --- the existing values in the data base may not satisfy the requirement of the new model.

If you have existing data in your data base, you would have to unload the data first before you change your QSCHEMAC. LL'SPIRIT knows which are the files that need to be changed due to the consolidation. It will prompt you whether you want to create the QTP source codes for unloading/loading the data in the affected files. If you answer 'Y' to the prompt, then LL'SPIRIT will create the file SPTUNLD for unloading data and SPTLOAD for loading back the data after you have changed your QSCHEMAC to the latest version.

TECHNICAL AUDITING - HOW GOOD ARE YOUR QUICK PROGRAMS

m692PIRIT's QUICK Auditor is an expert system with expert knowledge on quality and performance of QUICK programming. It is designed to read QUICK source programs and pointout the 'bad' codes and even suggesting better solutions.

Consider the following QUICK source codes:

```
PROCEDURE PROCESS QTY-ORD
BEGIN
   LET TOT-QTY = TOT-QTY + QTY-ORD
END
```

The every first time the operator enter say 10 for QTY-ORD, the TOT-QTY will be computed correctly as 10. However, if the operator backtracks to QTY-ORD and changed it to 20, the TOT-QTY will be accumulated to 30 rather than just 20. You can immediately see that so long as there is no correction of QTY-ORD, the TOT-QTY will be computed correctly (this is the sequence the programmer tested his program.) but it will be very wrong when the operator does back-tracking (which he sometimes does). Here we have a very difficult -to-reproduce dynamic error. There are many other similar types of 'potential' error codes in QUICK programs. (The correct coding in QUICK is to use the very powerful SUM INTO verb. This will be recommended by LL'SPIRIT.)

After analysing many installed QUICK programs, a set of knowledge on QUICK source codes that will cause dynamic error or performance problem has been accumulated. The knowledge is incorporated into the QUICK Auditor.

For example:

```
Enter QUICK Source >SPTQKAT3
Listing to terminal ?(Y/N)Y
Do you want auditing of source ?(Y/N)Y
Do you want file related statement ?(Y/N)Y
File ORDER-D
File ORDER-PART-D
File PART-M
File BACKORDER-D
 7      SELECT IF PART-NO OF BACKORDER-D = PART-NO OF ORDER-PART-D
  *P* Long chain read may give poor performance.
   \--- LL'SPIRIT points out potential performance problem
Procedure ENTRY
GET             PARM-M
 27       LET PARM-COUNT = PARM-COUNT +1
```

```
  *W* Recursive definition!
    \-  LL'SPIRIT detects potential error
PUT             PARM-M
END. 1 Screen only CPU:            1.849
```

ANOTHER EXAMPLE - INTELLIGENT QUIZ

LL'SPIRIT's Intelligent QUIZ provides 3 levels of interaction with the users to create his reports. The simplest level is a guided mode which prompts the user for his files through a menu, elements, sort etc. The menu-driven mode is for the more experienced user and a command mode for the expert QUIZ user. When a user saved his source codes, only the latest QUIZ codes are saved. IQUIZ also tries to help the user in programming QUIZ as well. For example, it will produce REPORT < > PRINT AT <> when the item is sorted.

CONCLUSIONS

I have shown how our knowledge could be transfered to a software to provide increased productivity and quality. The important concepts are:

1. The entire process of development needs to be automated

2. Use data centered approach for developing MIS system

3. Use prototyping approach in favour of conventional life cycle

4. Use automated testing tools

5. Use testing by inference

6. Accumulate your knowledge in software to auditing

7. Automate your documentation

8. Make use of others' work and stand on their shoulders

I hope you can catch the same spirit of automation and transfer your knowledge into a software and gain the productivity and quality not only in your work but in your life.

THE TOOLS OF STRUCTURED ANALYSIS: A TUTORIAL

MARK WALLACE
ROBINSON, WALLACE & COMPANY
11693 SAN VICENTE BOULEVARD
SUITE 168
LOS ANGELES, CA 90049

The specific tutorial that I'm going to give is drawn from a class that I teach for Robinson, Wallace and Company. It's our introductory class in Structured Analysis. It's licensed from Palmer Consulting and these people are among the originators of Structured Analysis who brought these techniques to the world of data processing some 10 years ago. Again, originally in a mainframe type environment.

The tools are used to construct a model of a system. That model of a system then will be reviewed with the users and with the implementers. The person who builds the model is called a systems analyst. A given employee may not have a hat that says "I am 100% a systems analyst and nothing else." You may have in your shop what we call programmer analysts. Those people may do coding and design and testing as well as going out to the user and finding out what they want. But, when they're talking to the user they're doing the job of a systems analyst.

They are a middleman between the user and between the people who are going to construct the application. In a previous talk on strategy, we had a couple of comments to the effect that people have actually built models using Structured Analysis and taken them back to the users or handed them over to the users and the users didn't know what to make of them. People sitting in the audience during that talk said, "This is a real problem. We built these models, we gave them to the users and they didn't know what to do. I taught this very class a couple of months ago and one of my students said, "In our company we built a structured specification. We had a lot of meetings with the users, gathered information, built a 200 page specification, handed it back to them and they didn't know what to do with it.

Well, if you're going to adopt this methodology, you've got to train several classes of people in dealing with it. It's not something you can pick up by the seat of your pants too easily. The analyst is a middleman between the user and the developer or implementor. All three groups need to be conversant with these tools. The end users need to be conversant with the tools. The analysts are obviously the focal point. They're the ones that will be building the models. They need to be conversant. The implementors need to be conversant because they will be reading the specifications to decide how to write the program.

People that were complaining earlier that their users couldn't work with these tools had never given their users any training. You wouldn't expect them to be able to work with it if they had not been trained in how to do it. The idea that this is not just something for DP technicians is embodied in a class that's taught by a company called Yourdon out of New York City. They have a formal three day class called "Structured Analysis for Users". No programmers allowed. It teaches people on one end of the communications spectrum how to communicate with the people in the middle. There are other classes that teach the designers and implementors how to communicate with the analysts as well. You can't just take these techniques and start running wild without laying the groundwork among everybody who's going to have to deal with them.

Let's talk a little bit now about what's involved in building one of these Structured Analysis models. First of all, the idea is that in an industrial stength system, there's a lot of detail to worry about. Structured Analysis is a way of coming to grips with that detail. We're going to build the specification that captures the detail and then from the specification we're going to build the system.

This is similar to an activity whereby, let's say a stereo receiver is built according to a specification. If you wanted an electrical engineer to build a receiver for you, I think you'd be pretty shocked if on day one without any kind of circuit diagrams they brought out their soldering iron and started wiring resistors and transistors together. That just wouldn't be appropriate.

Not only that, but the specification itself can be reviewed by product managers, in this case in order to verify that a product built to that specification will do what we want the product to do. In that sense, the specification itself serves as a kind of prototype. This is a very important idea. People are saying that you should prototype. We agree with that, but our prototypes take a very different form from the prototypes that 4GL vendors suggest that you build. Their prototypes are running screens and reports.

For industrial strength application that is not a good way to discover requirements. I can give you war stories for the next week that demonstrate that that's not a good way to do it. With tinkertoy application it's fine. For an industrial strength application we want to build a different kind of prototype using the tools of Structured Analysis.

Our strategy is going to be construction by specification. At the start, we'll gather up some information about the product. We'll use that information to specify the requirements that the product has to meet. That specification will be reviewed by the users on the one hand to verify that it meets their needs.

And by the implementors on the other hand to verify that it is something that is feasible to do with the technology that we have today. Because we could specify online audio response for a thousand conversations simultaneously with a 250 word vocabulary. It can't be done, because there's no computer that can do that today. So we need verification of this. The analyst has to get verification from both ends. The users, does it do what I want it to do? The implementors, can it in fact be built? Once it's signed off we will construct the product using the specification and the result is the product that we deliver.

This process of constructing a specification involves building a model or, if you want to call it that, a paper prototype of the running system. That model can take many different forms. You could build a model of one floor of a house. Okay. It's got rooms, dimensions, it shows doorways, windows, and so on. It does not show what color the walls are going to be painted. A model leaves out some details, otherwise it would be the whole system.

Here's a model of a requirement for a software application done on a PC. (At the end of this paper I'll tell you about PC-based support for Structured Analysis that eliminates the need for a lot of pencil and paper manipulation of these diagrams.) The way that you choose a model is a function of what your needs are. You want something that will be useful in communicating to the developers the way that the product needs to be built. And that is a function of what kind of a product it is that they're building.

So, let's take a look at what kind of a product we're going to ask our developers to build. We're going to ask them to build an interactive system. By interactive I mean that the system interacts with its environment. The system will be composed typically of both human and automated processors.

A common mistake that systems analysts make is to ignore the manual part. A good way to give the project two strikes before it's even delivered. There are real world people out there who are going to have to use your system, they're going to read the reports, they're going to have to do data entry at your screen, they're going to have to take your output and do further things with it before the mission of the company or organization is completed.

A payroll system that stops when the checks come off the line printer would leave a lot of employees unhappy. That's not enough, we need to get the checks into the hands of the people who have earned their pay for that week. That is typically done manually. I'm going to suggest that if you are specifying a payroll system, the analyst needs to consider the manual parts as well as the automated parts. A "payroll system" has not completed its job until the finished output, in this case the pay check, is in the hands of the end user, in this case the employee

So, consider both the automated and the manual parts. What we need the system to do is interact with its environment. It will receive stimuli and in response to a given stimulus it will generate some response. The two kinds of response that a system generates are either planned response or what we call an ad hoc response. That's a seat of the pants type response for which no rule can be given.

I'll give you an example of that. Suppose we need to select a vendor to fill an order for chairs. We need 100 more chairs. Lots of companies supply chairs. How do we pick a vendor? What might we look at in deciding which vendor we're going to get? What are some factors that we might look at? Cost. How much do they charge for the chairs? That's only one very important factor. Another factor might be what? Delivery time. How long is it going to take them to get them to us? If we have an urgent need we might pay a little more rather than wait for somebody who's going to take six weeks.

So there are many factors in a complicated procurement. The purchasing agent might have to consider things like, has this vendor got a labor contract that's up for renewal? Could they be subject to a strike that could prevent them from delivering an order? How far away are they as a factor that affects how long it's going to take? The trade-off or the procedure involved in making that judgment, is not something that could be programmed.

There's no way you're going to program whether another $3.45 a chair trades off against a one week longer delivery time. That's something that has to be weighed using a lot of judgmental factors. And therefore the function of choosing who the actual supplier is going to be, I call an ad hoc response. That is something the analyst should not waste any time on because you're never going to be able to lay out policy rules for that.

Now, the planned response system may assist that effort by delivering up from its data base a history of how long it has taken the vendor to deliver in the past, what they're charging for the chair, where they're located and so on. In other words, facts can be delivered in an information retrieval mode but the actual judgment will be an ad hoc response. So, what the analyst is going to do is break apart the system into a planned response half and an ad hoc response half, and the ad hoc response half will be factored out and will be considered a user.

In this case, the purchasing agent within the purchasing system will partly be a user of that system and will partly be inside it. That one person. Part of their duties will be outside becaues they'll be ad hoc, another part of their duties may very well be inside the system because they'll be subject to rigid policy that can be laid out in a policy manual. The ad hoc area would submit a request, "give me a list of all the vendors who sell chairs of this kind" and back from the planned response system comes "here's a vendor, here's their on time record, here's what they charge and so on".

So, we're going to break the system down into those pieces and we are going to study the planned response part of it. What we are responsible for is identifying the interactions between the system and its environment. Here's an example of a tinkertoy size system. This one is a wakeup system for a hotel. There are only two transactions. One is when the guests submits a wakeup call request. Our response to that is to record that request in the schedule. The other transaction takes place when it's time to wake up the guest. We tell the automatic activity to wake up the guest, which involves issuing that wakeup call.

The call request coming in and then the next morning the call going back out are interactions between the system and its environment. All of those need to be identified. Then we have to identify what are the responses. When the request comes in, what is it the job of the system to do? When it's 8 o'clock tomorrow morning, and that's when they wanted to be awakened, what is it the job of the system to do? Those are the planned response activities.

Then we have to identify the interactions between those activities and the essential memory of the system. The essential memory consists of those data elements that need to be stored regardless of the data storage technology. So we're not going to count in essential memory such things like redundant keys or IMAGE pointer chains or so on. Just the basic data elements. Now, that's the essential view.

The way that the analyst will typically find the system is looking something like this. Here's an implementation of that essential system. We find that the activities are carried out by real life people and machines. So the request comes in and it goes to the evening operator because we don't have any machines that are good at carrying out a conversation of this nature. The operator submits a scheduling transaction to the mainframe schedule package and files a hard copy of the request in a paper tray.

The next morning, the morning operator comes in, puts a request to the mainframe, gets the schedule listing, sends an auto call list to the easy up automatic operator and that's a machine that places calls. I don't know if any of you have stayed in a hotel or motel where they have that, but in the last year or two they've got the machine that actually makes the call. When you pick up the phone there's either a recording that says, "Good morning it's 7 o'clock" or just a beep tone. I've heard both of those. Or a third variety is a buzzer that lights up on your wall that you have to go shut off. Science marches on, right?

The automatic machine comes back with a confirmation list to the operator, and then anyone who has not been reached by the machine, the operator can call manually. Now that is an implementation, a specific implementation which depends on specific technology. There could be a lot of other ways to implement that wakeup system.

It could be done without any machines at all. Even without telephones. You could come in that night, leave a note at the front desk and then the next morning they'd send out the bell captain to knock on your door. Completely manual system. The next step up is the telephone technology. The next step would be something like this. Probably you could conceive of a fully automated one. You know, where you punched in on a numeric pad the time that you wanted to be awakened and that was stored and so on. So there are many different, shall we say, ways to skin a cat.

We have the essential requirement previously. Now we have a structured analysis model of solution, a technical solution. What you can do with this kind of a model is consider it as a prototype. Once you and your user have been trained in how to read these and you as an analyst have been trained in how to write them, you can sit down and track the transactions through this model.

(These models can be built on PCs now and that's something I'll talk about at the end of this paper.) You can track the transaction through the model. What you're concerned about is, "Do we have adequate information to generate the results that we need?" That can be verified from these data flow diagrams. We'll come back to that in a minute.

Let me go on now to show you a specification for our wakeup service. I want to repeat our claim that the structured specification is so effective at modeling the requirements that in fact it serves as a prototype on paper or on a PC. And I'm going to submit that for an industrial strength system it is a lot easier to change the spec than it would be to rebuild your 4GL prototype even with the fastest and most productive 4GL. For an industrial strength system.

Let's take a look at one. We start off with what we call a context diagram, the context diagram shows the user. In this case the user is the guest. It shows the system as one activity in the middle. This activity is called the wakeup service. And it shows the interaction between the users and the system. In this case the user submits a wakeup request and the same user later on receives a wakeup call. It wouldn't have to be just one user, you could have several.

Southern California Edison, which has been a pioneer in this kind of modeling for defining user requirements, has context diagrams with 40 different users around the system. And that's just for one system within their application. They definitely build industrial strength applications. Now, these pieces of information coming in and going out are called data flows. That's why this picture is known as a data flow diagram.

Those data flows are defined in a separate document called data dictionary. The data dictionary specifies for each flow the elements that make up that flow. Wakeup call is defined as a good morning message and the time of the call. The wakeup call request is composed of the guest name and the room number and the wakeup time.

The point is that the diagram serves as a picture, takes advantage of the graphic perception capabilities of the human mind. A picture is worth a thousand words. But if we want to get the detail we need someplace else to turn to get that detail, and that is the data dictionary.

Now, within this system, the wakeup service, there could be a lot of different activities going on. Imagine that it's a manufacturing system instead of a wakeup service. There could be dozens or even hundreds of activities. So, the way that we model those is with a zoom in feature. We zoom inside this overall activity which accounts for everything in the system (and remember, both manual and automated).

When we zoom in we create a model of what's going on inside here and that's called a lower level data flow diagram or a zoom in data flow diagram, or a child diagram is another phrase that's used. Here, in this case we see the two separate activities that make up this system. The one that records the wakeup request and the one that actually issues the wakeup call.

Notice that inside the activity we see some internal storage in the form of the wakeup schedule. As we zoom in more details of the internals of the system will be revealed. The user doesn't care about that internal source. All they care about is that we accept the input that they're sending us and that we generate the results. But the implementors are going to care. Because it's their model of how to implement the policy that's required.

These activities in turn could be further broken down if they were industrial strength themselves. We're going to take each of these activities, zoom in on it and break it down to a lower level diagram. The wakeup guest might go down to a lower level data flow diagram that's got your wakeup schedule, reads from that, has a couple of activities, finally issues the wakeup call. So there's an inside picture of what goes on inside here.

We can keep doing that. Our strategy is to keep doing that until we get down to a level where these activities are tinkertoys. And once they're tinkertoy we stop the zooming in and leveling down process. At that time what do we do? We take that tinkertoy activity or rather each of them, and we write a small specification or miniature specification or mini-spec for each of those tinkertoy activities.

If you leave something out of your requirements model by mistake, you're going to perpetuate the way that component does its work, even if that's 1950's technology or policy that could be improved. So you don't want to leave things out if they belong in. On the other hand, if you bring things in that you in fact have no control over, the cost is time. You're going to waste your time studying their activities in detail when in reality all you have from them is a given input or given output or both. It's a judgment that has to be very carefully weighed.

Once you make that judgment, you can decide which parts go in the system and which parts go in the environment. And that in turn will determine the interaction. Here we have a system tracking data, in this case from customer service about a new customer and from the metering service department about a new meter that's been installed.

The data that are coming in reflect actual things or persons out there in the environment of the system. And you want to identify what those things are because, typically, that will influence what your data storage structures are going to be. In this case we wouldn't be surprised to learn that inside the system there was a customer file or customer data set.

We need to identify the interactions between the system and its environment and we want to focus on the essential interactions. Those are interactions that are not dependent on technology. Let me give you an example. I read in Computer World a few months ago that IBM was shutting down its last plant that manufactures 80-column cards. This in 1985. That doesn't mean that you can't get 80-column cards anymore, that just means that there's not enough business for IBM to keep a plant operation.

Why is that so many people still use 80-column cards? Because even though they're on a 4381 or the latest IBM hardware technology, they never took the step of focusing on the essential requirements of the system. Whenever they got new hardware, they just made a simple A to B conversion and kept doing things the old way on a faster CPU.

We could fall into the same trap on the 3000. Just keep upgrading your CPU to faster and faster models, maybe take some batch things and put them on-line, but never really study the functional requirements of the system to determine what really has to be done versus what was done simply as an accident of the fact that we started out on an IBM System/3 before we bought first 3000. We perpetuate the RPG II approach of the IBM System/3 10 years later on a Series 68.

I'll bet you there are a lot of places where that's exactly what they're doing. They've never stripped out the effects of old technology. So one thing +that we study is what are the essential interactions, devoid of the implementation technology. And again, we're looking at the system as interacting with systems within the company, outside of the company, and throughtout the whole world.

One of these large activities might be a company with systems in it, those interact with systems in other companies and so on. That's why it's important to draw all your context boundaries correctly. Now, our job as analysts will be to identify the interaction of the system with the outside world, and once we are sure that we have done that correctly, we're there. We've got our requirements.

If you think about it, what are the requirements for a system? To accept certain inputs and generate certain outputs; even the inputs are only the means to the end. The fundamental requirement is always to generate a certain set of outputs to the users.

Therefore, once we've identified them we're home free. But, it's not as simple as that because again in an industrial strength application being sure that we've identified all the outputs and the data elements that make them up is a nontrivial task in itself. We need to ensure that we're obeying a couple of laws. John Palmer's law of connectivity says that it takes data to make data. You cannot generate output where there has not been input to account for it.

The corollary of completeness from Chris Gane, namely "what goes out must have come in at some prior time". Here's an example where we're preparing our Form 1040. What goes in is a W-2, deductible expenses, and a 1099 for any interest or dividends that we earned. Maybe that's not enough.

Suppose we received a refund last year and we itemize. We have to report that refund. The analyst's job is be sure that we have coming in all the information we need to generate the output, number one, and number two that the steps that are applied in calculating the outputs are in fact in accordance with the policy rules and regulations that management wants to apply.

The thing is that simply from the context diagram where the entire activity of the system is one bubble, we don't have enough information to do that. That's why we have to look at the more detailed picture. Including dropping down to lower levels and breaking down the system into separate activities. We have to look at the interactions between the processors and the system.

Here's an example where we have a mileage tracking system, Frequent Flyer type of a thing. When they put in a trip card, we keep a record of that. It goes to computer services, and it's stored in there. Every month we send out a mailing saying what award level you earned. If somebody wants to claim one, they send in a claim, that goes to data entry where it's prepared and that would have caused an award certicate to be issued and so on and so forth.

As an analyst we have the job of studying the interactions between the processors and the system. These processors are all involved cooperatively, like passenger operations (that's human), data entry (that's part human and part machine), computer services (that's also part human and part machine). All these processes are involved in the overall system which is the Frequent Flyer system.

We need to study those different activities, and we're going to do that with the aid of this kind of a model. This is a data flow diagram. Users, once they've been trained in these, can look at them and spot parts that are not correct. Passenger operations sends the application packet over here and not straight down there and so on. These models can be reviewed for correctness.

There are lots of shops where that's exactly the way they do it with their industrial strength applications. There are a few shops where the end users actually construct these models themselves. Those are in the minority. But you want to get your users involved at least as participants with the analyst in the building of these models.

You study the interactions between the processors. We might take say processor one which is computer services, again zoom inside that and look at the interactions within that processor. Here we see things like a front end conversion, a file maintenance activity, an evening shift operator, a month end report run and so on. All these are activities within the overall processor called computer services.

Then we might take a single activity like file maintenance and zoom in on that and break that down. We might find that it's composed of a master file update, a mileage assignment activity whereby for given trip we look up from point A to point B how many miles are involved and that's what they get, and whether there's any incentive bonuses for flying on that route this month and so on, a sort, and so on.

So, we break things down into a lower level of detail and we study within those activities the data flows between them, like from one activity to the next. Each data flow represents typically a form or some sort of an automated storage media, like a tape or a floppy or so on, or a printed report. It's a way that information is transferred from one activity to the next.

Typically the system will have to preserve that information over time. That gives rise to data stores. When the wakeup call comes in, we can't just turn around and call the guest. We have to wait until tomorrow morning. Well, tomorrow morning at 8 o'clock the guest is not going to call us and say, "Hey it's time to wake me up." We're going to have to remember that. That's called essential memory. It's something we have to remember no matter what kind of technology we use. That's represented as one or more data storers within the system.

So, we're going to break down the system into the processors involved, the interfaces among those processors, and connect them up according to what information they pass from one to another. So here with our wakeup call system we identify the processors within the system, the evening operator, the wakeup machine, the mainframe, and so on. We build a small model of each processor showing what it gets in and what it generates and then we hook them all up and come up with the overall model. And that's our physical or implementation model of the system.

Once we've done that, we still need to verify that these data flows in fact contain the elements necessary for the processors to do their job. And that involves us with the data dictionary. The data dictionary defines the data flows and also the data stores to be sure that we're storing enough information. The data dictionary helps us verify the system is generating correct output.

The wrong way to do it would be to lay these definitions right on the data flow diagram. If you did that, in fact in early techniques about 10 to 15 years ago they took this approach. Things just got too cluttered, number one, and number two, since the same data flow can appear on multiple diagrams, when you zoom in or when you pass from one to the other, you'd have redundant definition. The same flow would be defined more than once. And one of the goals of this entire approach is to make the requirements model maintainable by minimizing redundancy. So we don't want to do that.

We go back to the fact that the diagram declares data flows and data stores. Those were made up of data elements. Those data elements along with the flows and stores are defined in the data dictionary. The data dictionary definition as implemented either the way we teach it or on the PC uses some notational symbols that I'll outline for you very briefly.

We had something like, I'll give you an example, order header. We use the equal sign to mean "it's composed of", and what might be in an order header. There might be an order number and we use the plus sign to indicate another element. There might be a customer ID of the customer who placed that order. And a date that the order was placed.

Maybe we have a sales rep code so we use parentheses to indicate optional. But maybe this is a house account so there is no sales representative on this particular account. Parentheses indicate that optionally that data flow could contain a sales rep code, but it might be there or it might not. The symbols we use: = means is composed of, + means and as in this element and that element and so on, () indicate optional.

[] indicate choose one of. I can give you an example of that. Payment type is composed of and then choose one of cash or either (vertical stroke to separate the alternatives), check or credit card. So that's a choose one of. And the other one is the braces. {} means repetition. A sales history might be a repetition of the monthly sales figures for 12 months. And then the other convention is that asterisks in these definitions set off comments.

This is implemented on a PC and again I'll have information about that later. Now once we have verified that the elements are adequate to generate the output by using the data dictionary and the data flow diagrams, the next step we need to verify is that the calculation has been identified correctly. Yes, we've got the data that we need, but how do we know whether we're adding it instead of subtracting it.

That is the job of the minispec. So with the definitions we can prove connectivity as long as the processing is consistent with the definition. In other words, if we have gross pay going in, deductions going into an activity, we have net pay going out, we want to be sure that inside that bubble we're subtracting deductions from gross pay and not adding. The final step is the process description.

That's a simple example but you can imagine the policy would be much more involved and need to be specified. Again, the long solution to describing the process is to take a tinkertoy activity and break it down like this. Here is our wakeup call request, verify the request, find the time on the schedule, record the mass wakeup request on the schedule.

That would be more of a flow chart than a data flow diagram. It's too procedural. It's a waste of the graphics tool. You're going to have too many data flow diagrams floating around if you take it down to that level of detail. Also, all of the internal flows would have to be defined in the dictionary. So we want to get away from breaking down tinnkertoy activities. Instead, let's specify them with an English language description or what we call a minispec.

A minispec can be done with a decision table or decision tree. If it's heavily decision-oriented or if it's more calculation-oriented your fall back position is what we call structured English. Structured English vocabulary comes from the data dictionary, the structure of it comes from repetition, decision making and no control at all which is simply to go from one step to the next.

Repetition would be for each certificate level, set the number in circulation to zero or whatever. For each call that has to be issued at this time of the morning, make the call. A decision is if the transaction is a trip do something else and so on. They're similar to the control structures in the structured programming.

Those are the basic tools that are involved in Structured Analysis. The way that they get used, I discussed in my paper on strategy. To summarize, we start off with an activity called the blitz. The blitz is a fast, high-level look at the requirements, using some of these tools but not in full detail. The purpose of blitz is to guide project management as to how long it's going to take to do the rest of the steps.

After the blitz we can specify requirements, that's what I have just talked about. We can also do a data or information modeling activity where we would fit the entities or data sets for this application into the overall company information model. We take those stored data requirements and merge those and then we can go to the next step where we can deliver a running prototype with a 4th generation language for example.

But first we built our paper or PC-based prototype using the tools of Structured Analysis and data modeling.

There is a reference for the ideas that I've talked about this morning other than taking our three day class. If you want the book from which all of this came, it's called "Structured Analysis and System Specification," by Tom DeMarco, who is the father of most of the ideas in Structured Analysis. It was published originally by Yourdon Press in 1978. Yourdon Press has just been bought out by Prentice-Hall. You may find the book easier to obtain from Dorset House publishers in New York City. You can call them at 1-800-DH-BOOKS.

Building the data flow diagrams and the data dictionary entries is an activity that has been automated and can be done on a PC. It can be done on an IBM PC with suitable graphics or on a Vectra with suitable graphics. We're not sure whether the Vectra graphics board will work. The program is spec'ed for the Hercules card, which gives you the best resolution. It's supposed to support the color graphics card, which the Vectra graphics board can emulate.

Unfortunately, there is no implementation for the 150 so you're either in a IBM PC or clone-type ballgame. The Vectra is a clone. I've got a description that you're welcome to reference of one program which is a $1000 program that allows you to do Structured Analysis on a PC. Other programs are available that range up to $10,000 first copy.

Let me now answer some questions that previous attendees of this talk have asked. The question is, am I familiar with a book by Gane and Sarson? Yes, I am. Gane and Sarson were also associated with Yourdon in the early '70s and they produced a book that came out in 1977. Their company was purchased by McDonnell Douglas Automation, became the basis for McAuto's strata approach. I have a couple of problems with it. I think Tom learned from some of the problems that people had with the Gane and Sarson approach and he improved on it in his book.

One thing they don't do is level their data flow diagram. If you've got 50 activities in your system, you're going to have an enormous data flow diagram with 50 activities on it. Whereas with Tom's you'll have 5 or 6 high level activities and you can zoom into each one. So that's the major distinction where I have a clear preference for DeMarco. Philosophically they come from very similar roots. Their book is a useful reference.

The next question is when you're decomposing your data flow diagram by taking a bubble and breaking it down to the lower level diagram which itself will be made up of bubbles and so on and then you break those down, how do you know when to stop? The answer is, you stop when a bubble or activity can be specified in a page or less of minispecification. If you start to write the minispec and it's more than a page, then the activity's too big and you need to break it down to a child diagram.

The next question is, are systems that we're asked to build either industrial strength or tinkertoy or could there not be a middle ground? I think the answer to that is there could. There are going to be cases where as prototyping technology improves, the size of the system you could handle by directly building and then knocking it down if it doesn't meet the user's needs is going to increase. So first of all the boundaries shift over time with software development technology.

Secondly, yes there are areas that are not AC Sparkplug with an 80,000 line schema store that you still would want to build this kind of model on. Absolutely. The smaller the system, the easier it's going to be to build the model. And if you've got a PC-based tool, you can knock out one of these for a small to medium sized system in a couple of days. And it's a very nice thing to do if, for nothing else, training the analyst and the users before you have to take on one that's industrial strength, and to shake down your use of the techniques.

CREATIVE SYSTEM INTEGRATION TO ENHANCE PRODUCTIVITY

JILL C. RUTHERFORD
THE BOEING COMPANY
P.O. BOX 24346 M/S 6K-35
SEATTLE WA U.S.A 98124-0346

INTRODUCTION

In our mad dash to select the hottest, state-of-the-art software and hardware packages from the myriad of choices, we sometimes lose sight of the main objective. Development costs must be justified to management today, and the resulting software must continue to be efficient and productive years into the future.

So how do you keep your development costs down and still remain competitive? Most companies can't afford to constantly replace existing software or its supporting hardware.

In the Finance Department of Boeing Aerospace Company's Facilities Division, we're making creative choices to enhance our existing mainframe HP-3000 system without massive hardware replacement. And we're keeping costs down!

Through the integration of our HP-3000 environment and personal computers (PC) we are achieving our goal. Combinations of HP-3000 and PC software tools, as well as administrative, technical and organizational plans make it possible to combine these once separate entities. The result is a significant increase in productivity that is greater than either HP-3000 or PC-based applications alone.

Adapting our organization to the best available technology is an on-going evolutionary process that has led us down a path pitted with difficult decisions and frustrating mistakes. This is part of a necessary learning process that should eventually pay off in the form of a smoothly run, highly productive organization.

Ways to achieve optimum office automation include micro-to-mainframe links, networking with supporting software, multiuser systems, optimizing data storage and backup procedures, and new management ideas. The following subsections discuss these five categories in detail.

MICRO-TO-MAINFRAME LINKS

Micro-to-mainframe links are an attractive method of increasing productivity from the available system hardware in our office. By letting microcomputers, such as the popular IBM PC/XT, act as gateways to our HP-3000 mainframe, the PC's have become resourceful tools in our pursuit to reduce hardware costs and applications backlog. By using the right software tools and by applying a system life-cycle methodology, the PC can be a cost-effective alternative to applications development on the mainframe.

Your decision to use this method may depend on how much hardware your office has already acquired and how many workstations are needed. All links give users mainframe data but to varying degrees, generally reflected by cost differences. Micro-to-mainframe technology is constantly evolving. Simple connections can only look at mainframe data, while sophisticated links extract data from the mainframe to be downloaded into personal computer applications. This direct link saves time in retyping and gives access to current information.

Sophisticated software links provide the widest access to mainframe data. They permit data to become incorporated into micro applications, generally by sorting through mainframe data to download selected portions. There are open-ended links that can request data from several mainframe databases and conversely, download into several micro file formats. Links may be open or closed on either the micro or mainframe side. A few are open at both ends. A link's effectiveness is measured by its ability to download only the bits of data that are needed, and its capacity to carry data. Security is also provided for, and with the growing concern over the integrity of the central files, this will protect them from unauthorized access.

Reflection 3 is an example of a sophisticated software product our office is currently using. It enables an IBM PC to emulate the Hewlett-Packard 2392A terminal, giving it the ability to transfer data between the PC and our host HP computer. Reflection 3 can send data from the display memory of the PC to our mainframe peripheral printer, disk file or HP-LaserJet. A complete subsystem for transferring files between the PC and an HP-3000 is included.

Reflection 3 has a command mode with its own language, used for timesaving data communication operations. This can also be used in writing programs to provide faster execution of commonly performed operations. Reflection 3 can operate in remote mode, through a modem or a direct connection, or a local mode which emulates the keyboard entry and editing features of the HP 2392A terminal. When you exit, it leaves the PC logged on to the host computer. If the connection to the host was active, it also erases display memory and returns to DOS. Using Reflection 3 with your host HP computer is the best way to derive the full potential in productivity and cost benefits.

For micro-to-mainframe links the coming improvements are: greater data security, hardware adaptability and access, and a user interface that can be understood by the nontechnical employees. Software may soon have the capability of looking for data without user intervention. These developments should produce more micro applications with built-in intelligence that will open up systems, even to the most devout of your pencil-pushing, paper-shuffling employees.

In this evolutionary process some vendors are developing new kinds of links concerned with synergistic developments on the PC. This concept will be an important tool in applications development and office automation. Due to its early stage in development, the software tools available today have not solved all of the problems that arise from

integrating the micro into the mainframe activities. Problems include undocumented bugs, program limitations and slow performance.

The synergistic PC data entry applications software work on the concept of cooperative processing. These application programs run on both the micro and the mainframe, sharing functions but not processing power. Mainframe database files can be downloaded and worked on at the micro, while the connection to the mainframe can be broken. Since this reduces connect time it can reduce costs. When the local processing is completed the connection can be reestablished and work uploaded to the mainframe. The theory is to reduce the load on the mainframe.

The Synergist is a development of Gateway Systems Corporation. This software is an example of cooperative processing. Our office, in conjunction with Boeing Computer Services, is currently evaluating the possible use of this early fifth-generation product. The Synergist was designed for the development and maintenance of on-line applications using a PC (IBM PC/XT/AT, HP 150, Vectra, AT compatible) to do the work, while the database is stored and maintained on the HP-3000.

This product is intended to support programmer productivity as well as hardware efficiency and end-user interaction. Productivity is increased by using this simplified, nonprocedural type of programming to quickly develop applications, even by a "nonprogrammer". Complex routines may be selected, not coded through the product's expert-like features.

Hardware efficiency increases by integrating the PC with the HP-3000 through this "participatory processing" application development. Compiling, testing and maintenance can be done with the processing power of the PC.

To the end user this synergistic approach provides improved response time and customized features that reduce key strokes and automatically checks for input errors.

This concept in software is an idea our office is ready for, however we remain cautious in all of our product evaluations, to ensure that the benefits will exceed any software limitations.

Becoming available are is new class of micro-to-mainframe software (CMS, CICS AND VM/PC), promising to have a simpler user interface and powerful enough to compete with the traditional, mainframe-specific terminals.

NETWORKING

Linking PC's to the mainframe is a crucial, ground breaking event essential towards accomplishing total system integration in data communications - which to the corporate world is the ultimate goal. Data processing departments will no longer be the sole recipients of this integration. The entire company will eventually become linked to centralized information sources. They have accelerated into the Information Age, expanding into wordprocessing, databases, graphics, and telecommunications. All of which will change the office desk into an information center.

PC networking, known as Local Area Networks (LAN), Wide Area Networks (WAN), and other multiuser methods are a logical extension of linking PC's to the HP mainframe. These systems will provide faster ways for users to access either internal or external corporate databases. Over the next two years networking and micro-to-mainframe links are going to become very important. But it is still largely in the development stage.

Since information must be disseminated to be truly effective there is a need to connect these separate microcomputers together to share data. The LAN can be part of the total architecture developed for your office landscape to achieve true data integration.

A LAN is a logical grouping of microcomputers and peripherals linked by a short-range common communications path which ties once independent microcomputers together. These microcomputers can execute programs with their own processor and memory.

LAN's are used in office environments in which several people perform related tasks with different computers. Individual users can perform tasks independently from separate PC's while sharing resources - files, databases and spreadsheets. It is a cost effective method for sharing expensive peripherals such as hard disk drives and laser printers. Networks have been used for some time to link large computers. The time has come to use PC's as workstations (nodes) in the LAN network.

There are circumstances where it may not be cost effective to consider a LAN. If you do not intend to purchase laser printers, huge capacity hard disk drives, and your files do not require constant updating by multiple users, you may not need a LAN. If your company owns a mainframe or minicomputer with excessive capacity then consider simply linking the microcomputer that can create, upload and download data. If you own a small business and need to hook up only a few workstations, a multiuser microcomputer may be the best solution. A PBX (telecommunications) can send data down a telephone line very quickly and can be inexpensive. This may be an attractive LAN alternative, and a fairly reliable solution. Or if the majority of your computers and software are compatible, simply pass around the disks holding the common files.

None of these alternatives provide the flexibility, ease and speed of an LAN. Since companies are now heading back towards centralized operational structures, control and centralization of data are good reasons to get a LAN. If your office has many PC's it may be better to go with a LAN and try integrating existing hardware. Until standards are developed, difficult decisions will have to be made as to which LAN product to go with, knowing in several years it may have to be replaced.

Statistics show companies are unwilling to pay more than 15% of a systems price to link to a LAN. Unfortunately the real costs are much higher taking into account that it takes time for training, maintenance costs, and qualified consultants. The needs to network are here today and the benefits may already outweigh the costs. The personal stand alone computer is estimated to have 4-5 years left in the competitive office. Its likely that learning and making your mistakes now, before making a permanent investment, will cost less in the long run. The larger an organization the sooner you'll need a LAN.

How should a company evaluate their networking needs? To make it successful, planning is essential. Computer system administration must be thought through. If you don't, problems will surely arise. Initial equipment purchases are usually made haphazardly, before shared resources benefits are realized, resulting in substantial duplication of hardware, software, and peripherals. This will also increase the amount of technical tinkering required in setting up a network.

There must be decisions on organizing files, writing utility programs, system administration, backup and recovery, etc. It is crucial for your business to have working knowledge of things such as data transfer, rates, topologies, protocols, cabling etc. It will be an ongoing progression, a brutal learning curve with networking. It is hard to buy, learn and install. It is likely there will be an initial period full of regrets, disillusionment and pain.

For starters - LAN's have a specific lay-of-the-land for each type of configuration, which are called topologies.

1) star - links computers through a central host.
2) ring - a circle of microcomputers and peripherals. repeaters push the signal to the next PC.
3) bus networks - all terminals and servers share the same cable. an interface board is simply slipped into one of the slots in the back of each computers motherboard or bus.

Most networks depend on file servers. These are central computers responsible for directing the traffic of data between computers and manages access to shared peripherals. A Server may be a PC (XT,AT). Servers and workstations are linked by cable:

1) Coaxial - can support multiple signals simultaneously. Voice, video, data.
2) Twisted pair - inexpensive and easy to handle. Less capacity for high speed transition.
3) Fiber optics - promising technology for LAN, changes electric signals into pulses of light and transmits them along hair-thin lengths of glass and turns them back into electrical signals.

SUPPORTING SOFTWARE

In software marketing, four companies are the clear leaders: Microsoft, Lotus Development, Ashton-Tate and Software Publishing. Some computer companies currently supporting networks are: IBM, AT&T, Novell, 3COM, Nestar and Corvus products. IBM and AT&T are the leading contenders in the fight to dominate the LAN market and set the standards. Novell has, according to leading software publications, the best product on the market today.

IBM Sytek Network is a stripped down, technically sound, generic network package with potential, because it uses 20 different companies software.

The AT&T 3B2 network is expensive and has more features than most offices want to bother with. Most offices want a product that can be implemented with as little effort as possible.

Novell S-Net harddisk is the only true file server. It regulates access to files for network users. The Disk Server offers users a shared hard disk but minimal file or record protection. Novell S-Net may be the most powerful and flexible system, allowing up to 24 users.

Since most businesses graduated into personal computing on an as-needed basis, it is unlikely a LAN can link all PC's together. LAN's do not provide universal intervendor communications...yet! Integrated voice and data is reserved only for digital PBX. LAN's have problems: such as handling data requests by several users. And the danger exists of multiple users trying to update the same file simultaneously. LAN's also lack a central controlling processor, and file handling can't be taken care of by the operating system alone. Data transfer rates are slower because data is serially transferred down a cable to each PC.

To bring the LAN market into popularity there is a need for: 1) upgraded operating systems to enable users to boost memory capacity beyond the 640K barrier; 2) A standardized network product to allow PC users to communicate with each other and access centralized information sources; 3) Graphical interface and 4) the next generation of power like the 80386 chip by Intel - a prototype. For the foreseeable future voice must be supplied by PBX or Centrex (PBX's are phone systems that connect microcomputers via modems).

Dataquest predicts a 46% compounded annual growth rate for LAN's and LAN products through 1988 (7.1 million networked microcomputers with centralized and distributed servers). Since few companies want a network that requires the replacement or change of existing equipment or the addition of costly interface equipment, gateways will be vital to the products future.

MULTI-USER SYSTEMS

It is likely that the office of the future will be a blending of multiuser systems and LAN's. Multiuser systems are optimal for small clusters of users that share common data. LAN's can provide communications between multiuser system clusters. Telecommunications networks could connect corporate LAN's around the world.

Multiuser is the system of choice for the area that wants to share data among only a few workstations and doesn't need to incorporate a lot of existing hardware. Implementation and add-on costs for a multiuser system are usually much less than LAN's, because multiuser systems use inexpensive, "dumb" terminals.

Multiprocessors give tight resource control, and more expensive, faster performance. A multiuser system uses the time-sharing logic to link terminals. If a workstation uses only the processor and memory of a central computer, it is part of a multiuser system. A multiuser system has better data sharing strategy for tasks that depend on those applications

that require fast and frequent access to data. In some cases, a microcomputer-based multiuser system can be more effective than an LAN.

Northstar has the first multiuser computer to run an IBM-PC compatible operating system. It allows up to 12 users, each having a monitor and keyboard cabled to an independent workstation board in a central system cabinet. It runs a version of DOS under a multiuser operating system such as MP/M or UNIX.

The multiuser system has its problems. No industry leaders have marketed a micro-based multiuser product, therefore no standards have been developed for this concept. What is needed is a computer with a custom operating system designed to run MS-DOS software in a multiuser environment. Today manufacturers market with priority to make each system unique. Part of this is due to companies offering non standard enhancements. Another problem is on the production side; if the CPU is down all users are down, while LAN users could continue with local tasks.

Industry analysts predict that the future of the multiuser market lies in the development of machines that can run any number of operating systems concurrently and independently underneath a machine dependant BIOS (Basic Input/Output System). The writing is on the wall, products that sell are those that are interchangeable.

Standards for micro-to-mainframe links, area networks, and multiuser system links are emerging but remain elusive. PC to PC communications and corporate database access from desktop PC's are essential to remain competitive. A new MS DOS operating system for the PC/AT will be the breaking point in this vital development, which will tap the full potential of Intels 80286 chip. The current PC DOS has limitations because it doesn't use the full power of the AT's 80286 processor, second it is a single-user, single-task operating system. A multi-user, multitasking DOS is badly needed in the network environment.

OPTIMIZING DATA STORAGE AND BACKUP

Many micromanagers forecast a need for 100 Megabytes of storage based on anticipated applications. Total disk capacity has been projected to continue growing a 50% each year. What we see as a need in the future may not be met by todays technology.

In an attempt to solve todays data management problems, aggressive data processing managers are constantly on the lookout for new data storage media. Problems they must address include: tapes that are relatively slow and labor-intensive, as well as shortcomings caused by storage media capacity limits.

Kodak and Data Technology Corp recently announced a 5 1/4 inch floppy disk drive with a 10 Megabyte capacity. The 10 Megabyte floppy disk can serve as ordinary on-line storage and also back up a hard disk drive for the same price as a tape backup system, while doing it faster.

Other new products include a rectangular, 18-track cartridge, to replace the 1970's vintage IBM 3420 nine-track reel-to-reel device. This new

device has a data transfer rate of 3MB per second, twice the 3420. It stores 20% more data - 200MB (3480).

The latest media consists of optical storage technology that provides the ability to write only once but read many times. This may be used to archive data but otherwise doesn't have read/write access. Other data transmission possibilities are advanced telecommunications methods, such as high-capacity data lines and microwave transmission.

Even though these new products may be faster or hold more data, they are still not ending all of the storage and backup problems. Along with the mounting data capacity needs comes the problem of finding enough free system time between on-line and batch processing to complete the task of backing up critical disk file data to tape. The backup window has traditionally been between the day shift and the batch nightly runs, creating a severe operational problem for large users. Backing to tape requires costly, time-consuming human interaction. With additional problems of finding and organizing storage space for tapes, and the potential for human error. Thus the need for alternatives.

Possible solutions are: 1) incremental backups, the benefits may not be worth the performance problems; 2) making distinct business sectors with satellite facilities, rather than one large database. Break it up into divisions and separate databases; 3) and currently being researched is on-line vaulting. This requires the use of T3 communication lines (50Mb bandwidth) to send data to data backup centers off site. Once on-line vaulting becomes standard people could start sending large chunks of data to receiving (Sungard-type) facility on a scheduled basis. 4) Another future method is automated tape handling mechanisms to replace the human element, such as a bar code-readable device.

NEW MANAGEMENT IDEAS

The mulitivendor environment is a fact of life. Users can do more on multivendor systems. Incompatible equipment and software can lead to total paralysis when it comes to making large purchases of office automation technology. Today data processing managers are required to have a vision for what tomorrows technology will be available when the need arises. Only startup companies have the advantage of automating from scratch, and generally the older and larger the organization, the greater its need for multivendor systems.

Multivendor, Multilevel Architectures:

1) desktop - work group integration
2) work group - department integration
3) department - establishment integration
4) establishment - enterprise integration
5) enterprise to ex-enterprise integration

So how does a manager fight obsolescence, faced with the problems of large inventories of existing systems and data files, backlogs, and lack of skilled people? Changing some old management principals to make decisions better suited to the data processing climate is a start. The growth in

applications and small systems, micros, communication improvements, and falling costs are all in management's favor.

Managers should consider purchasing hardware with expandable memory options, so boards can be purchased latter when they are needed. They will likely be less expensive with greater capacity than those available today. Managers should ensure effective development of data base design, having a wide range of well-designed outputs. Demand good documentation, emphasize graphics, and make sure the user interface is carefully designed. More importantly, schedule costs strategically to ensure better timing in the budget cycle.

A good data base design permits easy access to the data, minimizing the chances for error, requires less memory, and is accessible to many people with diverse computer backgrounds. It is essential that information used today for one application can be used tomorrow on a different project. The automation of any environment is usually long-term. Structuring data so that it is useful for future projects is common sense. It will encompass the integration of previously unrelated data from a number of large databases and accommodate unique and variable requirements. Planning, design, and hard work is more crucial than that of new hardware or software selection.

Poor documentation is as good as no documentation. It must be pleasing to the eye - or it will go unread. It must be easy to understand and to the point. Lack of good documentation is one of the leading factors in high maintenance costs, it is not the answer to job security for the programmer.

In the future management will be more concerned with data acquisition, storage, manipulation, retrieval, and distribution. There is an increasing movement toward data-driven or data-oriented information resource management. This effects requirements definitions and software development and maintenance procedures. The objective is to standardize and reuse structures, and to go beyond the visible project boundaries.

Word processing continues to be the primary office function, but electronic mail is fast becoming the leading contender. There are more and more electronic networks available, but they don't have a common interface. People want information of all types to move faster and easier. Soon companies will have to meet this need for a universal data transmition system. Software companies that develop bridges between different office systems have already noticed the demand.

The future lies in very high-speed networks that integrate voice and data. Increasingly Artificial Intelligence will be used in Management's Decision Systems. Truly integrated systems will supersede traditional and standalone systems.

The key to survival centers on the principle of decentralization - for good communications give companies the competitive edge. It will be important to place investments in communications and processing to increase production and decrease costs, often without the traditional hard numbers to justify these expenditures to management.

Life at the End of a Phone Line, Part II
Online Information Services for HP Personal Computer Users

Bill Crow

Sysop, CompuServe HP PC Forum
and
R&D Project Manager
Hewlett-Packard Company
Personal Software Division

June 1986

Introduction

There is an information explosion underway. With little more than a personal computer and a modem, the average individual can instantly access an incredible array of information: Up-to-the-minute news, weather, sports, stock quotes, movie reviews, legal libraries, tax information, college catalogs, demographics, product support, software updates, free software, and much more. The explosion of online information services has revolutionized the availability and distribution of all types of information.

In addition to the ability to access online services, personal computers can also serve as the hub of a smaller network. A PC with a hard disc, modem, telephone line and the appropriate communications software is all that is required for anyone to provide an information service of their own. Today there are literally thousands of these personally operated, non-commercial electronic bulletin boards, providing everything from product support to government information to electronic dating services. Access is usually free, except for the price of the phone call. Many bulletin boards are networked together, providing nationwide electronic mail capability through a collection of independently operated nodes.

In the past five years, technology has opened the doors to a whole new way for people to communicate and access information. The telephone is no longer limited to a chat with Grandma on Thanksgiving; we now have instant access to virtually any information with the same ease and availability. And this is only the beginning.

The rapid growth of this new technology has created an almost mystic shroud around online services. The finicky nature of data communications, combined with the almost complete absence of comprehensive standards creates what appears to be a very steep learning curve. Many perceive online communications as the domain of computer wizards, and not to be attempted by mere mortal users. While personal computer data communications ease of use, reliability and standardization still has much room for improvement, it is a technology that most PC users can take advantage of today.

This paper provides an overview of the types of services available to personal computer users. It offers a detailed summary of the personal computer support services offered by Hewlett-Packard through CompuServe Information Service, and provides practical information to make the best use of these resources.

Online Services

Computers and communications networks have been used for many years in the corporate environment to access and distribute time-critical information. While the larger corporations often establish their own network facilities, many smaller corporations have turned to timesharing service companies to provide network services and access to mainframe computing facilities. Timesharing companies also provide common-carrier data communications services, facilitating communication among companies.

As the personal computer explosion began in the late 70's, a company in the online networks business started an experiment. CompuServe, the remote timesharing subsidiary of H&R Block, maintains an extensive data communications network and several mainframe computers to provide accounting, payroll, and many other services to their corporate customers. To make productive use of their mainframe's idle time, they began offering access to personal computer users during the evening hours. MicroNET was born in 1979, and for a one-time fee of $9 plus $5 per online hour, subscribers were provided with 128K of disc storage and access to a variety of language compilers, other standard programs and an electronic mail system.

Personal computer users responded immediately. As the user base expanded, so did the services. News, weather, stock quotes and online articles were added. No longer was MicroNET just an experiment; it began to account for a significant portion of the company's revenue. A new division was created to support what is now known as CompuServe Information Service (CIS). Competition appeared, and the variety of services continued to grow in the competitive environment.

Today, there are four major services providing full range, consumer-oriented online services: CompuServe, The Source, Delphi and GEnie.

CompuServe Information Service is the largest, with the most subscribers and the largest array of services. CompuServe is headquartered in Columbus, Ohio, and their service can be accessed through Telenet, Tymnet, Datapac (in Canada), local lines (in the Columbus area), or through their own CSnet communications network. The latter provides local telephone access for over 80% of their subscribers, with nodes in most metropolitan areas. The majority of this paper will focus on the services provided for Hewlett-Packard personal computer users via CompuServe.

The *Source Telecomputing Company* (STC) is a wholy-owned subsidiary of Reader's Digest, Inc., headquartered in Washington, D.C. STC provides online services in cooperation with Control Data Corporation. Access is provided via WATS lines, Telenet, Tymnet, and STC's own Sourcenet.

Delphi is a service of General Videotex Corporation, located in Cambridge, Massachussetts. While it has not yet matched the subscriber base or variety of services of The Source or CompuServe, it continues to grow.

GEnie stands for the *General Electric Network Information Exchange.* This latest entry to the online services community is growing rapidly, due in a large part to their expanding array of services and low rates.

Dozens of other commercial online services with products focused on specific markets are also available. *Dow Jones News Retrieval Service* specializes in stock quotes, historical financial information and news. *Dialog*, *Bibliographic Retrieval Service* and *Orbit* specialize in online encyclopedic databases. The *New York Times Information Service* and *NewsNet* provide news and specialized business information. *MCI Mail* and *Western Union EasyLink* offer electronic mail services. Both the *Official Airline Guide* (OAG) and *Trans World Airlines* (TWA) provide airline schedules and online ticketing. Scores of other companies provide a wide range of specialized online information services.

Bulletin Boards

In addition to the extensive services provided by commercial online services, personal computer bulletin board systems allow virtually any individual, club or small company to set up an information service of their own. Today there are literally thousands of bulletin board systems available, with scores more introduced daily. Books that attempt to provide lists of such services are typically outdated before they can be printed. While there are many to choose from, there are three primary criteria in choosing which bulletin board systems (if any) are worth your time to access:

1. ***Specialty***: Unless the service is geared towards your specific interests, there is little reason to spend time searching through the information provided.

2. ***Quality***: Unfortunately, there are far more bad bulletin boards than there are good ones. The quality of the board depends on the skills and knowledge of its system operator (sysop), the quality and timeliness of the information provided, and the participation of its subscribers. Separating the wheat from the chaff is often a difficult and frustrating experience. Furthermore, the best bulletin boards are also the busiest, making them the hardest to access.

3. ***Location***: Even the best bulletin board service may not be worth the cost of access if it requires a long distance telephone call. The most valuable services are nearby ones, providing free telephone access.

The Hewlett-Packard PC Forum on CompuServe

As an extension of Hewlett-Packard's personal computer support services, the company maintains an online Forum available through the CompuServe Information Service Network. The **Hewlett-Packard Personal Computer Forum** (HP Forum) is supported by a combination of HP telephone support staff personnel and two employee volunteer Sysops (myself, and Mark Horvatich of HP's Colorado Springs Division). It provides the HP customer with a source for official product support as well as unofficial assistance, opinions, suggestions and contributed software from the sysops and other participants.

The HP Forum is available to any CompuServe subscriber at no cost beyond the standard CompuServe online charges. HP currently includes a free CompuServe subscription kit, including some introductory free online time, with every MS-DOS personal computer shiped in the U.S.

The HP Forum is one of over 200 forums on the CompuServe network, with topics ranging from human sexuality to model building to aviation to personal computers. In addition to the HP Forum, HP personal computer users may also find it useful to visit the IBM PC Forums, the BORLAND Forum, the MICROSOFT Forum, the LOTUS 1-2-3 or SYMPHONY Forums, the DIGITAL RESEARCH Forum, the MICROPRO Forum, the ASHTON-TATE Forum, the PROGRAMMERS Forum, or the DR. DOBBS JOURNAL Forum, among others.

Each forum includes a *message area*, *data libraries* and several *conference channels*. These are each divided into *subtopics*, to help organize the activity on the forum. The HP Forum currently has eight subtopic areas organized as follows:

0 General Interest
1 Data Communications
2 Word Processing
3 Data Management (including spreadsheets and graphics)
4 System Utilities
5 Programming
6 Using the HP Forum
7 PC Instruments

The forums include a comprehensive and easy to use online help facility; at any prompt, type a question mark for a description of the various options. There is also an *Instructions* command from the main menu that provides a complete description of all forum functions.

The Message Area

The *Message Area* is used to leave or read the online messages between participants. By default, messages are public, allowing anyone to follow the various discussions. Private messages can be used for topics that aren't of general interest. There is also a link to *EasyPlex*, CompuServe's Electronic Mail service.

Messages are organized in *threads* within the individual subtopics; replies are connected to previous messages in a tree-like structure. Messages can be read in chronological order, or by following the conversation *threads*. You can read from only the subtopics of interest, and you can scan the message headers only, selecting the messages you want to go back and read. At any point you can jump in, adding another branch to the message tree by leaving your own reply.

The message area holds approximately 250 messages, and older ones are automatically purged as newer ones are added. The HP Forum currently averages over 100 messages a week, so the entire message area *turns over* about twice a month.

The message conversations in the HP Forum range from technical questions and advice, to support questions for HP, to equipment-for-sale notices, to discussions about new products (or rumors about future ones!) You can leave your own questions and usually receive multiple replies, comments, suggestions, recommendations or solutions within 48 hours.

The Data Libraries

The *Data Libraries* are the place to upload and download files; there is a separate data library for each subtopic. Each file includes an abstract and a keyword list (which can be used to search for files pertaining to a particular application or computer), in addition to the file name, date uploaded, size (so you can determine how long it will take to download) and the count of previous downloads.

You can browse the data library by filename (using optional wildcards for name and/or extension), keywords, age (showing only files uploaded since a certain date), and/or the id of the user who uploaded it. You can display the directory in chronological or alphabetical order, in an brief, one-line format, or in extended format with keywords and abstract.

Any file can be displayed to the screen and captured using text logging (binary files are displayed in an ASCII hexidecimal representation), or downloaded using either *XModem*, *CompuServe A*, or *CompuServe B* protocol.

Any user can upload files to the data libraries. Uploaded files go in a special holding areas, pending review by the sysops. They are usually added to the data libraries within 48 hours.

The data libraries currently contain over six megabytes of public domain, contributed, user-supported and HP-supported software for HP's family of MS-DOS personal computers.

The Conference Area

As a multiuser system, CompuServe offers a unique capability over single-user bulletin board systems: The capability for multiple users to communicate online in realtime. Each forum provides two separate channels in each subtopic area for this form of electronic conferencing. When in the conference area, whatever you type is immediately transmitted to all other conference participants. You receive messages from other participants, prefixed by their name and session number. A variety of commands are available for monitoring multiple channels, holding private conversations, or encrypting group conversations.

The HP Forum uses the conference channel to hold informal *chats* as well as special conferences to discuss specific topics or present special guests.

HP Support

The same engineers that staff the HP Helpline telephone support service also respond to questions left for them in the message area. While the response time is not as fast as the guaranteed 2-hour turnaround for the fee-based telephone support, there is no cost (beyond CompuServe's charges) for this service; typical response time is 48 hours. Because it is a public message area, the volunteer sysops and other participants usually jump in with their own ideas, suggestions and related experiences.

HP also uses the HP Forum data library to distribute selected supported software and software updates. For example, a complete library of Microsoft Word and MultiMate printer description files for LaserJet printer can be downloaded from the data libraries. Also available from (and supported by) HP are an update to the Lotus 1-2-3 PrintGraph utility (adding support for more devices), a utility to remove copy protection from HP-developed Touchscreen programs, and patch files for earlier versions of the HP Vectra firmware, among other files.

Online Software

Whether you use local bulletin boards, online product support services, or general purpose network information services, one of the most valuable assets is the ability to upload and download files. There is an incredible wealth of free (or very low cost), high quality software available through hundreds of online sources. With a little experience and the right tools, you can tap into this inexhaustible software resource.

Among the thousands of files available online, there is far more trash than there is quality software. Still, there are enough quality programs to satisfy even the most addicted software junkie. The key is to identify the useful programs, and not waste time on the rest.

The best way to find the best programs is to ask. Many sysops maintain an online list of their *best of* collections. Some systems (such as the Forums on CompuServe) display the number of times each file has been downloaded, indicating which ones are the most popular. You can always leave a message for the sysop or other participants, explaining your requirements and asking for opinions on the best solutions.

Software available for downloading falls into one of five categories: public domain, contributed, user-supported, corporate-supported, and pirate.

A **Public Domain** program is one where the original author has waived all rights to the work. You are free to do whatever you want with the software, including making modifications, enhancing it, or even selling it. The phrase *public domain* is one of the most misused terms in the software industry today. There is very little public domain software available, but the term is often used to refer to software in the next two categories.

A **Contributed** program is one where the author has made it available to anyone, but has retained the original copyrights and has usually specified some restrictions on its distribution. Typically, an author will prohibit the program from being sold for a profit, distributed in modified form, distributed under a different name, or distributed without its accompanying documentation and copyright notice. This is the most common way for individuals to *give away* their creative efforts; it protects their own rights and their investment in the work. Reputable sysops will always respect the conditions specified by contributed software authors.

User-Supported software (also known as *freeware* or *shareware*) has become a very popular method for software distribution. The author makes his/her product available through free or low-cost distribution channels such as bulletin boards, online services, or users group disc libraries. Users are encouraged to give the program a try, and if they find it useful, to send the author a payment (usually ranging from $10 to $50). In return, the author will often provide additional services, such as a printed manual, notification about updates, the source code, or access to a telephone (or bulletin board) support line. Many high quality software products are available in this manner, all at a very low cost. This is possible because there are no advertising, packaging, disc duplication, distribution or retailing expenses. To be effective, user-supported programs depend on users to compensating the authors, otherwise authors will simply turn to more traditional (and expensive) channels of distribution.

Many software companies, including Hewlett-Packard, make use of online services and/or bulletin boards to distribute software updates for their products. This **Corporate-Supported** software is almost always available through more conventional channels as well, but the use of online services provides a timely and cost effective alternative for many customers. Corporate-supported software is distributed on CompuServe by Hewlett-Packard, Microsoft, Micropro, Ashton Tate, Lotus Development Corporation, Borland International, and Apple Computer, among others.

As much as we hate to admit it, there is also a certain percentage of **Pirate** software distributed through bulletin board systems. In extreme cases, unscrupulous sysops distribute illegal copies of commercial products and encourage the exchange of this type of software among their participants. Other times, commercial products are modified to remove the original author's identification and copyrights, and then released to the public domain.

Another form of piracy are so-called *Worm* or *Trojan Horse* programs, which when run, will attempt to corrupt the file or directory structure of the user's system, or perform some other irrecoverable damage. Most Sysops work hard to eliminate pirate software, and online users should also do their part to prevent this crime.

The Online User's Toolbox

Like any other task, taking advantage of online information services requires the right set of tools. Beyond your PC and a modem, there are a handful of programs that are a *must* to be able to make the most of your online sessions.

The Communications Program

The first requirement is a good communications program; a few of the many possible choices are described below. The primary requirement is the ability to upload and download software using XModem. XModem is by far the most widely used file transfer protocol among bulletin boards and online services. The ability to log text to a file is also very valuable. Other features such as command files, additional file transfer protocols, and easy configuration options may also prove useful depending on your own personal requirements.

ARC File Compression/Library Utility

Uploading and downloading files takes time, and time is money. Therefore, anything that can reduce the time spent transferring files is of significant benefit to online users. One program, **ARC** from System Enhancement Associates (SEA), has become a de facto standard tool for online users. It is a *must* for anyone planning to upload or download files.

The **ARC** utility creates and manages compressed library files. One or more files can be combined into a single ARChive file, and data compression techniques are used to make this file as small as possible. **ARC** allows the user to add, extract, display, print, execute or delete files contained in the ARChive file, or to list the contents, verify the integrity, encrypt or decrypt the ARChive. Most MS-DOS based bulletin boards now use **ARC** to manage their online files. The resulting archive files allow the user to download all components of a submission (program file, documentation, data files, source files, etc.) as a single file, and the data compression reduces the download time anywhere from 30% to 50%.

Beyond its use with online services, **ARC** is also a handy tool for backing up files, or creating your own ARChive libraries to save important data. The ability to organize files into compressed libraries with optional encryption eases the task of managing large numbers of files.

There have been several versions of **ARC** since it was first introduced. While all are upward compatible, they are not backward compatible; the newer versions will read ARChive files created with previous versions of the program, but not vice versa. As of this writing (June, 1986) the current version of **ARC** is 5.12.

Other users have written **ARC**-compatible utilities. The most common is **ARCE**, a program that provides only the file extraction feature of **ARC**. **ARCE** is much smaller than **ARC**, and it is all that is really required to access downloaded ARChive files.

ARC is a *user-supported* program; **SEA** requests a license fee of $35 if the program is used in a commercial or government environment. They also have special terms for multiple copies or site licenses. The quality, performance and usefulness of **ARC** makes this one of the best values available. Get **ARC** and send **SEA** your $35, or you can send them $50 and they will send you a program disc (5-1/4" **IBM PC** format) and printed documentation. Their address is: System Enhancement Associates, 21 New Street, Wayne NJ 07470.

LU Library Utility

Before **ARC**, there was **LU**. **LU** (Library Utility) is similar to ARC, but does not compress the files. However, it is still used by various sysops and uploaders, so it is worthwhile to have a copy handy. **LU** is a *contributed* program, so no license payment is required. There are also several **LU**-compatible utilities that provide its various functions in individual programs.

SQ, UNSQ, et. al. File Compression/Decompression Utilities

There are several utilities available to compress (*squeeze*) and uncompress (*unsqueeze*) files. This technique is often used in combination with **LU** to create **ARC**-like library files. (Actually, the inspiration for **ARC** came from the widespread use of **LU** combined with file *squeezers*.)

Many *squeezed* files are found today in online data libraries. You can identify a *squeezed* file by the file extension: the middle letter will always be a Q. The *unsqueezer* program will properly restore the correct extension

Unfortunately, there are at least two or three different compression techniques used, and they are not compatible. Therefore, you must have the *unsqueeze* program that matches the one used to *squeeze* the files you download. **SQ, UNSQ, NSQ, NUSQ, ASQ, AUSQ** and **SQPC** are just some of the names of these various programs. Depending on where it comes from, the same program might have different names, or vice-versa. About the best you can do is collect these programs as you find them, so you have them available in your toolbox. You will invariably find the required *unsqueezer* on the bulletin board where you find *squeezed* files. However, when you begin uploading files, do your online associates a favor and use **ARC** instead!

Other Handy Utilities

There are many other useful utilities available to assist your online explorations, and most are available for free or for a small license fee. Depending on your own needs, you will make good use of different disc file and directory maintenance utilities, file browsers, hex file dump utilities, printer setup programs, file listing formatters, or floppy disc cataloging systems, to name a few. The best thing to do is explore the online data libraries,and select the programs that meet your needs.

Communications Program Features

While online services can be accessed with a terminal, they are more valuable when using a personal computer with local data storage and file transfer facilities. With most personal computers, a communications program is required to make this happen. There are all types of communications programs available for HP personal computers, providing a variety of features and capabilities. They range in price from free to several hundred dollars. Following are some of the typical features found in communications programs.

Terminal Emulation

All communications programs provide the ability for the PC to emulate a terminal, allowing bi-directional communications via the serial interface. Data entered via the keyboard is transmitted out through the serial port, and incoming information is written to the display screen. Some programs provide emulation of the control protocols used by different terminals, allowing more extensive host control of the display. Many online services support ANSI terminal protocol. HP block mode terminal emulation is most useful when accessing an HP 3000 minicomputer. Different programs provide emulation for graphics display protocols, text enhancements, extended character sets, color or other specialized terminal features. Some products emulate a variety of different terminals, with options to select the desired emulation mode. Others provide no special emulation; they can only act as a *glass teletype*, with no provisions for extended display control. While it has some limitations, this mode is more than adequate to take advantage of online services.

Text Logging

This feature allows a communications program to capture incoming information to a disc file or to the printer at the same time it is being displayed on the screen. Text logging is an essential feature for accessing online services, since you invariably want to make a copy of information that is accessed, to be referred to in detail at a later time. Virtually all communications programs provide this feature. A communications program must also offer some means of *handshaking* with the host system to prevent data loss due to delays caused by text logging.

Text Transfer

In addition to the ability to capture received text to a file, it is very useful to be able to send text from a file as if it were being entered from the keyboard. When using services that charge for access, or bulletin boards that provide a limited amount of online time, it is more efficient to compose messages or other information offline, and then transmit the file when connected. It is important for a communications program to offer a variety of *handshaking* methods for text transfer to insure data is not lost by overrunning the host system. This is an area that is woefully lacking in standards, so the handshaking used depends on the host being accessed. Many communications programs provide flexible configuration options to control text transfer handshaking protocols.

Protocol Upload/Download

By far the most useful feature of a communications program for access to an online service is the ability to transfer files using a communications protocol; this provides the facility to correct transmission errors, insuring accurate file transfer. Binary files, such as programs, formatted word processing documents, databases, or other non-text information can be transmitted. Any type of information that can be stored on a personal computer can be disseminated using an online service. Communications protocols are discussed in more detail in a later section.

Modem Management

Many communications programs provide automated facilities to manage the operation of the modem. They make it easier to dial a number, monitor for a connection and automatically log onto the service. They can also automatically re-dial a number if the first attempt was

not successful. This is especially useful when trying to access popular bulletin boards which are usually busy. Some programs provide built-in telephone directories, allowing you to select services by name rather than entering the phone number each time.

Command Files

The more sophisticated communications programs provide the ability to automate an online session by creating a file of communications commands. They offer an entire programming language tailored to communications, and the advanced user can build command files to assist during an online session, or make an entire operation completely automatic. Creating communications command files is not always easy, but the result can dramatically enhance the usefulness of the package and productivity of an online session. Unfortunately there are no standards in this area, so every package provides its own unique communications language.

File Transfer Protocols

Many different file transfer protocols are in use today. Each is optimized for its own purposes and environment, and has its own advantages and disadvantages. While all file transfer protocols share the same objective - to accurately and efficiently transfer information between two systems - each has different features. Following is a summary of the protocols most commonly used by online services, communications programs, and bulletin boards.

XModem

If there is any one standard in personal computer file transfer protocols, it is *XModem*. Also known as *Modem7* or *Christensen* protocol, *XModem* was originally developed by Ward Christensen as a means to transfer files between CP/M systems with incompatible disc formats. Ward freely distributed his programs and the details of his communications protocol, so it quickly caught on among the early hobbyist PC users. This simple protocol was very easy to build into any communications program, so it quickly became a de facto standard. Today it is one of the few protocols that can be found on virtually any system or communications program that provides file transfer capabilities.

Despite its widespread popularity, *XModem* has many limitations. Christensen admits that if he knew how popular his protocol would eventually become, he would have invested more time and thought than the two hours it took to originally design it! XModem depends on timing intervals between characters and blocks to determine if there was a transmission problem. The delays introduced by shared networks and timesharing computer systems can often cause communications programs to erroneously assume an error has occured. The protocol's limited error recovery facilities can't usually recover from this situation, causing the transfer to abort. *XModem* transfers files in 128-byte blocks with a one-byte checksum appended to each block for error detection. The one-byte checksum is prone to occasional undetected errors, and the absence of any header or file description information makes it impossible to perform batch file transfers, or automatically initiate the receiver. Files are always received as a multiple of 128 bytes, which can sometimes cause compatibility problems.

To address these and other problems, several variations of *XModem* have been developed. *Relaxed XModem* is more tolerant of transmission delays, reducing the probability of a failure due to heavy traffic on the network or timesharing system. *XModem*-CRC replaces the one-byte checksum with a two-byte *Cyclical Redundancy Check* (CRC) code, virtually

eliminating the possibility of undetected errors. *YModem* (also known as *Batch XModem*) adds file information blocks, allowing batch file transfer, automated receiver operation and accurate maintenance of file size and attributes. These and other *XModem* derivatives are becoming more popular with communications programs and bulletin board systems, but there are no clear or universal standards.

Kermit

Developed at Columbia University for file transfer among UNIX based systems, *Kermit* is a very widely used personal computer file transfer protocol. However, it is not currently supported by any of the major online information services. *Kermit* is a rich protocol, and provides file information headers to allow batch transfers, automatic unattended operation at either end and accurate preservation of file size and attributes. *Kermit* provides variable protocol parameters to optimize the transmission for the particular environment. A superset of *Kermit*, called *Sliding Window Kermit* or *SuperKermit* is starting to grow in popularity. It allows asynchronous block acknowledgement and retransmission, allowing for much more efficient transfers in a network environment. Few programs or systems support this new *Kermit* protocol.

CompuServe A

Developed for CP/M systems, CompuServe's first file transfer protocol is no longer officially supported by the company. However, because the protocol definition was freely distributed, it was widely used in a variety of communications packages. *CompuServe A* protocol is still used on the CompuServe system, and it can occasionally be found on independent bulletin board systems. It is a reliable, though not very efficient protocol, with the ability to detect when the sender initiates a transfer, and automatically start the receiver. CompuServe's implementation has a couple of nagging bugs which cause failures when transferring certain files and also creates file incompatibilities with the other protocols.

CompuServe B

Developed as part of their popular VIDTEX communications software, *CompuServe B* protocol offered many enhancements over *CompuServe A*. Computer-specific file types could be transmitted, allowing files to be downloaded directly to the personal computer's memory, or locally reformatted as required. Both efficiency and reliability were also improved. However, for several years, CompuServe would not release the protocol specification, reserving *CompuServe B* for their own communications software products. Therefore, developers and hobbyists continued to use the original *CompuServe A* protocol. Since then, CompuServe has published the specification, and *CompuServe B* is now supported by several communications programs.

Communications Programs for HP Personal Computers

The following are the most popular communications programs available for HP personal computers. A brief description of each program is provided, focusing on the features related to their use with online services.

AdvanceLink

AdvanceLink from Hewlett-Packard is the company's standard communications program for the Touchscreen and Vectra PC's. It is a successor to DSN/Link, first introduced for the HP125 CP/M personal computer. *AdvanceLink* provides an extensive communications

command language, HP terminal emulation, and two different file transfer protocols. A proprietary protocol is used for file transfer with other HP personal computers or minicomputers, and XModem protocol can be used when communicating with non-HP systems. Numerous handshaking parameters are provided for text logging and transmission. *AdvanceLink/2392* for the Vectra PC also provides ANSI terminal emulation and advanced modem management features.

Reflections

Walker, Richer and Quinn (WRQ) offer a family of products in the *Reflections* series for the HP Vectra, Touchscreen, and Portable PLUS computers. Reflections is similar in many ways to *AdvanceLink*, providing comparable features and performance. HP terminal emulation is standard, and an optional feature allows automated disc backup to an HP 3000 host. *Reflections* provides it's own proprietary file transfer protocol for communication with another PC running Reflections, or with an HP 3000 or DEC VAX minicomputer. XModem protocol is available for communication with other systems.

CrossTalk XVI

Available for the HP Touchscreen and Vectra PC's, Microstuf's *CrossTalk XVI* is one of the industry's most popular communications programs. Several terminal emulation options are provided, including ANSI, but not including emulation for HP terminals. *CrossTalk XVI* supports its own proprietary protocol in addition to XModem. It features an extensive command language and numerous features for text logging and transmission, and handshaking configuration.

TERMINAL and TERM

TERMINAL and *TERM* are the communications programs built into the HP Portable (HP 110) and Portable PLUS, respectively. *TERMINAL* offers XModem file transfer, and the ability to log or transmit text files. Limited handshaking configuration and terminal emulation is provided, and there is no command language. The program suffers from several anomalies, and other annoying side effects. *TERM* provides only basic terminal emulation and text exchange, with no file transfer protocols or command languages. It is useful only for the most simple access to online services.

ATOSIG

ATOSIG is a user-contributed program for the IBM PC (and compatibles like the HP Vectra PC) specifically designed for access to the Forums (otherwise known as Special Interest Groups) on CompuServe. It was developed as a group project by the sysops and participants of the IBM Forum on CompuServe. This powerful communications program provides one of the most efficient tools for accessing CompuServe's forums, as long as you are comfortable with the access conventions dictated by this single-purpose communications program. It is available for downloading from the IBM Forum.

ProComm

ProComm is a user-supported communications program for the IBM PC or the HP Vectra. Developed by Bruce Barkelew and Tom Smith, *ProComm* is one of the most extensive communications programs available. It offers several file transfer protocols and terminal

emulation options, an incredible array of configuration and handshaking options, and many other features too numerous to list here. *ProComm* is available from many bulletin boards and online services, including the IBM Forum on CompuServe. The authors request a $25 registration fee, making it one of the most amazing values available today.

Kermit

This communications program was first developed at Columbia University for file exchange among UNIX-based systems. The program, along with the Kermit protocol is in the public domain, and is available for scores of different computer systems. Versions of the *Kermit* program for the HP Touchscreen, Vectra, Portable and Portable PLUS pc's are available for downloading from the HP PC Forum on CompuServe. The Kermit protocol is also used by several other communications programs.

MH-BBS

This communications program for the HP Touchscreen PC was written by Mark Horvatich, one of the sysops on the HP PC Forum on CompuServe. *MH-BBS* provides one of the best implementations of XModem available for the Touchscreen PC, as well as features to implement a simple file transfer server system. Several other features are included in this free communications program available from the HP PC Forum.

PortTerm

Written by Mark Horvatich, *PortTerm* is a version of his popular *MH-BBS* program, modified to run on the HP Portable PLUS. It provides the same features and capabilities as its Touchscreen counterpart.

DC150

DataComm-150 by Jerry Bain is a versatile communications program for the Touchscreen PC. Version 1 is available at no cost from the HP PC Forum. It offers both *CompuServe A* and *B* file transfer protocols, and other features for text logging, and handshaking configuration. Version 2 adds many more features, and is available for a nominal fee directly from the author.

CISEXE

Originally written by CompuServe for CP/M based systems, *CISEXE* was the first communications program that provided file transfers with their system. It supports the *CompuServe A* file transfer protocol, but offers virtually no other features or capabilities. Versions for the HP Touchscreen, Portable (without modem support) and Portable PLUS can be downloaded from the HP PC Forum. A version that runs on the Vectra PC is available in the CompuServe IBM Forum.

CSVMDM

CSVMDM is a simple file transfer program for the Touchscreen PC that supports the XModem protocol, with enhancements to improve reliability for CompuServe access. Like *CISEXE*, it offers few other features. *CSVMDM* was written by Clarke Greene, and can be downloaded from the HP PC Forum.

How Free is Free?

While bulletin boards and online information services offer a wealth of valuable software and information, there is certainly some cost involved. As Robert Heinlein says so eloquently in his book *The Moon is a Harsh Mistress*: "There ain't no such thing as a free lunch!" It's up to you to determine if the costs are acceptable based on your own situation and needs.

The major tradeoff is between a fee-based online service such as CompuServe and the array of free bulletin board systems. At first appearance, the free bulletin boards look like the best deal, and in many situations, they are. However, for a bulletin board to be free, it must be within a local telephone call. If a long distance call is required, you can quickly rack up more telephone charges than you would spend in connect charges to an online service. Of course, this also assumes that you can reach the online service with a local phone call. With CompuServe, their extensive private network makes this possible for over 80% of their subscriber base.

There are many very good free bulletin board systems available, but there are far many more mediocre or poor ones. There are hundreds of bulletin board systems that cater to the top-selling personal computers like the IBM PC, but very few that specifically support HP products. If you can reach a high quality, HP-oriented bulletin board with a local telephone call, you're one of the lucky ones.

Few bulletin board systems can begin to match the variety and depth of services on CompuServe, not to mention the availability of official support from a variety of companies, and the large base of users to share ideas and experience. Also, as a multiuser system, CompuServe is always available. It can take hours of redialing to get past the busy signals of the popular bulletin boards. These are the major reasons HP chose to use CompuServe for it's online service, as opposed to setting up our own bulletin board.

When accessing CompuServe, there are several techniques to keep your online charges down:

1. Dial in a 300 BPS for browsing through the data libraries or reading messages online. You can't read at 1200 BPS, so why pay for the high speed access? When you have selected the desired files, log off and log on at 1200 (or 2400) BPS to take advantage of the lower cost per BPS to download files.

2. Log on to CompuServe before 8AM or after 6PM (local time) or on weekends to take advantage of the lower rates. Remember, your online rates are determined by the time *when you first log on.*

3. Make use of text file transfers to compose your messages and replies offline, and then upload them at full speed when the meter is running.

4. Learn the various commands available in the message area to scan message headers quickly, selecting the ones of interest. If you enjoy reading all messages, log them to a file and then read them offline. If you use an HP Vectra, check out the program ATOSIG to minimize your online time.

5. Learn how to use the forum's command mode; it's much more efficient than menu mode.

6. If you have access to a local, free bulletin board, make sure the file you want from CompuServe isn't available there instead.

7. If you access CompuServe for personal use, check with your tax accountant for potential ways to deduct your online charges.

Summary

So, what are you waiting for? If you spend more than a couple hours a day using your HP personal computer, you will undoubtedly find it worthwhile to explore the world of online information. If you are involved in supporting personal computer users at your company, then regular visits to the HP Forum on CompuServe and other bulletin board systems in your area is almost mandatory.

The technology of online information access is still emerging, and there is certainly a lot of room for improvement. Advances in modem technology, data communications and file transfer protocols, and communications software features and capabilities will continue to make remarkable improvements in the months and years ahead. Today, the learning curve to become a proficient online user is still a little steep, but for most personal computer users it is well worth the investment.

Planning Integrated Office Systems

by: Schram, W.P.

We regret that this paper
was not received for
inclusion in these proceedings.

Utilizing Telephone Usage Data - Automatically

Richard L. Burchett
InfoFlow International, Inc.
14644 S. E. 138th Place
Renton, Washington 98056

ABSTRACT

Telephone costs in many businesses rank high on the list of monthly expenses - sometimes as high as second or third. Yet, most businesses have done relatively little to manage this expensive tool, probably because management is unaware of how easily the computer can be used to track telephone usage and produce a variety of reports displaying the usage.

This paper discusses how large corporations, small businesses, government, and educational institutions can use the Teletrak system to obtain telephone usage information directly from their phone system. The paper suggests how TeleTrak features can be utilized in over two dozen telephone usage data retrieval and reporting ways. Key reports are described and illustrated.

Also included are discussions of the input requirements and how the telephone switch information is "massaged" to achieve desired results. The hardware aspects of utilizing telephone information, with regard to the HP-3000, are discussed in general.

Note: The subject of data communications is beyond the scope of this paper which focuses on the data retrieval and data processing aspects of telephone usage.

HOW CAN THE COMPUTER TRACK TELEPHONE USAGE?

Various recording devices have been developed for capturing data from telephone systems. Local specialists in that field may recommend specific devices to you. When used in conjunction with the TeleTrak system, these devices and your computer can track all telephone usage throughout your organization.

WHAT IS TELETRAK?

TeleTrak, developed by InfoFlow International, Inc., is a telephone information tracking software system which runs on micro computers, mini-computers, or mainframes. TeleTrak utilizes data from a multitude of recording devices and generates reports of telephone usage throughout an organization.

HOW DOES TELETRAK WORK?

The TeleTrak software places the call record information into a data base for efficient processing. Several lookup functions are performed within the data base which may contain, for example, the assignee and department code of a particular extension, when an extension was placed in service, or how much a call to the local sports line costs.

The TeleTrak tables are all easily maintained by authorized personnel through menu-driven programs.

HOW CAN TELETRAK HELP MY ORGANIZATION?

Your organization may use its telephone system for one or more of the following types of transmissions:

- MIS (either in-house or through RJE applications)
- sales-related incoming calls (e.g., from your customers)
- sales-related outgoing calls (e.g., telemarketing applications)
- miscellaneous in-house or other use (various corporations, small businesses, government agencies, educational institutions, etc.)

Because of the data base design and reporting mudule techniques utilized in TeleTrak, custom reports are easily generated. Thus, TeleTrak can produce several key reports to provide you with more information than can be gleaned from your telephone bill. For example, the information reported by TeleTrak may be utilized to breakdown telephone costs by department, to reduce telephone costs by pinpointing telephone misuse, to build sales, to determine trends, for client billing purposes, for long distance traffic engineering, to record equipment charges, for customer service records, and many other uses.

WHAT ARE THE MAJOR TELETRAK FEATURES AND USES?

Depending on your telephone usage according to your type of organization, TeleTrak may be effectively utilized to:

- generate customized reports
- track specific calls
- serve as management tool
- aid in cost reduction
- track calls by station
- calculate long distance usage
- track WATS calls
- report unauthorized phone usage
- report personal vs. business usage
- track calls made to/by individuals
- calculate time per call
- aid in service bureau applications
- aid sales service professionals
- project estimated phone bill at any time
- aid in EDP applications
- track telemarketing calls by salesmen
- merge ASCII files output (multiple switches)
- bill sold phone service
- bill client by time
- calculate departmental usage
- optimize carrier/vendor usage
- simulate "what if" situations
- report access vs. denied trunk usage
- chart traffic vs. time
- track data vs. voice usage
- report on-line calls to/from remotes
- track invalid calls to/from extensions
- report on autodialing system usage

WHAT ABOUT MY SPECIALIZED NEEDS?

TeleTrak interfaces with other telephone services offered by long distance companies (e.g., Sprint). Do you have an 800 number? TeleTrak can track all calls in your system. Also, the TeleTrak system can be integrated with TeleFile, a system which keeps track of your company's telephone equipment - from placing an order with the vendor to printing the corporate phone directory. For other specialized needs, data generated by TeleTrak can be ported into Lotus 1-2-3.

WHAT DO TELETRAK'S KEY REPORTS LOOK LIKE?

The following are examples of TeleTrak's key report formats:

(1) Detail report

This report shows one line per telephone call. The data is sorted by date and time within extension within account. An example of this report format is illustrated below:

04/16/86 *** Demo for InfoFlow Only *** Page 1
19:43:17 TeleTrak Detail Report for period 11/01/85 - 11/30/85

33-17 Customer Support G/L 934S

Station	Assigned To	Date	Time	Minutes	Number Dialed	Area Dialed		Charge
5796	K. Johnson	11/15/85	16:52	30.4	(206)851-4717	Big Harbor	MA	6.99
	STATION TOTAL			30.4	1			6.99*

figure 1

(2) Account Summary report

This report format shows one line per extension. The data is sorted by extension within account. An example of this report format is illustratd below:

04/16/86 *** Demo for InfoFlow Only *** Page 1
22:09:28 TeleTrak Account Summary for period 11/01/85 - 11/30/85

33-17 Customer Support G/L 934S

Station	Assigned To	Minutes	Calls	Charges
5796	K. Johnson	30.4	1	6.99
6064	M. R. Wings	1.5	2	0.41
6506	A. Lee Enation	0.9	1	0.21
7432	Audio Room	2.0	4	1.15

figure 2

(3) Summary report

This report format shows one line per account. The data is sorted by account. An example of this report format is illustrated below:

04/16/86 *** Demo for InfoFlow Only *** Page 1
22:25:53 TeleTrak Summary for period 11/01/85 - 11/30/85

Account	Department	Crossreference	Charges
1	Telecom Account 1	Account 950	0.27
33-08	District Attorney	45-9824-43-112-8	10.65
33-17	Customer Support	G/L 934S	9.03
49-01	Customer #4773	202-4773	19.75
5	Special Account 555	555	3.07
56-99	Overhead	220	0.82
6	Data Processing	DP6	5.09
7	General Administration	Sales	1.00

figure 3

(4) Traffic report

This report format shows usage by area code, prefix and/or destination code. An example of this report format is illustrated below:

04/16/86 *** Demo for InfoFlow Only *** Page 1
15:51:47 TeleTrak Traffic Report for period 11/01/85 - 11/30/86

Dest Code	Area Code	Prefix	Location		Calls	Tot Minutes	Ave Minutes	Total Cost	Ave Cost
WI	206	228	Renton	WA	2	0.4	0.2	0.08	0.04
		251	Renton	WA	9	12.9	1.4	2.94	0.33
		255	Renton	WA	7	3.6	0.5	4.13	0.59
		265	Arletta	WA	7	7.6	1.1	1.71	0.24
		271	Renton	WA	7	1.6	0.2	0.31	0.04
		283	Seattle	WA	1	0.9	0.9	0.20	0.20
		284	Seattle	WA	2	0.7	0.4	0.15	0.08

figure 4

(5) Error Log

This report format shows call records which are not acceptable. An example of this report format is illustrated below:

04/16/86 *** Demo for InfoFlow Only *** Page 6
22:39:49 TeleTrak Error Report for period 11/01/85 - 11/30/85

***** (TTERR 05) ACCOUNTING CODE NOT FOUND IN ACCOUNTS FILE *****

Station	Date	Time	Seconds	Number Dialed	Record
6433	11/15/85	16:58	166	235 (206)234-3365	67

figure 5

WHAT INPUT CONSIDERATIONS MUST BE MET?

TeleTrak reads ASCII data. If your telephone system uses something other than ASCII, InfoFlow International can provide a conversion program to use as a front end to TeleTrak.

TeleTrak can read fields of any length, located in any record. Upon installation of TeleTrak, tables are established which describe the various field locations.

TeleTrak will internally convert data into ASCII for output.

HOW IS THE TELEPHONE SWITCH INFORMATION "MASSAGED"?

By utilizing data base tables, TeleTrak validates data such as which extension makes the call, what area code and prefix is called, the cost per minute for the call, applicable authorization codes, which account should be charged, and if it is a third party call or a direct call.

Although everything that is dialed is recorded by the recording device, not all dialings result in valid, completed calls. TeleTrak can weed out incomplete or invalid calls.

WHAT ARE THE HARDWARE REQUIREMENTS?

TeleTrak runs on mainframes (e.g., the HP-3000), mini-computers, and micro computers. For input data gathering, you will need recording devices on your telephones. For output devices, you will need a printer.

Teletrak needs to be able to directly read the information from the phone system. Just exactly how that is hardwired or configured depends on the particular phone system.

SUMMARY

TeleTrak is a powerful, flexible, yet easily-used software system that runs on micro computers, mini-computers, and mainframe computers. It tracks the internal/external telephone usage of any type of organization - business, government, educational. TeleTrak can interface with other software and telephone systems. Its output provides several report formats to provide management aids, sales aids, personnel aids and other uses.

MIGRATING COBOL PROGRAMS TO SPECTRUM: A BATTLE OR A BREEZE?

STEVEN J. SPENCE
HEWLETT-PACKARD
19447 PRUNERIDGE AVE.
CUPERTINO, CA USA 95014

Migrating software from one architecture to another can be a rather painful process. HP's commitment to compatibility has made migration of COBOL programs from MPE V-based HP3000 systems to MPE XL-based HP3000 systems a very straightforward process for all ANSI standard-conforming constructs. There are areas where the COBOLII/V compiler takes advantage of, or is limited by, the MPE V HP3000's stack architecture, 16-bit address size and the MPE V operating system. These areas may require some migration effort but since most of these constructs are relatively obscure, few users should be affected by them. In other words, migrating COBOL programs to Spectrum should be a breeze.

New $CONTROL options have been added to assist in the migration process, however, differences in a few features will require source code changes. Some of these differences are detectable by the compiler and will generate error messages. Others are not detectable at compile time and other means of locating them are given. Information is also given on ways of taking advantage of HP Precision Architecture to facilitate improved performance. For the sake of completeness, all known areas which may require migration are presented. What follows is a description of each specific difference and what needs to be done to migrate.

Index data items

'Index data items' are data items that have the USAGE IS INDEX clause. 'Index names' appear in the INDEXED BY phrase of the OCCURS clause in the declaration of tables. The following difference affects only index data items.

Index data items are 16 bits long on the MPE V-based HP3000 and 32 bits long on the MPE XL-based HP3000 by default. Index data items within a record may change the byte offset of data items that follow it in the record. Any record that REDEFINES this record will also change accordingly. In general, index data items may be used with no changes on the MPE XL HP3000. However, when reading a record which contains an index data item from a file that was created on an MPE V system, a change may be required. In this case the index value will be invalid and all the data following it in the record will be skewed. The following example record illustrates this situation.

```
01 I-REC.
     03 I-KEY        PIC X(2).
     03 I-INDEX      USAGE IS INDEX.
     03 I-NAME       PIC X(10).
```

On an MPE V system, if I-KEY occupies bytes 1 and 2 of the record, I-INDEX occupies bytes 3 and 4 and I-NAME, bytes 5 through 14. On an MPE XL system, I-KEY is in the same place, I-INDEX occupies bytes 3, 4, 5 and 6, putting I NAME in bytes 7 through 16.

A new $CONTROL option, $CONTROL INDEX16, has been provided that will cause index data items to be allocated 16 bits. This allows the information in the file following the index data item to be read correctly. Remember however, the contents of the index

data items will not be valid and using them will give unpredictable results. This option is only provided to allow access to the remaining data in the record. $CONTROL INDEX32, the default on the MPE XL system, will cause 32 bits of storage to be allocated for index data items again. This option must appear before the 01 level of the record and cannot appear within a record. It may be placed in the COBCNTL file to make it the system default.

Another possible solution to this problem is to create a new data file on the MPE V system that does not contain index data items. A small program can be written on the MPE V HP3000 that will read the data in as 16-bit index data items and rewrite the data out as simple numeric data. The program on the MPE XL system should then be changed to read this new simple numeric data item instead of the index data item.

SYNC Data Items

All level 01 and 77 data items and SYNCHRONIZED data items are aligned on 16-bit boundaries on the MPE V HP3000 by default. On the MPE XL HP3000 the default for these data items is 32-bit boundaries. As with index data items, in general, no change is required here. The only potential problem is when SYNCHRONIZED data items appear in a record. When data items are aligned, they may create slack bytes of storage. The number of slack bytes on the MPE V system and on the MPE XL system may differ for a given record. Like index data items, this could affect the position of data items following the SYNCHRONIZED data items making data read from a file invalid. The following record exemplifies this situation.

```
01 O-REC.
     05 O-INITIAL        PIC X.
     05 O-VALUE          PIC S9(4) COMP SYNC.
     05 O-FLAG           PIC X.
```

With $CONTROL SYNC16 in effect (the default on the MPE V system), O-INITIAL occupies byte 1 of the record, O-VALUE occupies bytes 3 and 4, putting O-FLAG at byte 5. Byte 2 is a slack byte. With $CONTROL SYNC32 in effect (the default on the MPE XL system), O-INITIAL is byte 1 of the record, O-VALUE occupies bytes 5 and 6, putting O-FLAG at byte 7. Bytes 2 through 4 are slack bytes.

Using the $CONTROL SYNC16 option will cause SYNCHRONIZED data items to be aligned on 16-bit boundaries for the records following the option. $CONTROL SYNC32 returns alignment to 32-bit boundaries. The user can change the system-wide default to $CONTROL SYNC16 by using the COBCNTL file.

Note that if $CONTROL SYNC32 was used on the MPE V system to create the data file, nothing needs to be done on the MPE XL system. The SYNCHRONIZED data items will have the same alignment and the records will have the same byte offsets as the data in the file.

This idea may be used to create a new data file with SYNC data items on 32-bit boundaries. Simply write a program that reads in the 16-bit aligned data, moves it to a record with 32-bit aligned data and writes the record to a file. The program may then be compiled and run on the MPE XL system with no changes with respect to SYNCHRONIZED data items in records. This should also improve generated code since it is simpler to access 32-bit aligned data.

Parameter alignment

As stated above, 32-bit binary data items that are part of a record and do not have the SYNC clause may or may not be aligned on a 32-bit boundary. If such a misaligned data item is passed to a subprogram or an intrinsic that expects the data item to be aligned, an error will result. There are three different possible situations:

1. Calling an intrinsic: If an intrinsic called with the CALL INTRINSIC statement requires the parameter to be aligned on a 32-bit boundary and the actual parameter is not so aligned, the COBOLII/XL compiler will give an appropriate error message. This problem can be fixed by adding SYNC to the data description, with $CONTROL SYNC32 in effect, of course.

2. Calling a COBOL subprogram: No changes will be required when calling COBOL programs. All data is assumed to be byte aligned. However, a new $CONTROL option has been provided which will tell the compiler that all parameters are on 32-bit boundaries. This will generate more efficient code.

   $CONTROL OPTFEATURES = CALLALIGNED tells the compiler that all identifiers in the USING phrase of a CALL statement must be aligned on 32-bit boundaries. An error message is issued for any parameter not on a 32-bit boundary. If this option is not specified the compiler does not check the alignment of parameters, but uses the existing alignment of the data.

   This option does not apply to intrinsic calls. Alignment requirements for intrinsics are handled separately.

   $CONTROL OPTFEATURES = LINKALIGNED should be specified for subprograms which require that all parameters passed to them are on 32-bit boundaries. This allows the compiler to generate more efficient code for accessing formal parameters. This option is not meaningful for main programs. If this option is not specified then the compiler assumes all parameters are on byte boundaries.

   When calling COBOL subprograms, no problems will be encountered if the CALLALIGNED and LINKALIGNED options are not specified. Parameters are not expected to be aligned if these options are not specified. If CALLALIGNED is specified in the calling program, however, the COBOLII/XL compiler will issue an error message for any parameter that is not on a 32-bit boundary. There is more than one way to fix this problem. The data item specified in the error message may be moved to an 01 or 77 level data item. If the item is a COMP item, SYNC may be added to its data description. Or FILLERs may be added to the record declaration to align the parameter on a 32-bit boundary. A final alternative is to remove the CALLALIGNED and LINKALIGNED options, however this will result in less efficient code.

3. Calling non-COBOL subprogram: Subprograms written in languages other than COBOL may require parameters to be aligned on 32-bit boundaries. The CALLALIGNED option may be used to verify that all parameters are on 32-bit boundaries. Misaligned data should be aligned as described above. If, however, the CALLALIGNED option is not specified, the COBOLII/XL compiler will not issue an error message for misaligned data. However, when appropriate link options are used, the linker will give an error indicating that the parameter is misaligned. Align the data as described.

Note: The CALLALIGNED option may be specified temporarily just to check all calls to non-COBOL subprograms. Once data is aligned properly, the CALLALIGNED option may be removed. As specified above, misaligned parameters passed to COBOL subprograms will work (albeit less efficiently) if neither the CALLALIGNED nor the LINKALIGNED options are used.

.LOC. pseudo-intrinsic

Addresses on the MPE XL HP3000 are 32 bits long whereas on the MPE V system addresses are 16 bits. Therefore the GIVING data item on the .LOC. pseudo intrinsic must be 32 bits long. The COBOL compiler will issue an error message for the following statement:

```
CALL INTRINSIC ".LOC." USING A-ITEM GIVING A-ADDR.
```

when A-ADDR is incompatible with the requirements for the MPE XL system. A-ADDR should be described as PIC S9(9) COMP SYNC on the MPE XL HP3000.

Intrinsic calls

The COBOL compiler handles calls to intrinsics by looking in the intrinsic file for information about how to generate the call properly. Some interfaces to intrinsics on MPE XL have changed from the corresponding ones on MPE V. The best way to be sure these calls are correct is by using the CALL INTRINSIC statement for all intrinsics. This is mandatory for all HP system intrinsics and highly recommended for other intrinsics as well. CALLs to system intrinsics that do not use the CALL INTRINSIC statement may generate incorrect code, or may have unexpected results.

A new $CONTROL option, CALLINTRINSIC has been provided to assist in migrating intrinsic calls. When this option is specified, the COBOL compiler will check the intrinsic file at every CALL statement. If the specified subprogram is found in the intrinsic file, a questionable error will be given and code will be generated to call that intrinsic. The user should check all CALL statements where this message occurs and change them to use the CALL INTRINSIC statement where appropriate. The $CONTROL CALLINTRINSIC option is intended as a migration aid only, and should be removed when all CALL statements are checked and modified as necessary.

A utility is being developed that will scan a program file and give a list of intrinsics called by that program. The user would then have to check all calls to each listed intrinsic in the source file to make sure they use the keyword.

Obviously some intrinsics are specific to MPE V and either will not be available or will do nothing in native mode on MPE XL. Some of these are the extra data segment intrinsics (GETDSEG, ALTDSEG, DMOVIN, FREEDSEG, etc.) and the stack manipulation intrinsics (ZSIZE, DLSIZE, etc.). The COBOL compiler will give error messages for intrinsics that it is not able to find. These should be removed.

COBOLLOCK and COBOLUNLOCK

The COBOLLOCK and COBOLUNLOCK procedures will not be available in native mode on the MPE XL HP3000. Users therefore will have three options when migrating to native mode. The best option is to change all uses of COBOLLOCK and COBOLUNLOCK to use the COBOL verbs EXCLUSIVE and UN-EXCLUSIVE. The second option is to write replacement procedures for COBOLLOCK and COBOLUNLOCK in COBOL that will simply call the intrinsics FLOCK and FUNLOCK. Finally, the calls to COBOLLOCK and COBOLUNLOCK can be changed to call the intrinsics FLOCK and FUNLOCK directly. Note that the parameters must also be changed.

The compiler does not tell the user whether COBOLLOCK and COBOLUNLOCK are being used. At load time, however, the loader will give an error message stating that it was not able to find COBOLLOCK or COBOLUNLOCK. The user should then scan the source files for calls to either of these and change them to use EXCLUSIVE and UN-EXCLUSIVE respectively.

COBOLLOCK and COBOLUNLOCK were carryovers from COBOL/V (ANSI COBOL'68) and users were advised to change to the use of EXCLUSIVE and UN-EXCLUSIVE.

CALL identifier where the identifier is numeric

On the MPE V HP3000, the CALL identifier statement may be used to call a subprogram using the subprogram plabel. This is accomplished by calling the intrinsic LOADPROC to get the plabel of a subprogram, storing the plabel in a numeric data item and issuing a CALL of the numeric item. This gave improved performance over the regular CALL identifier statement. On the MPE XL HP3000, this feature is no longer supported. The performance of the CALL identifier statement is significantly better than on the MPE V system and the CALL plabel form no longer gives a significant performance improvement. The COBOL compiler will issue an error message upon reading a CALL identifier statement where the identifier is numeric. The user should change the statement to CALL the appropriate subprogram by name instead of by plabel. The numeric identifier can be changed to an alphanumeric identifier containing the characters of the subprogram name. The CALL identifier statement may also be changed to a CALL literal statement. Also the CALL INTRINSIC "LOADPROC" will have to be removed since LOADPROC is not on the MPE XL system in native mode.

External names

External names are those literals appearing on the PROGRAM-ID paragraph and the ENTRY and CALL statements.

There are three differences between the MPE V and the MPE XL systems in how external names are determined. These differences do not apply to intrinsic names on the CALL INTRINSIC statement. On the MPE V system, hyphens within external names are removed and the names are then truncated after the first 15 characters. On the MPE XL system, hyphens are not removed but are replaced by underscores. And external names will be truncated after the first 30 characters. The only programs that will be affected by this change are ones in which the user depends on the compiler removing the hyphens or using only the first 15 characters. The program will compile and link without generating an error for either situation. When attempting to run, however, the loader will fail and give an error message stating that the entry point specified on the CALL statement could

not be found. The user should look at the external that is undefined and either change the names on the CALL statements or the names on the ENTRY statement or the PROGRAM-ID paragraph so they match.

As an example, if a subprogram is defined as follows:

```
$CONTROL SUBPROGRAM
 PROGRAM-ID. GET-NEXT-RECORD.
      .
      .
 PROCEDURE DIVISION.
      .
      .
 ENTRY "GET-AND-CHECK-NEXT-RECORD".
      .
      .
```

The following CALL statements from a main program will fail because the external names are different.

```
CALL "GETNEXTRECORD".
CALL "GET-AND-CHECK-NEXT-REC".
```

The loader will issue the following error message at runtime:

```
Loader failed because of unsatisfied externals:
getnextrecord
get-and-check-next-rec
```

Either the calls should be changed to:

```
CALL "GET-NEXT-RECORD".
CALL "GET-AND-CHECK-NEXT-RECORD".
```

or the PROGRAM-ID and ENTRY should be changed to:

```
PROGRAM-ID. GETNEXTRECORD.
ENTRY "GET-AND-CHECK-NEXT-REC".
```

All external names are also downshifted unless explicitly specified by the user. In general this will cause no problems. The only problem will occur where a program has two external names that are identical except for case. When the compiler downshifts both names, they will no longer be unique. If the two names are defined in the same compilation unit, the compiler will give an error message. If they are in different compilation units, the compiler will not be able to detect the error. However, when attempting to run the program, the loader will give an error message stating that an entry point is multiply defined. All references to the two names will have to be checked and changed so that the names are unique irrespective of case. A less desirable option is to pre-append the backslash character to both literals which instructs the compiler to leave the name as is. This is the less desirable option because it propagates the case dependency.

Common exit points from PERFORMs and GOTOs out of PERFORMed paragraphs

Exiting at a common point from multiple PERFORMs is illegal by the 1974 and 1985 ANSI standards. Issuing a GOTO out of a paragraph that has been PERFORMed without returning to the paragraph is also illegal by both standards. In some cases on the MPE V system, however, a program that had either of these constructs would run without errors. Such a program may no longer run on the MPE XL system. Programs using either of these should be compiled with the $CONTROL BOUNDS option. When the program is run, if too many common exit points from PERFORMs or too many illegal GOTOs out of PERFORMed paragraphs are found, an error message will be issued and the program will abort. (See the section on Error Handling for other possible actions.) These constructs should be removed to get the program to run and to comply with the ANSI standard.

COBOL commands

The MPE V COBOL and COBOLII commands (COBOL, COBOLPREP and COBOLGO, COBOLII, COBOLIIPREP and COBOLIIGO) do not invoke the native mode compiler and hence no longer work for native mode programs on the MPE XL system. New command files are provided to compile, link and execute programs using the different entry points corresponding to the 1974 and 1985 ANSI standards.

The commands that will be available on the MPE XL system are:

```
COB85XL {sourcefile}, {objectfile}, {listfile}, {"info"}
```

This command invokes the native mode compiler using the entry point that conforms to the 1985 ANSI standard. It creates an MPE XL object module. Note that if the object file already exists, it will be overwritten. Multiple compiles to the same object file are not possible. Note also that the SEGMENTER is not used in native mode on the MPE XL system and therefore the PREP command does not work for native mode object files. The linkeditor on MPE XL is the utility that corresponds to the Segmenter on MPE V. The linkeditor provides facilities for building and maintaining relocatable and executable libraries as well as for linking object modules into program files.

```
COB85XLK {sourcefile}, {progfile}, {listfile}, {"info"}
```

This command invokes the native mode compiler, using the entry point conforming the 1985 ANSI standard. It also links the object file and creates a program file.

```
COB85XLG {sourcefile}, {listfile}, {"info"}
```

This command invokes the native mode compiler, using the entry point that conforms to the 1985 ANSI standard. It creates and runs a program file, in $OLDPASS.

```
COB74XL {sourcefile}, {objectfile}, {listfile}, {"info"}
COB74XLK {sourcefile}, {progfile}, {listfile}, {"info"}
COB74XLG {sourcefile}, {listfile}, {"info"}
```

These three commands are identical to the previous three except they invoke the compiler using the entry point that conforms to the 1974 ANSI standard.

Any COBOLII or COBOL commands in job files will have to be changed to the above commands according to the user's needs. The RUN command may also be used with the input and output files specified by the proper FILE equations. The primary entry point compiles the program using the 1985 ANSI standard as do the entry points 'COBOLIIX' and 'ANSI85'. The 'COBOLII' and 'ANSI74' entry points use the 1974 ANSI standard.

Validation of data

On the MPE V system, checking of ASCII and decimal data for validity and the automatic conversion of blanks to zeroes in certain cases is done by the microcode. The MPE XL HP3000 is a hard-wired machine and this type of checking must be done in software. This increases the performance cost of the checking. But since the COBOL language provides users with means for checking data, users taking advantage of these features may not need any other checking. For these reasons, the capability to specify checking or no checking has been added on the MPE XL system. The new options are $CONTROL VALIDATE and NOVALIDATE, with the default being NOVALIDATE. The default may be changed using the COBCNTL file.

If a program that has been compiled with the VALIDATE option encounters an illegal ASCII or decimal digit, an error message will be printed and the program will abort. (See the section on Error Handling for alternative actions.) To prevent these errors, the de-edited move can be used when moving from an edited field to check that no illegal characters or blanks get into a numeric field. The NUMERIC class condition can be used to detect invalid digits.

Error Handling

COBOL provides facilities for taking action at runtime when certain errors occur. For example, the ON SIZE ERROR phrase of the DIVIDE statement gives the user the ability to specify what to do when a divide by zero occurs. But what if a divide by zero occurs and the ON SIZE ERROR phrase is not used?

On the MPE V system, some of these runtime error conditions cause the program to abort, other conditions generate error messages and the program continues, while still other errors are ignored. On the MPE XL system, the user can specify what action should be taken whenever the following runtime errors occur:

1. Illegal decimal digit (Error 710). This error occurs if the program is compiled with the $CONTROL VALIDATE option and an illegal decimal digit is encountered.

2. Illegal ASCII digit (Error 711). This error occurs if the program is compiled with the $CONTROL VALIDATE option and an illegal ASCII digit is encountered.

3. Range error (Error 751). This error occurs if the program is compiled with the $CONTROL BOUNDS option and one of the following occurs: a GO TO DEPENDING ON identifier is out of bounds, a subscript or index is out of bounds, or a reference modification expression is out of bounds.

4. Divide by Zero or Overflow on Exponentiate (Error 707). This error occurs if a division by zero takes place without an ON SIZE ERROR clause, or if an exponentiation operation is performed without an ON SIZE ERROR clause, and an overflow results.

5. Invalid GO TO (Error 754). This error occurs for an alterable GO TO that was never ALTERed.

6. Address Alignment (Error 753). This error occurs if the program is compiled with $CONTROL BOUNDS and the $CONTROL OPTFEATURES = LINKALIGNED options and a parameter is passed that is not on a 32-bit boundary.

7. Paragraph Stack Overflow (Error 748). This error occurs when the program is compiled with the $CONTROL BOUNDS option and one of the following occurs: there is a recursive PERFORM, too many nested PERFORMs with the same common exit point, or too many illegal GOTOs out of PERFORMed paragraphs.

The default action when one of these errors occurs is to print an error message and abort the program. To specify an action other than the default the user needs to set a global variable, called COBRUNTIME, to a string of characters. (A global variable is similar to a job control word.) Each character position in COBRUNTIME corresponds to a particular error condition. The letter in each character position of COBRUNTIME tells the compiler how to handle that particular error, as follows:

Runtime Error Handling Options

Option	Meaning
A or blank	Print the error message and abort (default).
C	Print the error message and continue.
D	Print the error message and put the user in debug mode.
I	Ignore (continue without print the error message).
M	Print the error message, fix illegal digit and continue. (Only valid for illegal decimal/ASCII digit error.)
N	Fix illegal digit and continue without printing any error message. (Only valid for illegal decimal/ASCII digit error.)

The MPE XL SETVAR command is used to set the runtime error handling environment as follows:

```
SETVAR COBRUNTIME "string"
```

where string is a string of six characters from the possible letters A, C, D, I, M, N or Blank. The string can be in upper or lower case. Each character position in the string represents the action requested by the user for each error as described in the following table.

Runtime Error Specification

Char Pos'n	Error Type
1	Illegal ASCII or decimal digit
2	Range Error
3	Divide by zero or Overflow on Exponentiate
4	Invalid GO TO
5	Address Alignment
6	Paragraph Stack Overflow

Example:

```
SETVAR COBRUNTIME "M IDCA"
```

- will fix any invalid digits that are found, print an error message and continue running.
- will abort if a trap on DEPENDING on, Subscript/index or Reference Modification out of bounds occurs.
- will ignore any traps that occur on division by zero, if used without ON SIZE ERROR clause.
- will print the error message and put the user in debug mode if an invalid GOTO trap occurs.
- will print the error message and continue on an Address Alignment trap.
- will abort on an Overflow on Exponentiate trap, if used without ON SIZE ERROR clause.

Note that M and N can only appear in the first character position as this action only applies to the illegal ASCII or decimal digit error. If either letter appears in any other character position, it will be treated as a blank. Also note that use of these options will alter the offending source field.

Obsolete $CONTROL options

The $CONTROL ANSIPARM and BIGSTACK options on the MPE V HP3000 have no meaning on the MPE XL HP3000. They will have no effect on the user's program and can be ignored. Upon encountering them, the compiler will issue a questionable error message and ignore them.

The $CONTROL USLINIT option is the default on the MPE XL system. The COBOL compiler ignores this option and initializes the object file.

Changed $CONTROL options

The format and content of the output generated by the $CONTROL MAP, VERBS and CODE options are different on the MPE XL HP3000.

Optimum data type

The optimum data type on the MPE V system is 16-bit COMPUTATIONAL data items. These data items will work without change on the MPE XL system. To gain the best performance, however, the user may wish to change COMPUTATIONAL data items from PIC S9(4) COMP to PIC S9(9) COMP and from PIC 9(4) to PIC 9(9).

Subscripts and Indexes

The performance when referencing table elements using subscripts defined as PIC S9(9) COMP SYNC is the same as referencing table elements using index items. On the MPE V system, using index items yields better performance than using subscripts defined as PIC S9(4).

DISPLAYing unedited numeric data items

The format of output when DISPLAYing unedited numeric data items has been improved. For signed items, the signed overpunch is removed and a leading minus sign is printed if the number is negative and a leading plus sign if positive. Also, a decimal point is inserted where applicable.

SECTION numbers

No performance gain is achieved by including segment numbers after SECTION names. SECTION numbers will be ignored by the compiler except where ALTERable GOTOs exist. There is nothing the user needs to do. Note that the Segmentation module has been placed in the obsolete category in the 1985 ANSI standard.

Migrating PowerHouse Applications to New Machine Environments

Paul Elder
Jim Sinclair
COGNOS INCORPORATED
3755 Riverside Drive
Ottawa, Ontario
Canada
K1G 3N3

Abstract

One normally changes computer environments in order to reduce cost, increase performance, increase reliability, increase capacity or some combination of these. The new architecture machines from HP promise to deliver all of these.

Unfortunately, when one changes computer environments the old software does not necessarily work in the new environment. This paper describes how to migrate your PowerHouse applications from the HP3000 to HP's new architecture machines (MPE/XL machines) with as little pain as possible.

Some constructs in PowerHouse relate to specific features of MPE, IMAGE or KSAM on the HP3000. Although similar features may be available on the new machine the means of accessing them may be slightly different. This paper describes what constructs in PowerHouse are non-portable and what to do about them.

Migrating the software to a new environment is only half the question when moving PowerHouse applications. Migrating the data to the new machine also presents a problem. Migrating the data can be as simple as doing a STORE on your HP3000 and a RESTORE on your new machine, or as difficult as taking data apart bit by bit and putting it back together again in a new format. This paper discusses how to recognize when you have a data conversion problem, and how to go about getting the data converted properly.

Table of Contents

1. Migrating Your PowerHouse Applications

1.1 What Do I Do First?

When you receive your new MPE/XL machine contact your local Cognos representative. Provide your rep with:

1. The model and serial number of the machine you are upgrading from (if you are upgrading), and

2. The model number and serial number of the new machine you have received.

Your Cognos representative will arrange for you to receive a copy of Compatibility Mode PowerHouse licensed to run on your machine, a copy of Native Mode PowerHouse licensed to run on your machine, and the upgrade documentation set. These copies of PowerHouse can then be installed on your MPE/XL machine. It is not recommended that you attempt to install earlier versions of PowerHouse under MPE/XL, you are not licensed to use them on MPE/XL, and they are not guaranteed to work.

1.2 The Recommended Migration Strategy

When migrating your PowerHouse applications you do not have to do them all at once provided you follow a few simple rules.

1. DO NOT recompile a dictionary with Native Mode QDD until all of the PowerHouse modules (screens, requests, and reports) that use that dictionary have been recompiled with Native Mode PowerHouse. This is because Native Mode PowerHouse products will run off of a 5.01 created QDD dictionary, a Compatibility Mode created QDD dictionary, or a Native Mode created QDD dictionary, but Compatibility Mode PowerHouse products will only run off a 5.01 created QDD dictionary or a Compatibility Mode created QDD dictionary.

2. DO NOT convert your data files until all of the PowerHouse applications that use the data files have been converted to Native Mode.

3. DO NOT try to run Compatibility Mode compiled modules with Native Mode products.

4. DO NOT try to run Native Mode compiled modules with Compatibility Mode products.

Whether you convert all of your applications at once, or one at a time, your conversion must flow through the following five steps:

1. Prepare your application for migration under MPE/V.

2. Convert to running Compatibility Mode PowerHouse. (The brave can skip this step).

3. Convert to running Native Mode PowerHouse.

4. Convert your MPE, KSAM and PowerHouse subfiles to use IEEE floating point formats. (If you don't store floating point data you can skip this step).

5. Convert your TurboIMAGE databases to HPIMAGE databases. (Floating point conversions in databases must be done concurrently with changing databases).

2. Program Conversion

2.1 Preparation on MPE/V

Before bringing your application up on your MPE/XL machine, there are a number of things that must be done first on your MPE/V machine.

1. Upgrade your PowerHouse application to use 5.01 PowerHouse. There is no direct upgrade path to MPE/XL from earlier versions of PowerHouse.

2. If you have not done so already, upgrade your IMAGE databases to TurboIMAGE. There is no support for pre-TurboIMAGE databases under MPE/XL.

3. STORE all of your UDCs necessary for running PowerHouse.

4. STORE all data files, MPE, KSAM, permanent PowerHouse Subfiles and TurboIMAGE databases.

5. STORE all source and object PowerHouse code for your dictionaries, screens, reports, and requests.

2.2 Conversion to Compatibility Mode PowerHouse

Once your MPE/XL machine is up and running satisfactorily you can migrate your applications to Compatibility Mode by carrying out the following steps:

1. Install Compatibility Mode PowerHouse.

2. RESTORE all the files previously STORED on your MPE/V machine.

3. Modify your UDCs to point to the Compatibility Mode versions of PowerHouse installed in step 1 above.

4. Set all the necessary UDCs, at the system, account, and user level.

5. If you have hard coded the program names of the PowerHouse products (such as QUIZ.DD501E.COGNOS) in any of your application modules, you will have to go through and edit them to point to the new Compatibility Mode products and then recompile. The recommended way to do this is to point to the products in the group CURRENT.COGNOS. This way you will not have to change your source code again when you upgrade to a new version (such as Native Mode PowerHouse) in the future.

6. You are ready to run Compatibility Mode PowerHouse on your new MPE/XL machine. Note that there is no need to recompile your dictionary or any of your screens, reports, or requests, other than the changes you made in step 5 above. Test the new application. If all goes well, continue on to convert to Native Mode PowerHouse. If you discover problems, contact Cognos Technical Support.

.3 Conversion to Native Mode PowerHouse

. Install Native Mode PowerHouse.

. If you skipped conversion to Compatibility Mode PowerHouse, RESTORE all the files previously STORED on your MPE/V machine.

3. Change your UDCs to point to the Native Mode versions of the PowerHouse products.

4. Set all the necessary UDCs, at the system, account, and user level.

5. Recompile your PowerHouse modules using the Native Mode versions of the products. DO NOT recompile your dictionary with Native Mode QDD unless you have converted all applications that use the dictionary.

6. Test the new application. If all goes well, continue on to convert your data. If you discover problems, contact Cognos Technical Support.

3. Data Conversion

3.1 Conversion of MPE Files

All forms of MPE files supported in PowerHouse 5.01 under MPE/V are also supported on MPE/XL. Permanent MPE files can be moved by doing a STORE on your MPE/V system and a RESTORE on your MPE/XL system. The only data types that will need attention are floating point numbers.

3.2 Conversion of KSAM Files

For first release of MPE/XL, KSAM files will be supported on MPE/XL through KSAM INTRINSICS running in Compatibility Mode. This means that all accesses to KSAM files will result in a switch to Compatibility Mode to execute the INTRINSIC. This leaves KSAM applications with less room for performance improvements since the file system (KSAM) is not in Native Mode. KSAM files can be moved by doing a STORE on your MPE/V system and a RESTORE on your MPE/XL system. The only data types that will need attention are the floating point numbers.

3.3 Conversion of PowerHouse Subfiles

All Subfile formats supported by PowerHouse 5.01 under MPE/V are also suppored on MPE/XL. Subfiles can be moved by doing a STORE on your MPE/V system and a RESTORE on your MPE/XL system. The only data types that will need attention are floating point numbers.

3.4 Conversion of IMAGE Files

The central issue of almost every migration to the MPE/XL machines will be the choice of the IMAGE database system to use. Native Mode PowerHouse supports all the various forms of IMAGE available. Let's review the key features of each.

3.4.1 Types of IMAGE

The MPE/XL machines support both TurboIMAGE, and HP's new network database HP Image. Accompanying HP Image is TurboWindow, which provides access to HP Image as if it were TurboIMAGE.

TurboIMAGE. TurboIMAGE is a Compatibility Mode product. Everything should be identical to the way it was on MPE/V systems. PowerHouse applications require no changes to work with TurboIMAGE.

TurboWindow. TurboWindow allows access to HP Image databases as if they were TurboImage databases. Although there are a few TurboIMAGE features not supported by TurboWindow, PowerHouse does not use any of these. PowerHouse applications require no modifications to run with TurboWindow.

Note that TurboWindow will not permit utilization of most of the new HP Image features, such as relation sets. The objective of TurboWindow is to provide support for TurboIMAGE while taking advantage of the improved performance of the Native Mode database system.

HP Image. HP Image is a Native Mode network database system. HP Image features which affect PowerHouse are:

- HP Image is an IMAGE look-alike built on a relational core. The nature of a relational core means that:
  - All HP Image Intrinsic calls (HPI calls), such as HPIGET, HPIPUT, etc., must be within a transaction defined by HPIBEGIN and HPIEND calls. Changes made in a transaction are only committed and made visible to other database users when the transaction is ended.

- Once a transaction is committed (via HPIEND), all context is lost. This includes chain and current record, as well as all locks.

- HP Image provides automatic page level locking. Accessing a page via HPIGET or HPIFIND causes a shared read lock on the page. Changing a page causes an exclusive page lock. Locks are only released when a transaction is ended.

- Changes can be made to pages already having shared read locks, but the transaction changing the page cannot commit until all read locks are released.

- Explicit locks are allowed at the database and dataset levels. HPILOCK is only allowed within a transaction. Locks may only be released when the transaction is ended.

- All HPI calls except HPILOCK wait unconditionally until all required locks are granted. HPILOCK allows both conditional and unconditional locking.

- Transactions may be rolled back with HPIUNDO at any time prior to the HPIEND call. All changes made in the transaction are undone. HPIUNDO ends the transaction and releases all locks.

- If HP Image detects that a deadlock would be created by an HPI call, then the transaction containing that call will be rolled back and ended.

- HP Image supports a new dataset type, the RELATION set. A RELATION set incorporates the features of MANUAL MASTER and DETAIL sets. In addition, generic access is allowed on RELATION set key items.

- Each HP Image database belongs to a Database Environment (DBE). The DBE is the unit of transaction logging and recovery. Transactions may include several databases from the same DBE.

- HP Image has no mechanism for simultaneously committing transactions. Thus, changes to data in several DBEs cannot be guaranteed to jointly commit or jointly fail.
- HP Image allows up to eight sort items on each DETAIL and RELATION set search item. Extended sort items are not supported, and sort items may be positioned anywhere in the dataset.

3.4.2 Supporting HP Image

The preceding list of HP Image differences has meant that PowerHouse must handle HP Image differently from TurboIMAGE. The next sections will detail how HP Image is supported.

Dictionary Support. The FILE statement in QDD has been expanded to support HP Image datasets. In addition, the KEY syntax on the ITEM statement has been expanded to cope with multiple sort items. The FILE syntax for HP Image datasets is:

```
FILE name ORGANIZATION    { AUTOMATIC [MASTER]          }
                          { DETAIL                      }
                          { MASTER                      }
                          { RELATION                    }

                  OF databasename IN dbenvironment
                  [ PASSWORD string ]
                  [ TYPE HPIMAGE ]
                  [ OPEN datasetname ]
                  [ CAPACITY n ]
                  [ COPYLIB copylibkey ]
                  [ CREATE/NOCREATE ]
                  [ DESCRIPTION string [[,] string . . . ]
                  [ read/write lists ]
```

Note that a database environment must be specified in addition to the database name.

The ITEM statement for HP Image search items is:

```
ITEM element      [ storagetype ]
                  [CREATE]
                  [ [UNIQUE/REPEATING] [PRIMARY] KEY
                    [LINKS TO file
                       [SORTED ON item [ON item] . . .] ]
```

Reading and Concurrency. The automatic locking performed by HP Image can lead to severe losses of concurrency in on-line systems if care is not taken to limit transaction size. Of particular concern are what might be called read transactions. That is, a transaction that only involves database reads. QUIZ reports and the QUICK FIND procedure are the primary users of read transactions in PowerHouse. QTP will also use read transactions when there are no outputs to HP Image.

The obvious method of placing the entire read transaction within an HPIBEGIN/HPIEND pair can drastically affect database concurrency. A modest QUIZ report could end up holding read locks on large parts of a database, effectively preventing any updating activity. Even worse, QUICK FIND and SELECT modes could have much the same affect.

To circumvent these problems, PowerHouse implements read transactions as a series of much shorter HP Image transactions. Before moving to a new transaction, the database context must be saved, and then re-established in the new transaction. Figures 1 and 2 show how this is done.

```
Starting a chained read:
    HPIBEGIN
        HPIFIND

        repeat
            HPIGET mode 5
        save current record number
        HPIEND

Continuing a chained read:
    HPIBEGIN
        HPIFIND
        HPIGET mode 4 with saved record number

        repeat
            HPIGET mode 5
        save current record number
        HPIEND
```

Figure 1. Multiple transaction chained read in HP Image

```
    Starting a sequential read:
    HPIBEGIN
        repeat
            HPIGET mode 2
        save current record number
        HPIEND

Continuing a sequential read:
    HPIBEGIN
        HPIGET mode 4 with saved record number

        repeat
            HPIGET mode 2
        save current record number
        HPIEND
```

Figure 2. Multiple transaction sequential reads in HP Image

The optimal number of HIPGETs done within a single transaction is a balance between concurrency and optimal access time. One HPIGET per transaction would provide maximum concurrency. On the other hand, there is little value in locking a page only to immediately relock it. Also, the cost of the three or four extra HPI calls could easily become prohibitive. At this point, no good data is available to indicate what the best solution is.

QTP Locking and Transactions. The recommended QTP locking strategies involve opening all files and applying database or dataset locks that are held for the duration of the RUN or REQUEST. The nature of HP Image locks means that the lock duration must also be the transaction duration. Thus, REQUEST level locking means that each request will be contained in a single HPIBEGIN/HPIEND pair. Similarly, RUN level locking implies a single, RUN level transaction.

RUN and REQUEST level transactions have both advantages and disadvantages. On the positive side,

they can be used to guarantee that no changes are committed unless the RUN or REQUEST succeeds. On the other hand, an automatic rollback of 500,000 updates could take a long time.

Finally, a word about UPDATE level locking. When using this locking method, QTP does the following:

- During INPUT PHASE, a read transaction similar to QUIZ's is used to improve database concurrency.
- During the OUTPUT PHASE, QTP defines transactions about each individual output action. For example, changing an existing record would involve calls to HPIBEGIN, HPILOCK, HPIGET (reread, and checksum), HPIUPDATE, and then HPIEND. This is the same sequence of calls that would be used with TurboIMAGE.

While UPDATE level locking allows concurrent database access, it does not provide complete consistency, nor can it take full advantage of the HP Image transaction rollback facility. Performance is also adversely affected by UPDATE level locking.

QUICK Transactions. QUICK makes use of both read and update transactions. Read transactions are used in FIND and SELECT modes to retrieve each screen load of data. Read transactions are also used when performing edit lookups. Update transactions are used in the UPDATE procedure.

In FIND or SELECT mode, QUICK begins a new read transaction every time a new screen load of data is requested. The transaction is terminated before any terminal reads are done. QUICK does not maintain locks or transactions across terminal reads! QUICK saves the current context and then restores it again when the next screen load of data is requested. As with other file types, changed records are reread and checksummed as part of the PUT verb processing. Automatic read locks are not used to hold a lock on data being viewed.

UPDATE Procedure. The function of QUICK's UPDATE procedure is to commit all data changes to file. If the update fails, QUICK will rollback any changes made by the failed UPDATE procedure. Thus, the QUICK UPDATE procedure is the natural scope for an update transaction. QUICK begins the UPDATE procedure with an HPIBEGIN call. It is terminated with either an HPIEND or an HPIUNDO call. It is terminated with either an HPIEND or an HPIUNDO call. The following table relates the HP Image calls used to the verbs in the UPDATE procedure.

QUICK Verb	HPI Action
STARTLOG	Marks database for inclusion in update transaction.
LOCK	HPIBEGIN to start transaction if not already started. HPILOCK.
PUT	HPIBEGIN to start transaction if not already started. Re-read with HPIGET and recompute checksum. If checksums are the same HPIPUT, HPIDELETE, or HPIUPDATE
STOPLOG	HPIEND
UNLOCK	ignored
Any error	HPIUNDO

TABLE 1. HP Image calls corresponding to verbs in QUICK UPDATE procedure.

QUICK Rollback. When dealing with HP Image databases within the same Database Environment, QUICK does not maintain rollback information. Instead, QUICK relies upon HPIUNDO to do the rollback. Rollback information continues to be kept for other file organizations. QUICK will only commit an HP Image transaction after all updates involving other file organizations have succeeded.

When QUICK must update files from several Database Environments, rollback information must be kept for all DBE's except one. This is because HP Image cannot simultaneously commit several transactions. QUICK will commit all the DBE transactions for which it has kept rollback information after updates to other file organizations have succeeded, but before the last DBE transaction is committed. In this way, QUICK can attempt manual rollbacks on the committed transactions if the last commit should fail.

Coping with Automatic Rollback. When using HP Image either directly or through TurboWindow, PowerHouse must deal with a transaction being rolled back by HP Image itself:

- If a read transaction is rolled back, PowerHouse will retry from the last successful read. After repeated failure, it will give up.
- If HP Image rolls back a QUICK update transaction, the QUICK UPDATE procedure will fail, and result in a full rollback of any changes. The changes will still be on the screen, so that the user can retry the update.
- If an update transaction fails in QTP, further action is controlled by the error actions defined for the request. These allow either for processing to continue at the next transaction or for run/request termination. Under no circumstances will the transaction be retried.

Effects on QUICK Procedure Code. The FIND procedure has an implied HPIBEGIN and HPIEND pair defining a read transaction enveloping it.

The UPDATE procedure has an implied HPIBEGIN and HPIEND pair defining a read transaction with intent to write enveloping it. STARTLOG verbs encountered within an UPDATE procedure do not start a new transaction unless an explicit STOPLOG verb has been used to close the implied transact. It is VERY STRONGLY recommended that you do not create more than one transaction in your UPDATE PROCEDURE. If you do, roll back recovery may be impossible.

Outside of the FIND and UPDATE procedures GET and PUT verbs will be eveloped with a HPIBEGIN HPIEND pair to form a stand alone transaction, unless explicit STARTLOG and STOPLOG verbs are used to define a transaction.

The net result of these implied HPIBEGIN and HPIEND calls is that most QUICK procedure code should function without change. The one construct that must be reviewed is the use of STARTLOG and STOPLOG verbs because they now define HPI transactions.

3.5 Conversion to IEEE Floatng Point Format

If you use floating point numbers you will have to convert to the new IEEE floating point formats.

3.5.1 HP3000/IEEE Floating point types

The table below summarizes the floating point types available on the MPE/XL machines.

Format		Number of Bits		Range		Precision
		Exponent	Mantissa	Smallest	Largest	(Digits)
Float 4	IEEE	8	23	1.4e-45	3.4e38	7.2
	HP3000	9	22	8.6e-78	1.2e77	6.9
Float 8	IEEE	11	52	2.0e-323	7.0e307	15.9
	HP3000	9	54	8.6e-78	1.2e77	16.5

TABLE 2. MPE/XL Floating Point Types

There are restrictions on the usage of the IEEE and HP3000 formats:

- Compatibility Mode programs use HP3000 format floats. There is no supported mechanism for using the IEEE formats.
- Native Mode programs use IEEE format floats. A Native Mode intrinsic is available to convert between the various formats.
- TurboIMAGE only supports HP3000 format floats as item types.
- HP Image only supports IEEE format floats as item types.

The float format used in data files will affect the mix of programs that can access these files, and potentially affect program performance.

3.5.2 PowerHouse Support

Native Mode PowerHouse will support all four floating point formats as item types. Item type syntax has been changed to indicate this. See Figure 1. If IEEE/NONIEEE is not specified, a default format is taken from the dictionary options. A default format may be declared on the OPTIONS statement in the dictionary. If no default format is specified, then IEEE is assumed. Note, however, that when PowerHouse is run with a 5.01 compiled dictionary or a Compatibility Mode compiled dictionary, that the default float format will be NONIEEE.

```
float-item-type is:
     [IEEE     ] FLOAT [SIZE [4]]
     [NONIEEE]                 [8]

float-format-option is:
     [FLOAT FORMAT {IEEE     }]
                   {NONIEEE}]
```

Figure 3. PowerHouse float-item-type and QDD OPTIONS float-format-option definitions.

3.5.3 Changes in Precision and Range

Converting between floating point types can result in loss of precision or range overflow. There are techniques to detect each of these using PowerHouse.

Loss of Precision. Precision loss can be detected by converting the source type to the target type, and then back to the source type. If the original and final values differ, then precision has been lost. This simple scheme gets slightly complicated because the most precise float format available to Native Mode PowerHouse for internal computations is IEEE float 8. The trick is to use the CHARACTERS function to do the final comparison so that no calculation conversions must be made:

```
;; Precision loss if Original <> Converted

DEFINE Original    CHAR * 8  = CHARACTERS ( original-float )
DEFINE New-type    new-type  = original-float
DEFINE Old-type    old-type  = New-type
DEFINE Converted   CHAR * 8  = CHARACTERS ( Old-type )
```

Floating point precisions are shown in Table 1.

Range overflow. Range overflow may be caught by taking advantage of zero being the result of expressions with conversion errors.

```
;; Range overflow if Original <> 0 and Converted =0

DEFINE Original    old-type      = original-float
DEFINE Converted   new-type      = original-float
```

When using QTP, remember to include ON CALCULATION ERRORS REPORT on the REQUEST statement.

Approximate floating point ranges are shown in Table 1. Note that if you experience range problems, then you will have difficulty getting QUIZ to report these values, as PowerHouse does not support scientific notation.

3.5.4 Internal Calculation Precision

PowerHouse converts all numeric item values into Float 8 format when performing calculations. Native Mode PowerHouse uses the IEEE Float 8 format. As a result, calculation precision will be slightly less than was provided by HP3000 PowerHouse. This is likely a problem only when calculations require an accuracy close to 16 digits.

3.5.5 Preparing for Conversion

When preparing to convert data files to the new floating point formats several issues must be considered:

Redefinitions. If the floating point item to be converted is part of a redefinition, it is possible that it does not contain valid floating point data.

Figure 2 below shows the record layout for a file that tracks variable names, types and values. Whether FLOAT-VALUE or CHAR-VALUE is used as the value depends on whether VAR-TYPE indicates a numeric or character variable. Clearly FLOAT-VALUE cannot be converted without knowing the value of VAR-TYPE.

Note that files for which several record layouts are defined are another form of item redefinition, and should also be considered when planning conversion.

Conversion Viability. Prior to converting, it is strongly recommended that each file be edited to confirm that existing float values can be stored in the new formats. This is of particular importance when converting from HP3000 FLOAT SIZE 4 to IEEE FLOAT SIZE 4, as the target type has a smaller value range.

```
RECORD  A
  ITEM    VAR-NAME          CHAR     SIZE 20
  ITEM    VAR-TYPE          CHAR     SIZE  1
  ITEM    VALUE-AREA        CHAR     SIZE 20
   REDEFINED BY
       ITEM  FLOAT-VALUE     FLOAT    SIZE  8
       END
   REDEFINED BY
       ITEM  CHAR-VALUE      CHAR     SIZE 20
       END
```

Figure 4. A Record defining a variable name, its type and value.

Sample QTP runs have been included in Appendix A that will edit HP3000 floats and produce a subfile containing all problem records.

3.5.6 Converting

Once convinced of your success, you can start your conversion. Keep the following in mind as you go about your business.

- Only convert one file at a time, if at all possible.
- Remember to edit your data carefully before starting a conversion.
- Do not even think of converting without first making a complete backup!

TurboIMAGE to HP IMAGE. If converting from TurboIMAGE to HP Image, and if all float items are explicitly declared to IMAGE, and if float item values are not context sensitive, then DBMIGRAT may be used. DBMIGRAT is HP's TurboIMAGE to HP IMAGE conversion tool. In the above circumstances, it is likely to be the most effective tool available. Remember that DBMIGRAT converts the whole database in one fell swoop.

QTP and subfiles. This scheme will work for any file. It involves the following steps:

1. Unload the file to be converted into a permanent subfile.
2. Change the dictionary to reflect the new floating point type.
3. Use QUTIL to recreate the file.
4. Reload the file from the saved subfile.

Note that Step 4 will require conditional item final code if float values are context sensitive. Sample QTP runs have been provided in Appendix A to aid in this case. Extreme care must still be taken to guarantee that assignments only occur in the proper context.

One Off programs. A last option is to write a special conversion program in your favourite (next most favourite) programming language. This should not be necessary, but have fun!

3.5.7 Float Conversion Checklist

Here is a list of the things you should do (in the order you should do them) to convert floating point formats.

1. Backup the files you want to convert.
2. Use QTP to run edit checks on the items to be converted. The sample QTP runs provided in Appendix A illustrate each type of edit necessary.
3. If items to be converted are part of redefinitions, analyze these carefully to determine necessary contexts. Do not neglect the other items in the same redefinition!
4. If you are happy with your progress so far, choose the conversion tool:
   a. Use DBMIGRAT if moving from TurboIMAGE to HP IMAGE, AND if you have no redefinitions, AND if all float items are declared to IMAGE, AND if you want to do all the files in the database at once.
   b. Use QTP and subfiles to convert one file at a time and to cope with redefinitions. The sample QTP runs should help you with redefinitions.
   c. Write your own conversion program if you just can't wait to try out a Native Mode compiler.
5. Convert.

4. Additional Conversion Notes

4.1 Process Handling

Process Handling under MPE/XL will be very similar to process handling under MPE/V. Compatibility Mode PowerHouse will be able to call UDCs, other Compatibility Mode programs, and Native Mode programs. Native Mode PowerHouse will be able to call UDCs, MPE/XL script files, Compatibility Mode programs, and other Native Mode programs. All of the differences will be handled within PowerHouse with the COMMAND and PROGRAM statements and the RUN verb. There will be no syntax changes required.

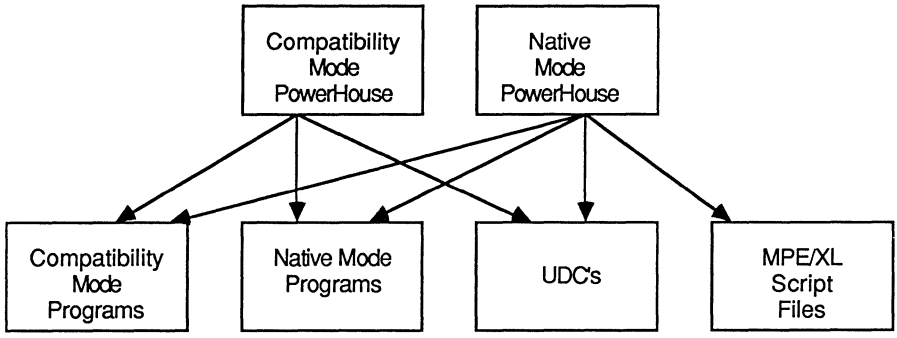


Figure 5. Process Calls Allowed

4.2 Obsolete Datatypes

A number of item data types were declared obsolete in 5.01. Table 3 gives a list of these obsolete datatypes. Compatibility Mode PowerHouse and Native Mode PowerHouse continue to support these obsolete data types, but it is strongly recommended that you move to the new forms.

Item Datatype Equivalents	
Obsolete	New
STRING	CHARACTER
BINARY3	INTEGER SIZE 6
BINARY4	INTEGER SIZE 8
DECIMAL	FREEFORM
DOUBLE	INTEGER SIZE 4
LOGICAL	INTEGER UNSIGNED SIZE 2
LONG	FLOAT SIZE 8
REAL	FLOAT SIZE 4
HPDATE	JDATE
QDATE	PHDATE

Table 3. Obsolete Datatypes

Appendix A - QTP Floating Point Edit and Reload Runs

This first run produces a subfile showing all problem records:

```
REQUEST Edit-floats        ON CALCULATION ERRORS REPORT

ACCESS  float-file

; ; For each Float 4 conversion add copies of these:
; ;
DEFINE    Original4-n       NONIEEE FLOAT SIZE 4 &
               = float4-n  OF  float-file

DEFINE    Converted4-n   IEEE       FLOAT SIZE 4 &
               = float 4-n OF  float-file

; ; For each Float 8 conversion add copies of these:
; ;
DEFINE    Original8-n       CHAR * 8 &
               = CHARACTERS( float8-n OF  float-file )
DEFINE    New8-n            IEEE       FLOAT SIZE 8 &
               = float8-n OF float-file
DEFINE    Old8-n            NONIEEE  FLOAT SIZE 8 &
               = New8-n

DEFINE    Converted8-n    CHAR * 8 &
               = CHARACTERS( Old8-n )

; ; Create Problems subfile
; ;
SUBFILE  <problems> KEEP  INCLUDE  float-file  &
               IF  Original4-1  <>  0 and Converted4-1  = 0. &
                                    or &
                                    ...
                   Original4-n  <>  0 and Converted4-n  =  0  &
                                    or &
                                    ...
                   Original8-1  <>  Converted8-1  &
                              or &
                                    ...
                   Original8-n  <>  Converted8-n
```

The second run loads and converts a floating point data file from a subfile. Conditional ITEM FINALs are added to handle context sensitive cases:

```
REQUEST Reload-Floats

ACCESS  *reload-subfile

OUTPUT  float-file ADD

; ; For each context sensitive float item include ITEM FINALs
; ; Remember to include ITEM FINALS for other items that
; ; overlap with this float, too.
; ;
ITEM  float-n                FINAL float-n  OF  reload-subfile  &
                               IF  float-context

ITEM  overlap-n FINAL overlap-n  OF reload-subfile  &
                               IF  NOT  float-context
```

THE HPIMAGE INTERFACE COMPONENT OF ALLBASE

SARA CHENG
HEWLETT PACKARD COMPANY
INFORMATION SOFTWARE OPERATION
19447 PRUNERIDGE AVE.
CUPERTINO, CA. U.S.A. 95014

The ALLBASE product on the HP3000 900 series systems is an integrated data base solution offering both a relational model interface (HPSQL) and a network model interface (HPIMAGE). In addition, a TurboIMAGE environment is included in ALLBASE so that existing TurboIMAGE applications may access HPIMAGE data bases. In the next few pages, this paper discusses how the HPIMAGE component carries on the successful familiar IMAGE functions while incorporating the advances made in relational DBMS technology.

Users who are familiar with the IMAGE interface already appreciate its simplicity HPIMAGE has retained that simplicity by keeping virtually the same programmatic interface. In addition, HPIMAGE has been made even more simpler to use in several areas through its expanded feature set.

HPIMAGE Features

o Automatic Locking

o Transaction Management

o Optional Dual Logging

o Automatic Rollback Recovery

o Optional Rollforward Logging and Recovery

o Generic Search

o Multi-level Relationships

o Dynamic Capacity Expansion

o Dynamic Security Modification

Automatic Locking

The HPIMAGE programmer is relieved of the burden of determining how to lock data because HPIMAGE takes care of user data locking automatically. Guided by user transaction definitions (through the intrinsics HPIBEGIN and HPIEND), HPIMAGE performs automatic locking. The granularity of the lock is a data page (4096 bytes). If a page is accessed for a read only operation (HPIGET intrinsic), a shared read lock is acquired. This allows other users to share the data page for read operations. When the user wants to perform an update operation (HPIUPDATE, HPIPUT, HPIDELETE), the page on which the data resides is upgraded to an exclusive lock. When a user determines that the transaction is complete (HPIEND), the locks are released so that maximum concurrency and data integrity is retained. On the other hand, if the user does not want to commit the work performed thus far in the transaction, the work can be rolled back, using the new HPIMAGE intrinsic HPIUNDO. In this way, complete data base integrity is guaranteed when reading and modifying data in a multi-user environment.

For those applications which require locking at the data set or data base level, the HPILOCK intrinsic is provided similarly to the TurboIMAGE DBLOCK intrinsic. This type of lock is an exclusive lock.

Transaction Management Overview

Transactions are units of work performed against one or more HPIMAGE databases. They are also units of multi-user isolation; the data base work performed by one user's transaction is not logically affected by the work performed by another user's transaction. This offers a high degree of consistency so that transactions are repeatable when recovery is necessary.

Recovery Overview

Recovery is the process by which the data base is brought back to a consistent state following either a system failure or data base loss. Recovery can be performed because of the ALLBASE locking and transaction management system. During recovery, all transactions which have committed their work are guaranteed to have their work reflected in the data base. Recovery will also ensure that partially completed transactions are not left in the data base. This leaves the data base in a logically consistent state. If proper locking and transaction management were not enforced by the data base management system, this level of consistency could not be achieved.

Data Base Environment

All data base modifications are logged by the ALLBASE system in order to provide the information necessary for recovery. The unit of logging, backup and recovery is called a DBEnvironment. The DBEnvironment consists of data base files, a primary log file (and optional dual log), catalog files which describe the data bases and a DBEnvironment configuration file. When a user creates a data base, the data base is added to the specified DBEnvironment. All data bases which are logically related should be placed in the same

DBEnvironment since it is the unit of logging, backup and recovery When the system administrator performs backup, an entire logical unit (the DBEnvironment) is saved. Similiarly when recovery is performed, all transactions which span work across the related data bases are logically recovered. Transactions cannot span DBEnvironments.

Optional Dual Logging

Since having a good log file is imperative to recovery, users can optionally invoke dual logging by defining a second log throught the HPIUTIL utility program. A dual log is an exact copy of the primary log and provides extra insurance in case the primary log becomes unusable. Although there is the cost of extra I/O's associated with dual logging, this option may be important to those customers who cannot afford to lose any data.

Rollback Recovery

Rollback recovery is the process by which the log file is used to bring the data base(s) to a logically consistent state following a soft crash. This assumes that the the data base files are not physically corrupt. Rollback recovery is performed automatically on the first HPIMAGE access following the interruption.

Optional Rollforward Logging and Recovery

In addition to rollback recovery, rollforward logging and recovery is optionally available. This type of recovery is used to bring the system up-to-date in the event of a hard crash where the data base(s) is corrupt. To perform rollforward recovery, a backup copy of the data base(s) must first be restored. Rollforward recovery can then be initiated through the HPIUTIL utility program. Recovery reapplies all completed transactions to the restored copy of the DBEnvironment.

Rollforward recovery requires that all data base work performed since the last backup be logged. Since the amount of work could be quite high, the system administrator may need to backup the log file periodically to tape in order to to allow the data base system to re-use the log file. Alternatively, a separate new log file can be defined.

Generic Search

To make application programming easier, an additional method of accessing data has been included in the HPIMAGE interface. This method is called 'generic search'. Subset of entries in a data set can be located by specifying a predicate as one of the HPIFINDSET intrinsic parameters. The predicate consists of a set of operands and operators where the operand can contain a special wildcard character (@) to locate all records which begin with a certain pattern. For example, all users who's names begin with 'WA' can be located by specifying 'WA@'.

Multi-level relationships

To make it easier for HPIMAGE users to model their hierarchical data relationships, HPIMAGE includes a new data set type, a RELATION. This allows multiple level relationships to be directly modeled without intervening automatic master data sets, as done today. Automatic master data sets are still available with HPIMAGE to allow migration of existing TurboIMAGE data bases. HPIMAGE RELATIONS are however more flexible and can provide the roles of both the manual master and detail data set all in one set. The only restriction placed by HPIMAGE is that a relation cannot be both a parent and child of another relation.

Dynamic Capacity Expansion

TurboIMAGE database designers need to carefully plan the data storage requirements of their data bases at schema definition time. With HPIMAGE a data set can be dynamically expanded. Without having to unload data, users can have more file space allocated to a specified data set by using the HPIMAGE HPIUTIL utility program. HPIMAGE does this by adding an additional file to the data set. The maximum number of files allowed per data set is 255.

Dynamic Security Modifications

One of the more common data base attributes modified by users is data base security. Through the HPIMAGE HPIUTIL utility program, the data base security can be altered without having to unload and load the data base. Passwords can be added, read/write lists altered and maintenance words modified. These modifications occur within the framework of a transaction so that the user of the HPIUTIL RESTRUCTURE command can control the modifications through the subcommands COMMIT or UNDO. The transaction interface provides a valuable user control over the restructuring process for now and for the future. Other types of restructuring will be available with future releases.

Implementation Strategy Summary

The HPIMAGE implementation strategy was to implement HPIMAGE on DBCORE, the internal component of ALLBASE which provides the basic database management services used by both the HPIMAGE and HPSQL intefaces. DBCORE was specially designed to meet the needs of an environment demanding high levels of concurrency while maintaining data integrity. By implementing HPIMAGE on DBCORE, the data base services required by any interface (HPIMAGE or HPSQL) need be implemented only once. Improvements to DBCORE will potentially benefit both interfaces. Future releases of ALLBASE will undoubtedly include improved HPIMAGE functionality and performance because of this implementation strategy.

Implementation Overview

Since DBCORE has services to handle physical storage definitions, HPIMAGE data sets are created by using the DBCORE data definition language (DDL). The HPIMAGE user still describes the data base by creating a schema file; the schema language is IMAGE compatible. The HPIMAGE schema processor then takes the schema file and produces an internal description of the user data base. This is called a catalog and is similar to the IMAGE root file in concept, although it is radically different in format.

Data sets are created as DBCORE objects. In TurboIMAGE, a single IMAGE record consists of both data and path information (pointers). In HPIMAGE, user data is stored as one DBCORE object and paths between data sets are established through separate DBCORE objects called indices. DBCORE supports several types of indices, including b-trees, hash indices and parent-child relationship (PCR) indices.

The PCR index was especially included in DBCORE to support HPIMAGE. Once a PCR is defined, DBCORE takes responsibility for maintaining the integrity constraint between the parent and children. A record cannot be inserted into the child if a parent record containing that key value doesn't exist. This holds similarly for the delete of a child record.

The DBCORE data manipulation language interface is used by HPIMAGE for all access to the data sets. The HPIPUT, HPIDELETE and HPIUPDATE intrinsics correspond fairly directly to DBCORE functions. HPIFIND, HPIFINDSET and HPIGET intrinsics utilize DBCORE 'scan' services. Scans accept parameters such as item lists and direction. They also utilize whatever index is specified in the request. Each type of HPIGET data retrieval corresponds to a different type of DBCORE scan. Examples are 'relation scans' (serial reads) and 'index scans' (chained reads).

Summary

This is just the beginning of a long list of possible improvements to the IMAGE interface provided by the HPIMAGE component of ALLBASE. The ability to provide HPSQL access of HPIMAGE databases is one important feature we are looking to provide in the near future. This feature will allow users to make ad hoc HPSQL queries on HPIMAGE data bases. Relational and network access to the same data - what a concept!

Using the MPE XL Link Editor

Cary A. Coutant

Hewlett-Packard Co.
19447 Pruneridge Ave., Bldg. 47LH
Cupertino, CA 95014

HP has developed a new Link Editor for MPE XL to take advantage of the Precision Architecture. The Link Editor replaces the Segmenter for all native mode program development. In addition to providing support for the new architecture, the Link Editor also supports more programming language features, offers an improved user interface, and implements new features that allow users more program development flexibility than ever before. Although some commands and terms have changed, it is still possible for Segmenter users to become accustomed to the Link Editor quickly.

This paper begins with a brief discussion of the Precision Architecture and the principles of object file management on the new architecture. The next section presents an overview of the functions of the new Link Editor, and the paper concludes with a discussion of migration issues that are important to today's Segmenter users.

1. Background

Accompanying the new architecture are completely new object file formats. An elementary presentation of some of the new concepts of both the Precision Architecture and the new object file formats is presented here. The major differences from the older HP 3000 architecture are highlighted.

1.1. Overview of the Precision Architecture

The Precision Architecture is based on 32-bit words and 64-bit virtual addresses. There are 32 general-purpose registers, denoted R0 through R31 (R0 contains a permanent zero). There are also 8 space registers, denoted SR0 through SR7; these are used to hold 32-bit space identifiers, which form half of a virtual address. The other half of a virtual address is a 32-bit offset within that space.

Programs are organized into spaces, each of which can be up to 4 gigabytes long. A program always contains one data space, which includes the global data area, heap, and stack, and one code space. (In practice, these two spaces are restricted to one gigabyte for reasons discussed below.) Additional code spaces may be loaded from libraries that are searched at program load time; this process is discussed under "Executable Libraries." During execution, a program can obtain additional spaces for extra data storage and mapped file I/O. Programs can also have unloadable spaces that are not needed for execution, but are kept in the program file. The most common example of an unloadable space is symbolic debugging information.

Memory is accessed exclusively by load and store instructions, which form a virtual address by combining a space identifier from one of the space registers SR1-SR7, and a space offset computed as the sum of the contents of any general register plus a 17-bit field from the instruction. A special form of addressing, called short addressing, automatically selects one of space registers SR4, SR5, SR6, or SR7, depending on the high-order two bits of the general register used in the offset computation. By convention, SR4 contains the space identifier of the current code space, and SR5 contains the space identifier of the data space, so that addresses in the first gigabyte ("quadrant") of the code space and in the second quadrant of the data space can be addressed in this mode. Thus, by placing all code in the first quadrant of its space, and all data in the second quadrant of its space, 32-bit pointers can be used to identify locations in either space.

Two of the general-purpose registers are reserved by convention to contain the base address for global variables (called the data pointer) and the current stack pointer. Thus, most local and global variables are directly addressable in a single load or store instruction (as long as they are within 128K of either pointer). Unlike the DB register on the older HP 3000 computers, the data pointer is not involved in indirect addressing, nor is it used for locating the heap. This fact makes it possible for MPE XL to allow executable library modules (roughly equivalent to segmented library segments) that reserve their own private data areas.

1.2. Object Modules

The MPE XL system introduces a new object file format and three new kinds of object files: Relocatable Object Files, Relocatable Libraries, and Executable Libraries. The first, Relocatable Object Files, are produced by the compilers and assembler, while the Libraries are produced and maintained by the Link Editor. Each is based on an atomic unit called an object module, of which there are relocatable and

executable varieties, as described below. As the names imply, relocatable libraries contain only relocatable object modules, and executable libraries contain only executable object modules.

Object modules produced by the compilers are called relocatable object modules, and generally represent the compilation of a corresponding source module. Depending on the programming language, a source module can be an entire source file (resulting in one object module for each compilation) or can be as small as a single procedure (resulting in as many object modules as procedures). All current compilers for MPE XL treat the entire source file as a source module.

The Link Editor can combine several relocatable object modules into a single larger one, and can link the result, producing an executable object module. Executable object modules have the same format as relocatable object modules, but are complete and have been bound to absolute addresses (or, more correctly, absolute offsets within a space). Any external references remaining after linking are summarized in an external reference table for processing by the MPE XL loader when the program is run.

1.3. Locality Sets

To maintain execution efficiency in a paged virtual memory system, programmers can organize code into locality sets. These inherit from segments the property that the MPE XL loader treats an entire locality set as a single unit, so that procedure calls within the same locality set will not suffer from a page fault. In addition, intra-locality procedure calls can often be implemented with a short branch instruction, resulting in additional savings. A short branch has a range of 256K bytes in either direction from the point of call, and costs only a single machine cycle. A long branch, however, can cost 3 or 4 machine cycles. Since the Link Editor chooses which length branch is needed rather than the compilers, the use of locality sets can significantly improve the size and speed of programs.

1.4. Relocatable Libraries

The Link Editor can search several relocatable libraries and include selected library routines in the same space as the main program (contrast this with the Segmenter, which could search only one relocatable library, including the selected routines in a single separate code segment). Like the Segmenter, the Link Editor builds, maintains, and lists the contents of relocatable libraries. Among the new functions of the Link Editor are copying object modules from one relocatable library to another, and extracting object modules from a relocatable library into a new object file.

1.5. Executable Libraries

Executable libraries are similar in spirit to the Segmented Libraries of MPE V. They consist of one or more executable object modules that may each contain one or more procedures. They are bound to the user's program at program load time, so they do not occupy any space in the program file, and the code is shared among all processes using the library. Procedure calls between executable object modules are routed through an External Reference Table (XRT), which is a direct analog to MPE V's Segment Transfer Table (STT).

Object modules are bound to fixed addresses at the time they are added to an executable library. To avoid any need for code relocation at program load time, and also to allow the desired code sharing, each executable object module is assigned to a unique space. The external call, therefore, must also save and change the currently executing space, and a return from an external call must restore the previous code space. This is also similar to MPE V, where the PB and PL registers were changed during an external procedure call.

The differences between the two architectures result in a subtle difference between segmented libraries and executable libraries: there is no direct analog to MPE V segments on MPE XL. On MPE V, segments are responsible for achieving program locality and provide the foundation for load-time binding. These two properties of a segment have been separated. Program locality is now achieved through the use of locality sets and the paged virtual memory system of MPE XL, while load-time binding is based on executable object modules. The new $LOCALITY directive associates a given procedure with a particular locality set, but does not imply any partitioning of the code into architectural units. (For source code compatibility, the $SEGMENT directive has the same effect as $LOCALITY; it should be realized that not all the effects of segmentation will be felt.) The physical partitioning of code occurs through creation of executable object modules, each corresponding to a unique code space.

It may be natural to create one executable object module for each locality set, thus preserving the mental association between segments and locality sets. It is equally valid to create an executable object module that contains many locality sets. Unlike segments, locality sets have no size restrictions, so there is more freedom in assigning procedures to locality sets. Likewise, since segments are not required for memory management, there is no compelling reason to keep executable object modules small. Calls between locality sets may suffer some penalty due to non-locality as mentioned earlier, but an external call (indirect through the XRT) is not necessary.

A program file is just a special case of an executable

library: it always contains a single executable object module, and it has a program entry point. Since it has only one executable object module, external calls are used only to call routines in executable libraries.

Some new features related to executable libraries are available in MPE XL. Users can specify at link time which executable libraries should be searched at program load time, and executable libraries may be given any legal file name. These two features simplify the use of executable libraries, permitting users to create many libraries and build programs that use the libraries without requiring any additional syntax on the RUN command. Another significant feature is that procedures in executable libraries may now define static data areas for their own private use. There is no way, however, to share data between object modules except through procedure and function arguments. The Link Editor also allows a user to copy an object module from one executable library to another, which was not possible with the Segmenter.

2. Link Editor Functions

The primary purpose of the Link Editor is to produce executable programs given relocatable object files. This process is called linking, and the LINK command performs the task (it is analogous to the Segmenter PREPARE command). The other important function of the Link Editor is to build and maintain relocatable and executable libraries. Most of the remaining Link Editor commands are devoted to this task.

2.1. Linking Programs

The linking process involves three distinct, but inter-dependent, stages. First, the Link Editor combines relocatable object modules into a single, larger object module. Second, while combining modules, it relocates all the addresses in the input object modules to reflect the actual addresses assigned by the Link Editor. Third, it resolves external references between object modules.

At compile time, if a reference to a symbol (such as a procedure or global variable name) that is not defined by that compilation unit is encountered, the compiler marks that symbol as imported (or unsatisfied). Similarly, all symbols that are defined by that compilation unit and designated as global in scope are marked as exported (or universal). At link time, the Link Editor expects to find an exported symbol in exactly one module for every unique symbol that is imported from any other module or modules.

The Link Editor can conditionally include object modules from relocatable libraries, depending on whether they export any symbols that have been imported by any other modules;

this feature is called library searching. The Link Editor can also unconditionally include all object modules from a relocatable library; this occurs if the library name is listed among the input files to the LINK command instead of among the relocatable library files.

If the Link Editor finds more than one exported symbol with the same name, an error is reported, since it is likely that two object modules have exported two different procedures or global variables with the same name. If an imported symbol remains unresolved, the Link Editor will take one of two actions. If the symbol is a data symbol, an error is reported, since there is no load-time data binding in MPE XL. If the symbol is a code symbol, the Link Editor assumes that the symbol will be found by the loader in an executable library. To indicate this to the loader, a small sequence of code called an import stub is provided and an XRT entry is allocated for that symbol. The import stub contains the code necessary to make an external call through the XRT entry, which the loader is expected to fill in at program load time after it has searched the appropriate executable libraries.

To illustrate the use of the LINK command, a subset of its syntax is shown below.

```
LINK  [FROM=file [,file]...]
      [;TO=dest_file ]
      [;RL=rl_file [,rl_file]...]
      [;XL=xl_file [,xl_file]...]
      [;CAP=cap_list]
      [;PARMCHECK=check_level]
      [;NODEBUG]
      [;MAP]
      [;SHOW]
```

As one can see from the syntax, all parameters are optional, and the simplest command "LINK" links a relocatable object module from $OLDPASS and produces a program file in $NEWPASS. Among the new features found in the Link Editor are the ability to name several input files (FROM=) in the LINK command as well as the ability to name several relocatable libraries (RL=), a list of executable libraries (that is stored in the program file for use by the loader), an override for parameter type checking level, an option to omit all symbolic debugging information from the program file, an option to print a symbolic map of the program, and an option to display the names of all object modules as they are being processed during the link.

2.2. Relocatable Libraries

It has already been mentioned how relocatable libraries are

used during linking. The Link Editor also contains a repertoire of commands to build and maintain relocatable libraries. These commands are BUILDRL, ADDRL, COPYRL, PURGERL, EXTRACTRL, HIDERL, REVEALRL, LISTRL, and CLEANRL. In addition, the command RL selects a current relocatable library that serves as a default for most other commands; the command SHOWRL displays the name of the current relocatable library.

A library is built with the command BUILDRL, and object modules are added to it with the commands ADDRL (to add modules from compiled object files) and COPYRL (to add modules from other relocatable libraries). The ADDRL command is capable of adding many object modules individually to the relocatable library, or combining them into a single larger object module that is added to the library. The COPYRL command can select one or more object modules from one library based on exported entry points, locality sets, or module names.

The PURGERL and EXTRACTRL commands can select one or more object modules from a library based on the same selection criteria as COPYRL, and purge them from the library or simply copy them into a freestanding object file. The LISTRL command prints a list of modules and the symbols imported and exported by each module. The CLEANRL command improves the efficiency of library searching by compacting the library symbol tables; this also reduces the amount of space required by the library file.

The HIDERL and REVEALRL commands are similar to the HIDE and REVEAL commands in the Segmenter. As the names imply, they operate on relocatable libraries, but their effect is only evident when the object modules are added to an executable library. In the Segmenter, hidden symbols had global scope within their segment, but were not visible from outside that segment. In the Link Editor, hidden symbols are invisible only from outside their executable object module. Since object modules from a relocatable library are included in the main program's executable object module at link time, hidden symbols in a library are still visible to the rest of the program. Therefore, the HIDERL command (or the equivalent compiler directive, $OPTION INTERNAL) is only useful to hide a procedure entry point that is to be placed into an executable library.

2.3. Executable Libraries

Executable libraries are built and maintained by a nearly equivalent set of commands as relocatable libraries. These commands are BUILDXL, ADDXL, COPYXL, PURGEXL, LISTXL, and CLEANXL. The command XL selects a current executable library that serves as a default for most other commands; the command SHOWXL displays the name of the current

executable library. For the most part, these commands function just like their relocatable library counterparts.

The ADDXL command is of particular interest in how it differs from the Segmenter ADDSL command. Like the ADDRL command, it offers the option of adding many object modules individually (creating one executable object module for each relocatable object module), or merging them all into one executable object module. In the latter case, the Link Editor can even search relocatable libraries while building the executable object module. Essentially, the ADDXL command differs from the LINK command only in that it does not expect a program entry point, and that it can add more than one object module to an executable library.

With virtually no limit on the size of a code space, the question is raised of how large an executable module should be. In general, there is no significant performance differences between an executable library with a single large module and an executable library with several medium modules. (Executable libraries with many small modules should be avoided.) The primary consideration in this case would then be one of library maintenance. It is probably easier to group library procedures by function, and include each group in a separate executable module, so that updating an executable library involves replacing one of several modules instead of the entire library.

Since relocatable libraries can be searched to produce an executable object module, it should be noted that it is not always appropriate to do so. If the routines in the relocatable library are expected to be useful in several distinct executable modules, it would be more desirable either (1) to add the entire relocatable library to the executable library as a separate executable module, or (2) to link the entire relocatable library (rather than search it) when adding one of the executable modules that depend on it. In the second alternative, the relocatable library routines would be an internal procedure call for the one module, and an external procedure call for the others. It is still possible, however, to search the relocatable library for each executable module, but all the entry points in the relocatable library should be hidden to prevent duplicate symbol errors. With hidden symbols, each executable module has its own private copy of just those library routines that are needed within that module.

2.4. Other New Features

The Link Editor also offers a thorough help facility through the HELP command. The HELP facility is the same one used by the Command Interpreter and other MPE subsystems. It contains help screens describing the syntax and parameters and provides examples for every Link Editor command.

Like the new Command Interpreter, the Link Editor also keeps a list of the last twenty commands entered. Users can edit and redo any of the commands in this list.

Several Link Editor commands (e.g., LINK, ADDRL, ADDXL) allow the user to specify a list of files as input files or as relocatable or executable libraries. In any of these situations, an ASCII file can be prepared in advance containing the list of files, and this "indirect" file can be specified, preceded by a '^', in place of the list of files. The Link Editor will read this list of files as if they had all been typed on the command line.

Two additional convenience features are also available. First, any MPE XL command may be executed from the Link Editor by preceding the command with a colon. Second, any Link Editor command may be executed directly from the Command Interpreter by running LINKEDIT.PUB.SYS and including the entire command in the INFO string. In this mode of use, the Link Editor will terminate after executing the command.

3. Migration

The process of learning to use the Link Editor after becoming accustomed to the Segmenter is more than just learning the new command syntax. There are many similarities between the two products, but the significant differences and the potential pitfalls of drawing parallels are discussed below. This section ignores Compatibility Mode, where the MPE V Segmenter is available and functions exactly as it has on all previous HP 3000's.

3.1. Using RL files as USL's

Probably the first difference that an MPE V programmer will notice is that MPE XL does not have USL (User Subprogram Library) files. The MPE XL compilers, instead of placing object modules into a USL file, now create new object files with each compilation. This change reflects the observed tendency of many MPE V programmers to use separate USL files for each source file, combining them only in a long Segmenter script with lots of AUXUSL and COPY commands. Since the LINK command now accepts many file names (and even accepts "indirect" files that contain a list of file names), it is now much more convenient to develop programs with this technique.

It is still possible to use relocatable libraries as a substitute for USL files, however. With a UDC or command file, users may automatically add the object file resulting from a compilation to their pseudo-USL. The relocatable library thus produced can be named as an input file in the LINK command, and the Link Editor will link the entire file

unconditionally (as opposed to searching it if it were named as a library file). In this sense, the relocatable library behaves just like a USL.

Relocatable Libraries, however, do not support version management. The Segmenter CEASE and USE commands do not have any equivalent in the Link Editor, and compilers do not automatically deactivate old object modules as they create new ones. Because of the error-prone nature of this feature (old versions of an object module were denoted by a chronological index, with no form of user annotation available), and the increasing use of (more effective) source-level version control systems, this Segmenter feature was chosen for obsolescence.

Another significant difference between Segmenter and Link Editor use results from the fact that MPE XL compilers do not maintain a USL. As mentioned earlier, the compilers create one relocatable object module for an entire compilation, whereas MPE V compilers created one RBM per subprogram in the compilation. Because of these facts, there is no practical way to do a subset compilation (a variant of the $SUBPROGRAM directive, where a subset of procedures in the source file is named in the directive). Since a relocatable object module, like an RBM, is an atomic structure, there would be no way of replacing a subset of the procedures from an earlier, full, compilation with the new versions compiled in a subset compilation. The only successful way that such a feature could be used would be to compile disjoint subsets at all times.

Also because of the fact that MPE XL compilers place all procedures in a single object module, the Link Editor can copy and purge procedures only by copying and purging entire object modules. For example, even though the PURGERL command allows the user to specify an entry point name, the entire object module that contains that entry point is purged, including any other entry points contained within that module. Therefore, unless all source files contain only a single procedure each, the Link Editor PURGERL command is not an exact replacement for the Segmenter PURGERBM command.

3.2. Relocatable Library Differences

Relocatable libraries are not much used on MPE V, mostly because of the restrictions associated with them: a program could be linked with at most one RL, and all modules included from the RL were placed in a single, separate, segment. Nevertheless, those who used RL's on MPE V should have no trouble using them on MPE XL. With one exception noted below, the only noticeable difference is that no separate segment is created; therefore calls to procedures from a relocatable library can be just as fast as internal

procedure calls.

The Segmenter ADDRL command can be translated directly to a Link Editor ADDRL command by ignoring the new features. If USL's are being emulated with RL's, however, the Link Editor COPYRL command becomes the appropriate Link Editor command to use. Since the ADDRL command requires one parameter -- an entry point name -- the COPYRL option to select by entry point name is a direct translation.

It was mentioned above that the MPE XL compilers create one relocatable object module for an entire compilation, and that this creates some differences in perceived behavior between the Segmenter and the Link Editor regarding USL files. It also affects the behavior of relocatable libraries at link time: the Link Editor includes an entire object module in the program file even if only one procedure in that module is referenced by the program, since object modules are indivisible. This difference could affect more than just the size of the program file; the user program may define an entry point that conflicts with one of the extra procedures that just happens to share an object module with a procedure that was needed. In such a case, a duplicate symbol error occurs when the link would succeed if each procedure had been in a separate module. If this problem does indeed occur, it would be necessary to recompile each library procedure separately (that is, place each one in a separate source file), and add the resulting object modules individually to the library. This technique will produce a relocatable library equivalent to the MPE V RL.

Users who did use RL's on MPE V will probably want to modify their source code over time to take advantage of locality sets that are available with RL's on MPE XL. In addition, RL management should become easier since several libraries can be used for different purposes instead of maintaining a single library for potentially several applications.

3.3. Segmented Libraries vs. Executable Libraries

There is little perceivable difference between MPE V segmented libraries and MPE XL executable libraries. An executable object modules possesses all the significant properties of a segment with regard to the functions of a segmented library. In particular, an executable object modules and a segment are both indivisible, so a reference to one procedure results in the entire module or segment being bound to the program at load time.

For the user who is making a straightforward translation from Segmenter usage to Link Editor, a series of ADDSL commands that add the contents of a USL into an SL can be replaced by a single ADDXL command with the MERGE option to add an entire relocatable library or set of object files

into an executable library. The MERGE option causes all the input modules to be placed into a single executable object module; there is little to be gained by adding more, smaller, modules to the library. It is reasonable, however, to translate segments into locality sets, and a more literal replacement for an ADDSL command would be an EXTRACTRL using the locality set selection (assuming an RL is being used as a USL), followed by an ADDXL to add the object file just extracted to the library.

3.4. PREPARE vs. LINK

Most of the parameters for the Segmenter PREPARE command are either available in the Link Editor LINK command or meaningless on MPE XL. Those that are meaningless can simply be ignored. The only significant difference in command operation is the default object file that is to be linked: the PREPARE command always uses the current USL, while the default FROM file for the LINK command is always $OLDPASS (which is the same as the default for the MPE V PREP command in the command interpreter). If an RL is being used as a USL replacement, the PREPARE command can be replaced by a LINK command specifying that file as the FROM file.

3.5. HIDERL and REVEALRL

As mentioned earlier, the HIDERL and REVEALRL commands themselves function just like the Segmenter HIDE and REVEAL commands, in the context of using an RL as a USL replacement. The effect of hiding a symbol, however, can be different if there is not a one-to-one correspondence between segments and executable object modules. If exactly the same effect is necessary, then the technique presented above for translating the ADDSL command literally should be used.

4. Summary

In summary, the Link Editor adheres to the philosophy of the Precision Architecture itself -- that completely new technology can be compatible with previous technology without sacrificing in design. Complete and identical Segmenter functionality is available to all customers in compatibility mode; with few exceptions, customers can migrate to native mode with only a minor effort in learning the new product and converting job streams. The only exceptions are that some applications using RL's may need to restructure (but not rewrite) some of their source code. A complete migration to take full advantage of the new architecture and Link Editor can be done at the customer's own pace.

Acknowledgments

The Link Editor was implemented by Bruce Blinn, Jon Bresler, Dan Magenheimer, Tuan Pham, Chris Schoppa, Lam Tran, and the author. I would also like to thank Jon Kelley and Debbie Coutant for providing valuable help in preparing this paper.

Adolescence at Age 137; The Study of a Change in Corporate Culture

By

Mark Indermill
Amfac Distribution Corporation
81 Blue Ravine
Folsom, CA

Amfac is 137 years old this year. It's revenues in 1985 were in excess of $2.6 Billion. It is a conglomerate which consists of five major groups; Food, Retail, Resorts, Amfac Hawaii, and Distribution. All of the glamour and half of the revenues come from four of these groups. The Food Group is one of the largest suppliers of potato products to America's fast food chains, as well as Fisher Cheese and Monterey Mushrooms brands. The Resorts Group operates the Fred Harvey Hotels at the Grand Canyon and Death Valley in addition to five resort properties in Hawaii; Kaanapali is perhaps the most famous. Amfac Hawaii manages and develops the company's 57,000 acres of owned and 97,000 acres of leased land and their diversified agricultural operations. Amfac is the largest member of the C & H sugar cooperative. Amfac Hawaii also has an Amfac Energy Division which produces more electricity than many small utilities on the mainland. The Retail Group manages Liberty House stores in Hawaii and Village Fashions, an off-price retailer located in six east coast states.

The other half of the revenues, about $1.3 Billion, is generated by the low-profile Distribution Group. This group is involved in three lines of business, all as a wholesale distributor. The Health Care Division is primarily a secondary supplier of prescription and over the counter drugs with 37 branches in 22 states. The Electric Division has 87 branches and the Supply Division has about 130 branches. Together these divisions serve 42 states and parts of Canada. Because of its role as a middleman, the wholesaler is traditionally anonymous. Amfac Distribution is no exception to this rule. Even though it is one of the largest wholesalers in each of the markets in which it competes, Distribution has virtually no name recognition outside the industry or financial community.

Amfac Distribution recently undertook a committment to spend about 20 million dollars with Hewlett Packard to automate and modernize each of its branches with HP3000 computers. It is the Distribution Group which serves as the focus for this paper.

In 1849 a China trader, Heinrich Hackfeld, landed in Hawaii with a 156 ton ship named the Wilhemine loaded with $8,500 worth of goods from his home in Germany. He set up shop in a Honolulu storefront on October 1, 1849. His business then catered to trade generated by the California gold rush and the 300 to 400 whaling ships that wintered in the Islands. Within four years he had become a "factor" for many of the fledgling sugar plantations in Hawaii, handling most of their overseas purchasing, shipping, insurance, and marketing. His first store quickly became more of a transshipment operation for industrial goods than the retail and dry goods store he had originally envisioned. He then expanded to a second "upper store" away from the waterfront. As a precursor to the Distribution group, he began importing building materials and lumber in the 1850's and his endeavors grew as did the Hawaiian economy throughout the gold rush era. Holdings totaled nearly $4,000,000 in 1897 when H. Hackfeld & Co. was incorporated.

The infant sugar industry struggled with the business end of agriculture. Hackfeld's experience as a China trader and his willingness to use his business acumen as a business manager for the Hawaiian plantations was a perfect match for the needs and the time. This arrangement of factoring for the plantations was refined to a large degree by Hackfeld's partners and his successors.

The end of World War I brought an end to the German involvement in the Hackfeld company. In 1918 the assets were sold to American Factors, Ltd. a consortium including Alexander & Baldwin, Castle & Cooke, Matson Navigation Co., and Welch and Company. American Factors, Ltd. was shortened in the vernacular to Amfac and the company remains with that name today.

The early fortunes of Amfac were tied directly to the cultivation of sugar cane in the Islands, not only the production and sale of sugar, but also the development of the land that accompanied the growth of the plantations. Since the turn of the century, Amfac has diversified into several areas, further strengthening its financial base. Prior to 1968, substantially all of its revenues were derived from operations within Hawaii. 1968 marked the year that the company began to extend operations beyond Hawaii. These ventures returned roughly three-quarters of Amfac's total revenues in 1985.

One of the most profitable of Amfac's groups has been the Distribution Group. Since its expansion to the mainland in 1969 it has grown consistently and phenominally. The years from 1970 through 1980 were characterized by tremendous growth through acquisition. The strategic plan for the Distribution group was to continue to expand in the 1980's both through the building of new branches and through acquisition and to leverage depth and

diversity of experience as one of the oldest, largest, and most successful distribution operations in the United States.

There are approximately 8,000 wholesalers competing with Amfac in its primary lines of business in the U.S. These competitors are typically single-house independent distributors, each about the size of one of Amfac's 266 branches. It is these independents that serve the bulk of the market place, not the other large, multibranch distribution companies. Amfac has always attempted to emulate these small, high service independents and has been very successful in competing with them. Key to this success has been the independence with which Amfac's own branches are managed.

Decentralization was the watchword through the 1970's and early 1980's. Grass roots decision making, on the spot responsiveness to the needs of the customer, and total awareness of market trends and the shifting desires of the customer base were encouraged through a compensation program for branch personnel that was tied directly to performance. The effort was made to duplicate the spirit, responsiveness, and innovation of the small independent distributor. Amfac attempted to avoid the bureaucracy imposed by the trappings of a large unyielding, public company by giving the branch manager the autonomy to create his own product mix, inventory levels, pricing, and service levels. The role of the central management group in monitoring and measuring the performance of each branch was reduced to the relatively non-participative involvement such as would be provided by the independent wholesaler's banker and/or CPA.

Evidence for the degree to which the Distribution Group was decentralized could be found simply by counting heads at the corporate headquarters. A staff of less than twenty people directed the course of a business with at one time nearly 300 branches and revenues in excess of 1.2 billion dollars.

The man primarily responsible for the implementation of this philosophy was John P. Richardson. Richardson came to power in 1964 when he went from salesman to department head literally overnight. As a manager of the Electrical Supply Department, Richardson became well connected with the business of distribution, its trade associations, the people, and the community of interests that bound them together. He was intimate with the knowledge of what worked and what did not in the industry and he was convinced that good profits could be made in the right lines of business and through the acquisition of the right type of independent dealer. When he became chairman he put these ideas to work.

Richardson was noted for his dogged determination to avoid "bigness". His philosophy was best articulated by this quotation:

> This business is successful only if it is conducted on an eyeshade, crackerbarrel basis. For example, in the electrical supply game you've got to know your customer's problems so well you can help in bidding on contracts and in the course of it sell him your lines. In the drug field the pharmacist has to know you so well that he can call you at home after hours in the certainty that you'll go down and open up the warehouse to get some rarely needed item to him in an emergency. The minute you get too big this intimacy is lost. So we want small companies and we want them to stay that way, growing with the communities they serve. This means keeping the owner on as manager along with the staff he has built through the years.

Richardson was convinced that the place of the middleman was not only secure, but remained essential in certain lines of business. Among these were both pharmaceuticals and electrical supplies. It was not the product line that had primary importance, it was the service offered with the product. Where health is concerned, price is not the object; where complex circuitry or environmental controls are involved, knowledgeable advice, practical alternatives, availability of supply, and timely delivery are the most important considerations for customer loyalty. To maintain service levels, depth of inventory is important as well as having the right people on staff to describe properties of newly developed drugs with the pharmacist or to sit with the contractors and help them work up a critical bid. Richardson believed that service is the product.

Richardson was also convinced that "bigness" and computerization were synonymous. He was determined to avoid the impersonalized way of doing business that he thought computers brought to a "big" organization. While the rest of Amfac Incorporated had pursued a more aggressive stance on computerization, the Distribution Group remained with the green-eyeshade, know-your-neighbor operations that were the basis of its strategy. Richardson opted to require those distributors and others acquired by Amfac to persist in this backwoods approach to business.

Richardson was so persistent in his philosophy that Amfac Distribution made no attempt to capitalize on its size whatsoever. There were no programs for centralized purchasing,

or for large central warehouses. There was no concerted effort to standardize cash management or credit policies. There was no way of dealing with national accounts except on a local basis. There was no real way to manage inventory and market demand in different locations.

It wasn't until 1978 that the Distribution Group moved into the world of bits and bytes with some Burroughs B700 machines. These 8K computers were maintained and programmed by a staff of one. They handled the accounting function and provided some loose inventory control. In Richardson's one condescension to advancing technology, and, at the same time in his attempt to maintain the flavor of the small responsive business that he so greatly admired, B700's were placed in virtually all the branches. Deviation from this computerization policy was not tolerated. A few new acquisitions had to go through the embarrassment of converting from much more sophisticated systems to relatively simple, and minimally functional B700's.

The new computers were all batch oriented systems. The accounting data was current enough, but inventory reporting consistently lagged what was on the shelf by anywhere from two days to two weeks. Computer operators were employed to nurse data through these systems. The computer became a means to communicate the state of the business to Corporate, but not to the business itself. It was not capable of providing the timely information that was so valuable to these business operations. Nor were the B700's capable of electronic data communication, so the financial numbers from the branches were sent to headquarters on data cassettes. Month-end was a beehive of activity as nearly 300 cassettes flooded the corporate offices for consolidation.

Despite the apparently solid performance of the Distribution Group in the late 70's and early 80's. It was becoming increasingly evident that to remain competitive in sophisticated and highly competitive market place Amfac must take advantage of not only its size, but also advances in technology that its competitors were beginning to utilize.

The market place was beginning to change, to grow, and to move in directions that the mom-and-pop operations could no longer keep up with. The most dramatic of these changes was in drug distribution. Even though Amfac Health Care branches were only secondary suppliers to most of their customers, they could not even begin to offer the same services to the pharmacist and drug retailer that a primary supplier could. The primary suppliers were grabbing more and more market share by offering an abundance of value added services such as product stickers, shelf labels, remote order entry, and automatic reorder initiation. Amfac could not begin to approach this level of service without sophisticated tools to help it run its business and modern computing power to help customers run theirs.

Even though the Electric and Supply divisions served a customer base that was at that time far less advanced, the implications for what could happen to those business segments were very clear. If Amfac was not able to position itself to offer sophisticated services to a market place with growing needs, it would fail to provide the service that had become its trademark.

There were other advantages that could accrue to an innovative wholesale distributor. Centralized purchasing could save a tremendous amount of money on volume purchases. Those savings could be passed on to customers or could contribute to the bottom line. National accounts could be handled more consistently. In fact, all customers could be segregated by the volume of business into A, B, and C categories, to standardize pricing and insure appropriate discounting. With proper controls, cash management could become a real source of funds. Inventory might be managed more effectively, if branches were able to share such information with more ease and reliability.

Amfac Inc. saw that it could no longer afford to let the Distribution Group ignore the march of time. The notion of "service" in the market place had evolved beyond that which Distribution had routinely offered. It was with great irony that the man who touted "service" above all else failed to recognize the changing perception of what "service" really was and what real benefits could be had as a by-product in pursuit of satisfying the customer.

September 1983 marked the replacement of John Richardson with John E. Gomena, former Chairman of the Food Group. (The Food Group had long been considered one of the best run of the groups.) This year was also marked by a significant swing in profitability from $43 million in 1982 to a $9 million loss in 1983. This drop was due primarily to a $30 million write down in assets consisting mainly of inventories and receivables. Acquisitions reported a net loss. The already thin margins characteristic of the wholesale distribution industry were lower and expenses were 25% higher than the previous year. This was the backdrop that Gomena inherited.

The challenge that was before him was aptly stated by Gomena; The Distribution Group "will be the best run, people-oriented, most profitable distribution company in existence." Gomena promised that Amfac branches would be efficiently laid out and staffed; that employees would be trained in new systems and procedures; that training programs would be developed for all levels; that there would be communications and information to help in running the business; and that the Espirit D'Corps would be second to none. He added that Amfac must be the most profitable distribution company in any segment of business in which it competed. Big is not better, profitablility is.

Gomena also laid out a time table for accomplishing these rather ambitious goals. "1984 is a time to reorganize; 1985 a time to implement; 1986 a time to enjoy the fruits of our efforts." When one considers the size and scope of this undertaking one can begin to appreciated the enormity of the task evoked by this schedule.

The restructuring of the company was from the ground up. Thirty of the company's top forty managers were to be replaced, including the presidents of each of the divisions. The corporate offices, then located in Burlingame, California, were to be moved to a location with lower lease costs, lower housing costs, and lower commute times. This was necessary because the corporate staff was to increase eleven fold to over 200 professionals and the costs of attracting and relocating people to the Bay Area were simply too high. Operations were to be centralized, and management information was to be readily accessible for running the business.

The keys to the success of this restructuring were; 1) Correctly identifying the problem areas, 2) finding the right managers to head the divisions, and, 3) implementing sophisticated computer systems to help manage the business and to provide those value added services that were so important to the continued viability of the group. Another element was timing. It was critical to implement these changes in the least possible time. With a $30 million write down in 1983 and an additional write down of nearly $5 million expected in 1984, the company could not tolerate a delay.

The environment at Amfac up until 1984 was simply stated. There were essentially no systems. There was no systems staff. There was no management information. There was no means of supporting competitive business services. And finally, the basic needs of the business were not being supported.

In April 1984, Amfac Distribution hired its first data processing professional. The company was so new to the concept that it had trouble with the title, waffling back and forth with Manager and Director. To provide staff for this new function, Deloitte, Haskins, and Sells was hired on consulting assignment to put together a requirements definition and Request for Proposal based on site visits, interviews, and review by an Information Systems Planning Team made up of approximately 15 Amfac managers from various staff and line positions. Because this activity was concurrent with replacement of key players in the organization, the membership of the Information Systems Planning team was unstable.

It was in this environment that many of the most critical systems decisions had to be made. While there were interviews with representative branch sites and fairly thorough examinations of the business as it existed and discussions with the most forthright and forward thinking branch managers, whatever systems were selected had to be flexible enough to handle current business requirements as well as requirements that this management team might not even consider. The three new division presidents had not yet been selected. Their input would be important if the systems were to accommodate their own styles of management. The system architecture was not defined. Centralized management does not necessarily mean centralized computing. The business needs of the three divisions were quite different as well. A Health Care branch was simply not run the same way a Plumbing Supply branch was. The difference in the level of sophistication required for managing either business was night and day.

The Board of Directors met quarterly. It was important to lay before the Directors a proposal for meeting the systems needs of the business at the earliest opportunity. The proposal had to be accompanied by a capital budget request with substantiated, well documented supporting arguments. It was important too, that the proposal meet the schedule that Gomena had laid out, that the expenditures were predicted with more than reasonable accuracy and that the whole project had the ring of credibility and perhaps, more importantly, do-ablility.

The proposal that went before the board in July 1984 outlined the installation of about 120 computers, serving 286 branches, for a total capital expenditure of $26 million. The total implementation was to be completed by the end of 1986. This presentation was accomplished just four months after hiring its first ever MIS professional and having no other full time staff on the payroll.

The computerization of Amfac Distribution proceded at a full gallop. The first equipment order was signed in September 1984. The first Series 68 was installed in late November and was live December 10th. There were 74 more HP3000's installed in 1985, eight Series 68's, one Series 48, fifty-six Series 42's, and nine Series 37's. There were nearly 100 personal computers deployed in 1985. Overall, 109 branches were implemented and a systems staff of 120 professionals was established. There were 45 more computers to install in 1986. (12 more as of June 1, 1986.)

It is important to reflect here that these milestones were not accomplished without cost, both in a monetary sense and also from the shear weight of change in business methodology, information reporting, and organizational changes. Some of these changes will be explored below from various perspectives.

Because the MIS organization did not exist in prior years, it alone is perhaps the greatest contributor to the change in corporate persona and to the coming of age of Amfac Distribution. It is now a pervasive element at Amfac. The original MIS organization consisted of a Manager and a support staff of outside consultants. By late 1984, the organizational structure looked like figure 1, with a Vice President of Management Information Services and six section managers. (Figures 2-5 show current organization charts.)

The Managers of the Divisional groups were responsible for the implementation of the software at the branches. The Managers of Telecommunications and Hardware Planning & Operations were responsible for working closely with the divisions to provide the communications ability for the systems and "dial tone" at the branches. This was very much a matrix organization with no one manager being able to successfully function without working with at least two others. This fostered extraordinary communication among the MIS staff and kept the project on track with natural checks and balances.

What made this organizational structure possible was the fact that all divisions shared a common hardware base. The application software was different for each division, but it all ran on HP 3000's. Even though the divisions did not share a common system configuration, Amfac was able to affect tremendous cost savings in volume discounts and being able to take advantage of special promotions from HP, for example, purchasing 7933 disc drives in sets of three's rather than one at a time.

Staffing was a considerable problem from the beginning of the project. In order to meet the milestones of the project a large number of systems people was required. The schedule was so pressing that to slow the pace of the implementations enough to devote resources to hiring was very difficult. The volume of paperwork and interviews involved in hiring 100 experienced data processing professionals was staggering. Each candidate was interviewed by at least three people who based their judgments on expertise, experience and organizational fit. One person was hired for every four or five interviewed. For every person interviewed about ten resumes were reviewed. To get each of those resumes, advertisements were regularly placed in the Wall Street Journal, the Sacramento and Bay Area papers as well as key regional papers across the country. Literally thousands of candidates were considered.

Not only was finding candidates with the right mix of skills and fit difficult, but there were a number of aspects to the work that were not at all appealing. Many of the positions called for 90 or 100% travel, most of it to glamour spots like Lawton, Oklahoma, or Livonia, Pennsylvania. The project had a definite

lifetime -- complete implementation by the end of 1986. There was an obvious staff overloading required during the 3rd and 4th quarters of 1985 when the bulk of the implementations were scheduled. Amfac addressed this last issue by utilizing consultants or contract employees to fill the gaps. The number of full time employees was kept to a reasonable minimum to avoid layoffs at the project's scheduled conclusion. Employees were also screened for their adaptability to a stressful environment and a skill mix that would allow management to use them in some other aspect of the project if it were necessary.

Perhaps one of the most challenging aspects of the employee/employer relationship was maintaining morale of MIS staff and of branch employees undergoing conversions to the new systems. Most of the MIS professionals were excited enough about the project and their participation in it that personal gratification and job performance were inextricably woven together. For each individual, their role made a difference. One could see and measure progress everyday. Mis turnover was very low despite the stress, virtually non-existent in 1985.

Morale at the branches was quite another problem. The branch's first concerns were, Was this system implementation going to be another B700 implementation? Were they going to get proper installation help and instruction? Were the systems going to be supported after implementation? Were the systems really going to be of help in running the business? The time that the implementation team had to spend at each branch was necessarily short, 3-5 weeks. Their training had to be concise and to the point. They had to be knowledgeable in both the distribution business and the application software. Their first implementations had to be the most successful so the "grapevine" would be supportive of the automation efforts rather than serve as a detrimental detractor. The systems must prove their own worth in providing support to the business. Schedule changes dictated by corporate headquarters were met with extreme disappointment in some cases and equally extreme relief in others. By and large most branches were very eager and willing to do whatever they could to assist in the process. Branch morale has maintained a very high level throughout this project even though there has been a routine amount of frustration at various points during each implementation cycle.

There have been some significant changes at the corporate level (ADCORP). Prior to this systems project there had been no mechanism for spending money at corporate. The very first purchase order number was requisitioned in late 1984 to buy a purchase order tablet from the local stationery store. The second P.O. number was issued to Hewlett Packard for purchase of a Series 68. (Amfac has since become expert at spending money.) Another function that most businesses take for granted is

accounting. Because ADCORP had really never purchased anything on its own, it did not have an accounting department to keep track of such things or mind the payables. There was no standardized chart of accounts. Each division kept their own. There was no well defined mechanism for cutting checks, nor clearly stated policies on vendor terms or signature authority.

Because staff had never really been large enough to "administer" to, there was no personnel department, there were virtually no rules or regulations, no viable personnel policies, no policies for hiring or firing, and no written documentation regarding benefits. All of these had to be developed from scratch.

In December, 1984, ADCORP moved from Burlingame, CA to Folsom, CA. An Administration Department was created to administer the relocation of the business and about twenty employees. This department was also responsible for managing the new personnel and purchasing functions. A travel agent was brought on board to assist in making travel arrangements for 50-75 constantly traveling employees. A lease administrator was hired to assist the divisions in negotiating new lease space for branches. For the first time Amfac was beginning to function like the billion dollar business it was.

A concept that was very new to Amfac, and to most companies, is the Information Support Center (ISC). This department was established to help educate and train new users in the disciplines of personal computing, office automation and decision support. In the first full year of this project, nearly 100 personal computers were deployed. It was critical that all of this computing power be harnessed for useful and productive work. The ISC provided both classes and personal instruction to any users who wanted them. ISC helped develop Lotus applications and also performed more advanced database design work. For the most part the project could not have been accomplished without personal computers. Amfac went from the backwoods to advanced computer literacy literally overnight.

Change was not limited to the corporate organization. Each division went through extraordinary change as the new presidents each brought with them their own style and staffs.

The quickest to complete the implementation of its branches was the Health Care Division. Health Care topped the other divisions in revenues with nearly $485 million in 1985. The new president was hired early in 1985 to guide the division from its role as a secondary supplier to being a primary supplier in three of its major markets, independent retail drug stores, chain drug stores and hospitals. The implementation of Series 42's in all 38 branches was complete by April, 1986. Coincident with the system implementation, each branch warehouse was

significantly modified to take advantage of the old 80/20 rule - 80% of the orders are filled from 20% of the inventory. That 20% of the inventory was organized and localized into a few"fast pick" shelves and made easier to reach. New warehouse transport systems made picking and packaging more efficient. Without adding personnel, the branches are able to process upwards of 35% more volume.

The computer systems have allowed Health Care to compete as a primary supplier for many customers. Among the new value added services offered are shelf labels, product stickers, and remote order entry. Customers are provided with hand held terminals into which they enter orders. They then transmit these orders to a speech aided modem (SAM) for validation. Customers hear the computer say thank you when the ordered has been received.

The new systems have also let Health Care pursue different lines of business. The division has been very successful at winning new contracts from hospital buying groups. These contracts have traditionally been left only to primary suppliers.

These systems have made some significant changes not only in the way Health Care does business, but also the way it looks at the business it has. It may not be in the big leagues yet, but its suited up and ready to play.

The Electric Division had revenues of $471 million in 1985. It consists of 88 branches and they market a full line of wire cable, conduit, lighting, switchgear, and a host of other electrical products to the residential and commercial construction industry, retailers, industrial users, utilities, and government. Their product catalog is recognized as the most comprehensive in the industry.

Electric will complete their implementations by December, 1986. They are about three quarters completed now. Depending on the size of the branch it is serviced by either a Series 37 or Series 42. Those branches with Series 42's generally support a second or a third branch remotely.

Electric has instituted a consolidated purchasing program to take advantage of their size and presence in the market place. They have developed sophisticated matrix pricing programs to increase profitability. They have also begun to monitor goods that contribute only marginally to profit and are beginning to actively market those items to clear their inventories. They have created a business development department to consider such risk ventures as energy management systems and other new product lines. They have analyzed the accounting functions thoroughly to determine how best to handle the whole cash management problem, and as a result have begun to consolidate payables and

receivables at higher organizational levels. Electric is beginning to position itself as a true innovator and leader in the industry by offering remote order entry and stock checking by salesmen in the field. This is a value added service that indicates real change for the industry as a whole.

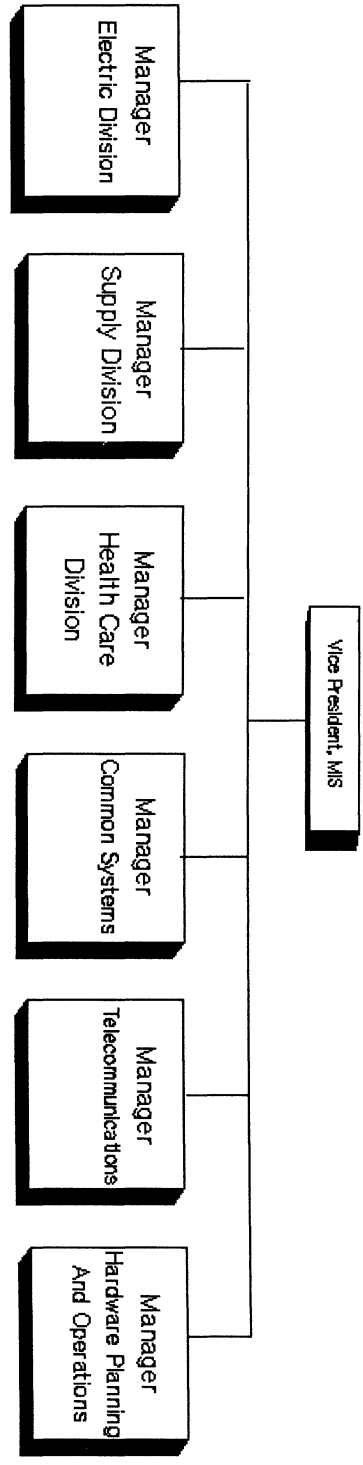
The Supply Division consists of 128 branches in 16 states and has revenues of $367 million. The industrially oriented pipe, valve, and fittings businesses are closely aligned with the same customer base as Electric, but are most closely affiliated with the petrochemical industry. The plumbing segment serves primarily residential and commercial construction market places.

The systems configuration for Supply consists of large regional and market area Data Centers with one or two Series 68's serving from six to twelve branches each. The branches typically have three terminals and a printer. This configuration provides for a shared database for those branches that are geographically related. It makes it easier for branches to sell out of each other's inventories, though this is typically not the case.

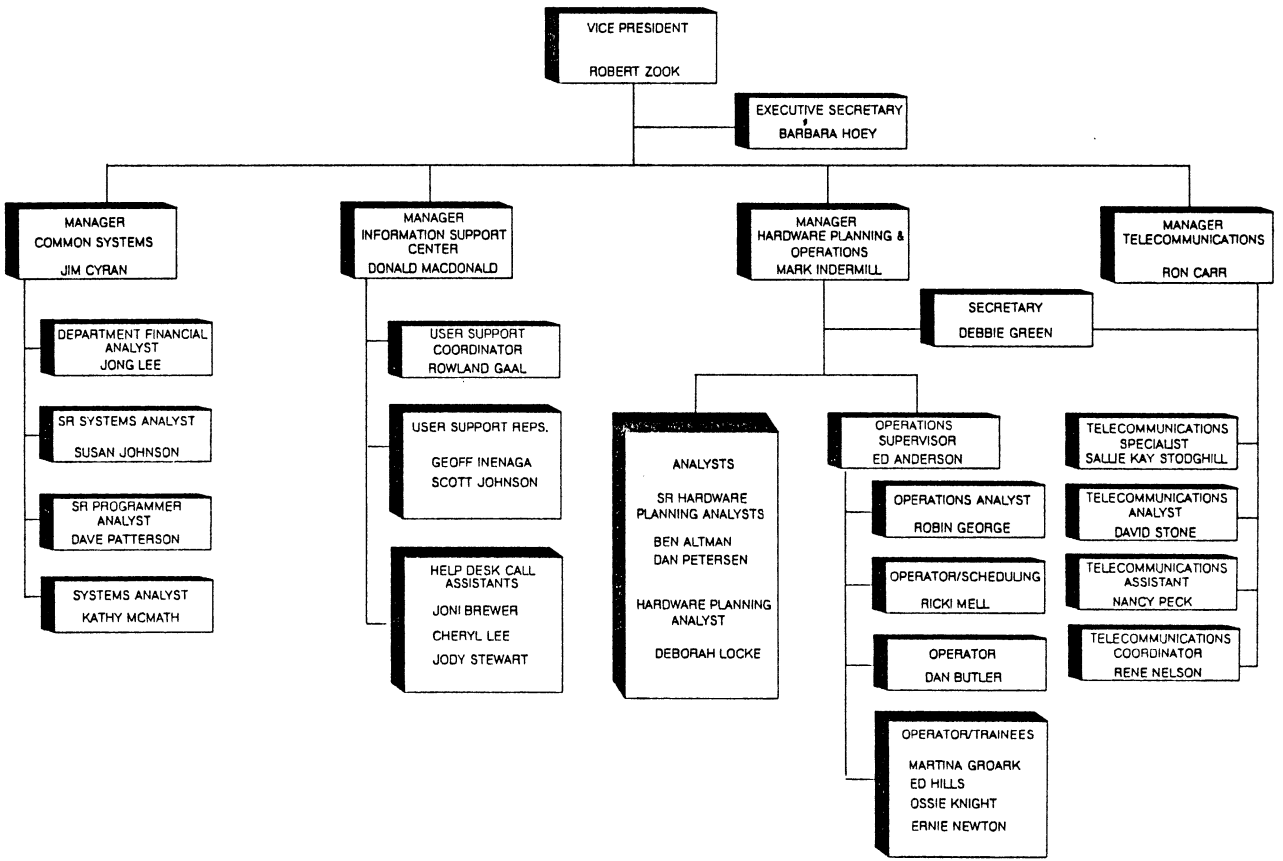
All accounting functions have been centralized at the regional offices. Tighter inventory control has been critical given the dependency on oil field related businesses. For Supply the computers have been used first and foremost to apply solid business controls where previously there were few.

The branches of all three divisions have come to rely heavily on a new entity at Amfac called "Help Desk". The Help Desk, modeled loosely on the HP Response Center, first served as a focal point for all application and system related problems. It has since grown to encompass inquiries about virtually everything related to the branch's business. On over 100 installed systems the call rate is about 5000 calls per month. The call rate alone is a good indicator for how reliant the branches have become on the Help Desk for information. Help Desk phones are answered 24 hours a day by real people. It is important not to forsake a user when help is needed the most.

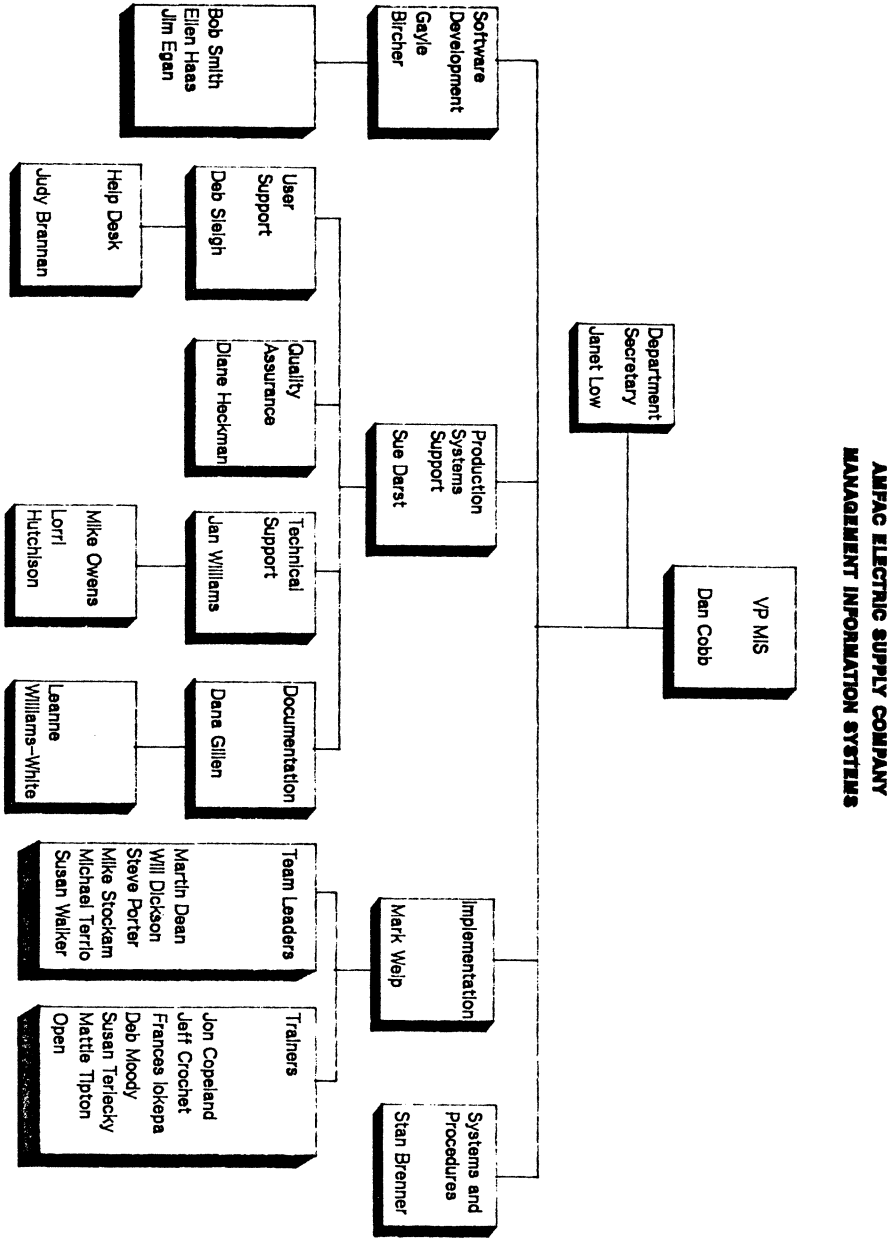
Twenty million dollars later, where does Amfac Distribution stand? This revitalization has had considerable impact on everything it has touched. The pocket book has been mentioned. The effect on the business has been equally deep. Amfac Distribution is no longer the sleepy little distribution company that it was up until the early 1980's. The concept of service has changed and Amfac has risen to meet that change with decisive action. The business has changed fundamentally. It is now a big business and it is operated as such. Not only has computer literacy become routine, information i.e., management information is vital to the control of business activity. The turnaround that began only 2 years ago has a long way to go, but the seed has been sown.

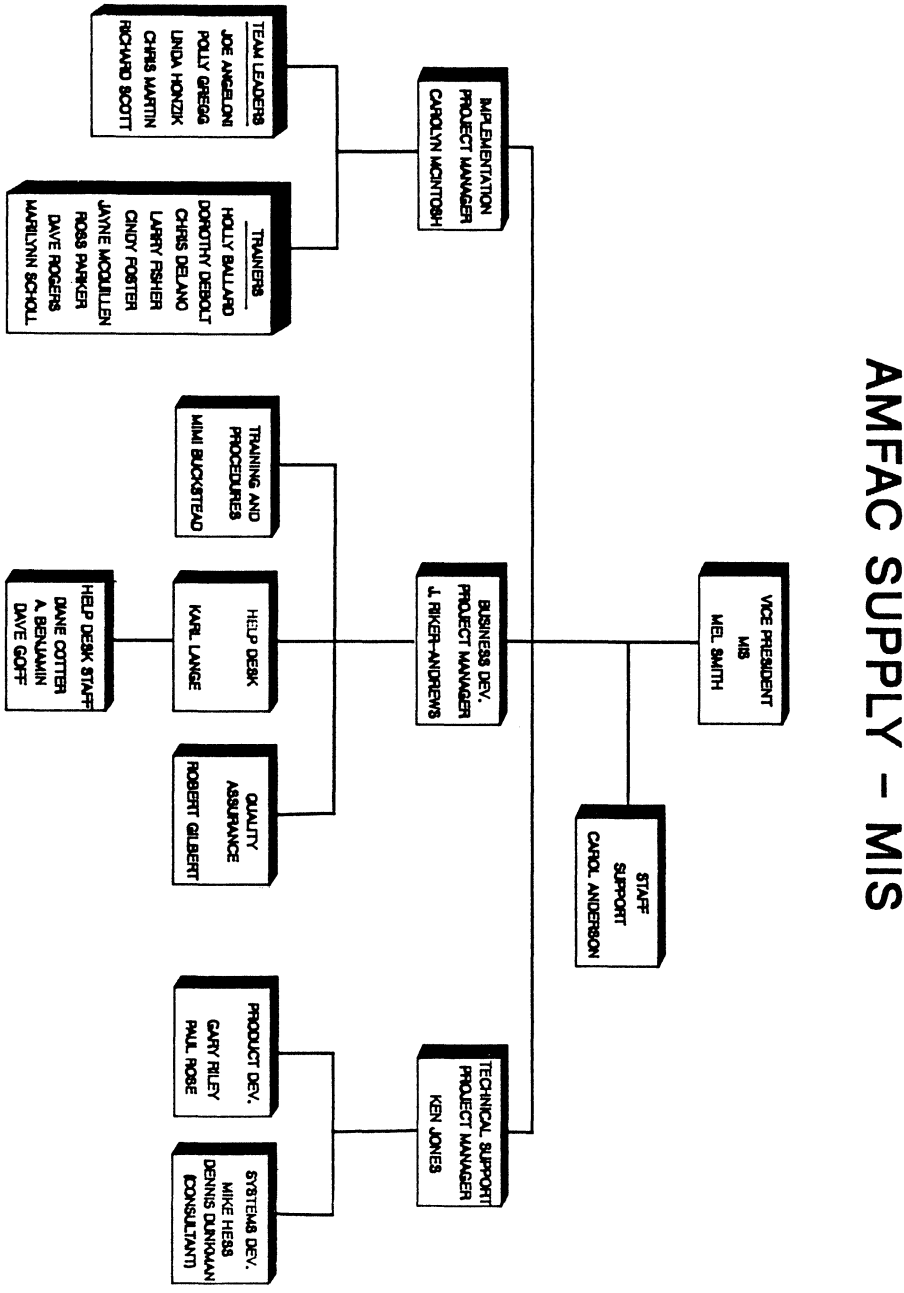

Vice President, MIS
Manager Electric Division
Manager Supply Division
Manager Health Care Division
Manager Common Systems
Manager Telecommunications
Manager Hardware Planning And Operations

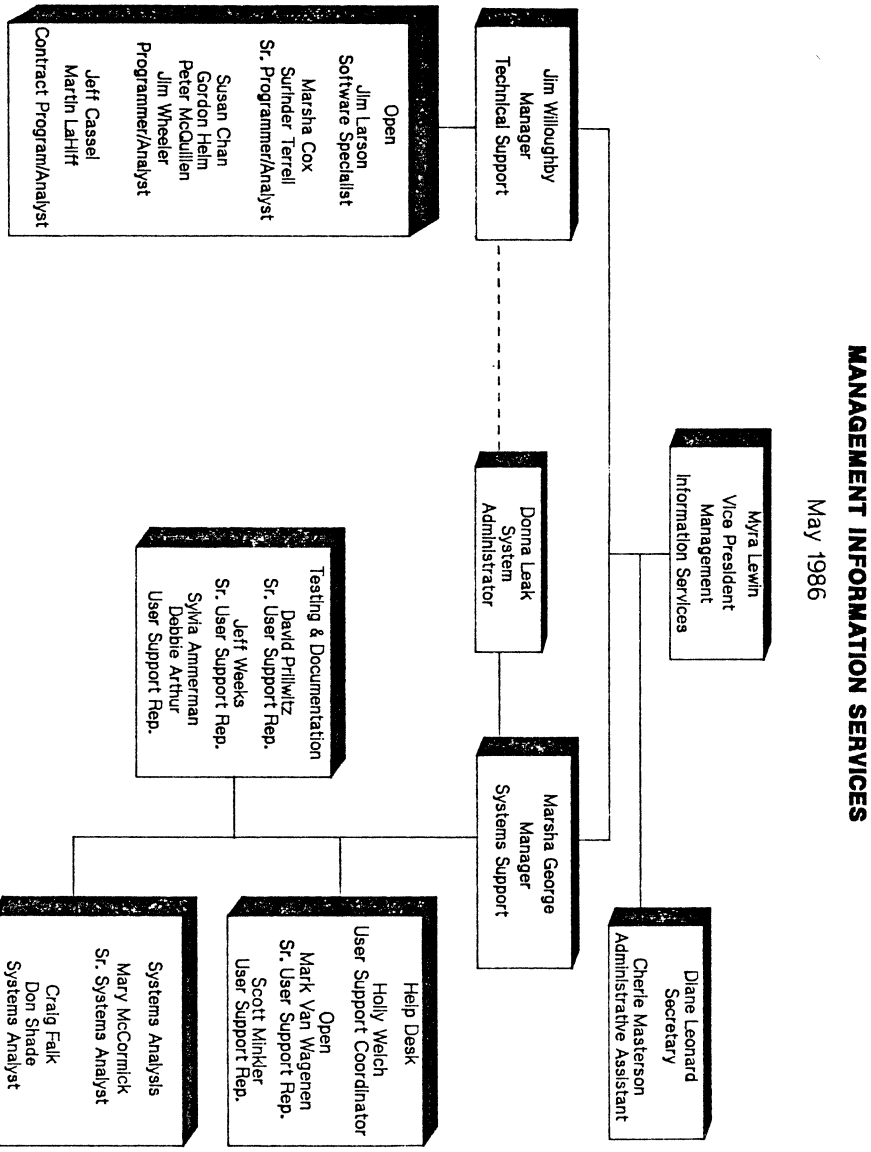
ADCORP - MANAGEMENT INFORMATION SERVICES



APRIL 1986


AMFAC ELECTRIC SUPPLY COMPANY
MANAGEMENT INFORMATION SYSTEMS
VP MIS
Dan Cobb
Department Secretary
Janet Low
Software Development
Gayle Bircher
Bob Smith
Ellen Haas
Jim Egan
Production Systems Support
Sue Darst
User Support
Deb Sleigh
Help Desk
Judy Brannan
Quality Assurance
Diane Heckman
Technical Support
Jan Williams
Mike Owens
Lorri Hutchison
Documentation
Dana Gillen
Leanne Williams-White
Implementation
Mark Welp
Team Leaders
Martin Dean
Will Dickson
Steve Porter
Mike Stockam
Michael Terrio
Susan Walker
Trainers
Jon Copeland
Jeff Crochet
Frances Iokepa
Deb Moody
Susan Teriecky
Mattie Tipton
Open
Systems and Procedures
Stan Brenner


AMFAC SUPPLY - MIS
VICE PRESIDENT MIS
MEL SMITH
STAFF SUPPORT
CAROL ANDERSON
IMPLEMENTATION PROJECT MANAGER
CAROLYN MCINTOSH
TEAM LEADERS
JOE ANGELONI
POLLY GREGG
LINDA HONZIK
CHRIS MARTIN
RICHARD SCOTT
TRAINERS
HOLLY BALLARD
DOROTHY DEBOLT
CHRIS DELANO
LARRY FISHER
CINDY FOSTER
JAYNE MCQUILLEN
ROSS PARKER
DAVE ROGERS
MARILYNN SCHOLL
BUSINESS DEV. PROJECT MANAGER
J. RIKER-ANDREWS
TRAINING AND PROCEDURES
MIMI BUCKSTEAD
HELP DESK
KARL LANGE
HELP DESK STAFF
DIANE COTTER
A. BENJAMIN
DAVE GOFF
QUALITY ASSURANCE
ROBERT GILBERT
TECHNICAL SUPPORT PROJECT MANAGER
KEN JONES
PRODUCT DEV.
GARY RILEY
PAUL ROSE
SYSTEMS DEV.
MIKE HESS
DENNIS DUNIGAN (CONSULTANT)


AMFAC HEALTH CARE
MANAGEMENT INFORMATION SERVICES
May 1986
Myra Lewin
Vice President
Management
Information Services
Diane Leonard
Secretary
Cherie Masterson
Administrative Assistant
Jim Willoughby
Manager
Technical Support
Open
Jim Larson
Software Specialist
Marsha Cox
Surfinder Terrell
Sr. Programmer/Analyst
Susan Chan
Gordon Helm
Peter McQuillen
Jim Wheeler
Programmer/Analyst
Jeff Cassel
Martin LaHiff
Contract Program/Analyst
Donna Leak
System
Administrator
Marsha George
Manager
Systems Support
Testing & Documentation
David Prillwitz
Sr. User Support Rep.
Jeff Weeks
Sr. User Support Rep.
Sylvia Ammerman
Debbie Arthur
User Support Rep.
Help Desk
Holly Welch
User Support Coordinator
Open
Mark Van Wagenen
Sr. User Support Rep.
Scott Minkler
User Support Rep.
Systems Analysis
Mary McCormick
Sr. Systems Analyst
Craig Falk
Don Shade
Systems Analyst

Archaeology of the HP 3000 Computer

Robert Green

Robelle Consulting Ltd.
8648 Armstrong Road, R.R.#6
Langley, B.C. Canada V3A 4P9

Did you know that the first HP 3000 was a complete disaster and was withdrawn from the market for a short time? Did you know that Hewlett-Packard originally planned on a 32-bit machine, not a 16-bit one? I worked at Hewlett-Packard in Cupertino during those turbulent times; this is how I remember the introduction of the HP 3000 computer.

Background. Time-Shared Basic (1968-69)

The 2000 time-sharing system was Hewlett-Packard's first big success with computers. The 2000 line was based on the 2116 computer, basically a PDP-8 stretched from 12 to 16 bits and still supported by Hewlett-Packard as the HP 1000. HP inherited the design of the 2116 computer when it acquired Data Systems, Inc. in 1964 from Union Carbide. The 2000 supported 16 to 32 time-sharing users, writing or running BASIC programs.

This product was incredibly successful, especially with schools. The original 2000A system was created by two guys working in a corner. Heavy sales of the 2000 finally pushed the computer division into a profitable state, produced a healthy cash flow, and generated an urge to make a contribution among the engineers and programmers in the Cupertino Lab. They said to themselves, *"If we can produce a time-sharing system this good using a junky computer like the 2116, think what we could accomplish if we designed our own computer."*

Abortive First Try. The 32-Bit Omega (1969-70)

The project to design a new computer, code-named *OMEGA*, brought new people into the Cupertino Lab, people who had experience with bigger operating systems on Burroughs and IBM computers. The Omega team came up with a 32-bit mainframe: stack-oriented, 32-bit instructions, data, and I/O paths, 8 index registers, up to 4 megabytes of main memory, up to 4 CPUs sharing the same memory and bus, both code segmentation and data segmentation, a high-level systems programming language instead of an Assembler, multi-programming from the start, and support for many programming languages (not just BASIC as on the 2000).

The Omega computer was designed to compete with big CPUs. But Omega looked too risky to management, so it was cancelled.

The 16-Bit Alpha. Big Ideas for a Little CPU (1970-71)

Hewlett-Packard seldom fires people. When Omega was cancelled, a few of the project members left HP, but most stayed and were re-assigned to the Alpha. This was an

existing R&D project to produce a new 16-bit computer design. The Omega engineers and programmers were encouraged to continue with their objectives, but to limit themselves to a 16-bit machine. Alpha was the Omega squeezed into 16 bits: 128K bytes maximum main memory, one index register, and Huffman coding to support the many address modes needed (P+-, DB+, Q+-, S-).

Same People, Smaller Hardware, Bigger Software

The original design objectives for the Omega Operating System were limited to multiprogrammed batch. The designers put off time-sharing to a later release, to be supported by a front-end communications processor. The cancellation of Omega gave the software designers another year to think of features that should be included in the Alpha operating system. As a result, the software specs for this much smaller machine were much more ambitious. They proposed batch, time-sharing, and real-time processing, all at the same time, all at first release, and all without a front-end processor.

To make the Alpha fit within budget and microcode restrictions, data segmentation had to be dropped. As a compromise, the expandable DL area of the stack was added. This strange feature, which provides for both positive and negative data addresses, is the one serious design error in the HP 3000. It makes byte addressing ambiguous and has led to numerous bugs in MPE (see my paper on *Byte Addressing* for details).

The instruction set of the Alpha was designed by the systems programmers who were going to write the compilers and operating system for the machine. The prevailing "computer science" philosophy of the day was that if the machine architecture was close to the structure of the systems programming languages, it would be easier to produce efficient, reliable software for the machine and you wouldn't need to use Assembler (i.e., a high-level language would be just as efficient).

The Alpha was a "radical" machine and it generated infectious enthusiasm. It had virtual memory, recursion, SPL instead of Assembler, friendly MPE with consistent batch and on-line instead of OS/360 with its obscure command syntax, segments instead of pages, and stacks instead of registers. The Alpha was announced as the HP 3000, and prospective users were assured that it would support 64 users in 128K bytes.

Harsh Realities. 200 Pounds of Armor on a 90-Pound Knight (1972-73)

The first inkling I received that the HP 3000 was in trouble came in an MPE design meeting to review the system tables needed in main memory. Each of the ten project members described his part of MPE and his tables: code segment table, data segment table, etc. When the total memory requirements were calculated, they came out bigger than the 128K-byte maximum for the entire machine.

The programmers worked seven days a week, 10 to 14 hours a day, attempting to get MPE to fit. Managers were telling their bosses that there was "*no problem*". They hadn't had a chance to "optimize" MPE yet. When they did, it would all turn out as originally promised. As the scheduled date for the first shipment approached, the Cupertino factory was festooned with banners proclaiming, "November Is A Happening".

The first HP 3000 was shipped November 1, 1972 to Lawrence Livermore Hall of Science in Berkeley, California. But it was incomplete (no spooling, no real-time, etc.),

it supported only two users, and it crashed every 10 to 20 minutes. The customers who had been promised 64 terminals and were used to the traditional HP reliability became increasingly frustrated and angry.

Eventually, the differences between HP 3000 reality and HP 3000 fantasy became so large and well-known that the product was withdrawn from the market for a short time.

Re-introduction. Struggling to Restore Lost Credibility (1973-1974)

Hewlett-Packard had no experience with bad publicity caused by low-quality products. Paul Ely was brought in from the successful Microwave Division to straighten out the computer group. First priority was to help out the existing users, the ones who had trusted HP and placed early orders. Many of them received free 2000 systems to tide them over until the 3000 was improved. Second priority was to focus the programmers' energy on fixing the reliability of MPE.

Once the HP managers realised (or admitted!) the magnitude of the 3000 disaster, the division was in for lean times. Budgets and staffs that had swollen to handle vast projected sales were cut to the bone. Training, where I worked, was cut from 70 people to less than 20 in one day. HP adopted a firm "no futures" policy in answering customer questions. If it wasn't working today, they wouldn't talk about it. The new division manager was strictly "no nonsense". Many people had gotten in the habit of taking their coffee breaks in the final-assembly area, kibitzing with the teams testing the new 3000s. Ely banned coffee cups from the factory floor and instituted rigorous management controls over the prima donnas of the computer group.

By continuing to work long weeks, the programmers managed to reduce MPE crashes from 48 per day to two and increase users from two to eight. Marketing finally took a look at what the 3000 could actually do, and found a market replacing IBM 1130s. HP no longer sold the 3000 as a souped-up version of 2000 time-sharing. But they could sell the 3000 as a machine with more software capability than an IBM 1130 and available to a number of users at once instead of just one. Eventually the 3000 became a stable, useful product. To my mind, this occurred when someone discovered the "24-day time bomb". If you kept your HP 3000 system running continuously for 24 days, without a SHUTDOWN or a System Failure, the internal clock would overflow and the system would crash.

The Comeback. Fulfilling the Promise (1975-76)

The original 3000 had a minimum usable memory size of 96K bytes and a maximum of 128K bytes. The Series II went beyond that 16-bit limitation by adding "bank" registers for each of the key pointers (i.e., code segment, data segment, etc.). Thus, the Series II could support up to 512K bytes, a much more reasonable configuration for MPE.

The Series II made one change that was not backward-compatible. LONG floating-point was expanded from three to four words. This caused the CHRONOS intrinsic to be dropped, replaced by the CLOCK and CALENDAR intrinsics. CHRONOS was the only MPE intrinsic that I designed. With that exception, the upgrade from the Series I to the Series II was simple and painless.

The choice of SPL as the machine language instead of an Assembler truly began to pay off now in an avalanche of excellent software: the IMAGE database (again, two guys

working in a corner) was soon joined by compilers for COBOL and RPG, a screen handler, and other tools to support transaction processing.

Concurrent, consistent batch and time-sharing was now a reality and the goal of concurrent real-time processing was finally dropped as unrealistic. The HP 3000 hardware now matched the software written for it. Business users discovered that the 3000 was great for on-line transaction processing; they dragged Hewlett-Packard firmly into the commercial DP world.

The years that followed were merely an unfolding of the original specifications. MPE C, II, III, IV and V were all attempts to complete what had been promised to the first users. The development of the 3000 was an **evolution**, not a **revolution.** At last, with the Series 64 in 1982 the 3000 reached the 1972 target of 64 users on a single machine. Many of the key contributions in this history were small, bootleg projects. Disc caching, for example, was again the work of two guys in a corner.

What Has Changed About the HP 3000?

Each new model of the HP 3000 has had more main memory, more computing power, more software, and more disc space. Users of the systems are now predominately commercial, not technical. The cabinet shapes have grown, shrunk, turned into desks, and turned back into plain boxes, but the computer inside the cabinet has always acted the same as far as the user is concerned.

What Has Not Changed?

The Series 70 of 1986 uses basically the same instruction set, the same SPL language, the same MPE operating system, and the same reliable IMAGE database as the original 3000A of 1972. Information is still moved around in 16-bit words and the program "stack" (the space available to store the data variables of a program) is still limited to 64K bytes. Amazingly, the price of a basic HP 3000 has remained at about $100,000 through the years, despite the intervening inflation and improvements in technology.

Despite the numerous expansions and revisions to the HP 3000 hardware and software, upgrades have been painless. Often the user just rolled the new system in on a Sunday, plugged it in to the power, reloaded his files, and resumed production. The original 1974 MPE Pocket Reference Card is still useful; everything on it works under MPE V, except the Cold Load instructions. I have programs that I wrote in 1972, for which the source code was lost years ago, and they still run! The key words that describe the history and appeal of the HP 3000 are: **flexible, functional, and compatible.**

About the author:

Robert Green became involved with Hewlett-Packard when he found that his degree in Ancient Greek Philosphy did not lead quickly to a paying job as a philosopher. He worked at the HP factory as a junior programmer, technical writer, training instructor and course developer. Robert left HP to venture into the real world, trying to solve the DP problems of a distributor on a Series I. In 1977, Robert Green formed Robelle Consulting Ltd. and began developing software tools, the best known of which are QEDIT and SUPRTOOL.

How To Design A Chinese Style Network System

by: Hwang, Hen-Jey

We regret that this paper
was not received for
inclusion in these proceedings.

Power to the People: Experiences of a Start-up DP Manager

by

Clifford W. Lazar
President, SystemsExpress

15015 Ventura Blvd. * Sherman Oaks, CA * 91403

Table of Contents

He who learns from experience is a smart man;

He who learns from the experience of others is a wise man.

--Benjamin Franklin

The purpose of this presentation is to pass on experiences that could help new DP managers to avoid the chuck-holes that pock the road of data processing.

While I have had an exciting career in data processing I haven't had all the experiences that there are to be had. I strongly invite anyone else to contribute to this memoir and thereby contribute to the knowledge base of the whole community. I promise attribution for any experiences and lessons contributed and included.

This will be a living document to which I will add more of my experiences and those of others.

My mailing address is 15015 Ventura Blvd. Sherman Oaks, CA 91403. My phone number is 818/907-4800.

Selecting the HP 3000 in 1979 -- HP vs. IBM

My idyll with Hewlett-Packard began the right way. My company needed a marine payroll and a purchasing system. The size of the problem and the technology, then available, indicated we needed a minicomputer or reliable access to the company mainframe.

At that time, 1979, the company IBM mainframe was crashing four times a day. Every time I approached an IBM user for a demo the machine was UNAVAILABLE. HP 3000 series III's were crashing about once every three months. One of my potential users, a controller, asserted that if we went mainframe she would stick with her manual calculator.

Our search for software disclosed the MAP package by Frank Greis and a payroll system by Collier-Jackson. The software looked like the beginnings of a solution.

The alternatives to the HP 3000 were the IBM 43XX and the IBM Series 1. There was no Cobol, then available, on the 43XX and the Series 1 was not a business machine--a single user could over-write everyone else's memory spaces.

The combination of cost, reliability and software availability drove the recommendation towards the HP 3000. The next problem was bringing management along in a company that was committed to IBM as a computer supplier--as a provider of computer solutions.

Educating Management: CYA!

CYA means cover your ass. In any company big enough to afford an HP 3000 there will be enough politics to require a DP managerto cover his ass--to protect his hind side from attack. A story that had been circulating at my company recounted that a DP manager, in an east coast location, recommended an HP. The next day an IBM director who lived next door to a senior company executive demanded to know who the idiot was who ordered an HP computer. The idiot was fired and the HP deal was scrapped. That was an impressive story.

As I researched the alternatives I issued status reports that telegraphed my decision. In staff meetings I warned my Vice President of Planning and Control of coming political pressures if we went HP. At the same time I pointed out that it appeared that HP was the only choice if we were to get our payroll online, on schedule. On schedule meant meeting a union-negotiated deadline to deliver checks.

By the way, my VP later had to be reminded about my warning of the political pressures to come.

Management agreed to go HP and that was the beginning of a long series of problems and solutions.

As an indication of the company's commitment to IBM as a solution was the fact that when I ordered the company's first HP 3000 the corporation reserved 51 IBM 43XX's.

Acquiring the Software and Hardware

We bought the HP and the software from an OEM who also signed a fixed price contract to modify the purchasing software.

Failing at No Surprises

A failure of management is to promise a level of performance for a cost and under-achieve the performance or be forced to go back to the controller and ask for more money.

We put pressure on the OEM to assure us that the configuration of hardware would do the job in the RFP. We needed accurate costs for the capital request. Well, the HP 264X is like a Barbie Doll and it required one more board than originally estimated.

The best way to avoid such failures is to only buy clones of hardware/software systems, used by other companies. Two factors militate against this convenient choice:

1. Your problem is always unique.
2. Clones are copies of older configurations that usually cost more.

So what happens if you want to configure a cost-effective, one-of-a-kind HP system: NO ONE KNOWS HOW TO CONFIGURE A ONE-OF-A-

KIND-SYSTEM. No one understands the configuration manual. Three years ago HP couldn't tell us what the maximum connections were for a Series 44 or how it would perform if we maxed it out.

One of the current mysteries of the HP is how much main memory is the right amount. Too many megabytes can bog down the memory management system by giving it too big an area of free space to manage.

Still another truism should bear upon your decision to be out there in front:

Pioneers get arrows in their backs.

Economic Justification of the HP and Add-ons

Granted that the mission to do payroll and on-line purchasing were adopted by management, justifying an HP when compared to reasonable cost estimates on the IBM mainframe was very easy. Five years ago. With more agressive IBM pricing this may no longer be true. Hypothetically, the mainframe should be cheaper than a minicomputer. The gigabyte disk drives cost less per megabyte than HP. The CPU costs less per MIP.

So why could an HP come in for less. First, IBM processing overhead. The MVS or VM operating systems probably consume 50% of their CPU power in overhead processing.

Second, IBM support overhead. IBM mainframes require a small army of systems programmers. Because the projects were generally so large, the IBM way required volumes of preliminary documentation:. Volumes of in-process documentation. Volumes of heavy maintenance documentation.

Contrasted against the IBM environment was the smaller, and more efficient, HP environment. There were no system programmers required. The jobs, even though spec'ed out and contracted, generally did not involve the rigid development cycles of IBM.

A significant comparison was that my HP shop cost $41 per user-connect hour--with all costs considered--versus $96 per user connect hour for a competing IBM 4341--with many costs hidden.

Lesson: Our survival and growth were supported by our apparent efficiencies. At the same time the efficiencies created jealousies and antagonisms on the part of the IBM sales force and supporters.

The Computer Room: Power Plays for Space

The scarcest resource in a large corporation is not money or people, it's space. A division president has signature authority to spend $10,000,000 or hire hundreds of people, but he could not get one extra square foot of space without securing Executive Committee approval.

When you are starting out with a new computer department, you need new space and you find yourself fighting for the scarcest resource in the company. When I made my request for modest space, a 11 by 12 foot room, I was put off. I went through the proper channels and got nowhere. Finally I approached the manager responsible for getting out the marine payroll checks and told him "No space, no checks." Then I got the space. The next problem was installing the power and the wiring.

Adventures Installing the Wiring

HP terminals only require connecting pins 2,3 and 7 (read, write and ground). It's simple enough, except few electricans have computer wiring experience and fewer have wall plates for DB-25 plugs. Also there is the usual confusion over what sex the plugs should be and where.

The official company contract electrican didn't understand computer wiring and wanted to be paid a lot for his on-the-job-training. The company contractor-coordinator told me to wait until the main electrical work had been installed and inspected and then bring in my own contractor.

That's what happened. My man installed phone wires in the plenum of the air conditioning system with numbered labels on both ends of the cables for identification. The cables were tested and we went on the air. Everything was fine until a concerned citizen (read SOB) called in the city electrical inspector and showed him my wires without conduit.

The inspector issued an order to pull the wires within 24 hours or he'd shut the building down. We pulled out $2000 worth of wire. It made a four cubic foot pile of pink wire.

I hired the offical electrical contractor and we wrote specs that said the conduit must only have gentle turns -- no 90 degree angles. I got a lot of 90 angles and a lot of wires without labels on both ends. The cost of the new replacement wiring was $10,000.

I did have a policy of installing three times as much cabling as the current need. This paid off within months. In the long run it still wasn't enough.

Mysteries of RS-232

We had a central site for the HP 3000 and remote sites where the users were concentrated. We installed a matched set of HP 2631B printers at both remote sites and connected them as follows: CPU, ADDC, Cable, Modem, telephone line, modem, cable and printer.

We expected them to work and to work the same way. The unit in the south remote would run fine for about twenty minutes and then it caused the whole system to freeze.

We would run the same job on the north site printer and not get the failure. We brought in HP, the modem vendor and the phone company and a room full of test equipment. They all said it was the other guy's hardware and went home.

We could recover from the freeze by turning off the wall power to the CPU. One night I was walking next to the North site printer and noticed that it was a blue Inmac cable. It had three lines: 2, 3 and 7. I called my operations manager and asked him to look at the grey cable that the contractor ahd installed at the south printer site. It has lines 1, 2, 3, 4, 5 and 7. I told him to cut lines 4 and 5. After that we suffered no more failures. The apparent cause was a signal the printer sent the modem which was passed through to the ADDC which told the CPU to wait for another transmission. It waited and waited.

Lesson: Don't install more pin connections than are necessary. Don't assume that two cables act the same if they have different pin connections.

Ungrounded Power

Ungrounded computers, that use 208 volts, are extremely dangerous and HP recommends that a seven inch copper spike be hammered into the earth as a ground. The offical electrical contactor was given the HP specs. They were in the work order. However, the contractor simply connected the ground to a painted pipe with flexible metal strap. As a result we had a floating ground for a year. No one had inspected the ground.

Later the same contractor installed a Topaz power conditioner and surge protector. The contractor installed the unit on the far side of the air conditioner so that every time it came on the surge went through the HP.

The Topaz paid off within six months when lightning struck a nearby power pole. Every tube in the place lit up but the Topaz breakers popped and the computer was saved.

Lesson: Don't believe that the contractor cares about your specs and don't believe they will be followed.

This lesson works two ways: First, if you don't check things for yourself they probably won't happen correctly. Second, in a large corporation you are supposed to delegate that to the designated department and your looking over their shoulder won't be appreciated.

In spite of the technical problems we got the HP up and running in a very short period of time--especially compared to our IBM counterparts.

Beware of the Painters

A year later we got permission to expand our computer room by knocking out the adjoining wall and putting in a sliding glass

door. I got a call from the operator to come check on what was happening. The painters had thrown plastic sheets over the disk drives and had begun to spay paint the walls.

Two things could have happened: the drives could have overheated and the platters could have suffered paint damage.

I stopped the work. We shut down the machine and let the work continue for an hour.

Lesson: Don't let contractors schedule rennovation work without your control.

The next issue was training the users.

Training the Users

Our initial training was one-on-one training with the entry clerks who would be loading the files with employee payroll data. Henry Kramer, the payroll programmer, and now the programmer of :DBEXPRESS was principally responsible for our success. Also critical to getting running, was Rita, the main payroll clerk. She entered over 50,000 characters of data with only three errors.

One of our tricks was to teach the users how to play some of the games on the HP because it reduced the stress and made learning keyboard skills fun.

Later we began to give classes in MPE, EDITOR, QEDIT, Math/3000, and Basic. Ron Ellis, then on our staff, lead that effort. He was responsible for providing hundreds of user training hours.

The user training hours were extremely important to our policy of leveraging our programming staff by having the **users** do as much as possible. It also made the users more understanding of inevitable problems.

Installing User Tools

I have to make a distinction between user tools and programmer tools. Users have different thought processes than programmers. They get frustrated sooner, they don't understand processes as easily in a computer context and they blame malfunctions on the DP department, and not themselves.

We installed Math/3000, Qedit, Inform/3000 and Basic and had EDITOR available. To my knowledge no user, even though trained, ever wrote a Basic program to do anything. Our biggest hit was Inform/3000. Users, without any programming background, could generate listings of their data from the databases we had built for them. Bob Murray, of the purchasing department was our biggest user and fan. Lee Richards of the Marine Controller department was also a major user and fan.

Unfortunately Inform/3000 requires the Rapid Dictionary, and as a

result, means that you have to spend $9,000 before you get your first user-generated report. SystemsExpress is about to release :DBReport which provides the ease of use of Inform/3000 but doesn't require Dictionary/3000. It will be priced at $1,000.

Math/3000 was also a major success. Math/3000 re-created the functionality of VisiCalc on the HP 3000 for multi-users for less than VisiCalc cost on the microcomputers in the early 1980's.

I trained a naval architect in its use and a week later he cancelled a $10,000 consultant project and did the entire job himself, in a few days. At that time, Math/3000 cost $200. It now is sold by Tymlabs for $900, in a much enhanced version.

The Marine Department also used Math/3000 to project demand for oil tankers and its requirements for chartering in or out--a multi-million dollar a year decision.

Management Comfort

While users are important, in a corporation, they are not the most important consituency. Keeping your manager's comfort level high is your highest priority.

Your manager's comfort level will go down if anyone can claim your system lacks control or security.

Building Security

Our first major projects were payroll and purchasing. Both projects have major opportunities for fraud and theft. The HP security system of user, group and account passwords do precious little to maintain security.

The company's auditors began asking that we change the passwords every week and use nonsense combinations of characters and numbers as passwords. No one could memorize such garbage and so you would find their passwords written on slips of paper, stuck to the terminal. Auditors could walk into many offices and sign on as the user and gain access to critical data.

I felt we could do better and so contracted with VESoft Consultants to write Security/3000, which gave each user multiple and secret personal passwords, such as "Your favorite ice cream or the color of your first car", that only the user would know. Eugene Volokh also programmed the system to store the passwords in one-way cipher, so even the systems manager couldn't know the passwords, as he could using the the HP password system.

The major advantage of Security/3000 was that it gave us integrity of user name. JOHN.PAYROLL was really John! As a result we put the username in with any new vendor or employee as a sponsor. Ghost vendors or employees were a major source of potential fraud.

When users could be held responsible for all acts committed under

their username they would be very unlikely to pass around their passwords.

Living with Auditors

An early event in my tenure as a DP manager occured at a controllers meeting that I wasn't invited to attend. The Corporate controller, who had dotted line authorty over all the division controllers and direct authority over the in house auditors, stated "That Lazar is a maveric." In the next three years my shop was auditied seven times. We survived all the audits, but they were time consuming.

I discovered that DP auditors are smarter than the average corporate employee and far more dedicated to truth and justice. I also discovered that they look for problems based upon a check list and that they have interim audit memos already prepared on their word processors.

It saves a lot of time if they can find an infraction that fits the memo. In one case they cited my computer room for not having an "Emergency Exit" sign over the exit. When I pointed out that they didn't cite the men's room, which was the same size and also only had one exit, they withdrew the memo. I couldn't evade not having a no smoking sign.

Lesson: HP 3000's are meant to support business functions and so are justifiably subject to audit. A new DP manager shouldget hold of a DP auditor's check list and build his security accordingly. This includes locking up tapes, having fire extinguishers and training in their use and having a disaster recovery plan.

Tape Backup

One gap in my security that somehow evaded the sleuthing of the auditors was an early failure in my tape backup policy. We had a policy of backing up changed files every night and the whole system once a week. In the case of databases, saving only changed files is not enough. The fourth weekly tape was to be kept for a year and quarterly tapes were to be kept for seven years. Once, when I was looking for an archive tape and couldn't find it my operator explained, in order to save the department money she was recycling the monthly tapes every year.

Lesson: Have a written tape backup policy. Check periodically, that it is being followed. Keep your monthly and yearly tapes off site at a repository.

Emergency Shut-down

The auditors required that we install a manual emergency power shut-off button and an automatic over-heat power shut-off. By now the company had administrators who handled such mundane issues. One day I arrived at the computer room to discover a three inch diameter red button mounted outside the computer room door in the hall. I pointed out to management that any yo-yo

could hit the power off button. It was moved inside the room.

Later that summer the sun came in the window and shined its face upon the over-heat sensor and off went the machine. We installed reflective window blinds and over-problem went away.

Lesson: Review modification plans before the contractors start.

Thanks to HP we had the systems up and running on schedule and below budget. Our next concdern was the attention we were receiving from the IBMer's.

The Attack of the IBM People

In many cases the difference between HP and IBM shops is a difference in culture. IBM shops are run very much in tune with the culture of large bureaucratic corporations.

HP shops are more like small businesses.

The IBM sales organization has been working closely with senior corporate management for years. IBM account managers, who are very well paid, are in the same country clubs, serve on the same charity boards and live in the same neighborhoods as corporate VP's, controllers and MIS czars. They have been cultivating the relationships since the 1950's, in some cases they have helped incumbents get promotions.

IBM sponsors computing technology briefings, attended by senior management. The theme of these seminars usually was, "You can trust IBM to provide your company with the solution."

Surprisingly, another theme was that the complexity of the IBM systems exceeded any company staff's ability to assure throughput, but that IBM would analyze your problems and always pull you through.

These briefings occured every three to six months. The result was a lot of contact, a lot of credibility and a lot of sales.

There have always been better and cheaper alternatives to IBM but they involved risks. If IBM failed in its promises, it was IBM's fault. Almost an act of God. If HP or Burroughs or some other lesser company failed, and you had chosen them, it was your fault.

The IBM Trojan Horse -- The BSP

During a competition IBM is relentless. The manager of a major IBM-using department at a major corporation walked into his office, one day, to discover the IBM sales executive sitting at his desk, going through his in-box.

That was an intelligence gathering activity, in what could be considered a major campaign. IBM never gives up, even after a competing brand has won a competition.

To promote sales of IBM equipment, IBM offers a stacked deck analysis called a Business Systems Project, the BSP. The central, and reasonalble assumption of the BSP is that company information is a company asset that needs to be controlled, collected, protected; shared, as appropriate; stored and accessed on a computer--preferrably an IBM computer.

If management buys off on the first part, about the information asset, and who wouldn't, then the BSP begins. IBM contributes one or two free information analysts and the company dedicates 5 to 20 middle to senior managers to the study.

The study is massive and time consuming and the results are massive and confusing. The core of the study is a survey of practically every information provider and user. They tell what they need and when--all good stuff. They are also asked what is wrong with the current (HP) operation/management.

After months of diligent effort, the team issues a 500 page report that includes a giant information sources and sinks matrix, that would give credibility to Marcos' election, a prioritized systems development project slate and a litany of what's wrong with current (HP) management.

The individuals who signed off on the report each worked on a separate section. Only the chairman and the IBM advisor were involved in all aspects of the study because they were the only real full-time members. It was a team project, however, and the team was built from each of the major DP user departments. It was hard work, with a lot of study, effort, late nights, comraderie and "team building." All the members supported the report, though, if confronted over a specific, they didn't write that part--like a firing squad with one blank bullet.

Lesson: Don't let IBM develop a BSP about your shop. A preferred approach is to maintain your own permanent revolution and have a loyal staff member perform a BSP for you and report to you. If you letyour user feel you don't have their interests at heart and that you are not continually trying to improve the service the IBM rep will be there with the BSP and you will have it imposed upon you.

Lesson: Don't narrow your focus to delivering on your projects or keeping your systems efficiently tuned. Within six months of your startup, your supporters are focusing on their own problems and your enemies have re-grouped. Also your company's needs have evolved away from your currently perfect match.

User Councils

A useful, but not sure-fire, way to keep in touch with your users and give them a feeling of participation is a user council, with the DP manager as the chairman. This is a good forum for debating priorities and to involve them in your problems. This is also a good place to have the users understand and support your

need for new hardware and software. Ninty minute meetings, once a month are enough. Have an agenda, use slides and serve coffee and donuts. It's also a good idea to have one or two users do a show-and-tell about their projects.

Interfacing with the Mainframe

When the HP 3000 was originally selected one of the screening criteria was that it should communicate in the IBM network. At that time, SNA was just being implemented and bisync was the common IBM method of doing remote job entry (RJE) processing. The HP can do RJE under bisync just fine. Unfortunately, HP doesn't give you a solution, they give you a problem. What HP did for us, was steer us to Hughes Electronics who were gracious enough to actually give us copies of working code.

For us it was all new stuff and a painful learning experience. My number two man was assigned the responsibility of setting up the RJE connection with the IBM mainframe. After weeks of failure we discovered that he didn't know that 2780 and 3780 protocols were different. After that, everything went fine. Subsequently, he became a company expert in tapping the information on the IBM mainframe, bringing it to the HP 3000, loading it into Image and Dictionary/3000 or flat files and selling it back to mainframe users who couldn't get at it as fast as he could.

Buying Solutions with Money

One of our major projects involved collecting data via space satellite, loading it into the HP, formating it, transmitting it to the Mainframe for corporate processing and retrieving the processed images and printing them on the HP 3000. The corporate organization that we had to deal with was proudly one of the most bureaucratic of all the corporate units. We were up against a critical deadline, and they chose to engage in negotiations that never seemed to lead to a conclusion.

One of their tactics was to never have decision makers and technical people in the same room. If we met with the technical people they would say they couldn't make a decision. If we met with the decision makers, they would say that they didn't have the technical input and couldn't make a decision.

Finally, the decision makers said they didn't have enough programming support to create a gateway for us to the data that we needed. At that point, having lots of money in my budget, I offered to hire them all the consultants that they might need to get the job done on time. Suddenly, they had the assets required and the job got done.

Problems with Flakes

The computer business is a new business and, as a result, is full of adventurers, trail blazers, con artists and flakes.

Included in the flakes that a DP manager has to deal with are

Consultants,
Head Hunters, and
Employees.

Consultants

Consultants are hired because the department lacks the staff necessary to complete the mission on schedule. Because of this, consultants get more money per hour than employees. Some consultants are worth their weight in gold and some aren't.

One consultant that I dealt with objected to using productivity tools because he didn't want to get fewer hours. Later I discovered he was shorting us on his time because he was working two jobs at once. A review of his performance and comments by people who were nearby led us to fire him. While my second in command gave him the bad news I changed his passwords, thus thwarting his next move, which was an attempt to purge his work-in-progress files.

Head Hunters

DP managers get calls from head hunters every week. Most head hunters will send you anyone that they are lucky enough to come in contact with, regardless of the match in requirements and skills. Often no screening whatsoever is done. That is what you are paying a head hunter for.

Lesson: Have applicants bring samples of their code. It's more revealing than a resume.

Lesson: Warn head hunters that they get only one chance to send you an unqualified applicant. And stick to it.

Employees

One of my key employees was turning out very impressive status reports, every week, indicating substantial and timely progress. I visited him, and requested to see the reports that he said he had programmed. There weren't any reports. There were no programs, just itemizations of reports.

I requested that he develop a work-around plan to get back on schedule and he refused. Corporate employee relations being what they are, three weeks later, a security guard and I escorted him to the gate.

Lesson: Don't believe programmers. Don't leave them alone to do their jobs. Break all work down into lumps that can be completed in two weeks. Have code walk-throughs -- even if you don't understand that language. Demand structure and comments so that the main ideas are understandable to anyone.

Another employee had the problem, even though he was very competent and professional, that when he ran into an obstacle, not

covered in the manual, he would call PICS and then stop work. He didn't do work-arounds or experiments.

Lesson: Don't assign uncreative programmers to exploratory type tasks.

Lesson: Both the consultant and the terminated employee worked at the computer site while I worked at headquarters. Programmers must be supervised by a manager on site.

I took the responsibility of designing, furnishing, installing and assigning office space.

Lesson: Programmers care very little about the quality of their working conditions, but they care alot about the quality relative to their colleagues. Avoid office facilities where the offices differ in size and amenities.

Lesson: If management agrees that you need space let the space department have the whole responsibility for design and installation. Assignment is still a management problem.

Promises, Promises

When you are starting up a new shop there is a lot that you can do for your users, and a lot you can fail to deliver on, if your raise their expectations too high. There is nothing you can't accomplish with seven diligent, hotshot programmers and theother resources you need.

Sometimes, in order to get the resources you may be tempted to do what Teddy Roosevelt did: Send the US Fleet half-way around the world and let Congress figure out how to get them back.

That is equivalent to promising information systems for everyone, using their appetite to push management and then only delivering half the jobs to half the people because money is short or you're out of sync with the budget cycle.

If you're the president the worst that can happen is that you won't get re-elected. If you're a DP manager, you could end up writing RPG reports on an IBM 360/40.

Administrative Support

According to the consulting firm of Knowland-Norton, most DP shops have a similar life-cycle. They begin with a senior programmer or an entrepreneurial manager who goes through a phase of rapid and euphoric growth. This manager has the characteristic of John Wayne, decisive, fast, blunt can-do. Like Napoleon or Hitler he'll march into Russia and like them he'll find himself bogged down in the winter snow. With a myriad of operations launched, he will be extended too far and the auditors will be able to show that he is "OUT OF CONTROL".

The next stage of development finds senior management bringing in

an assistant controller, who may know nothing about computers but a lot about control. Everything slows down. The job loses its appeal because administration becomes more important that information.

Lesson: Find an administrator to work for you before you work for an administrator.

DP Costs as a Function of Company Budget

Surveys have shown that DP budgets vary between 1% and 5% of a company's total budget. If you are closer to 1% you don't have the administrative infrastructure that you require to satisfy the demands of most corporations. If your are running at 5%, people will probably be able to prove you're gold plating your computer room.

Data Processing is high technology and requires a reasonable appearance of professionalism. That means you should buy good furniture and accessories and don't try to save money on minor items like file cabinets or shelving. The computer operation should reflect the standard for the rest of the corporate culture or you will be suspect as one not of the inner circle, who can be dispensed with, when it gets time to spend some serious money.

Conclusion

I have attempted to capture some of my experiences and observations in an attempt to help new DP managers both avoid some of the likely pitfalls that are inevitable with the territory but, also hopefully, to prevail.

Again, if anyone has more gems to add to this collection I'll be pleased to include them, with attribution, in a future release.

THE SECOND MARKET: BUYING AND SELLING USED HP EQUIPMENT

BUCK BUCHANAN
FIDELITY SYSTEMS
11000 RICHMOND AVENUE
SUITE 460
HOUSTON, TEXAS U.S.A. 77042

Let me start by welcoming all of you to this presentation. Many of you may have read my article on the subject in the June issue of the Chronicle, and while our conversation here today will take the same basic tack, we will get a bit more involved in specifics. There will be time at the end for your questions, but there is nothing particularly sacred about that, so if you have a question or you would like clarification as we go along, please feel free.

Used data processing equipment is nearly always substantially cheaper than new equipment, and quite often available sooner than possible when ordered from the manufacturer. The risk you take when either buying or selling used equipment is that a mistake made by one of the involved parties can significantly reduce the value of the equipment , or, indeed, render it totally worthless. The two areas we need to consider to avoid this are contractual requirements and technical requirements. We will go over these parameters shortly, but first let's take a look at the market for used HP equipment.

When buying used equipment, your sources are HP's own remarketed equipment which are trade-ins and demos, other users like yourself, and a third-party dealer such as myself. Buying remarketed equipment from Hewlett-Packard will give you the same basic assurances and guarantees as you would get buying from a reputable, knowledgeable third-party dealer or when buying newly manufactured equipment from Hewlett-Packard. HP remarketed equipment is cheaper than new equipment but usually substantially more expensive than purchasing from a third-party dealer. If you can find another user selling the equipment you require, and it happens to be configured as you require, you may be able to save about another 10%, unless the seller intends to save that same 10%. Of course, the real downside to this scenario is that while you may save a little money, you more than proportionately increase the risk of problems. This is due to the fact that most industry professionals aren't generally familiar with the requirements to retain warranties, transfer title, ship and insure properly and document these procedures. This is also the reason that the legal departments of many companies will not guarantee that their equipment will be acceptable for maintenance.

Disposing of surplus equipment is much the same. You may

trade your equipment for credit against new equipment purchases. This is particularly advantageous when you are disposing of very old, out-dated equipment with little or no market value and buying recently announced products unavailable on the second market. Again, I cannot recommend a user-to-user sale unless it is something like a terminal or a modem, and the purchaser gets to use the equipment on a trial basis before acceptance.

Now that we've identified the players, let's take a deeper look into the program, and go over Hewlett-Packard's policy on Acceptance of Maintenance Agreements on previously maintained equipment. Since HP maintenance agreements agree to keep products in good working condition and to provide engineering improvements, they assume that the product was in good working condition at the time the agreement was terminated, and, if not, they had the obligation to make it so. The same is true regarding updates. Therefore, the only thing the customer is liable for is what has happened to the equipment since the end of the agreement and the associated costs to make it operational. These costs could include damage due to shipping or storage. Shelf-life failures or worn-out parts are warranted for 90 days, and the shipping and storage damage can be insured against with the shipper. Insurance on shipment of data processing equipment is not the place to save money. Never be under-insured when shipping equipment such as this.

Also remember, when budgeting costs for the purchase of such equipment, to include site-planning verification costs and installation costs.

At Fidelity Systems, we now require that all products we purchase whose warranty classification includes installation and on-site maintenance must be de-installed by Hewlett-Packard or other authorized third-party maintenance organization. I also recommend this requirement to anyone buying used equipment. Hewlett-Packard requires that they perform the standard installation services for these products in accordance with their Warranty and Installation Terms. The maintenance agreement is effective the day following installation.

HP will accept maintenance agreements on "customer-installable" products without inspection. However, the agreement will not be effective until 30 days after the signed agreement is accepted by HP. If repairs are required during this 30 day interval, the customer must pay for any repairs which do not involve installation of updates of replacements of worn parts (A product in this category which has just been repaired by HP is eligible immediately following the repair for a period of 90 days).

Now for my suggestions on selling your surplus. The four

simple, but very important guidelines on selling your surplus HP equipment are as follows:

1. All "HP installed" machines must be de-installed by Hewlett-Packard or other authorized third-party maintenance organization.
   Make sure that Hewlett-Packard provides you with the HP Certificate(s) of Maintainability BEFORE THE PICK-UP DATE. As soon as you have received the letter(s), forward copies of them to the PURCHASER.
2. Discontinuance of Computer Equipment.
   As individual machines from your system configuration may be re-installed at various user sites, please ensure that the Customer (Field) Engineer who performs the discontinuance service packs the cables and logic & maintenance manuals, as well as any other accessories, WITH THE APPROPRIATE MACHINE TO WHICH THE ACCESSORIES BELONG.
   Under no circumstances should the accessories (such as cables and manuals) be lumped together and packaged physically separated from the machine to which they belong. HP will usually handle this part for you provided the proper discontinuance information is communicated and appropriate packing material is provided.

When selling a processor, be sure to include back-up tapes of the Operating System. When selling disk drives, be sure to include the disk pack and the error - or faulty - track log.

3. Preparation for Shipment.
   If there are some structural peculiarities about your building (narrow stairs, doors, elevators, steps in building, etc.) please let the PURCHASER know immediately so that they may make the appropriate arrangements with the carrier. The carrier will then contact you directly in order to assess the degree of difficulty of performing the internal move and determine any associated costs. Also, if your shipping dock is in a restricted area or has special hours of operation, please let the PURCHASER know immediately.
4. On the Pick-up Date.
   When the transportation carrier arrives, check the bill of lading to make sure he is acting on the PURCHASER'S behalf. Also, ensure that all accessories (such as cables, manuals, etc.) are duly recorded on the bill of lading BEFORE YOU SIGN IT. In the event of loss of machines or accessories, failure to record them on the bill of lading may put the burden of proof on you that they were ever picked up.

You can sell your surplus equipment under a variety of terms and conditions: as is, where is; guaranteed eligible for manufacturer's maintenance; a percentage down, net on delivery, installation, or some other finite point in time; and payment prior to pickup. "As is, where is", is certain to get you the least amount for your equipment, and if your equipment is under maintenance there is no real reason for selling it under those conditions. "Payment prior to pickup", is a common industry practice when sold by a user to either a dealer or another user.

However, if you are selling complete systems or groups of peripherals, you may want to consider a little "creative" financing. This is commonly handled by requiring a down payment upon receipt of a signed contract, with pre-determined payments being made prior to pickup of individual systems, or groups of peripherals. If necessary, all equipment can be moved to a bonded warehouse and released only upon your approval.

Should you decide to lease equipment, you have three options: Lease with a Requirement to Buy, Lease with an Option to Buy, or an Operating Lease. The "Lease with Requirement to Buy" is obviously the least expensive monthly charge, but be comfortable with the means of determining what the sales price will be at the end of the lease. An "Operating Lease" is the next in line because you are only buying the right to use the equipment for a specified period of time with no obligation to buy. The "Lease with Option to Buy" is the most expensive. You are buying the right to use the equipment for the specified period plus the option to buy at lease end for either a pre-determined price or a pre-determined method of pricing.

The best recommendation I can give you on buying used equipment is to find two or three reputable dealers you are comfortable with, and stick with them, and don't be afraid to ask for references.

If you are buying equipment to go overseas, there are a couple of additional bases we've got to cover. When used HP equipment goes overseas, the Certificate of Maintainability is acceptable at the discretion of the installing organization's service manager. Now this generally isn't a problem as long as you contact the installing service manager overseas. You may have to put him in contact with the de-installing service manager here in the States. If you don't handle this before anything moves, it could be a very expensive oversight.

It is also a good idea to go over changes necessary in power supply, cabling, and data communications that may be necessary to meet local technical requirements before you buy, and I would certainly do it before you ship. I would

be cheaper to change out RS-232 ports on an ATP to RS-422 ports here as opposed to there, regardless of wherever "there" might be.

When shipping a CPU overseas, be sure the cold load tape of the operating system accompanies the CPU, and that a copy remains here in the U.S. in case of damage during shipment. Tapes should always be shipped in anti-magnetic pouches and clearly labeled as "MAGNETIC MEDIUM - DO NOT X-RAY".

Financial requirements on an international sale are usually arranged by letters of credit (LC's) and should be drawn on a U.S. bank. Twenty-five to fifty percent should be payable when the equipment leaves the U.S. with the remainder split between the date of arrival at port of entry and the date of installation.

Since import/export restrictions can be quite confusing at best, I would suggest using a reputable customs broker to handle it for you.

I'd like to thank you all for attending; I hope this has been of help to you.

Data Base Design in an Imperfect World

Marcia Shonnard Clarkson
The University of the South
Sewanee, Tennessee 37375

At The University of the South I have responsibility for the development of administrative computer systems, have responsibility for many of the user departments such as the bookstore, the in-plant printing department, housing, and purchasing, and also teach computer science. For twenty years I have considered myself a data processing professional, first with a computer vendor and later with the University. In my more recent roles as a user and especially as a teacher, I have been forced to look at the way we in data processing go about our business from two new points of view. I hope to combine all three points of view (systems developer/database designer, user, and academic) in discussing the methodologies and procedures for designing database systems.

Let's first look historically at how methodologies and procedures developed in data processing. As program development became more complex in organizations with tens of programmers and millions of lines of code, the need for programming standards and a programming methodology became obvious. Today we teach our beginning programmers structured programming, that all programing logic is made up of sequential operations, conditional statements, and repetitive statements (GOTO's are never mentioned). We also teach both top-down and bottom-up program design, methods which provide an atmosphere in which program design has become more formal and rigid. This need for formality is necessary in a complex industry where user needs and programming personnel are constantly changing. Using structured programming and program design methodologies, programs written in third generation languages are more readable, reliable, and maintainable. Together with the increased use of fourth generation languages these methodologies for programming in third generation languages have helped us in data processing control our backlog of user requests.

The complexity of our database systems has also evolved. Batch oriented systems supporting one organizational function have evolved to online, multiuser systems for both adhoc and standard reports supporting multiple, interrelated organizational functions. As complexity in programming led to programming standards and methodologies, increased complexity and scope of database systems must lead to more formal database design procedures and techniques in order to insure that our database systems will meet the needs they were designed to serve.

In this paper I will talk about goals in designing databases, database design procedures, techniques and tools for database design, and the implications of design methodologies in our IMAGE 3000 environment. In this process I hope to call on the

academic, the user, and the professional data processing points of view to show that this increased complexity presents significant problems in the database design process. I will also suggest that the academic emphasis on formal design procedures and precise language may help bridge the communication gap between end users and our professional data processing staff. The application that I will use as an example is a student information system. We have all been students and have at least some appreciation of the data and relationships that are involved.

A database according to David Kroenke, the author of the textbook I use in the course I teach on database processing, "is a *self-describing* collection of *integrated* files."* This is a very short and simple definition that suggests a number of characteristics of a database. First, the database contains not only data but information about the structure of the data. This self-describing characteristic makes data/program independence possible. Also, if the files are integrated, the relationships among the records in the files must be internal to the database description.

Kroenke's definition describes a database in data processing terms. A user might think of a database as a representation of the condition of an organization. Certainly no database can be a complete representation of every facet of the organization. In a student information system we may not be interested in the fact that student ABC is now sitting at the third desk on the right in the main reading room of the library. Neither is it a perfectly timely representation. There is always some lag in the time between when the condition of the organization changes and when the database changes.

What then are the characteristics of a self-describing collection of integrated files such that the database does a good job of representing the condition of an organization?

1. If the database cannot be a complete description of an organization, we must aggregate and generalize the data so that the only unanswerable questions are the ones that are never asked. Kroenke says this is "like the securities advice to buy low and sell high", easier said than done. When we aggregate we combine data. For example, we may decide not to keep individual test scores for a particular student in a particular class. We may only keep his semester grade. This decision to aggregate makes certain questions unanswerable. And, when we generalize we ignore some differences in objects. We may set up two categories of courses. One category would be courses with problem sessions, and one category would be courses without problem sessions. Or we may decide to set up just one category for courses ignoring the fact that some courses have a required problem session and some do not. Again, when we generalize we make some questions unanswerable. A good database stores only data necessary to answer the important questions that will be asked today and in the future concerning the organization. The idea that in the future we

*Page 11. Mr. Kroenke's Database Processing is listed along with a few other helpful works at the end of this paper.

may ask questions about data we are gathering today makes the questions of aggregation and generalization even more complex. Deciding whether a piece of data is going to be important five years from now is like trying to decide if a particular short skirt will be fashonable the next time short skirts are in style. How do you know which details might be important at some future date? Certainly our direct access storage devices are roomier than our closets; but, unused data are costly to gather and unused data encumber our systems.

2. A good database has an architecture with three distinct levels or three separate audiences to serve. a) The external level is the database as it appears to users and application programmers. Most users and programmers require an abstract representation of just *some portion* of the database. These abstract subsets are called user views of the data. In a student information system we can talk about the registrar's view, the director of financial aid's view, the treasurer's view, the dean's view, etc. b) The conceptual level is the abstract representation of the complete database. It is used by the person or persons responsible for coordinating the use of the database. This person, many times called the database manager, should be a technically oriented user or a user oriented data processing person. c) The internal level is the physical appearance of the data to the computer. It is the description of how the elements are grouped into physical records, the records into files, and the access paths and relationships among the files. This level should be transparent to users and application programmers and possibly even to the database manager. It is the domain of a specialist in techniques for storing and retrieving data on direct access storage devices.

3. In a good database redundant data is kept to a minimum. Not only does redundant data waste space; but, redundant data can lead to inconsistencies. If we keep a student's phone number in two different places, every time the phone number changes, both occurrences must be changed or the data will be inconsistent. Removing inconsistencies from our databases is discussed at great length by the relational theorists. Their process of putting relations into the various normal forms leading ultimately to domain/key normal form is designed to eliminate modification anomalies or unexpected consequences of modifying data.

4. Access to data in the database must be rapid, and multiple users must be able to access the data concurrently. This is especially true for a system like student information which is used by many departments within the organization.

5. Since the organization being modelled is almost always volatile, it must be easy to change the model or the structure of the database.

6. Data in the database must be secure. In an era of computer hackers universities are almost paranoid about student records stored on a central system like ours which is used for both academic and administrative computing.

Designing a database to meet the above criteria is an intuitive and aesthetic process and almost always an iterative process. What, then, are the steps necessary to develop such a system? Database design, according to Kroenke, "is a two-phased

process. First, we examine the users' requirements and build a conceptual database structure that is a model of the organization. This phase of database design is called *logical database design*. . .[the second phase] formulating the logical design in terms of DBMS facilities is the *physical database design*." It is important to understand that phase one should result in a logical design which describes the data in terms of the pieces of information and the relationships among those pieces in a form that can be easily understood by all personnel in the organization who will eventually use the database. It does not describe the data in terms of linked lists, inverted lists, hashed records, or flat files. It is in the second phase, the physical database design, where we worry about how the data is actually stored on a direct access storage device and what access paths must be provided to individual records.

These two phases can be broken down into seven steps.

1. Define the system requirements in terms of data elements and define the data relationships needed to model the organization.

2. Define the system requirements in terms of system use, that is user reporting and access requirements for both online and batch operations.

3. Relate the information in steps one and two. That is, specify the user requirements for each class of user in terms of the data needed and the specific relationships and access which that user will require. This third step produces the first level of the database architecture or what is commonly called the users' views.

4. Develop the data model. This is the second level or conceptual level in the database architecture, the abstract representation of the whole organization.

5. If more than one database management system is available, choose the one that can best be used to convert the logical design or conceptual level into a physical design or internal level.

6. Analyze the requirements in quantitative terms. What is the frequency with which the data is accessed? At what times and in what places must data be available? What is the volume of data? How will the data be captured? Answers to these questions are necessary to make decisions related to the physical structure of the data.

7. Finally, use the chosen database management system to convert the logical design into the physical schema or the third or internal level of the database architecture.

It has been said that the cost to correct errors in design increases exponentially as the design cycle progresses. It is therefore very expensive to find flaws in the database logic at the time when the physical schema is being created. Certainly there are many pitfalls in the whole design process. The technical considerations in the physical design phase of the database development cycle are many times enormous. Before a databse is operational one does need to worry about Image masters and details and sorted chains; but, my interest in this paper is not with these technical

problems or the internal level of the database architecture, but with the very costly problems involved in errors in logical database design.

The problems in logical database design are obvious, yet difficult to rectify. Without complete understanding between those who know the organization to be represented and the data processing development personnel who know how to describe the organization in terms of a database model, the logical database will NOT represent the organization. Also, because no database can be a complete representation of the organization, compromises in terms of aggregation and generalization must be made. But the compromises can not be intelligent unless all involved understand the nature of the consequences of each compromise. Finally, the time constraints for getting the database design complete can put tremendous pressure on the designers to rush through the logical design phase of the development cycle.

A good data model or logical database design, then, is one that is precise in terms of describing the data and yet one that users can understand. An Image schema can certainly not serve as a data model because even very technically oriented programmers find it hard to read. The following information, presented in a manner that is easy to understand, should be included in the logical design. Items one through three are common to almost any data dictionary product available today. Items four through six are not.

1. A description of all the types of records.

2. A description of the fields within each record.

3. A description of the keys for each record.

4. A description of the relationships among the records. In my hypothetical student information system there is a many to one relationship between a student and his dormitory, a one to one relationship between a student and his matriculation number, a many to many relationship between students and courses, and a many to many relationship between students and home addresses. Many colleges have more than one student from a family and we have students who list as many as eight home addresses. This many to many relationship between students and home addresses may well NOT be obvious to the data processing professionals involved in the database design; and, the importance of specifying that this is a many to many relationship early in the logical design, may NOT be obvious to the users. Relationships among records and data within the records, then, must be clearly described in the logical design. Many of the modeling tools in use today use a graphic presentation to specify and categorize these relationships that is very easy to understand.

5. A description of each user view. Not only are the dean, the financial aid director and the treasurer interested in different subsets of the complete student information system, but many times each perceives the same piece of data differently. Student ABC receives prestigious scholarship XYZ. To the treasurer this means a credit to ABC's account and a debit to the XYZ restricted fund. To the financial aid director this means a reduction in the amount of loan in ABC's aid package and that there are only fifteen more XYZ scholarships to be

awarded. To the dean this means that student ABC may cut classes the day before and after spring break.

6. A description of the constraints on the data. This includes edit constraints for fields, intrarecord constraints, and interrecord constraints. An edit constraint may be that grade point average must be a number between zero and four inclusive with at most three decimal positions. An intrarecord constraint might be that only student classifications junior and senior may have a value in the fields for major. An interrecord constraint may be that all students with a classification of freshman must have a billing transaction for dormitory housing.

Modeling tools available today range from user oriented models meeting most of the specifications described above to the traditional data dictionaries available on the Hewlett Packard 3000. Dictionary Plus or Dictionary 3000 meet the specifications in one through three above and include some limited information on edit constraints. They do not clearly describe relationships among records, user views, or intrarecord and interrecord constraints.

More user oriented models include semantic data models (models designed to display the meaning as well as the structure of the data), the entity relationship model, and the relational model. Although these more sophisticated models cannot be directly implemented on the Hewlett Packard 3000, we in data processing should be able to map or convert any logically consistent, complete model into a physical schema such as Image 3000. Problems in design occur in misunderstandings between users and data processing personnel about the data required and how it needs to be accessed. The logical design, therefore, must be very user oriented and must include this carefully described information about data relationships, user views, and constraints on the data.

Possibly the most user oriented modeling tools are the semantic data models. Semantic data models are "like pseudocode, but instead of describing the structure of programs as pseudocode does, SDM describes the structure of data." Three advantages of semantic data models are that they provide a method for expressing meaning about the data in a database, that they allow data to be described in context (show various user views), and, that they allow constraints on data to be defined.

The second problem, the problem of deciding how detailed a representation is necessary, how much to aggregate and generalize, and how many relationships to express directly requires many compromises. These are really user compromises because they involve weighing the cost of getting data (in terms of cost of data collection and processing) and the value of the information that can be obtained from the data. It is the responsibility of the data processing personnel to explain clearly to the users the consequences of alternatives, but ultimately the users must make these decisions. Without excellent communication among the users and between the users and the data processing personnel, it is almost impossible to make good decisions in this area.

The final problem is one that we as Hewlett-Packard users very often ignore. Most of us have one database management system and a very small systems development staff. We know we are going to have Image masters and details, so we interview the

users and immediately begin translating their needs into a physical database. We may use a dictionary product that produces something a little more intelligble than a TDP listing of an Image schema, but from the start we think in terms of the network database management system that we all use. But, it is difficult if not impossible even for sophisicated users to think in Image terms or in terms of a network database structure. In consequence the logical design phase must not be skipped. It is absolutely necessary if users are to determine whether the database is really *the model of the organization that they need.*

Too many times the users just let data processing do the design. Then when the system is in place they complain that it is not what they need. In my role managing many of our users at Sewanee, I symphathize with their frustration during the design process. They do not understand the implications of the information they are providing and they do not understand the communications provided by data processing that describe the system they will be receiving. Their first real involvement in the process is when the system is finished and they can point to something that they understand like a report that does not serve their needs.

Most of us in data processing started our careers programming, many of us in an era when we were using assembler type languages and machines that were less reliable and without any real user interface compared to the machines today. We were used to just getting things done and many times even we did not know exactly why a particular routine worked. This led to an informal working atmosphere where we did not always explain, even to our colleagues, how we were getting the output in a particular report. As an academic, I see more and more the necessity for formality in the process of systems development and the need for a precise, yet English-like language in which we in data processing can communicate with the users whom we attempt to serve. This language of logical database design must be a language common to both users and data processing people. It must be precise and it must as clearly as possible describe the data and their uses.

Finally, let me suggest that we as database designers in the Hewlett Packard community, need to do the following:

1. Investigate logical design tools such as the semantic data models, the entity relationship model, and the relational model.

2. Insist that vendors of dictionary products allow us to express user views and constraints on data that are more sophisticated than the simple edit constraints that we have today.

3. Educate both our users and our data processing personnel on the new methodologies for database design and insist on a database design cycle that has a model such as the one I described for logical database design.

Suggested reading:

Kroenke, David. Database Processing. 2nd ed. Chicago: Science Research Associates, 1983.

Hawryszkiewycz, I. T. Database Analysis and Design. Chicago: Science Research Associates, 1984.

Date, C. J. An Introduction to Database Systems. Reading: Addison-Wesley Publishing Company,1986

Vesely, Eric Garrigue. The Practitioner's Blueprint for Logical and Physical Database Design. Englewood Cliffs: Prentice-Hall, 1986

MANAGEMENT - THE FORGOTTEN PART OF MIS

BRUCE EDWARDS
BRIDGE OIL LIMITED
60 MARGARET STREET
SYDNEY NSW 2000 AUSTRALIA

Before we can look at the role of management in the MIS function we must first examine what is meant by Management Information Systems or is it Services?

MIS is a relatively recent term. When I started my computer career over 20 years ago there was no such thing as MIS, in those days it was EDP or Electronic Data Processing. In some companies the term MIS is not much more than an up-market way of defining what is still essentially data processing but in some enterprises and for the purpose of this talk, MIS can be regarded as encompassing not only the traditional function of data processing, but also office automation and communications technology. If you do not as yet have control over communications or office automation, don't despair because the principles will apply equally to running a data processing department.

In considering the role of management in relation to computer technology the title of the executives is irrelevant. Thus when one talks about a Vice President MIS, or Data Processing Manager or a Chief Systems Analyst or Chief Programmmer, the concepts apply equally to all of them. And at the very outset we must consider the often discussed question of whether MIS is different in principle from other disciplines or functions within an organisation. Are we as MIS managers or chief programmers etc., so different in what we are trying to do from chief accountants or engineers or administrators?

Perhaps because the computer industry and the widespread use of computers is so recent, a mystique has grown up about the people who control it. The difference between what we are trying to control and other corporate functions is that in our industry there is a tremendous rate of change. We all know that the technology which is being developed in research laboratories today will be a commercial reality in a very short time and equally that technologies which we thought we had mastered are already being replaced.

Thus we can claim with some justification, that as MIS managers we have a special problem in that our very basis for knowledge is changing so rapidly. When I entered the DP world the concept of disc drives was very new. Indeed the first computer I worked on had no disc drives. To explain to a junior programmer in your organisation how you wrote and developed systems without disc technology would seem to him to be very strange. But I suspect in a few short years that to explain to people how we wrote systems without the use of video disc technology or speech recognition, will appear equally strange. And whilst we had many years of computer usage before disc technology was introduced, I suspect that video disc technology will be commonplace in a much shorter timespan.

We are faced then with the unusual problem that our knowledge of our own operations becomes outdated so quickly. But surely we are not unique. The medical profession, through the use of our own technology, has changed dramatically over the last few years, and this will be true of many other management disciplines. So we should start off by trying to dispel the idea that we are different from other people. Although the rate of change of technology is more rapid than for other managers in our organisations, they have similar problems too. What about the chief accountant who is now being forced to utilise a personal computer to do spreadsheet work when he would much rather stick to his old, tried and proven manual ways? He is being faced with this same impact of changing technology that we are, and in many ways for other managers this rate of change is perhaps more difficult to appreciate.

Many managers in other parts of the organisation have no real time to plan for change. They spend most of their time trying to catch up with the problems of that morning and don't see what is coming up tomorrow. The concept of planning a week or year ahead is not difficult for them to understand, they simply don't have time to do it. But we have by our technological involvement become more used to the concept of rapid change and planning for it.

Thus in spite of our peculiar problems of changing technology, the conclusion we must draw is that we are like all the other managers in the organisation and therefore the concepts of management that apply to running other parts of the operation must apply to the way we operate.

This is something which many of us in the DP industry have tried to ignore for far too long. Because we have been in a growth industry we have believed we are different from other

managers and have allowed ourselves to operate in ways which whilst they might have served us well in the past are no longer suitable for future development. I am not talking about the great hardware and software entrepreneurs who have taken ideas and brought them to fruition. These people have played, and are continuing to play, a tremendous part in the development of our industry. But for those of us who are operating within a company management structure, their methods are rarely suitable.

This leads us to a very important principle when we are discussing MIS management which is that as managers we must be concerned with the goals and objectives of the enterprise for which we work.

It is not good enough for us to see our role as that of a specialist, that is, to be concerned with the implementation of better computer systems, communications or OA and not to regard ourselves and our fundamental goals as being those of our employer.

As the MIS manager for an oil company, it is vital that I consider the objective of myself and my group to further the interests of the company in its search for oil and gas. My job is not to operate a DP development company. I and my staff cannot work in some kind of ivory tower concerning ourselves only with the latest release of MPE or investigating a 4GL. Our role is to provide an important part of the support function of the company and therefore my group must be interested in whether we find oil and gas.

Why do so many people from the DP function have problems in either integrating with the management of the enterprise or being seen by other managers as not being part of the operation? Part of the problem stems from the fact that our industry is very young and many of the people in it are correctly seen as technocrats.

There is no doubt that a large number of people involved in the computer industry do not align themselves with the goals of the enterprises for which they work but see themselves purely as technicians involved with data processing. We are not entirely alone in this approach. The research and development operations of companies can breed the same type of thinking, but it is vital that we try very hard to eliminate it.

The reason why we as MIS managers must not remain outside the mainstream of our enterprises is that the very nature of the technology of which we are in charge, is reaching into

all aspects of enterprises' operations. In the past, computers were brought in to do tasks, very often accountancy, which were seen as tedious, involving relatively low level staff who could be replaced by computer systems. But now the dramatic increase in the scope and scale of computer usage has meant that the introduction of our technology cuts across all department levels and boundaries. This places us in a unique position in that we are in one of the very few management positions which cover all aspects of a company's operations.

Those of us who have been involved with the implementation of any aspect of office automation, be it electronic mail or word processing, will know that we have had to gain an understanding of the operations of engineers, accountants, marketing people and so on. Very few managers have to operate outside their own discipline as much as we do.

We have an important role in that we are the introducers of new technology. Unlike other managers whose job is to control and operate their own function, such as accounting, for us to meet our corporate objectives in introducing new technology we have to liaise with managers and staff at all levels in other operations. I would suggest that the majority of chief accountants, sales managers and so on do not.

There is one other role that we have to play and that is the control of information and this is going to be our biggest challenge in the years to come. We have all read articles about the introduction of personal computers into organisations and it is instructive to consider what the British industrialist, Sir Michael Edwardes, has to say on the subject. As a man who has achieved considerable success in the management of such varied enterprises as the British Leyland car manufacturer, the Dunlop rubber group, and who also has held the top position in more than one computer company, he believes that to have a multitude of systems being created as individual departments go off and do their own thing leads to anarchy and wasteful duplication. His view is of local autonomy but central financial and strategic control.

I believe that it is an important function of the MIS manager to make sure that he or she is in control of the introduction of all new technology into the enterprise and particularly to make sure that the use of personal computers is done along planned and constructive lines.

To give an example from my own organisation. Our engineers use IBM PCs because the programs they wished to run were only available at the time on the PC and correctly it was practical to purchase an IBM PC to do the job. But once having got their hands on a PC and remember that these engineers like many people today had experienced training in computing as part of their university education, these people then saw no reason why they should not continue to develop their own little programs to help them do their work. There is, of course, nothing wrong with this until it comes to the point where they start generating data for their own operations without realising that this data is being duplicated elsewhere in the company.

It is a vital role of the MIS manager to convince his management colleagues and top management, that data is a corporate resource and that the control of this data should be vested in the MIS manager. We were able to convince the engineers that whilst there was nothing wrong with them having a number of programs, (most of which we knew nothing about!) it was wrong that they should consider the data that they were collecting and using to be the personal province of the engineering department. It is not my purpose here to tell war stories about disasters of PCs in organisations, there are better people than I equipped to do so, but just remember that the control of the data used in the enterprise as a corporate resource ultimately rests with you as MIS managers.

With these ideas of the goals and objectives that we have and the functions of the MIS manager, let us consider how we might achieve all this. We have talked about the problem of being seen as technocrats. What are we going to do about this? Well, one problem is in our approach to users when trying to define a problem. One of the many buzzwords of our industry is "user friendly" and we talk about this a lot in relation to the screens and the way in which they present information to the user, but have you ever thought how user friendly you are?

When we go to talk to the user who wants our help in designing a new computer system, or is simply trying to express to us a problem and ask our advice on how to solve it, I would suggest that the majority of us are hardly friendly at all. How many of us half way through his third sentence are already beginning to work out the schema of the data base? How many of us before he has really explained half of what he is trying to do, are already finding ways why it can't be done? How many of us see his problem in terms of files and numbers and don't really see what he is

getting at? As a classic example, how many of us have vendor systems which have a supplier number? Why? Why don't we have systems which are based purely on names? Why is it necessary on the payroll that everybody has a number? Surely it is a throw-back to early batch processing DP days. I would hasten to plead guilty and admit that in my company all employees have numbers and all suppliers also!

Can we just consider a few fundamental misconceptions about managing. Managing is about getting work done. It is not about doing it yourself and this leads of course to the question of delegation. The hardest thing for managers in their early days to understand is the concept of delegation. The problem is you can always do it better yourself, and indeed in your early days of moving up the management ladder that is true. But as you move to higher levels, comes the realisation that you can't program any longer, that you don't really know how to work out a schema and with that comes a new understanding of what your job is all about. I would suggest one of the biggest problems we have as managers is not being able to let go of the technology. Those of us who have come up, and most of us have, through the ranks of operator, programmer, analyst, or all three will at some stage in our career have been extremely proficient in those functions.

There comes a time when you take your first steps to management, say as a team leader, when you realise that although you can do it better than all those around you, you can't do it all. You can't produce what the three programmers under you are doing, it simply isn't possible, and gradually as you move higher up the scale you are less and less able to correct on technical grounds, what your staff are telling you. I regret that a large number of managers of computer installations simply are not able to recognise this. They still prefer at the end of the day to get in there and write a COBOL program themselves, but the truth is that this is not the job of management. Managing MIS is not doing the job yourself. It is one of the hardest facets of management to understand that your role is to see that the work gets done.

I would however add one important rider and that is, don't forget that if a job is not worth doing it is not worth doing well. For many years I was a management consultant and I regret to say that there are a large number of occasions when I saw managers working extremely hard to achieve objectives which simply weren't worth achieving. Do not forget that as managers it is your job to set the objectives and if you don't set them right, then you will

waste a great deal of your company's resources having staff achieve work which shouldn't be done in the first place.

With all these worrying edicts behind us, perhaps it is time to look at some contemporary management theory and see how it applies to us in the DP environment. I suspect the majority of you will have heard of, or better still read, "In Search of Excellence" which was written by Thomas J. Peters and Robert H. Waterman. The book resulted from the authors' work as consultants and those of you who have read it will know that Hewlett-Packard was one of the companies that was used as an example of the best run companies in America. But I wonder how many of you who have read the book have followed it up and examined the lessons that were stated, to see how they could be applied to your own jobs.

It is not my purpose to teach you management theory in such a short time as we have available today and indeed I would not wish to be involved in any copyright accusations that might follow from using examples from this excellent publication, but I have picked out four examples that I think specifically apply to us in the MIS role and these are:-

Management by walking about
Encouraging experimentation
Who are your customers?
Simplifying systems

Peters talks about managing by wandering around, I talk about managing by walking about not because I disagree with Peters but because I believe I was actually doing it before I read his book. It really is very simple yet very few people do it. Do you know what your staff are doing? It is very likely that you don't, so management by walking about has two objectives. One is to find out what your own staff are doing, which in turn encourages them, particularly if you stop and talk, because it correctly makes them believe that you are interested in what they are doing thus leading to an improvement in morale and productivity. Secondly, we as MIS managers have a peculiar role particularly when we start looking at the communication methods of our companies or installing office automation ideas, as it is essential for us that we understand what the rest of the company does, and how else can you do that other than by walking about?

If your company is engaged in manufacturing, go and have a look at the production line. In the case of my own company, go and visit an exploration oil well just to see what people do. You will not only learn a lot but you will be surprised

to find that people are interested in the fact that **you** are interested in what **they** are doing. Any operation which has branch offices knows that when someone from head office appears there is interest. There may also be concern as to why this person from head office has bothered to turn up and a certain amount of resentment shown, but that is something we have to live with. If you visit as many parts of your enterprise as you possibly can, you will have a much better understanding of what your enterprise is trying to do and when you try to implement new methods of communication or office automation, you will be met by a far better response from the people than you could possibly have by simply thinking up a good idea and issuing a memo to implement it.

Perhaps I have been fortunate. It has always been my policy long before I read Peters and Waterman's book to spend as much time as I could possibly allow exploring all aspects of what the various companies that I have worked for over the last twenty years have been doing, and I can assure you that apart from being extremely interesting in seeing how other people carry out their duties, it is very rewarding and will serve you very well when you subsequently have to operate in those areas. After all when you pick up the phone and call a remote location and find you that have met the person who answers, it has to be much easier to establish a good working relationship because you know the people, perhaps have had a beer with them, but importantly have established some sort of rapport.

The second idea which I think is very appropriate to MIS, is to encourage experimentation, or "try it and see what happens". But how often do we? How often do we theorise at length about a possible outcome of something when all we really have to do is try it, and how often do we explore manuals to see how it is done when a bit of commonsense would work it out anyway. Remember the first rule of computing is whatever you want to know it is not in the manual!

In computer terms of course what we are talking about is prototyping. For many years the biggest problem in designing computer systems has always been that the user never knows what he wants. This of course is completely wrong, the user nearly always does know what he wants, what he doesn't know is how to tell you. We have all been frustrated by designing systems which after a great deal of discussion and consideration, when presented to the user have not been what he wanted. How often have we heard "Oh! but I assumed that you would know that we don't do that" or

"We don't do it that way", as if to say, well anybody would know that.

I admit to more than one occasion when this has happened to me, and that leads us to one of the biggest problems I think that we have developed in our industry and which we as managers must ensure that our subordinates do not do. This is the "Holier Than Thou" approach, to take the view that this is what was asked for, this is what will be provided and if it is no good its not my fault. It is your fault. You are there to provide the users with systems, technologies or techniques which will enable them and ultimately your enterprise of which you are an integral part, to do a better job, so it is no good taking the view that you did your bit and that somebody else didn't do theirs. You may well be right that somebody else didn't do theirs, but that is not the point.

An easy solution to this is to encourage experimentation. If we can go to the user and very quickly produce a possible solution to his problem, which he can discuss and change, then this is a far better way of developing systems than spending months of detailed interviews writing some massive manual which you expect the user to sign as an indictment of what he wants to do and ultimately produce a system which is so far out of date that it is useless. However be careful when prototyping. As in word processing it is always possible to improve the final product with very little extra work and the temptation to keep on adding one little bit is very great. As we shall see later, cost control is also one of our most important functions.

I believe that the role of MIS is to provide a support function, even where the company is directly involved in the computer industry itself. In my own case everything we as MIS, or the accountants or the marketing people do, is subordinate to the fact that we are an oil and gas exploration company. Thus we are a service department and our customers are of course the rest of the organisation.

Earlier we talked about being user friendly in our approach to other departments. This is the first part of servicing our customers and there are three basic rules that we must instruct both ourselves and our staff on before we go near the people outside our own departments. They are of course obvious and therefore are always overlooked.

Firstly we must learn to listen, which I suspect we don't. We must learn to listen to the whole problem, users never tell you the whole problem in the same way as you don't tell

people how computers work because you assume a certain basic knowledge. They equally will assume a certain basic knowledge from you, so it is essential that you allow your users to explain in their own jargon, what it is they want to do. Let me give you another example from my own company.

Recently we were discussing with our engineering executives a problem relating to the balancing of the quantities of oil, gas and condensate produced at the end of the pipeline to that produced from the various wells around the country. The executive in explaining to us what his problem was went into a great deal of technical detail of how his pipelines, valves, and processing plant operated. Now many a young analyst would not only have switched off very early in listening but ultimately would have been unable to grasp the complexity of the problem because he simply was not prepared to listen to what the engineer was saying. It is true that in order to work out what we had to do to design the computer system, we were given a certain amount of superfluous information or so it seemed at the time. But none of the information in the end was superfluous. What it did was add to our knowledge of the problem and by understanding the problem we were able to ask pertinent questions to arrive at the information that we needed in order to produce the system. So as managers it is very important that both we and our staff learn the importance of listening to the users' problems.

The second rule is to prepare beforehand. If you are going to talk to accountants it does help if you understand a little bit about what accountancy is. It helps enormously if when words like discounted cash flow, ledgers, debits and credits are discussed you do have some understanding of what they are. Before you talk to the paymaster do try to understand a little bit about how the taxation system works, not only will it make your life much easier but much more importantly the user will appreciate that you have taken the trouble to understand what he does and I can assure you that the necessary rapport that is essential to good systems work will be greatly facilitated by this knowledge.

The third rule is to keep your technology in the background. Nothing is worse than the egotistical analyst who believes that technology will solve all problems especially ones that have not been identified. In other words do not go in with this great solution which will do everything the user wants if only he will change his problem. I am not going to get into an argument about packaged systems here, they have their place, but only consider your package **after** you have identified the user's problem. Even if your solution does

fit the problem do not underestimate the need to convince the user that it fits and do not expect him to take quantum leaps into the unknown of new technology.

One last point on prototyping and servicing your customers, don't get excited if the user says that what you did is all wrong. You may very well feel that you didn't get it wrong and that you did provide what the person asked for but the reality is that both they and you exist to service the enterprise and if what you produced isn't what is required then change it. It is an unfortunate part of our industry that we seem to breed a large number of prima donnas. I was perhaps fortunate in that my very early training prior to being involved in computers, was working in the clothing industry. I can assure you that I had all delusions of grandeur knocked out of me very quickly at a very early age.

Of course it is annoying when after all your hard work you present a system which the user says is a load of rubbish, but getting angry with the user will not help. With experience you will be able to convince the user that half the problem was his own fault anyway and you will then sit down together to produce what was really required. As an important aside from this, remember also that good service and being polite and helpful doesn't compensate if you do get the system wrong, but most of all remember that there is a thing called cost. If the user wants a Volkswagen don't produce a Rolls Royce because perhaps the reason he wanted a Volkswagen was that a Rolls Royce wouldn't fit in his garage!

We have entered a time when we can provide better service to our customers through the use of 4GLs but even without 4GLs, good service on a straight forward basic batch system can work wonders. Some years ago I was operating an internal bureau doing accounting work. The very fact that we guaranteed to produce a 24 hour turn around and did it, although it meant sometimes working night shifts and having backup machines available to us, went a very long way to providing a good MIS service to the customers within our diverse group.

The last management technique which is particularly applicable to MIS is that of simplifying systems. An example of this, which I am sure you will recognise, is the spaghetti program: an entire system consisting of one program which is so complicated that if ever a change is made the chance of success is nil. The whole of the DP industry is full of quotes and sayings about keeping systems simple yet very few are.

One of the biggest dangers that we run into in MIS in designing systems for our users is to assume that because we have this information within the system it would be "useful"to produce a report. Very often it is not and all we do is confuse people by inundating them with superfluous information. Just because you can sort the information three different ways and produce three different reports it does not mean that the user actually wants any more than he has asked for, so don't produce it. Another problem is the production of information on which we simply do not act. Indeed a useful criterion to apply when producing any report is: "If I had this information what would I do?" and if the answer is I wouldn't do anything, then one should seriously question the value of producing the information in the first place.

These are some of the very few ideas presented by Peters and Waterman and I hope that you will be sufficiently encouraged to research them further. But I think it highlights that although we and our staff often receive considerable training in technical matters we perhaps lack the incentive to learn more about management. I recently sent two of my staff on a course for improving their presentation techniques, because all of us at some time will have to give an explanation of our systems to the users as part of the implementation process. Sure we will write, (or will we?), good training manuals for the system, but as part of that training it is necessary that we are able to explain to the users how the sytem works. And yet how many of us have ever received any sort of training in how to do that?

Those of you who have been fortunate at some stage to have had exposure to sales training will know the value of role playing. Embarrassing though it often is at the time, how much more confident are you as a result of these classroom simulations. We should be far more aware of the necessity to balance technical training against a more general role. Most of our companies have training facilities or departments and it is your job as an MIS manager to ensure that you and your staff learn not only about your company and the way it operates but also about necessary skills such as report writing, presentation, and general management in addition to learning about database design, programming techniques, 4GLs and so on.

Within your own department and under your own control perhaps the most important contribution you could make to training is delegation. As I said earlier delegation is probably the hardest thing that any manager has to learn. But it is important that you encourage even your youngest

and most inexperienced staff to take some responsiblity for some task. But make sure it is within their capability. There is nothing worse then delegating to people a job which is too difficult for them and therefore one which they fail to accomplish properly. It is an old adage that managers delegate all the jobs they don't like doing themselves. But you must guard against this.

I know, you would like to delegate, you **intend** to delegate. Its just that you can't find the time to do it. Make the time, but remember also that delegation without authority serves no purpose. If you are going to ask your staff to do a task you must give them the necessary authority to carry out that task. It is like giving a person responsibility for the day to day operations of 2 million dollars worth of computer equipment but not allowing him to place a purchase order for parts for more than 50 cents, and yet in many companies this is precisely what happens.

You have to learn that you must allow people to make mistakes which is one of the hardest things in management. Sometimes it is important that you do stand back and watch people make mistakes and allow them to do so, providing of course they are not going to cause catastrophes. It is a sad part of the human learning process that very few of us are prepared to accept at face value what people say. Most of us have to blunder in order to realise that the advice we were given was in fact true. If you tell people that the chair has wet paint on it, how many will touch it to check?

One of the most important and rewarding roles as a manager is the development of your staff. Training, formal or on the job, under proper supervision and control is an important aspect, but delegation of small tasks leading in turn to greater authority and control is a major component of your job as an MIS manager.

The computer industry has come of age and with it have come managers with many years experience in the industry. The bulk of this paper has been considering our role within our organisations as MIS managers and how we can do that job better. But where do we go from here?

The traditionally held view of executive development is of continuing progression until one becomes the managing director or chief executive of the enterprise. Apart from those companies who are actively involved in the manufacture or sale of computers or computer products there are few examples of MIS managers reaching these goals.

The reason for this lies in the development of the computer industry. For many years, computer people have been regarded with a great deal of suspicion by senior executives in organisations, at best regarded as friendly technocrats, but hardly ever regarded as part of the management team. I think we have ourselves to blame for this because undoubtedly we have traditionally been far more concerned with the technology then with how to channel it to the enterprises' goals.

But we have now matured to a point where we are taken seriously as managers in our own right. And we have one big advantage over our management colleagues in our challenge to reach the top because as MIS managers we will have seen all aspects of our various enterprises operations. There will be few of us who have not over the years gained great insight into personnel systems, accounting systems, financial modelling, forecasting, production control and so on. We have been able to gain that broad perspective which many of our contempories have not, so we are very well equipped to try to move into general management or chief executive control.

But obviously not everybody can be the CEO and most importantly not everybody wants to be. A friend of mine refused a headmastership because he preferred the face to face student contact to the administrative role offered although to most people he is turning down a promotion. How many Chief Programmers have reluctantly become MIS managers because they felt it was the only path open to them ?

We must recognise that the most important people in every organisation are the "solid citizens". These are people who enjoy what they are doing and see their personal goals as doing it to the best of their ability. They occur at all levels in the organisation and without them the organisation simply couldn't operate. As soon as we understand that this "plateauing" is not "dropping out" nor is a sign of inferior talent or ambition then we can see that there is a very clear role for us to play and it may come as no suprise to learn that personnel studies have shown that solid citizens are among the most contented people in the organisation.

Earlier this year the great mountaineer Tenzing died. He and Sir Edmund Hilary were the first men to conquer Mt Everest back in 1953 and they were justly remembered as heroes for doing it. But what about the people in camp five just below the summit, can any of you recall their names? I can't. But without those men at camp five and indeed without the men who never left base camp Hilary and Tenzing

would never have set foot on the summit of Everest that year.

Those men in camp five were the solid citizens I referred to when talking about the company hierarchy. Without solid citizens as members of a team very little would be achieved. Although you and I don't know their names I can assure you that in mountaineering circles those camp five men would be regarded as highly as the two climbers who made it to the top. And this leads us to a very important point, because for the vast majority of us engaged in business, getting to the top is not all that counts. Very often of far more importance is recognition by one's own peers. If you are a good chief programmer and are recognised by other chief programmers as being amongst the best in the industry, then this in itself may mean far more to you then taking a promotion to an MIS managerial position which you do not really want and which you do not do well. Thus I emphasise that we must not think of plateauing as being in any way a sign of failure.

In this context, as an MIS manager it is very important therefore to recognise those of our subordinates that wish to continue in an upward career path and those who have reached a level which they thoroughly enjoy.

Part of our responsibilities in motivating our people must be to motivate for achievement and not necessarily for positions within the hierarchy. We often read articles about the theory of having a happy and contented staff, this is of course not easy to achieve, but it is certainly better to have that objective than to have a staff which is constantly striving to out do each other in the promotion race.

But we must recognise the necessity for healthy competition providing it is kept within bounds. There can be great merit in encouraging staff to improve their performance by competition but be careful that you do not make the competition so intense that the losers get so disheartened that they may seek to further their careers elsewhere.

In many organisations the functions and role of the MIS manager is now recognised as being at the very highest level of executive responsibility. There are many Vice Presidents Information Services, General Managers of Information Services and so on amongst major corporations. Corporations do recognise that the MIS function is equally important to that of sales, finance and so on.

So we may have to accept that we have reached our plateau and that as the Vice President of Information Services we have achieved the highest level which is obtainable. We should recognise this with enthusiasm not despair because we are after all in charge of a very exciting function. Advances in technology continually present challenges to us in appreciating not only how they work but how we can best use them.

Innovative techniques such as expert systems popularly referred to as artifical intelligence are going to provide new opportunities, and it is to us that the enterprise will look to assess and implement any new technology. The changes over the last 20 years in the use of computers are just the beginning of the challenge.

The introduction of new technology has always been extremely difficult. It was Machiavelli in the sixteenth century who wrote "there is nothing more disturbing than the introduction of a new system" and no doubt the ancient Greeks said something similar.

The computer industry has been lucky in that the dramatic reductions in cost and proformance of computer equipment have made it easy to install and implement new methods of working. The introduction of OA has changed the way in which enterprises perceive new technology and this will continue.

Up to now MIS managers or DP managers as they were once called have been able to bumble along because the cost saving from the new technology was both obvious and easy to understand. From now on managing MIS is going to get more difficult and we have to be more professional.

There is much more to be said about our job. I have not covered the importance of planning, the use of Management By Objectives techniques, how to negotiate, or how to control a meeting. All of these are important parts of the job of managing. Above all remember that managing is about money and you have the responsibility to relate all future investments in technology or systems to cost. If you are in doubt ask yourself whether you would spend the money if it were coming out of your own pocket. The answer will usually solve your dilemma.

What about the fascination of the technology, because new ideas and inventions **are** interesting. Many of your management colleagues enjoy playing with PCs, colour graphics and so on and so do we. Earlier I commented on

letting go of the technology in the sense of not doing your subordinates' work, but there is everything right in keeping a keen interest in experimenting with the latest gadgetry and encouraging your colleagues to do so. After all you are employed to be innovative and introduce new technology. Just be careful to keep your involvement at the right level.

But above all remember it is the manager part of your MIS Manager title which is the most important and which is the key to your future success.

METHODS AND PRACTICES OF MIS MANAGEMENT, ...FOR THE PREVENTION OF CRUELTY TO EXECUTIVES

MITCHELL KLEIMAN
CONSOLIDATED CAPITAL COMPANIES
2000 POWELL STREET
EMERYVILLE, CA 94608

Introduction

"What do you mean you need another disc drive?"
"When can I get the memory upgrade for my PC?"
"Is is okay for me to buy a ....., I can get a really good deal on it?"
"Why does it take 3 minutes for the loan calculation to run?"
(or 2 hours for the GL job)
"When will my system be ready?"

The questions above represent a small sample of typical questions to MIS management. As data processing professionals, particularly in the management area, we spend much of our time communicating with the end users of our computer systems and in applications development. Some of our time is spent preparing information for and communicating with executive management. While we spend much time organizing, analyzing, documenting, and supporting other departments and functions within organizations, we seem to have missed accomplishing these same objectives within our own areas - like the accountant whose checkbook is never balanced. Standards and practices for managing the MIS function and measuring performance are rare and vary widely between companies.

This paper outlines past and current management practices covering system performance, capacity planning, staffing, computer usage, and computerization developed within a company of over 400 employees, undergoing rapid growth with two HP3000 Series 68's and more than 200 microcomputers.

The methods described in the paper are applicable to both large and small organizations, as they were developed and refined through the growth of the MIS department from 4 people to its current staff of 16. They are the basis of establishing standards within the data processing organization and providing management with the information necessary to answer questions concerning critical needs and issues. While some of the ideas and methods I describe may seem simplistic, they are what currently work, and that is what I use as the test. Good ideas about how to are just that, good ideas.

The paper is organized as follows:

- Overview
- The Eight Papers (or Reports) that an MIS manager should take to every meeting
- Staffing and criteria for measuring staff performance
- Planning the use of your time and why "the best laid plans of mice and men..."
- Hopes for things to come
- Questions to review and for which to prepare the answers for the question/answer/discussion period at the end of the presentation

Overview

Consolidated Capital is a syndicator of real estate investments, in 1985 the country's fifth-largest. It recently merged with Johnstown American Companies, which previously had managed many of its properties. Con Cap began purchasing PC's on a large scale in late 1983 and early 1984. Management also decided to develop an in-house computer capability to replace the use of outside time sharing bureaus and to answer the needs of other departments that were going off in their own direction using consultants.

I started at Consolidated Capital in mid-1984, when they had decided to purchase a Hewlett Packard computer. There were two people officially supporting the microcomputers in the company and they were managed out of the finance department (as a special project). They supported approximately 75 IBM PC's, for which they provided purchasing administration, installation, maintenance, training, and on-call support.

In November '84, we installed an HP3000, Series 68 and moved all existing HP applications in-house and began the development and installation of other applications (financial systems - GL, AP; property tracking; mortgage servicing and administration; investor and investment tracking, etc.). Usage grew rapidly and by June of 1985 we had overrun the ability of a single Series 68 to meet our needs and added a second one. Our current configuration is two Series 68's (soon to be 70's) each with an average concurrent 35-43 users of a possible pool of 140 users. 90 terminals are connected to the two systems besides 50 (of the 224) PC's equipped with telecommunications software (that enables them to act as terminals and to transfer files). The staff, supporting and maintaining all the equipment and the users, has grown to 16; comprised of 3 people full-time supporting the use of PC's, 4 people in the operations group (covering 18 hrs/day, 5 days/week and Saturdays 8 hrs/day), 7 people responsible for software development and maintenance; one administrative support person (doing all the purchasing and vendor coordination; scheduling class and coordinating use of "loaner" Compaqs and working on special projects), and a manager coordinating activities with other departments, responsible for planning, budgets and obtaining executive support.

The Eight Papers (or Reports) that an MIS manager should take to every meeting

1. Schedule Paper (or Appointment Book) - An invaluable resource for a manager is a method for keeping track of meetings, deadlines, birthdays, and time not scheduled. For an MIS manager it is useful to have, in addition to their own personal schedules, a calendar of development project's critical dates and deadlines and a calendar of system production peak periods.

   The personal schedule (diagram 1) is necessary because very often at a meeting another meeting will be scheduled, or a new critical date is decided upon. The development project (diagram 2) and systems demand calendars (diagram 3) are critical for planning preventative maintenance periods, installs of new equipment, for avoiding scheduling software project deadlines in the middle of heavy production periods (those last minute compiles competing against the 150 jobs in the queue), and for answering questions like "What would happen if the production of financial statements were moved up 5 days?".

   If your company publishes a schedule of company dates, this should also be added to your schedule collection. Knowing filing dates, fiscal year-ends, annual conferences, and company picnic dates will aid in scheduling.

2. Monthly Projects/Goals List - This list is a one line per major item list that in non-technical language identifies a project that will be finished, a major production accomplishment scheduled, or notes the completion of a module that is part of a project. I prepare this list for and distribute it to management. I also provide a copy to each person within the department showing who is working on each of the projects (diagram 4).

   Each month I review the previous month's list, prepare a proposed list for the upcoming month, and distribute the proposed list asking for comments and at least 1 project per person to note. I then follow-up by group and prepare the composite list. This way, each staff member has a chance to input what they are going to accomplish, they know what I am telling management they are working on, and they know what I (and management) are expecting from them.

   This list is also handy for quickly seeing what the impact would be moving a person from one project to another.

3. Department description (including a list of services offered, who to call about what, and an organizational chart) - Several times over the past year, I have been asked to prepare a "short summary" of my department by executives or by the corporate administration department for the inclusion in the company administration book. Also, when first starting to work with a group of people not familiar with the department, (especially if they are new to computers) I find it helpful to give them a description (diagram 5) that provides a quick overview of the structure of the department, how it functions, who does what (therefore, who to call) (diagram 6), a list of services (diagram 7), and a list of supported hardware and software (diagram 8).

   The lists (services offered, who to call) come in handy when you need to shift job responsibilities, replace personnel, make staffing plans, or compare services to related costs.

   An organization chart (diagram 9) is the easiest way to communicate the structure of group functions, of personnel depth (cross training availability), and management time involved in your department.

   Variations of the organization chart with titles and/or salaries are useful in developing a detailed personnel plan for your staff covering education, job titles, responsibilities, compensation rates, and a development schedule.

4. Annual Budget (with current status and the 3-year plan) - As an MIS manager the topic about which I most often interact with executives, is costs. For some reason, managers are particularly concerned with the costs of additional computers and peripherals.

   Developing an annual budget (whether required or not) provides many benefits. Adding purchases, shifting equipment deliveries, increasing tape backups, etc. all affect the expenses of your department. The process of developing the budget (diagram 10) will leave you with a better sense of how work and paper flow through the department, the relationship between different items in your budgets, and an increased awareness of background costs (the ones you or others don't budget for that are necessary - for example, PC to printer cables). Do not expect to catch everything the first time through developing a budget. Live with it, note the items you did not plan for and add them to next years'. Above all, do your budget on a PC using a spreadsheet package. There will always be adjustments and you'll be asked or asking the HP question, "What if...?".

   Knowing how you are doing compared to the budget is what executives are interested in after they've seen the budget. Hopefully, you can get a report from your financial people displaying year-to-date actual expenses, monthly actuals, and budget vs. actual year-to-date totals. I use these to see if I have underestimated in the budget, to review current monthly expense levels (as an indicator

of the future annual total expenses and expense trends - more or less than last month), and in planning the ordering of additional equipment so as to keep to the budget (replace one item with another more important one, move a purchase up in the schedule to take advantage of special incentives).

The 3-year plan (diagram 11) sets the expectation level of the executive for the current year and for the future. If they were expecting you to buy everything the first year and you have hardware expenses budgeted for succeeding years, that issue would come up in reviewing the long-range plan. Very often executives do not understand that systems grow, requiring more machine resources, and that maintenance costs increase as old equipment is replaced by newer (technology) versions. This is a useful tool for working with management to keep a long-term perspective on computerization of a company. What seems like a large expense compared to the monthly total expenses is just a "drop in the bucket" contribution to the comprehensive plan.

5. Project(s) Status Report (diagram 12) - Right after seeing the budget and actual reports, management wants to know what benefits they are getting from all of this. The project status report should be designed for your management, to answer their questions.

   I always provide a project description or overview, current status of the project, budgeted costs, department the project is for, and who is working on the project.

6. Systems Status Report(s) - As Consolidated Capital has a significant investment in HP hardware and software, I consider it critical to monitor the use of the two HP 3000's. As there currently is not readily available a package that meet all of our needs, with the Operations staff I have built a set of system management graphs that show:

a)	CPU and connect time	(diagram 13)
b)	System uptime and system availability	(diagram 14)
c)	System free space (highs, lows, averages, and working levels)	(diagram 15)
d)	Operations requests (monthly total)	(diagram 16)
	(monthly detail)	(diagram 17)
e)	Production time summary	(diagram 18)

   These are used to indicate growth trends, justify peripheral and utilities purchases, and justify operations staffing requests.

   The information for the graphs is compiled from various places: our resource allocation package, daily system reports designed to provide the information which is then entered into Lotus), written production request forms, and review of the console logs and backup jobs.

7. Requested Projects List - Since the typical MIS manager supposedly has a 1 to 2-year backlog of application projects, I am surprised at the lack of tools to manage and track this backlog. If your manager does not ask about the 15 projects you have not started yet, the other department managers will. They will want to know what priority you have given to their request, when you will start on it, when you will finish it, how much it will cost (why so much), and who will work on it.

   That is a lot of information to manage for projects you are not working on. At a previous company, we developed an in-house computer system to track and allow us to sort and print these waiting projects. Currently, I have a short backlog which I manually keep track by keeping all requests in one folder and I maintain a list with a word processing package (diagram 19).

8. Inventory Reports (and a report on how the inventory is used) - As a manager of computers, you are expected to know how many you have, where they are, how much they cost, how much they are worth, and what they are being used for.

   I typically use three different inventory reports:

   a) A detail report of all equipment sorted by user, equipment type, and department (diagram 20).
   b) A company equipment summary report (diagram 21).
   c) A computerization by department and location chart (diagram 22).

   The computerization chart reveals possible growth areas, percentage of saturation, and who has the most. I combine this information with the software inventory information to generate Lotus graphs analyzing how the PC's in the company are used (a series of these graphs will be shown in the presentation).

Staffing and criteria for measuring staff performance

The staffing of the MIS area is one of the leverage points in building a department. I subscribe to the theory that you can always find someone with better technical skills and that the key ingredients to look for are the communications skills with both technical and non-technical people and a commitment to doing quality work. People costs keep going up, equipment costs are going down - make the most of each person and position (give them the tools to be most effective and efficient). Invest your time in coaching your people once you have hired them. The interest you shown and time you invest pays off many times over.

I measure the staff's performance on:

- The quantity of work they accomplish
- The quality of their work
- Their level of technical knowledge and analytical skills
- Project management skills
- Communication and cooperation skills

Planning the use of your time and why "the best laid plans of mice and men..."

Things will change! Schedules will be moved, projects will be late, new higher priority projects will bump others, and someone will go on vacation or become ill.

The payoff on the planning is in these situations; when you can adapt quickly, take specific actions to recover from emergencies, and rapidly figure costs and time delays and adjust appropriate resources.

So leave some extra time in your schedule and in your staff's schedules for handling emergencies. Unless you have people backing up one another, run your operation with a little extra capacity than is required to handle the demands of the job. Management wants the task accomplished, not an explanation why it did not get done and a complaint about a lack of resources.

Hopes for things to come

HP has begun to develop tools for large systems management, trend analysis, and capacity planning which should aid considerably in matching computer resources to company needs.

Tools for enhancing, documenting, and speeding systems development are beginning to appear. While these tools help they do not currently meet all development needs.

As more and more people in the business world learn to use computers, the mystique surrounding them will dissipate and MIS professionals can get on with the job of using them most effectively in business with the support of management.

Questions

1. How do you manage the projects that have been requested that you are not currently working on?

2. How do you manage the projects you are working on?

3. How do you explain about the need for sort space, the need for available free space distributed on drives, and the need for excess computer capability to your manager who knows nothing about data processing?

4. How do you insure that you're not bringing the computer down (for that new disc drive install or the CPU upgrade) on the day before the finance department promised the CFO they would have the quarterly statements ready (on the new schedule, of course)?

5. What is the business your company is in, is it in a cyclical industry and if so, what types of cycles and how long are the cycles? Does your MIS plan take this into account?

6. What is the exact percentage of projects accomplished on schedule by length of project 1-2 days?
   3-5 days?
   5 days-6 months?

23 Week Beginning June

May 1986 — June 1986 — July 1986

Week Endin Jun

MON., JUNE 23 174/191	TUES., JUNE 24 175/190	WED., JUNE 25 176/189	THURS., JUNE 26 177/188	FRI., JUNE 27 178/187	SAT., JUNE 28
8	8	8	8	8 WEST COAST COMPUTER FAIR	8
8:15	8:15	8:15	8:15	8:15	8:15
8:30	8:30	8:30	8:30	8:30	8:30
8:45	8:45	8:45	8:45	8:45	8:45
9 DEPT. STAFF MTG.	9	9 DEPT. HEAD MTG	9 OPERATIONS GROUP MTG	9	9
9:15	9:15	9:15	9:15	9:15	9:15
9:30	9:30	9:30	9:30	9:30	9:30
9:45	9:45	9:45	9:45	9:45	9:45
10	10 MICRO GROUP MTG.	10	10	10	10
10:15	10:15	10:15	10:15	10:15	10:15
10:30	10:30	10:30	10:30	10:30	10:30
10:45	10:45	10:45	10:45	10:45	10:45
11	11	11	11	11	11
11:15	11:15	11:15	11:15	11:15	11:15
11:30	11:30	11:30	11:30	11:30	11:30
11:45	11:45	11:45	11:45	11:45	11:45
12	12 LUNCH: HP REPS	12	12 RACQUET BALL	12	12
12:15	12:15	12:15	12:15	12:15	12:15
12:30	12:30	12:30	12:30	12:30	12:30
12:45	12:45	12:45	12:45	12:45	12:45
1	1	1	1	1	1
1:15	1:15	1:15	1:15	1:15	1:15
1:30	1:30	1:30	1:30	1:30	1:30
1:45	1:45	1:45	1:45	1:45	1:45
2	2	2 COMPUTER APPLICATIONS	2	2	2
2:15	2:15	2:15 COMMITTEE MTG	2:15	2:15	2:15
2:30	2:30	2:30	2:30	2:30	2:30
2:45	2:45	2:45	2:45	2:45	2:45
3 I.S./M. STATUS MTG	3	3	3 MTG - ALICE & JOE	3	3
3:15	3:15	3:15	3:15 RE: BUDGETS	3:15	3:15
3:30	3:30	3:30	3:30	3:30	3:30
3:45	3:45	3:45	3:45	3:45	3:45
4	4 DENTIST	4	4	4	4
4:15	4:15	4:15	4:15	4:15	4:15
4:30	4:30	4:30	4:30	4:30	4:30
4:45	4:45	4:45	4:45	4:45	4:45
5	5	5	5	5	5
5:15	5:15	5:15	5:15	5:15	5:15
5:30	5:30	5:30	5:30	5:30	5:30
5:45	5:45	5:45	5:45	5:45	5:45
6	6	6	6	6	6
6:15	6:15	6:15	6:15	6:15	6:15
6:30	6:30	6:30	6:30	6:30	6:30
6:45	6:45	6:45	6:45	6:45	6:45
7	7	7	7	7	SUN., JUNE 29
7:15	7:15	7:15	7:15	7:15	
7:30	7:30	7:30	7:30	7:30	
7:45	7:45	7:45	7:45	7:45	
8	8	8	8	8	
8:15	8:15	8:15	8:15	8:15	
8:30	8:30	8:30	8:30	8:30	
8:45	8:45	8:45	8:45	8:45	
9	9	9	9	9	
9:15	9:15	9:15	9:15	9:15	
9:30	9:30	9:30	9:30	9:30	
9:45	9:45	9:45	9:45	9:45	

66

B:\TRESWK04 as of 24-Apr-85 9:59am

TREASURY SYSTEM - WEEK 4 (4/1 to 4/5)

```
                                        Apr
                                Status  1       1       2       3       4       5

Screens - DE completed          4-1 JK  D #####  .       .       .       .       .
System - revies/demo            4-1 JK  D      # .       .       .       .       .
Screens - DE updates            4-1 JK  D       ##       .       .       .       .
System - user review            4-2 JK  D         .#      .       .       .       .
Database - updates              4-2 JK  D         . ####  .       .       .       .
Screens - tables                4-2 JK  D         .     ###       .       .       .
System - user review            4-3 JK  D         .       . ##    .       .       .
Screens - updates               4-3 JK  D         .       .   ######     .       .
Screens - updates               4-4 JK  D         .       .       .   ######      .
Screens - DE updates            4-4 JK  D         .       .       .       . ##    .
Screens - table updates         4-5 JK  D         .       .       .       .   ## .
Screens - DE updates            4-5 JK  D         .       .       .       .     ##

-------------------------------------------------------------------------------
Legend:  D Done           === ASAP task       _ Slack time (==___), or
         C Critical       XXX Fixed Date        Resource delay (___==)
                          +++ Started         > Conflict
         R Resource         M Milestone
           constrained    ### Done
Scale: Each column equals 1 hour
-------------------------------------------------------------------------------

Time Line Gantt Chart Report                                          Strip 1
```

May

COMPUTER PRODUCTION

LOUIE

Sun	Mon	Tues	Wed	Thurs	Fri	Sat
APRIL S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30	JUNE S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30			1 PFP-------Heavy Processing--- CF--------Heavy Processing--- MS--------Heavy Processing--- EASTERN ORTHODOX HOLY FRIDAY	2	3
4 PFP----------) CF----------) MS----------) EASTERN ORTHODOX EASTER	5 PFF-------Heavy Processing----------)	6	7	8	9	10
11 PFP----------) PFF----------) MOTHER'S DAY	12 LASER DOWN 8-12 PM	13	14	15	16	17 ARMED FORCES DAY
18 PFF----------)	19	20	21	22	23	24
25	26 HOLIDAY MEMORIAL DAY	27	28	29 SYSTEM DOWN 3P-MIDNIGHT DISC DRIVE INSTALLATION	30	31

LOUIE - SYSTEM SCHEDULE

The system is OPEN (with an Operator on duty) from 7:00am till 1:00am Monday through Friday and from Noon until 9:00pm on Saturday, except during scheduled backups.

From Midnight to 7:00am weekdays, before noon on Saturday and all day Sunday, the system is **available** but **unsupported** (there is no operator on duty).

LOUIE IS CLOSED:

Monday	9:00pm - Midnight
Tuesday	9:00pm - 10:30pm
Wednesday	9:00pm - 10:30pm
Thursday	7:00pm - Midnight
Friday	9:00pm - 10:30pm
Saturday	6:00pm - 7:30pm

CF = Corporate Finance

MS = Mortgage Servicing

PFF = Program Finance Funds

PFP = Program Finance Properties

TO: WALLY AND LOUIE USERS

FROM: ALICE J. MARSHALL, OPERATIONS MANAGER

SUBJECT: HP COMPUTER PRODUCTION CALENDAR

Attached is the computer production calendar for May. It shows when heavy production is expected in a number of major accounts. Please use this when planning your production, and if there are conflicts of scheduling, please contact me as soon as possible (at ext. 2554).

On the back of the calendar are definitions of the terms we use for system availability. Use these in conjunction with the calendar to determine when an operator will be available to retrieve reports, and to plan your work so that jobs **finish and print before** scheduled system closed or down time.

Wally's backup schedule is changing. Beginning May 1, **Monday** and **Thursday's**, backup will begin at **7:30pm**, on **Tuesday, Wednesday and Friday** backup will begin at **8:30pm.** Saturday's backup time will remain unchanged, starting at 6:00pm. This change is being made to increase the number of hours available for order processing, particularly in handling heavy volume periods (such as IRA season).

As you remember from April's production calendar, we have shifted full backups from Thursday and Friday to Monday and Thursday. Every morning now, we send the previous day's backup tapes offsite. These changes have been implemented to better insure our systems against data loss.

Lastly, I want to remind you that operator support is now provided every Saturday from Noon to 9:00pm. The operator will retreive reports from the printer room, fill production requests and do the Saturday night backup.

If you have any questions, please contact me at ext. 2554.

THE CON CAP HP3000 SERIES 68 COMPUTER SYSTEMS

DEFINITIONS OF SYSTEM AVAILABILITY

UNAVAILABLE:

The system is not available to users, due to backup, scheduled maintenance, equipment installations or unscheduled down time.

When the system is scheduled to be **down**, all jobs must complete executing and all reports must finish printing at least 15 minutes before the scheduled system shutdown.

After the system has been **down**, all scheduled or waiting jobs must be restreamed, and spoolfiles will be restored by the operator.

CLOSED

- Operator is on duty.
- System is up but is unavailable.
- All jobs streamed are waiting till the system is **open** again.
- Spoolfiles may have reassigned numbers when the system is open again.

DOWN

- Operator may not be on duty.
- System is shutdown.
- Jobs that did not run or finish before the scheduled down time period will be lost and therefor should be restreamed.
- Spoolfiles will have reassigned numbers.

AVAILABLE:

The system is up 24 hours a day, seven days a week, and except for regularly scheduled backup times and maintenance, is available. From midnight till 7:00am, and on weekends, the system is **unsupported** (there is no operator on duty).

OPEN

- Operator is on duty.
- System is available to users.

UNSUPPORTED

- No Operator is on duty.
- System is available to users.

COMPUTER TECHNOLOGIES DEPARTMENT
JUNE 1986 GOALS

CG	1.	Complete the testing of the generic computerized mailing/contact system.
CG, JC	2.	Develop the accounting and billing modules and reports on the Risk Management project.
AJM, JS	3.	Upgrade "Louie" computer to new computer operating system (UA-MIT).
MT	4.	Upgrade 4th generation programming language to new version (Powerhouse 5.01E) for all Investor Services/Marketing programs (QUIZ and QTP).
JS, AW	5.	Complete 4th generation programming language (Powerhouse 5.01E) conversions on "Louie" computer (CCCG, PFP).
AJM, Oper.	6.	Work on K-1 printing and microfiching: - Print 4 funds in-house - Print and mail 2 funds off-site - Fiche 11 funds
MK, AJM, JS, JC	7.	Prepare 1987 (fiscal year) department business plan and budget.
MK, AT, JS	8.	Support Program Finance department: a) Complete the financial reports b) Develop special reports and interfaces c) Multiview training
AW	9.	Implement Sales Materials Management system used by Marketing department.
AJM, Oper.	10.	Printing of quarterly distributions and Broker/Dealer History Report: - 37 hours printing Broker/Dealer History - 100,000 distributions checks printed
AJM, Oper.	11.	Review tape library procedures and propose necessary changes.
JS, AW	12.	Implement Mortgage Administration Contact System using the Generic Contact System.
JS	13.	Provide a 3-day training seminar for QUIZ users.
CG	14.	Complete restructuring of Program Finance notes database (Alicia Anderson).

COMPUTER TECHNOLOGIES DEPARTMENT

The mission of the department is to support the various departments in doing their work by providing appropriate and effective computer resources and related services. The department is organized into four functional groups:

1. Hewlett Packard (HP) Operations - This group maintains two HP 3000 Series 68 super minicomputers and provides operations support from 7:00 a.m. until midnight Monday through Friday, and from Noon until 9:00 p.m. on Saturdays.

2. Systems Development - This group develops, coordinates, and maintains HP applications software currently used by more than 175 people.

3. Micro Support - Installs, maintains, and trains the more than 250 users of our over 200 microcomputers.

4. Administration and Management - Manages the PC loaner program, purchases all computer equipment, and coordinates all activities within the department and the relationship of the department with all other departments.

COMPUTER TECHNOLOGIES DEPARTMENT PHONE LIST

DEPARTMENT MANAGEMENT

MITCH KLEIMAN	3094	Director of Computer Technologies - Interface with all departments, planning, budgeting, equipment requests review

MICROCOMPUTER SUPPORT & DEVELOPMENT

ANDREA WALDMAN	6804	Computer Programmer and PC Specialist - Multimate, Wordstar, Dataease, DOS, lost files/recovery, dead machines
CHRIS GILBERT	6224	Microcomputer Specialist and Trainer - Micro problems, training, Dataease, Lotus, DOS, Wordstar, Multimate
JIM CLARK	7234	Microcomputer Support Specialist - Micro problems, hardware and software requests, data communications

HP SOFTWARE DEVELOPMENT

JOE SEIBERLICH	3054	Software Projects Coordinator - Mortgage Servicing, Treasury, Mortgage Administration, Payroll Reimbursement project, Johnstown ASMI project, SMMS
KIM EVERINGHAM	2624	Programmer/Analyst - Investor Services/Marketing systems, Investor Services, Marketing, CCCG
ARLEIGH TAYLOR	6644	Programmer - Corp. Finance, Program Finance, Multiview GL and AP.
ED HARRIS	2114	Programmer/Analyst - Investor Services/Marketing systems, Investor Services, Marketing
MANNY TOLOUI	2264	Programmer/Analyst - Investor Services/Marketing systems, Investor Services, Marketing

MICRO SUPPORT AND BACKUP HP OPERATOR

Laura Schiro	2584	Install/repairs of micros, PC troubleshooting, weekend operator

COMPUTER OPERATIONS AND SYSTEM MANAGEMENT

ALICE MARSHALL	2554	Systems Manager - Operations scheduling
TONY CHEUNG	7484	Senior Computer Operator - 7:00 a.m. - 4:00 p.m. - Terminal additions, moves, repairs
YOUSSEFF ABED	2574	Computer Operator - 10:00 a.m. - 7:00 p.m. - Terminal additions, moves, repairs
RICKY RIMPLE	2634	Computer Operator - 4:00 p.m. - 1:00 a.m. - Terminal additions, moves, repairs

ADMINISTRATIVE SUPPORT

JAN DARWIN	8254	Executive Secretary - Vendor relations, purchasing, telephone support, contracts administration, special projects support, scheduling of training and classes, arranging of loaners, computer library

COMPUTER TECHNOLOGIES DEPARTMENT

SERVICES OFFERED

MICROCOMPUTER GROUP

- Responding within 2 hours to all requests for micro support.
- Repair of PC's within 8 working hours or swap in of a replacement unit).
- Providing training and on-call support for all of our supported hardware and software:
  - spreadsheet
  - word processing
  - database
  - graphics
  - telecommunication
  - enhancement utilities
- Database system development and maintenance.
- Purchasing of all hardware, software, and peripherals.
- Evaluation of new products to remain current with technological advances.

OPERATIONS GROUP

- Operators are on duty 7:00 a.m. to 1:00 a.m., Monday through Friday and 8 hours/day on Saturdays for production requests and user support.
- Off-site storage coordination and administration.

ADMINISTRATION

- Training classes are offered on an on-going basis for supported software.
- A training room is made available for computer training classes and other functions requiring a large video monitor connected to a PC or to the HP's.
- Computer Library offers a wide selection of books, periodicals, and technical journals that can be checked out.
- Compaq portable computers and a wide variety of software and videos are available to be loaned out to individuals so they may use them with self-paced learning courses, either in the office or at home.
- "Bits and Pieces" newsletter, published bi-monthly, provides schedules of training classes offered, and updates on new developments in hardware and software that have been added to the Computer Library and tips and news items of interest to computer users.

SYSTEMS DEVELOPMENT GROUP

- Review of application specifications.
- Technical consulting, systems design, and applications programming.
- On-call support for applications.
- Technical project management.
- System and utilities software maintenance and updating.

Hardware/Software Standards

The following is a list of our currently supported products:

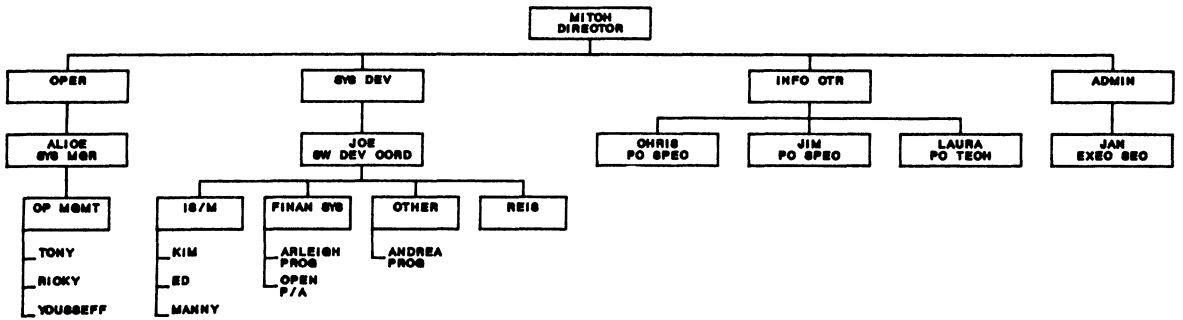
Hardware:
- IBM PC/XT/AT;
- Compaq portable;
- Compaq Deskpro;
- HP Portable;

Printers:
- HP Laserjet;
- Epson LQ-1000/1500;
- Epson FX/MX 80/100;
- HP Thinkjet;
- Okidata 2350;
- IBM Quietwriter;
- Anadex;
- NEC 3500/7700;

Software:
- DOS 2.1/3.1;
- WordStar;
- WordStar 2000;
- MultiMate;
- Lotus 1-2-3;
- Lotus ReportWriter;
- DataEase;
- dBASEIII;
- Sidekick;
- Timeline;
- Reflections (PC2622);
- Fastback;
- CompuServe EMail
- Crosstalk.

COMPUTER TECHNOLOGIES DEPARTMENT

ORGANIZATIONAL CHART



ORGCHART/JC/5/86

Consolidated Capital - Computer Technologies - Budget Worksheet - 8/14/85

	SEPT	OCT	NOV	1st QTR	DEC	JAN	FEB	2nd QTR	MARCH	APRIL	MAY	3rd QTR	JUNE	JULY
HARDWARE MAINTENANCE														
HP Series 68, SMMC contract														
HP Series 68, off hours														
HP terminals,printers, micros(VRS)														
Off-site storage														
Power Equipment Maintenance														
Air Conditioning Maintenance														
Tape Cleaning														
Hard Copy Terminal Maintenance														
TOTALS														
**********														
SOFTWARE MAINTENANCE														
HP FOS and Software														
Robelle - Supertool,Qedit														
Cognos Powerhouse														
Adager														
HP special consulting														
VESOFT - Mpex,Streamx														
DISC - Omnidex,Dbmgr														
RA ASSOC - Dcas/3000														
Other														
TOTALS														
**********														
VENDOR EXPENSES														
Paper														
Special Forms (eg. checks, invoices, etc.)														
Labels														
Ribbons														
Tapes														
Cabling														
Tape Drive Cleaning Supplies														
Terminal/Printer Cleaning Supplies														
Additional Equipment														
Outside Consulting Services														
TOTALS														

COMPUTER TECHNOLOGIES DEPARTMENT - THREE YEAR PLAN
OPERATIONS GROUP

	Year One	Year Two	Year Three	Total
Hardware Maintenance				
Software Maintenance				
Vendor Expenses				
Paper				
Operating Supplies				
Operating Staff				
Department Overhead				
Training				
Dues and Subscriptions				
Travel and Entertainment				
Hardware Purchases				
Software Purchases				

PROJECT STATUS REPORT

DEPARTMENT	: RISK MANAGEMENT	BUDGET	: ESTIMATE
PROJECT	: CLAIMS MANAGEMENT SYSTEM	TARGET DT	: ESTIMATE
PROJECT MANAGER	: DEBRA MOORE	ACTUAL DT	:
PROJECT LIAISON	: CHRIS GILBERT/JIM CLARK	STATUS	: 15% DONE
REPORT STATUS	: New		

Risk Management needs a system to manage insurance claims made by properties we insure and to handle premiums and payments.

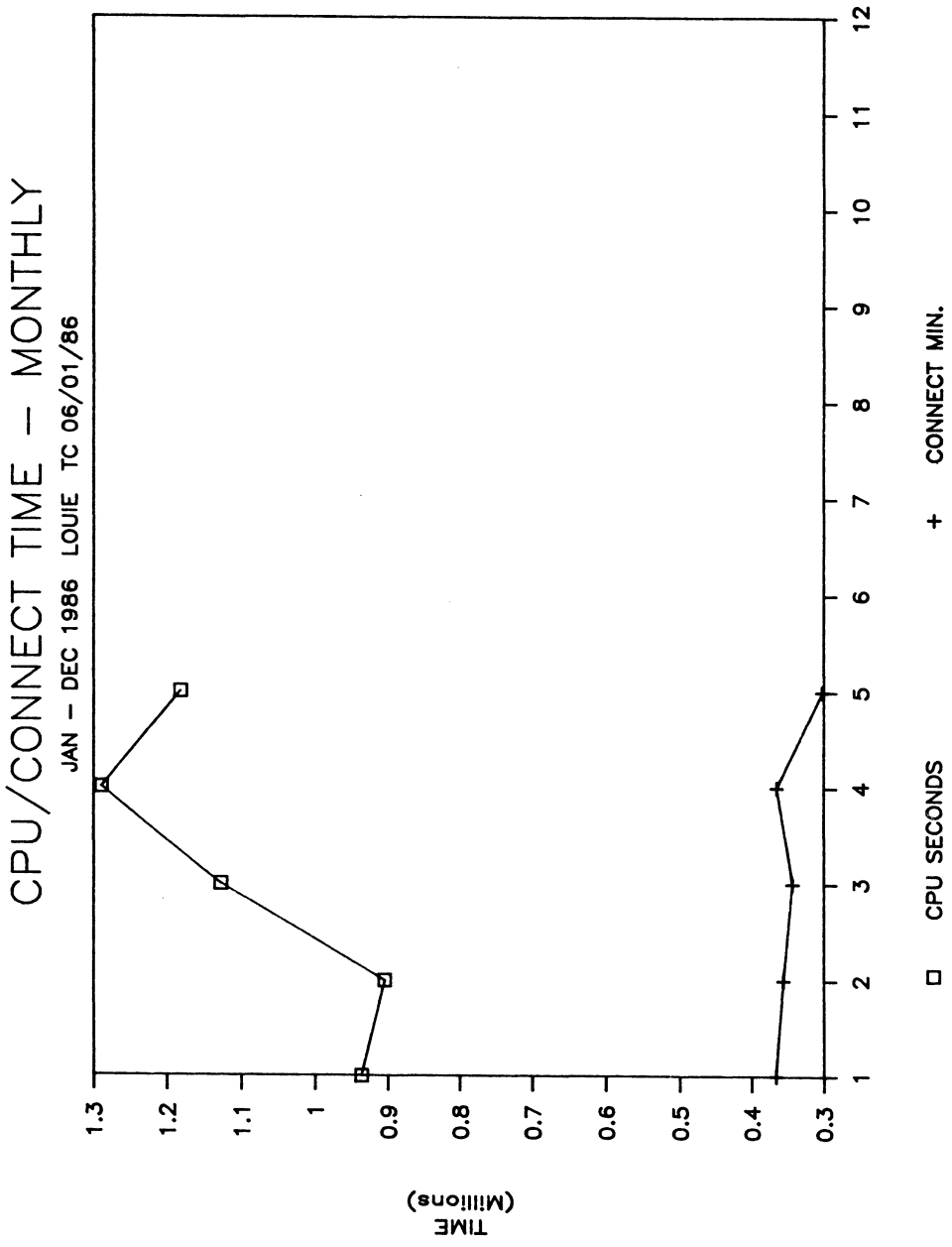
The initial specifications are being completed and will be reviewed by Risk Management. Test screens and databases are also being reviewed.

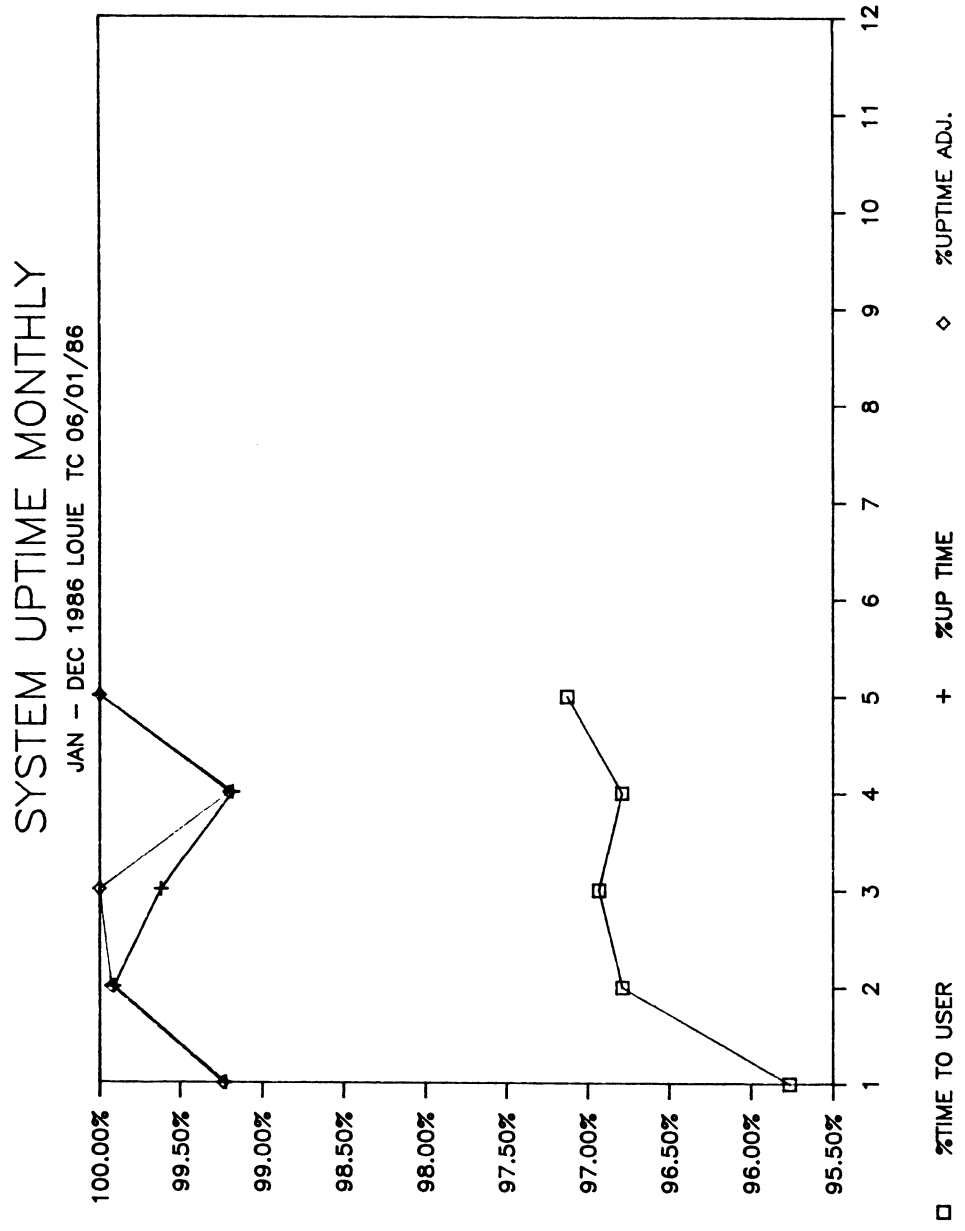
Time and cost estimates will be completed during May.

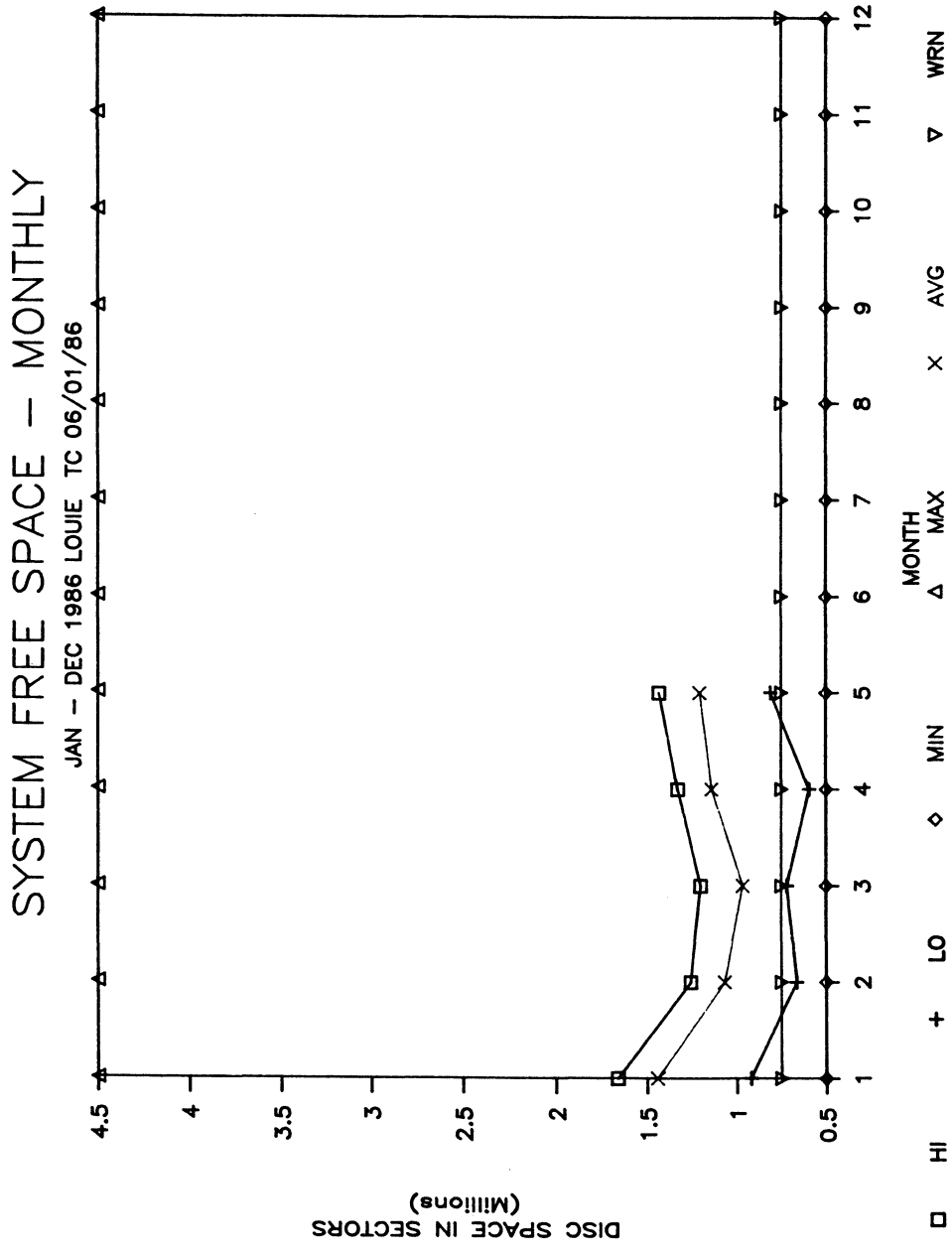
DEPARTMENT	: TREASURY	BUDGET	: 33,600.00
PROJECT	: LOAN/GNMA SYSTEM	TARGET DT	: 2/28/86
PROJECT MANAGER	: MADALYN SCHAECHER	ACTUAL DT	:
PROJECT LIAISON	: JOE SEIBERLICH	STATUS	: 95% DONE
REPORT STATUS	: Updated		

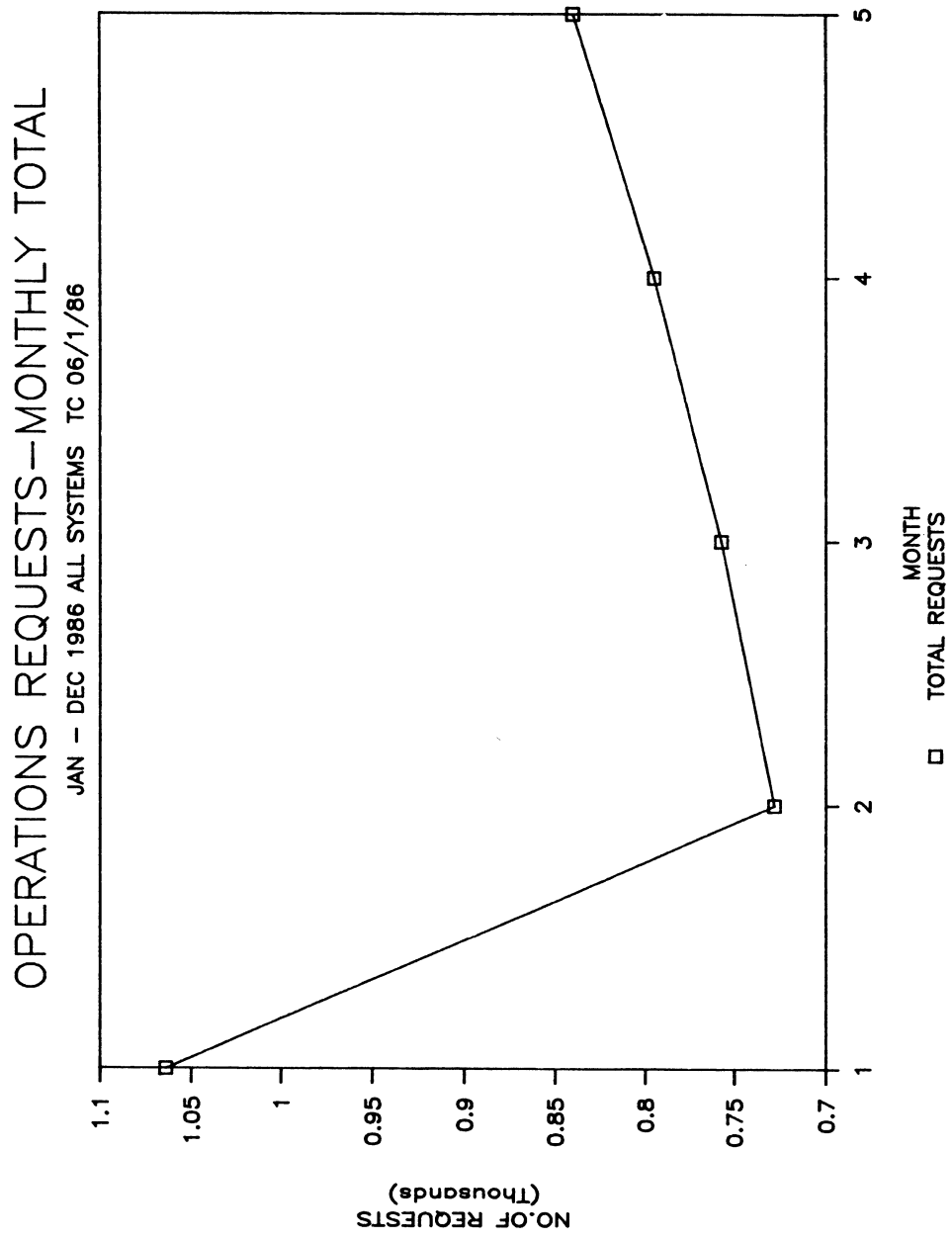
The Treasury Information system will manage Consolidated Capital's portfolio of investments which in the past was done manually. The new Treasury system was installed in two parts, the Investments module and the loan/GNMA module. The final installation of these was completed on 4/31/86 with a final cost of 33,460.00 dollars.

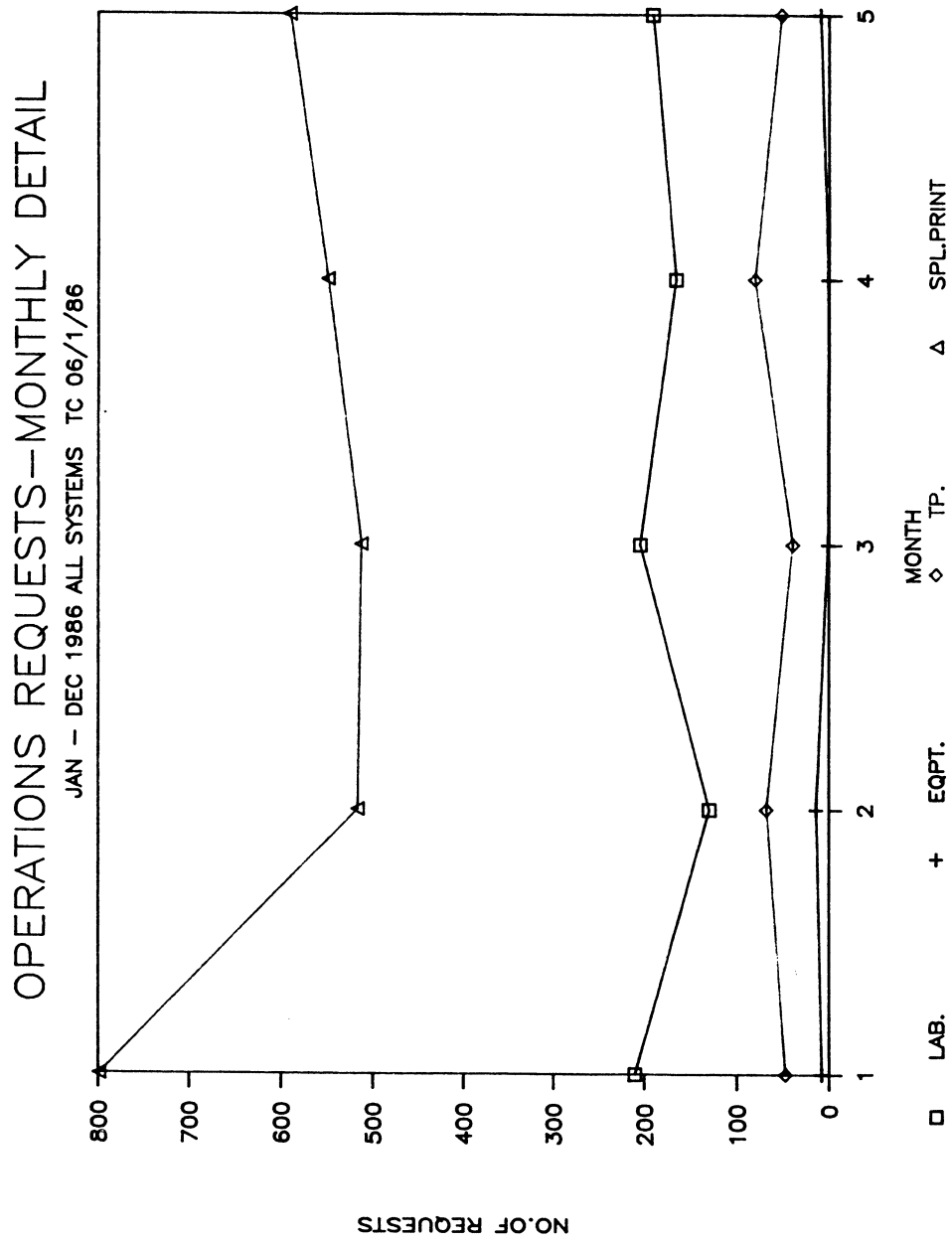
Programming and testing continues on the interface to Program Finance.


CPU/CONNECT TIME – MONTHLY
JAN – DEC 1986 LOUIE TC 06/01/86
TIME (Millions)
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
1
2
3
4
5
6
7
8
9
10
11
12
CPU SECONDS
CONNECT MIN.


SYSTEM UPTIME MONTHLY
JAN – DEC 1986 LOUIE TC 06/01/86
100.00%
99.50%
99.00%
98.50%
98.00%
97.50%
97.00%
96.50%
96.00%
95.50%
1
2
3
4
5
6
7
8
9
10
11
12
□ %TIME TO USER
+ %UP TIME
◇ %UPTIME ADJ.

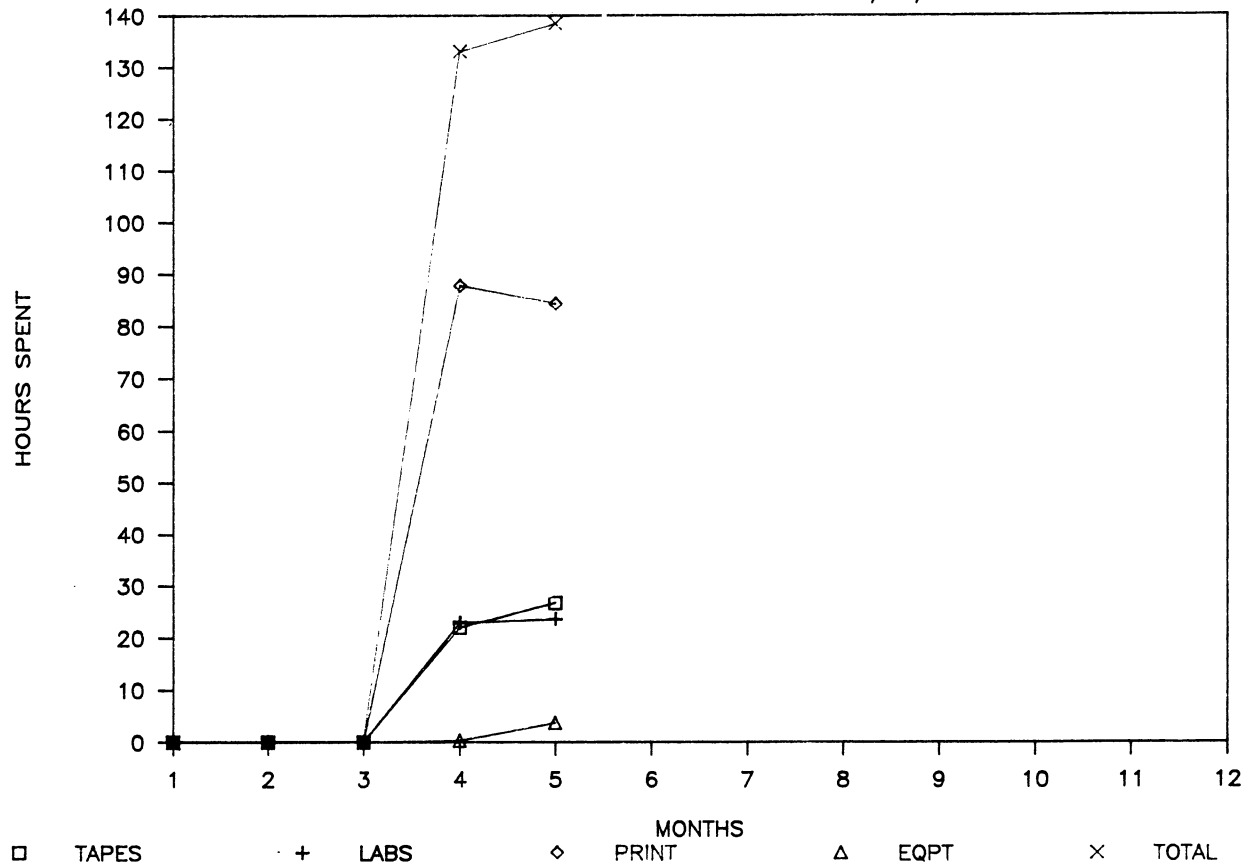

SYSTEM FREE SPACE – MONTHLY
JAN – DEC 1986 LOUIE TC 06/01/86
DISC SPACE IN SECTORS
(Millions)
4.5
4
3.5
3
2.5
2
1.5
1
0.5
1
2
3
4
5
6
7
8
9
10
11
12
MONTH
□ HI + LO ◇ MIN △ MAX × AVG ▽ WRN


OPERATIONS REQUESTS—MONTHLY TOTAL
JAN – DEC 1986 ALL SYSTEMS TC 06/1/86
NO.OF REQUESTS (Thousands)
1.1
1.05
1
0.95
0.9
0.85
0.8
0.75
0.7
1
2
3
4
5
MONTH
TOTAL REQUESTS


OPERATIONS REQUESTS—MONTHLY DETAIL
JAN – DEC 1986 ALL SYSTEMS TC 06/1/86
NO.OF REQUESTS
800
700
600
500
400
300
200
100
0
1
2
3
4
5
MONTH
□ LAB.
+ EQPT.
◇ TP.
△ SPL.PRINT

PRODUCTION TIME SUMMARY — MONTHLY

APR – DEC 1986 ALL SYSTEMS TC 06/01/86



04/22/86
MK

List of Requested Systems

1. DW Computerized Address System (3/17/86).

2. Prospect Tracking and Tickler System for Anthony Papadakis (3/24/86).

3. Real Estate Legal System to maintain a directory of outside counsel, keep records of each transaction by attorney and billings, and perform comparison studies, requested by Eric Horodas. (4/1/86)

4. Mortgage Database conversion to new Dataease to take advantage of multiple file structures, etc., requested by Alicia Anderson/Rich Dooley. (4/10/86)

5. Replacement of data entry system and customized report system on Odin by an inhouse system, requested by Payroll/Corporate Finance - Mindy/Maurice. (4/17/86)

6. Property Sales Contact System moved to HP - develop a system with security, lookup limitations, multiple user and sorting capabilities, download abilities, requested by Mike Bosch/Weston Munselle. (4/22/86)

7. Add to Investor Servies/Marketing System - Bar Coding of all mail so that return mail is bar coded and then loaded into a job that flags records not to allow mailing from them until the address info is updated, requested by Janet Gillespie, she estimated it would save $30,000/year in mail costs. (4/22/86)

CONCAP MICROCOMPUTER INVENTORY BY DEPARTMENT
CORP FINANCE
04/17/86

ITEM	ASSET #	SERIAL #	USER	UPDATE
PERSONAL COMPUTER				
PC/FLOPPY DISK	0610	61370505160	N.ZIMMERMAN	03/29/86
	0671	61307665160	T.YEH	03/29/86
	0023	05898355150	A.YEH	03/28/86
	0095	05898545150	N.KAPELLAS	03/28/86
	0029	06171575150	T.DAVIDSON	03/28/86
	02162	15444195150	J.AMAR	03/29/86
	0564	14238015150	D.HARVEY	03/29/86
PORTABLE/COMPAQ	0572	1504050741	C.FONG	03/29/86
	03788	1422020459	R.RICKS	03/29/86
	0058	32598-SD	M.HIDALGO	03/29/86
	0258	1416010189	GROUP	03/29/86
	0241	3492232-S	J.TUCKER	03/29/86
	0617	1443050827	KEN RAVEY	03/29/86
	0289	1416010197	W.TAYLOR	03/29/86
	0632	1431021716	K.BODHAN	03/29/86
	0439	1432021201	D.MITCHELL	03/29/86
	0449	1432021050	M.EDWARDS	03/29/86
MONITOR				
AMDEK 310A	0670	5090293	D.HARVEY	03/29/86
	0683	5091269	N.ZIMMERMAN	03/29/86
	0669	5090295	T.YEH	03/29/86
	0186	4105714	A.YEH	03/29/86
MONOCHROME/IBM	03751	0483011	J.AMAR	03/29/86
	0097	0094356	N.KAPELLAS	03/29/86
	0030	0743175	T.DAVIDSON	03/29/86
PRINTER				
LQEPSON	0333	035012	D.HARVEY	03/29/86
	0340	012093	GROUP	03/29/86
	0360	012094	N.ZIMMERMAN	03/29/86
	0616	88081	T.YEH	03/29/86
	0614	079953	N.KAPELLAS	03/29/86
	0233	012539	A.YEH	03/29/86
NEC3500	03787	L28255	T.DAVIDSON	03/29/86

TOTALS

COMPUTER:	17
PRINTER:	7
MONITOR:	7
MODEM:	0
EXT.HARD DISK:	0

MASTER DEPT LIST

CONCAP INVENTORY - SUMMARY ITEM-NAME REPORT
04/17/86

ITEM	COUNT	COST
FX/185 EPSON	1	540.00
FX100/EPSON	21	17,177.00
FX80/EPSON	2	1,345.00
IMAGEWRITER	1	450.00
LASERJET	11	27,302.00
LASERJET PLUS	1	2,879.00
LQ1000	2	1,543.00
LQEPSON	15	18,405.00
MANNESMAN/TALLY	1	1,000.00
NEC3500	7	10,230.00
NEC7700	6	16,528.00
OKIDATA	16	17,171.00
QUIETWRITER	23	23,693.00
QUME	1	1,600.00
RX80/EPSON	1	499.00
THINKJET	19	8,064.00
THINKJET HP-IB	2	870.00
TOSHIBA	6	11,325.00
TRANSTAR	1	1,000.00
	160	207,007.00

PLOTTER

COMPUTERIZATION BY DEPARTMENT AND LOCATION
AS of MARCH 1, 1986

DEPARTMENT	TOT % COMP FLD	TOT PC/TERM HDQTR	TOT % COMP HDQTR	% COMP FLD & HDQTR
ACQUISITIONS	100%	10	125%	115%
CCCG (1)	0%	11	92%	92%
CORP COMMUNICATIONS	0%	12	63%	63%
COMPUTER TECHNOLOGIES	0%	25	179%	179%
CORP FINANCE	0%	28	97%	97%
CORP ADMINISTRATION (2)	0%	3	18%	18%
HEADQUARTERS	0%	4	36%	36%
INVESTOR SERVICES	0%	49	129%	129%
LAND SYNDICATION	60%	7	88%	77%
LEGAL CORPORATE (3)	0%	10	38%	38%
MARKETING	7%	14	61%	25%
MORTGAGE ADMINISTRATION	50%	11	100%	87%
MORTGAGE FINANCE	0%	5	45%	45%
MORTGAGE SERVICES	0%	23	100%	100%
OFFICE OF THE CEO	0%	1	100%	100%
OFFICE SERVICES	0%	2	22%	22%
PERSONNEL	0%	5	100%	100%
PORTFOLIO MANAGEMENT	0%	3	100%	100%
PROGRAM FINANCE	0%	48	100%	100%
PROPERTY SALES	75%	6	60%	64%
REAL ESTATE LEGAL	0%	4	50%	50%
TELECOMMUNICATIONS	0%	2	29%	29%
TREASURY	0%	6	100%	100%
WORD PROCESSING	0%	1	13%	13%
TOTALS	25%	290	82%	73%

STRATEGIC PLANNING IN SMALL MIS SHOPS

TERRY W. SIMPKINS
Spectra-Physics - Laser Systems Division
959 Terry St.
Eugene, Oregon 97402

Strategic Planning, what is it, why is so much attention being paid to it these days, why should it be done, and assuming that it should be done, how does one go about it? All very good questions that don't have obvious answers but need to be understood by every MIS manager; especially in small shops where resources are extremely limited.

WHAT IS STRATEGIC PLANNING ?

This paper does not pretend to be the definitive explanation of the topic, but rather to highlight the experiences of one small shop that has gone through the exercise. The state of the literature on strategic planning for MIS outlines a massive project covering every conceivable aspect of systems, lasting well over a year, employing several dedicated people, and having a considerable price tag. The results of this process is a document of considerable weight and volume. Every detail of future systems would be explained, and a considerable amount of the initial design work for these future systems might well be included. Detailed forecasts of the expected volume of transactions going through the systems may be outlined along with pages of explanations for the volume changes. While this may be possible (even reasonable) for a large installation (say Ford Motors), the concept is laughable in a small shop of say 5 people. The small HP3000 shop normally consists of 1 to 6 people and is completely buried already, the thought of adding something of this magnitude is simply not within reason. Faced with this situation, we undertook the task of developing a workable strategic plan. However before we could begin the process, we had to develop the methodology that would be used. This is a description of that methodology and some of the reasoning that went into it.

For the small shop it is very important to plan, perhaps even more important than for large installations, since we as small entities are more sensitive to change. But we must also consider the very limited resources available to us and keep the amount invested in the planning process in perspective. Because of this, our effort will by definition be much smaller and of less detail. The crux of the issue is to develop realistic expectations of the planning process and to resist the desire to develop an all encompassing document that is all things to all people. Define exactly what you expect the process to produce and focus on the activities that will address those goals, and stick to them. In my opinion, realistic expectations are developing an outline of what the

upper management of your company (or division) expects the environment to be for the next several years, and what are the types of systems required to support that environment. Any more detail, and the level of confidence drops quickly.

WHY PLAN?

If you are considering the development of a Strategic Plan, I assume that you are aware of the reasons planning is important. Perhaps you are here because your boss has told you that you are going to develop a plan. Whichever is true, a quick review of the major reasons that I see for planning is in order.

1) Businesses change over time, as do the systems requirements of the business; while installed software stays perfectly constant without human intervention.

2) Non-trivial computer systems take a considerable length of time to develop and install. This may include definition, design, selection, coding, training, etc. Whether you create or purchase the software, it is not an overnight project.

3) All resources are finite, and we want to make the best possible use of them. If we can avoid spending valuable resources on short term needs in favor of addressing needs that will be with us for a longer period of time, we have probably gotten more value for those resources. Likewise, if we address the most important needs of our users first, we develop credibility with management and that has several benefits.

4) By having a better understanding of where we are headed we minimize "mid-course corrections". This saves valuable time and effort and gives the appearance that we know what we're doing.

5) We generate enthusiasm and user buy-in when they understand the direction systems are taking. Even if they don't totally agree with the direction, they will be more cooperative if they at least know what is going to happen.

There are more reasons why a Strategic plan should be developed, but as MIS professionals, you should already understand them and if you don't there is plenty of published material on the subject.

PURPOSE OF THE MIS PLAN

1) The MIS plan should provide an overview of the state of the current systems and how they got where they are. This historical perspective may not add value to the future plans, but will provide good background understanding for why we are planning now and some of the problems that can be avoided by planning.

2) The plan should provide insights into the direction of MIS both from a broad general perspective as well as some of the specific needs that are identified. And a general timeframe for when these changes should be in place.

3) You need to measure how far you currently are from where you want or expect to be. This provides a basis for priority setting and resource allocation. It also prepares management for the requests you are going to make and gives them a measure of how reasonable and realistic the plan.

4) Finally, once the target is established and you understand how far away you are from that target, you can outline what is needed to move toward your goals and how quickly you can expect to get there.

COMPOSITION OF THE MIS PLAN

I. Introduction/Narrative Summary

In order to provide a basis for the plan a brief historical recap of the current set of systems is included to help define where we are today. Trends are discussed as well as the reasons for the major decisions that have been made. The purpose is not to justify or condemn, but to provide better understanding of why we are where we are. The next section covers where we want to go in very general terms. This is the visionary portion of the plan, my chance to gaze into the crystal ball and present my vision of what MIS should be and the role it should play in the business. Included in this are the type of resources required to support that vision, people, hardware, software, business commitment, etc. All of the requirements may not exist, but I can describe what I need anyway. A review of the current role of MIS is included to compare and contrast the current environment with my vision of the future. Next, a review of the MIS organization; describing how we are organized, the equipment currently in place and a recap of the department's strengths and weaknesses. Finally, a brief comparison with other MIS departments in organizations similar to ours. Size of staff, machine capacity, services performed, and budget as a percent of sales comparisons provide a relative measure of our performance.

II. Existing Systems Profile

For each system, a narrative is created covering in detail the basic function of the system, state of the code, documentation, user understanding, and MIS understanding. Also discussed are the things the system does well and the areas that the system doesn't address or that it addresses poorly, including major known bugs or omissions. A general description of the backlog of change requests for the system provides a basis for management to judge the validity of our assessment of the system. We then recap the expected impact of these changes in terms of cost, time, stability and probability of success.

Ideally your business has or will develop a Strategic Plan for the entire business, if so you will be able to use that plan as input to the MIS plan and build from it. If not then portions of that general business plan must be developed either explicitly or implicitly in order to proceed further.

III. Comparison of Existing and Future Operating Environments

For each functional area of the business a projection of the future environment is required if we are to understand and project the information and systems requirements. This projection must be that of the functional manager, it is not required that he/she actually create it, but he/she must agree with it and be willing to sign his name to it. The purpose of this projection is to compare the current environment with that of the future and to understand the impact of the changes on the information needs of the business and the ability of the current systems to meet those needs. Each functional area is divided into the processes that compose it. For example Marketing might be broken into the following subcategories: Promotion & Advertising, Sales, Market Research, and Market Planning. For each of these areas, a profile is created listing the important attributes, a description of the current environment and the expected future environment (see form #1). The information on this form is strictly business oriented, it should be completed by the users and not consider systems at all. If a Strategic Plan already exists, or is being developed in conjunction with the MIS plan, this should be a part of that plan. If not, you must develop it at this point in the process, since the rest of the plan builds upon it.

The next step in the process identifies the important "information systems" used in the operation of the business. These systems are not necessarily computer systems, but rather the way information is collected, arranged, or used. Systems could be defined as processes or functions (i.e., Master Scheduling, Purchasing, etc), or along the lines of the current computer software packages if that is felt appropriate. The important thing here is not the way "information systems" are defined, but the listing and explanation of the critical factors that impact these systems and the informational nature of these critical factors. The intent here is to highlight the type of information that is needed to support the future business environment. Form #2 is one method of ferreting out this information. Each critical factor is then described according to how is is impacted by the seven information characteristics listed across the top of the form. Any factor listed must, by definition, relate to at least one of these characteristics. It may relate to more than one, but seldom to all of them. These descriptions are then the basis for evaluating current systems and any alternative systems or designs (see forms #3 & #4). Because these factors are "critical" and they are impacted by certain information characteristics, they are the obvious basis for analysis and comparison of alternative systems. Of course there will be other criteria in the selection process such as cost, complexity, support available, interconnectivity with other systems, etc.; but this list of critical

factors and the information requirements will be a vital part of the requirements definition.

In our case a proposed replacement system had already been identified and we focused our attention on that alternative. The process being that if that alternative didn't sufficiently meet the user's needs, then we would start from scratch to evaluate other systems. It should be noted here that the outcome of completing form #3 could be that the current systems satisfactorily meet the needs of the enterprise. In fact this conclusion would be expected where systems had been recently replaced. This would not mean that the exercise was wasted effort, but rather that those involved in the planning process had verified that business needs were being met. Businesses periodically examine the market segments they participate in to insure that they are in the correct ones, and should likewise examine their systems to insure that they are appropriate and providing the required information.

It is critical that users complete these forms. The MIS staff should assist to insure that a proper level of detail is included and that the descriptions are measurable and quantifiable; however, in order for there to be a meaningful result the users must "own" the information that comes from the exercise. In my experience, the generation of the forms used in the project is very important. The format of the information gathering must assist in the extraction of the information required to construct the plan or you will find it very difficult to communicate your goals to all involved parties. The forms should lead the users through the process and force them to provide the required focus on business issues and information requirements. Another tool to help and guide the users is to create a "straw man" list of business attributes, information systems and critical factors for each of the areas. This list should not be cast in stone, but should provide "food of thought" for the users, seeds to start their thought process.

As the future information needs of the various areas are assembled, the cost of meeting the needs and the impact of not meeting these needs must be understood, along with the relative importance of the needs in order for management to make correct resource allocation decisions.

IV. Plans to Meet Future Needs

Once the future needs of the business have been identified, quantified, and ranked, a plan to meet those needs must be developed. This may include the selection process for new software, a list of modifications to current software, and an installation/project plan. Project management skills and tools play a very important role in this section of the plan. It is not critical that the actual details of the process be included in the plan, rather that the process that will be used is defined and understood so that management can buy off on it. This section will also include the sequence of projects to be undertaken. This will reduce the "mid-course corrections" caused by misunderstood

priorities and increase the confidence level in the plan since management has reviewed and "blessed" it. This is the place to address the hardware required to make the software plan possible. Where possible, link hardware needs with specific projects, to reflect the true cost of various alternatives and projects. Keep in mind that hardware requirements are usually a step function and seldom follow a smooth curve.

HUMAN RESOURCE PLANNING

As a part of your plan be sure to address the people portion of your department. This is your greatest asset and deserves attention just like your software and hardware. Succession planning, career development and professional training are all part of a complete Human Resources Planning effort (see form #5). Personnel planning is important for several reasons. Much time and money has been invested in your programming and operations staff to get them to their current level of understanding of your systems. By failing to provide an effective career plan for these employees you risk loosing them and having to reinvest in training their replacement. This retraining effort also has opportunity costs associated with it in addition to the cost of the training; that being the value of the projects that cannot be undertaken while this training is taking place. these costs can very often be avoided with good career planning. Remember too, that happy employees are more willing to work the long odd hours often required in our profession and are always more productive than unhappy ones. By developing your staff and expanding the knowledge base of each employee you also reduce your exposure in the event that an employee does leave. By having several people who can perform each job or support each system, you have designed a back up for everyone, and you improve the quality of systems because interactions and interfaces are better understood by everyone.

As stated at the beginning, this paper is not designed to be an exhaustive study in the art of Strategic Planning, but rather to reflect some of the insights gained by performing it in a small shop environment. All portions will not be applicable to every installation, but the general requirements and approaches I feel are common through all companies. Feel free to use the forms and modify them to best serve your needs.

BUSINESS ENVIRONMENT

Today vs. Future

Functional Area

Department

Business Attribute	Today	Future (5 Years)

Functional Area

Department

Change in Information Needs
Due To
Changes in Business Environment

Information System (Not Computer)	Critical Factor That Drives or Impacts the Information System	Timeliness — How Quickly Updated & Available	Detail Level	Information Commonality — Same Data Across User Groups & Which Groups	Arrangement — Indexes, Sorted Order	Time Horizon — How Much History Is Required	Accessibility — Who Can Access Data	Accuracy — How Critical Is Absolute Accuracy

GAP ANALYSIS

Information Needs vs. Current System

Functional Area _____

Department _____

Information/Functional Requirement	Ability of Current System to Provide

GAP ANALYSIS

Information/Functional Requirement	Ability of Current System to Provide

GAP ANALYSIS

Information Needs vs Proposed System

Functional Area

Department

Information/Functional Requirement	Ability of ASK System to Provide

GAP ANALYSIS

Information/Functional Requirement

Ability of ASK System to Provide

Are assumptions upon which the strategic plans are based realistic regarding human resource requirements?

Over time, what skills will become obsolete, change in nature, or be eliminated?

For what functional skills/positions are we likely to encounter a shortage of qualified candidates in the marketplace, now or in the future?

Do the present managers within the function have adequate technical/managerial skills to meet the strategic changes occurring at LSD?

What are the principal H/R obstacles to achieving the function's strategic objectives?

Are age patterns in the organization imbalanced, suggesting high future attrition or career path blockage?

Is there adequate or excessive turnover in any group, at any particular level?

Is there a proper balance (staff mix) of managerial, professional, technical, and support staff within the function?

What are the most significant skill deficiencies within the function organization? How will such gaps be addressed?

Are the organization and structure and staffing of the function appropriate for the achievement of strategic objectives?

To what extent will qualifications for existing positions change in light of strategic plans? How will such changes be addressed?

Do the strategic plans/objectives call for projects or processes that have no precedent at LSD? What are the implications for staffing requirements? Design of the present function organization?

Which positions, if not filled, will have the most detrimental effect on achieving the function's objectives?

What impact would a product line de-emphasis or discontinuance have on the responsibilities of the function staff?

The Future of Data Processing Management

by Richard A. Frost
FUTURE IDEAS Inc.
P.O.Box 5406 Chicago, IL 60680

In many ways the nature of data processing management in the future will be no different than it is today. Those persons who expect you to predict their needs for tomorrow and satisfy them yesterday will probably still continue to do so tomorrow and the day after as they have yesterday and the day before. Those persons who do not understand why it takes data processing so long to fulfill their needs when they're sure they could do it much faster themselves on their personal computer will probably, in all likelihood, continue to feel that way in the future as they have in the past. So the human element of data processing in the future will not change from what it is today, since humans are not expected to change drastically in the next ten to twenty years.

The nature of other aspects of data processing management will undoubtedly be drastically different in the future than it is today. These aspects being affected by the hardware advancements, increasing computer power and decreasing cost and size, data communication software and hardware. These advancements bring possibilities and options regarding the distribution of computer systems, and fourth generation languages and similar products which drastically change the way we look at system development.

While all these technical advancements bring with them the promise of untold wealth and ease and glory, we also know that there is probably, as there always is, some price to pay for these technical advancements. This price, as I see it, is the complexity which these advancements can bring about into the data processing environment. In many ways, I see this complexity as one substantial threat which could face many data processing departments and information centers of the future. Complexity in and of is own rights carrys a price tag: a price tag of time and money for the management and control of this complexity. How then can this complexity be controlled? The first approach to controlling complexity is the typical fight fire with fire.

The Future of Data Processing Management

by Richard A. Frost
FUTURE IDEAS Inc.
P.O.Box 5406 Chicago, IL 60680

By this I mean the data processing department becomes larger and more complicated with a broader scope and more levels. There is however a turning point which one must be aware of, in the decreasing economy of size. By this I mean simply that once a department grows to a certain level of complexity, it in itself will stop being the cure for the problem and become part of the cause of the problem. The other basic approach to attacking the problem of complexity is the old KIS rule - Keep It Simple. This in contrast to fighting fire with fire is to fight complexity with simplicity and thereby avoid the problem. By this approach I do not mean to forget fourth generation languages and stay with third; forget hardware advancements or data communication or other advancements. Just the opposite is needed in order for a business and a data processing center to remain viable in the future. They will need to use many of these tools. Realize them for what they are - tools. The end product that they produce does not need to be complicated. In contrast, in order to make it more manageable we will want the product of these sometimes complicated tools to be simple. As an example, rather than have an application written partially in third generation, partially in fourth generation and with a variety of hardware and database management utilities, it would be much better to have it all written in one language using a common database management system and hardware, since multiple languages would mean multiple staff or staffs capable of supporting both languages for maintenance of the application.

The other advent of advancements and complications which come with them is that now and in the future, while the questions may still be simple and the problems at their core may still be simple, the solutions are becoming more and more complicated.

The Future of Data Processing Management

by Richard A. Frost
FUTURE IDEAS Inc.
P.O.Box 5406 Chicago, IL 60680

By this I do not mean that the applications need to necessarily be more complicated, but merely that the decision processes used by data processing management must become more thorough and face a barrage of options and possibilities. The question of "Are programmers going to need to become more productive?" can bring hundreds of considerations of hardware and software that would make the development staff more productive. Quite commonly we might find ourselves forgetting this raft of possibilities or maybe even voluntarily ignoring them in lieu of making a decision. The decision process itself, what products to use; software, hardware, data communication to apply to problems, becomes more and more complicated in light of the technical advancements both currently and into the future.

What follows is a checklist of points to consider when attacking these problems in the area of technical advancements and choices on what products to be used. This list might not be definitive, but it will hopefully serve as a good outline to keep us all honest in our parts of the decision process.

1. What is the level of complexity in the end product or result of using this tool or tools? The very simple rule here applys that the more complicated the end result, the more likely to have bugs in the code or difficulty in the data communication connections, etc.

2. Know exactly what your needs are. It seems almost trivial to say that you must know the question before you can come up with an answer. However, this is a very simple rule that we sometimes forget.

The Future of Data Processing Management

by Richard A. Frost
FUTURE IDEAS Inc.
P.O.Box 5406 Chicago, IL 60680

3. Know your current operating environment and resources for solving any problem or need. At some point we're all quilty of thinking we have more resources or abilities and over committing ourselves.

4. Be prepared with the knowledge of what resources or tools are potentially available to you. Essentially this means be prepared and know what products are out there. Simply because there are so many products, both hardware and software, available in the current marketplace, each with different attributes, even for solving the same problem, you must have some basic knowledge and some prepared knowledge of what is out there even before the question or problem arises. Otherwise you will have typically never have had a chance to collect a thorough base of the available solutions in time for when the decision has to be made.

5. Identify and prioritize the attributes you want and the tool to be used. This is to say quite simply, know what you want, not just in the answer, but in the solution you take to get there, not only qualitatively but also quantitatively.

Finally, examine each of the options, apply the rules both qualitatively and quantitatively in what you are looking for in a solution and last be not least, make sure it really does solve the whole problem. Hopefully these rules can help in attacking the complexity of the tools available for solving the current and future data processing problems.

The other issue to address is the fact that none of us have a true crystal ball. None of us know what the products or hardware or software of tomorrow will be capable of and how we might be able to get there from where we are today. Perhaps the golden rule for data processing management in the future is to allow flexibility in the tools and the solutions chosen. This flexibility and the products chosen may very well turn out to be a saving grace in the future.

The Future of Data Processing Management

by Richard A. Frost
FUTURE IDEAS Inc.
P.O.Box 5406 Chicago, IL 60680

Another trend in the future of data processing management is that of internal vs. external resources. In the not so distant past, the majority of software products were created internally. The majority of the utilities for development and for system managment were developed internally. Gradually that trend has been changing and a larger number of the utilities and products used in data processing management come from external sources. It is forseeable in the future that the majority of software products, applications, and utilities will originate from outside third party vendors, and hardware and software vendors, since no one really wants to reinvent the wheel. Here again, apply the rules which we discussed previously to insure a good relationship between all hardware and software resources used.

Introduction to Step-By-Step

by Michel Kohon
Tymlabs Corporation

Background

Since we introduced the Step-by-Step idea, we have found some discrepancies between what it means and what people think it means. The objectives of this document are:

- to explain Step-by-Step for end-users,
- to specify the Step-by-Step method for DP professionals,
- to give Step-by-Step's main advantages and its possible adverse consequences.

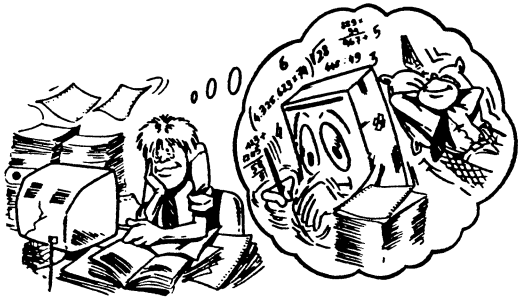
Toward Step-by-Step

Description of Data Processing:

To understand Step-by-Step, it is essential to analyze the DP function. Like all other human beings, DP personnel have to deal with two concepts:

- the reality,
- the users' dreams (also called requirements).

The reality is a difficult thing to change, to transform or simply to understand. Similarly, the users' requirements are difficult to change, to transform or simply to understand. From the reality, the system analyst or programmer will try to develop a system which fulfills the users' dreams. Users can see the impossibility of their trying to accomplish this themselves.



Packaging:

The traditional DP approach to such a problem has been:

- to obtain the user's requirements;
- to formalize the requirements;
- to design a system which would deal with all aspects of the requirements, and define programs covering all requirements;
- to program and implement all these programs, together or, at least, to try.

This is called "packaging." Let's examine why it is likely to fail.

- Users do not know "specifically" what they want;
- users do not think enough about exceptions or specific circumstances;
- skilled users sometimes do not exist.

One way to reduce these problems is to educate the users and to improve the skills of the DP staff in questioning users. But whatever is done to improve the situation, the facts obtained from the users are only a partial statement of their requirements.

The second way, extensively used in the "packaging" method, is to have endless interviews and studies. This investigative period is sometimes so long that the company changes its product lines in the meantime, creating a high degree of confusion in the requirements. The dreams tend to be a mirage.

Formalizing the Requirements

- From a written document, users can only visualize the general lines of a system, not the details or the consequences.
- Specifications tend to be too detailed for parts of the system, while lacking specificity for other parts.
- System design is likely to generate a huge amount of paperwork consisting of the translation of users' requirements into program specifications, database structure and paths of information in and out of the computer.

- Program design will include all possibilities requested by users, from the most useful to the least, from the most complicated to the least.

- Detailed analysis is likely to discover that new functions are necessary. This means new programs, an increase in program complexity, or design changes. In any case, it will be the first delay in the project.

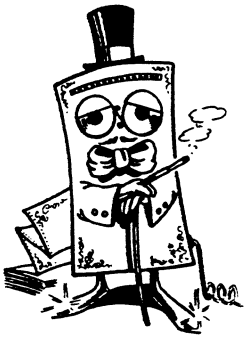
Programming

- From the system analyst's specifications, the programmer will create a program, probably to meet the technical objectives rather than the users' objectives.

- For interactive programs, he will have to think like a user, which is impossible because DP people are computer professionals.

- The programmer will find some errors in the analysis. This will delay the whole project because an essential program will not work correctly on time.

- Like the designer or the system analyst, the programmer would like to have a perfect system, so he will add beautiful but useless features. This is called the "galloping elegance" syndrome.



THE 'GALLOPING ELEGANCE' SYNDROME...

As we said earlier, the users' requirements are likely to change. Why is this so?
The users' requirements are determined by the type of business the user is involved with. The business can be production, sales, marketing, service, etc. Business can vary depending on season, crop, day or other factors. The consequences for data processing from a change in business are new and different information needs. One way of taking these changes into account is to make an exhaustive list of the possibilities and program all of them.

We also said that it is difficult for a user to visualize the end-result of a program, and even more difficult when it is an interactive one. Why is this so?

A program is not static. The actions it performs vary dynamically, depending on the information that is entered. A program is a moving body and is unlikely to be adequately described without using jargon. The same applies to mathematics, or astronomy, or films. How can we visualize a film from a script? This is why the sooner you show the program to the user, the better he will understand it.

The following two concepts are the foundation of Step-by-Step:

- to discover the users' actual requirements and to program all of them;

- to give the programs to the users as soon as possible.

This is nothing really new, but keep these two concepts carefully in mind. You will be surprised by the methods of achieving them.

Getting the requirements:

What are we really interested in? To know what a user wants. If you ask a user "What drug do you want?", he will answer, "Vitamin C." In fact, the question is too specific. What we should ask the user is: What are your problems?" He will perhaps tell you: "I can't recover payments quickly enough. I have to pay too much in financing costs." If he knows why, you can start the second part of the study; otherwise, you will have to go into a problem-solving routine. Problem-solving is also a service; we advise you to use some of the Kepner & Tregoe methodology to help you. In any case, you will have to find out why the payments are not cashed quickly enough.

We seem to be far from data processing. Not so. We are only processing non-structured information (the user's explanation) without a computer. But

you know that the computer is not the most vital element for data processing.

Anyway, going back to the example (a real situation, by the way) we still find that "the invoices are not processed quickly enough." It appears that our user needs an invoicing system. Mind you, if you had asked, "What do you want?", he would probably have answered, "I want an order-entry system," because this was his overall, long- term objective, and because it will solve his cash problem.

The confusion arises partly from mixing long-term objectives with short term problems. If you could provide, overnight, an order-entry system, including the invoicing package, there would be little or no problem.

But, of course, you cannot implement it overnight, not even in a fortnight. You will conduct a long study and come back with phase one of the project ready, i.e., the order input. You will not solve the problem which has created the need for your expensive intervention (the cash is still not arriving faster).

Both you and the user will be very frustrated.

Breaking the Problem Into Steps

Obtaining the user requirements is a two-fold process:

* to identify the problems which have created the need for DP assistance;
* to set long-term objectives as well as short-term ones.

Identifying the long-term and short-term objectives will permit you, with the users, to draw a line of actions within the overall strategy. You will move from point A to point Z through points B,C, D,..., with each point being an objective. But how to order these points?

To provide a solution to the top problem means that you will give the maximum result in a minimum of time, and you will repeat this with each successive point. Order the objectives from the maximum payoff to the minimum. These will be your steps. Now you can make your design and organize the system in a structured way. Do not go into details. Remain flexible. The document you are preparing is the final report. It consists of:

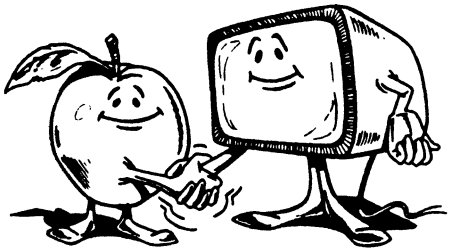
* long-term objectives,
* objectives for the first and all following steps,
* sub-systems or programs related to each step, and
* priorities.

When this document is approved, you must write the programs - those of the first step, of course!

Programming a Step:

A few remarks beforehand:

* The final aim is the program, not the analysis. So, until the program or its results are in the hands of the user, nothing is completed.
* There is a general law governing the whole world or any part of it: the offer-demand law. It applies to DP as it applies to apples. We will see how.



...APPLIES TO DP AS IT DOES TO APPLES...

A step is usually a move of two to three programs, but it could be more. To write the programs, you will have programmers, but never enough. Why is this so? Because the demand (user's request) automatically adjusts itself to the offer (programmer resources). We won't prove this here, but you can believe it.

The strange thing is that the more programmers you get, the more complicated the resulting system will be. This is the very well-known law stating that "adding people to a project will delay it in direct proportion to the number of added men." The explanation is easy to state: by increasing the offer, you increase the demand.

The only logical way to escape this dilemma is to limit the offer. How can we do that?

One way is to limit resolutely the number of programmers working on a project. This approach is already frequently used, but for different reasons: to shorten communications and avoid misunderstandings, and to add flexibility and improve accountability. This is a method we still recommend using.

A second way is to limit explicitly the amount of time allocated to a program or system. Let's imagine for a moment that we've said we have two weeks to program our step with the existing manpower. No more than two weeks. How can we best solve the problem in the amount of time given? The natural way will be to put on paper what the *musts* and the *wants* are. If both can be produced in two weeks, we will program both, but that is unlikely. Therefore, how do we determine which steps will be *musts* and which will be *wants*?

It really depends on their nature. If they are, or some of them are, necessary before we can program the next step, they have to be part of the next step *musts*. If they are not necessary before step X, insert them into the *musts* of step X. The most important objective is to find the absolute *musts* which can be produced with the current staff in a limited period of two weeks.

Well! That's it! There is nothing more to Step-by-Step. However, we will come back to two important questions.

How to Establish a Time Limit:

In our examples, we took two weeks, and this was not arbitrary. We think that two weeks is a manageable period. Less is difficult and dangerous, as stepping too quickly will permit neither the proper education of users nor the corrections to the programs. Programs must still be of outstanding quality, and, of course, you need some tools and structured design to step correctly.

More than two weeks is too much. Generally, users do not wait for you. They have other activities, very often repeated on a monthly basis. Therefore, to give them the program(s), you will need another week, which means that the program(s) will go into operation only after three weeks.

Experience also shows that two weeks is a "manageable" piece of time, possible and realistic.

How to Establish the Discipline:

This is the most difficult part of Step-by-Step, since it deals with human willingness — both the users' and the DP team's willingness. Here are a few tips which might help, but success depends more on the company and personalities than on anything else.

It is good to introduce Step-by-Step to the users. The more of a selling job you do before starting a project, the easier it will be. If the users do not want it, don't undertake the project at all, as users without commitment will never help you.

Use the *musts* and *wants* method with the users. Look at all the problems - theirs and yours. Some *musts* are yours. Explain why. Again, try to solve their problems. It is amazing how often you will find that a program can be easily replaced by a simple manual procedure.

Challenge the users. Everything is a must for them at the beginning. Ask always and again - why? Quickly, some of these *musts* will turn out to be *wants*.

Never go back on the two weeks allowed. It must be done in two weeks. Try to imagine that in two weeks' time, it will be the End of the World. Users will laugh, but they will, as well, appreciate your concern.

Step-by-Step: Pros and Cons

One of the major bonuses for the users is that they take control of the product as soon as possible. This means that users can see the program reacting to their input and can visualize how the program will work day after day. In general, users ask for small modifications. It is a *must*, at this time, to avoid asking for and, therefore, to avoid making modifications which are lengthy, or which should be included in a further step.

A request may look like a *must*; the analyst and the users should determine whether it is. Both should remember that a *must* means that the program will not work when put into production. In other words, a modification needed a few days after the scheduled implementation is not a *must*.

It is the analyst's responsibility to plan this *must* for a further step. The users should remember that when they start using the program, they may discover that the request was irrelevant or inadequate. Irrelevant, because they might find that what they have proposed will make the program very cumbersome to use.

One of the major pitfalls with Step-by-Step is the possibility of stepping forward without clear final objectives. There is nothing more expensive than programming a step, then being obliged to re-program it during a further step. This will happen if the original objectives are absent or forgotten, and the result will look like a drunken man's walk.

IT LOOKS LIKE A DRUNKEN MAN'S WALK...

A second problem can come from "stepping" too quickly. If two weeks is the agreed pace, stand firm on that. Rushing will decrease the quality of the programs in terms of reliability and documentation; it will also kill DP team communication, as few people will be aware of a step's content.

A third problem is a switch in priorities. It is in the nature of Step-by-Step that steps may be swapped due to reshuffling of priorities. This is not dangerous, so long as the DP team is aware of it, and organized for it.

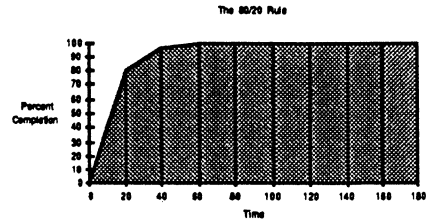
Step-by-Step Programming

Step-by-Step can also be applied to implementing a particular program. The methodology, known as "step-wise refinement" is not very far from Step-by-Step.

Like step-wise refinement, Step-by-Step needs a structured programming approach. Programmers can use almost any language to program in a structured way, with or without GO TO statements. It will be very dangerous to program Step-by-Step in an unstructured way, as the code, which will constantly change, will look like a war theater.

...OR A WAR THEATER!

With Step-by-Step, unlike step-wise refinement, the analyst has to bargain with the programmer to obtain a workable piece of code in two or three days, then a second piece of code in a similiar amount of time, and so on till the end of the step. This is a kind of sub-stepping, where the analyst has to play the user. It requires teamwork and dicipline to keep the programmer form testing the exception module when the general one is not yet working.



The 80/20 rule

Managers in many other fields have long applied the "80/20 Rule" as an indicator of the relationship between results and resources. For example, in a bookstore it is expected that 80% of the income will be produced by 20% of the titles, i.e. the "best sellers."

Although the 80/20 ratio is not absolute, it suggests the general principle that only 20% of the available resources (in term of money, people, supplies or time) are required to produce a large proportion of the desired results. Conversely, the ratio indicates that the cost of a marginal action can be disproportionate relative to the cost of a standardized action.

Although Step by Step is informed by the statistics which back up the 80/20 rule, we believe that a simplistic application of the rule to the planning of a software project would be very detrimental, if not fatal. Here's why.

Say you have a 9 month deadline (180 working days) to complete a software project. You have worked with your users to formulate a set of *musts*. These *musts* represent 80% of the project. Since the completion of these *musts* should produce a basic working system which can be given back to the users, in Step by Step terminology they comprise the first step.

Strict application of the 80/20 ratio to your 180-day deadline yields the following breakdown: The first step will be finished in 36 days (20% of 180). The second step, is composed of 80% of the objectives not included in Step 1, will be achieved in 29 days (20% of the remaining 144.)

A chart shows the following progression:

While progress is quite rapid at the outset, users still have to wait 36 days to get anything they can use. During the latter part of the project, very little headway is made, yet the users get a new step every day.

Though this example may be a little too simplistic, it shows that trying to complete 80% of the users' requirements in 20% of the allotted time leads to a disordered implementation of the project, from the user's point of view. Furthermore, you never finish!

By defining steps of equal length, you avoid disturbing users all the time, or at any time. To do so, you must pare down the original *musts* so that they fit in 10 days rather than 36. The point is to get a working system back to the users as soon as possible. This is much more productive than waiting until the 80% point is reached.

Back-to-Back Programming

In developing almost any large application, it is necessary to define and program a set of common tools and utilities that will be used by various components of the application and by the programming team. Some of these tools are already there, whether from third party software vendors, from Hewlett-Packard, or from within your own organization.

Other tools must be created specifically for the project. For example, you might need a unique and safe routine to access system tables for the purpose of controlling user access to a terminal. Or you might want a routine to log transactions so they can be picked up by a later process which posts them to IMAGE databases. There are nearly always a number of system-level programs — some tricky, some trivial — required for the development of the application.

Most of the DP shops I have encountered will start a project by programming these development tools to build a framework for the final application. In principle, this is reasonable since good tools are even more essential to good programs than good

programmers are. However, there is a pitfall in this approach. By starting with the tools, you delay completion of Step One and the demonstration of results to the users.

This is a serious dilemma. On one hand, it is necessary to achieve long-range productivity by creating and using the appropriate tools. On the other, a concentration on tools delays the short term implementation of the application. This is difficult to explain to managers and users.

One method I have always used is back-to-back programming. It means that instead of doing one project at a time you do two! Since the tool and the actual application are interdependent, each can be used to test the other. For example, when I began to write Mirage (an IMAGE database for micros), I simultaneously wrote the screen handler required for the user interface of the product.

Instead of delaying the actual implementation, the synergy of back-to-back programming strengthens both projects. Michelangelo ,the Italian Renaissance painter, used this method long before I did, with greater success indeed!

Presenting technical issues to users and managers -- the next step.

Lets us take a short detour into the land of politics (short, I promise.) We have all been subject various political situations in our companies. Some people, like me, refuse to play politics (I just quit.) Others love it. But however you feel about it, politics are everywhere.

Because of this, you may be tempted or even forced to select the *musts* of a project according to the power of the user who requests them -- much like politicians do on Capitol Hill and elsewhere in the world. Yet there is an important difference between you and, say, a senator. He is elected; you are selected. By being selected, you have certain powers you may not realize.

The style in which you choose to carry out your duties is uniquely yours. Yet every person responsible for the completion of a company project shares at least one objective. You must get the project(s) finished in time for the company to benefit from it. Depending on the results, your "power" will increase or decrease.

Therefore, the first action to take is to decide who will evaluate the quality of those results. A one-man decision made by your boss is too risky. To avoid this risk, you can set up a EDP Steering Committee to meet every 4 to 6 weeks. The committee (not you) will be responsible for the EDP strategy. However you, as secretary of the committee, will control the agenda and monitor certain power struggles.

The chairman of the committee must be the highest officer in your group, division, or firm with any responsibility for EDP. It is better to involve the president than the comptroller but sometimes you just don't have the choice. In my experience, two times out of three I was able to get the president.

The charter of this Steering Committee should include:

- Evaluation of current progress
- Review of outstanding problems
- Resolution of priority conflicts
- Allocation of resources
- Review of steps

The first four items are pretty much standard. The fifth one is what will make Step by Step work in a political environment. As MIS director, EDP manager, or project leader, you come to each committee meeting to present conflicting priorities and to suggest a plan of action -- that is, steps. These items will be discussed and resolved, meaning that at every meeting you get a consensus on the upcoming steps. Political struggles take place in the committee, rather than at the coffee machine. And because you are the secretary rather than the chairman, your position will be more secure.

Users, through their committee representatives, are making the decisions and your job is to implement them in term of programming projects.

How do you break a project into steps?

Steps are a series of tactical moves within a strategic plan. Actions which do not contribute to the established *musts* are not included in any step Yet you must maintain an open-minded attitude toward problem solving.

For example, I remember a MIS director asking me how to find *musts* and *wants* in a payroll project. The solution was to computerize first the 80% "standard" employees in the first phase and the 20% "others" after that, rather than trying to program and implement everything at once.

Then it comes down to the tactical move, or one 2-week steps. The questions to ask yourself to determine whether a particular objective should be included are:

- Does it contribute to one of the project objectives?
- Does it contribute to one of the *musts* in this step?
- Does it provide the most immediate payoff?
- Does it produce something to show the users?
- Do we need the action to perform a later step?
- Does it fit into the current 2-week plan?

If you cannot answer yes to the first 2 questions, forget it. Then rate the objective in terms of the next 3 questions on a scale of 1 to 5. If it produces a lower total than other possible actions, forget it. If it doesn't fit into your 2 week plan, try to find how it will by going back to the users. There is no easy way!

I created Step by Step as MIS director for Cargill in Europe. They are still using it there, and now in the States. After leaving Cargill, I used Step by Step myself to write Mirage with very limited resources and time.

Software houses are struggling to deliver products quickly because of the extreme brief life of most hardware, and also because of the competition. A 6 to 9 month lag between planning and actual shipment can absorb most of the profits. This is why companies like Robelle, Tymlabs, and others are using Step by Step.

****

We like to make analogies between data processing and other technologies or sciences. Analogies are very important, as you often create a new technique in your field which is very old in another field. Knowing this ancient method may save you a lot of time.

When the first rockets were launched toward the planets, they were fully programmed. Everything was automatic and followed an unchangeable program. It all seemed logical, as all the parameters needed to find the target were known: the distance between planets, their relative speed and weight, the rocket's acceleration, weight, etc. But the project failed. The objectives were not reached.

The next rocket generation included a new feature. Technicians were able to change the rocket's direction slightly during the flight to counteract the non-programmed consequences fo the environment's reactions. This time the objectives were reached. Implementing an on-line system in a commercial environment is like sending a rocket toward a moving object. You need to adjust and control the direction. This is Step-by-Step.

RESPONSE CENTER FROM THE OTHER END OF THE PHONE

CHERIE SCHOONOVER
HEWLETT PACKARD
3300 SCOTT BLVD
SANTA CLARA, CA 95054

The Hewlett-Packard Response Centers provide remote support services to HP customers, HP field personnel, and other HP entities. When calling the Response Center, you speak directly with engineer specialists who provide a centralized pool of technical knowledge. This paper is not intended to provide an overview of the services offered, but rather to give an inside look at the Response Center from the perspective of a Response Center Engineer (RCE). This perspective comes from 16 months as an on-line RCE handling customer calls and more recently as a Support Engineer with the Software Support Engineering group at the Western Response Center.

This paper is divided into two parts. The first will look at the process flow of a call, the resources available to the RCE, and the various activities of the RCE. The second part examines some of the current and future programs of the Software Support Engineering organization and how they are important in improving the support services of the Response Center.

A Brief History

Prior to the Response Centers (RC), telephone support was available through the local (area/country) level. This support was often referred to as "PICS" -- Phone-In Consulting Service. To prepare for future support needs, localized PICS activities were consolidated into two Response Centers in North America and one Response Center in Europe. Implementation of that plan began in October of 1983 and the Centers opened in February, 1984. Since then, the Response Center's have expanded to include Regional and Country Response Centers in Japan, Australia, Europe, Latin America and the Far East.

Process Flow

A call placed to the Response Center via a toll free 800 number is initially answered by a Call Coordinator who will verify that you have purchased a support contract and that the phone number where you can be reached is correct. They will also "log" the HP application, operating system, or subsystem involved in your question, and a one-line description of the problem. After entering/verifying this information, a unique PICS-ID is assigned and the call is sent (via software) to the "Team Leader".

The team leader monitors the incoming calls and assigns them to teams based upon the specialty needed. (Each online engineer rates their level of knowledge across technical topics.) By accurately describing your problem and the subsystem involved you can help ensure that your call will be most efficiently routed. Once the calls are assigned they are sent to the designated team. The engineers on the team choose to take new calls from the pool based on their expertise, preference, and number of open calls. Once an engineer chooses to work on a call they assume ownership of the call. The number of calls handled by an engineer on a daily basis varies greatly based on call loading, engineer expertise, and call difficulty, but is on the average seven calls per day per engineer across all of the product lines supported in the Response Center.

Although a single engineer will assume ownership of a call, calls may in fact be worked on by many engineers. The RC works under a team concept. This means that more than a single engineer may work towards a solution -- a team of specialized engineers with different backgrounds may work together to isolate and analyze your problem. Software engineers work closely with hardware and application engineers and vice versa.

While the RCE is working on your call they are also responsible for a variety of other calls. For this reason, it is important that you communicate to the RCE the severity of your problem and the extent of assistance that you need. The RCE works with a variety of customers as well and strives to deliver the particular support each customer needs.

When you place a call to the Response Center what type of engineer can you expect to return your call? There are basically three types of engineers at the Response Center: hardware, software (systems) and application. Commercial Application engineers are divided by Office Products, MM/3000, FA/3000, etc. A general "systems" engineer supports the Fundamental Operating System (FOS) which includes the operating system (MPE), VPLUS/3000, EDITOR, IMAGE/QUERY, and KSAM. Additionally, engineers can choose to "specialize" -- i.e., train and develop a set of skills in a specific area of expertise. Examples of specialities include: Datacommunications, Languages, and Operating System. Finally, among specialists, even more defined levels of expertise are developed.

After the initial call to formulate an accurate problem description, the RCE will usually set up a time to call you back (i.e., if the problem wasn't resolved during the initial call). If it is difficult to reach you it is important that you convey this to the RCE and agree upon a mutually appropriate time for the engineer to call you back. Likewise, once the problem resolution process has begun, if you want to check on the progress of your call or if you have additional information to pass on to the RCE, you can call back on the 800 number at any time.

Callbacks are not made directly to the engineer since they may be on the line with another customer, or away from their phone working on a problem, etc. Instead, by leaving a message for the engineer you are

ensured that they will not miss your call nor will you be interrupted when you are on the phone with them. Within moments after you call in, a message is printed on a dedicated printer located near the engineers workstation. Thus, the RCE is quickly aware of your callback. The RCE will return your call as soon as possible, so, to help avoid "telephone tag" you should be available after placing the call for the return call.

From the time you place a call through the time that your problem is resolved, an online system is used to track the progress of your call. Each time an engineer speaks with you or works on your call, an entry is added to the call "history" (and is available to all engineers and the field). This is extremely important when a call is "turned over" to another engineer (this will be discussed further in the "RCE Activities" section). The new engineer working on your problem can retrieve the existing information to familiarize themselves with your problem, then update it accordingly until the final resolution of your call. The database is also used in identifying common problems and as a resource for known problem solutions.

Once your problem has been resolved it will be closed in the tracking system. This should happen as a mutual agreement between you and the RCE working with you. If the problem reoccurs you may open a new call referring back to the original one. Since all closed calls are maintained in the tracking database, we treat your new call as an extension of the old one even though a different engineer may be working on it.

Resources Available to the RCE

The engineers most important role is that of resource manager. The first part of this section will examine the various resources available to the RCE. The second part will take a look at a sample call to see how the resources are used in the problem resolution process.

Along with the standard HP Manuals and an extensive library, the engineer has an On-line Data and Information Network system which stores the call documentation on all PICS calls, Known Problem Reports (KPR's) and Service Requests (SR's), etc. Using keywords (such as "SF311" or SPOOLEE I/O ERROR") the RCE can search through the text contained in all the documents for a reference to the selected keywords.

In conjunction with the online databases, RCE's also have a number of people they can use as resources: team members, RCE specialists, the System Support Team (SST), online support groups at the divisions, and the local field support teams. The next section of this paper will examine these resources in more detail.

The SST is a group of offline engineers dedicated to providing advanced levels of technical support. As mentioned, the RCE can also contact

factory support personnel. With a single phone call the RCE can (and does!) contact the division responsible for a product when further information or expertise is needed.

Another notable resource is the diagnostic center which contains software and hardware available to the RCE for use in duplicating your system environment. Sometimes referred to as "crash and burn" machines, the RCE is in most cases able to match your system configuration while attempting to duplicate and isolate your problem.

The RCE may also choose to dial in to your system. The engineering workstations are set up with modem access on every phone; without leaving his desk, the RCE can access your system as well as expert and diagnostic systems at the Response Center.

Although not yet mentioned, the RCE's most valuable resource is his own knowledge and experience. RCE experience ranges from newly trained college graduates to personnel with many years of field and industry expertise. Likewise, the continuance of the Field-On-Loan (FOL) program* guarantees that current field experience is always available to you. Supplemented with field experience, the RC is dedicated to offering the RCE a comprehensive training program.

The Training Program is tailored to supplement the RCE's background and specialization interests. The RCE has the equivalent of all of the basic customer courses as well as internal Systems Engineer (SE) training classes. As well as the external training available, the RC provides an extensive variety of internal training courses. Courses have been expanded upon and created to meet the training needs specific to the RCE. In addition to technical training, RCE's take a number of analytical courses and people skills training.

Problem Resolution: A sample call

What does an engineer do on a call from the time he first reaches the customer to the time the call is resolved? As an example consider a system failure call. The first step involved in any problem is to formulate an accurate description of the problem. This is the most important step (and can be the most difficult as well!). The RCE will gather data from you to find out as much as possible about the failure in question. The RCE can then confer with other RCE's and check internal

*When the Response Center first began operation it was almost entirely staffed by HP field engineers on rotating two week tours of duty. As the permanent RC engineers completed their training they began to assume call responsibilities and the number of FOL's decreased. The FOL program proved so beneficial to the process of exchanging information between the field and the RC that today, roughly ten percent of the online RC staff is composed of FOL's.

knowledge databases to determine if the failure you experienced is a known problem. If it is, then a solution, such as a workaround, software patch, or the need to update, may exist.

If further diagnosis is necessary, the RCE may dial in to your system to run diagnostics or check for logged errors. It is also common to work with a hardware engineer to determine if the failure is actually a manifestation of an underlying hardware problem.

The RCE may also need to examine your memory dump for further information. Note however that dumps are not always conclusive, and they can take anywhere from five minutes to weeks to examine. Often, when analyzing the dump, the RCE will contact the lab engineers at the factory for assistance.

In some cases, the problem may be identified as a new problem and a Service Request may be submitted to the labs or division responsible for the code causing the failure. In any case, if the problem is still continuing without resolution, the RCE may need to notify other resources in the field.

Although the cause of all failures can not always be found, the RCE will do everything possible to prevent a future occurrence and ensure that proper steps are taken for gathering the needed information in case the problem does reoccur.

RCE Activities

The next section of this paper will examine the various activities of the RCE. It will first cover the internal activities such as rotating on the System Interrupt Team, "turnover calls" (i.e., when responsibility for a call is transferred from one engineer to another), Service Requests, working on projects, and teaching classes. Next it will cover external activities such as field and division exchanges. Finally, it will touch upon the topic of Application Notes.

Engineers participate in a number of activities other than solely phone support. From the onset of the Response Center, management realized that engineers would need time to maintain and expand their expertise in their chosen specialty areas. Thus, a model was developed where forty percent of an engineers time is devoted to activities other than online support.

Offline activities are scheduled based upon the needs of the customer coupled with each individuals career and growth interests. By offering a wide range of activities to increase the technical expertise of the engineers, the RC attracts highly motivated, technical personnel. RCE activities include phone support, team leading, System Interrupt Team (SIT) duty, open weeks, projects, classes, preparation for instructing,

instructing, field and division exchanges, turnover duty, Service Request (SR) duty, and writing Application Notes.

Along with "regular" phone duty engineers can also be scheduled to "sit on the SIT." (Although "sit" is at the other end of the spectrum from whey they actually do!). The SIT is a special team which handles only (and all) system "interrupt"-type calls: System halts, System hangs, and System failures. Interrupt calls were not always handled by a dedicated team but rather were assigned to teams within the regular call flow process. By implementing the SIT the RCE on "regular" phone duty was freed from the interruptions of "down" system calls (while you gain the expertise and consistency offered through the dedicated team.)

If an engineer has any open calls at the end of a week, and he or she is not scheduled to be online the following week, then the open calls are "turned over" to another engineer. Turnover calls are distributed among all online engineers after being analyzed for the reason that they are still open. If you are told by an engineer that they will be turning your call over, you should be sure to communicate to them any time sensitivity of your problem. They can then document that information with your call history and the turnover RCE will be better equipped to respond to your needs appropriately.

While working with you on a call, it sometimes becomes necessary for the RCE to submit a Service Request (SR) on your behalf. (An SR is an enhancement request or problem report.) In this situation the RCE assigned to the call usually submits the SR. You can also send SR's directly to the Response Center. It is the responsibility of the engineer assigned to "SR duty" to attempt to duplicate and then submit these SR's.

Other activities include working on projects such as developing software tools for use by the Response Center or participating in the technical review of a new or new release of a product. RCE's also spend a considerable amount of time taking, preparing to instruct, and instructing classes. In conjunction with the instruction of in-house classes, occasionally RCE's also teach customer or factory classes.

Further, RCE's can participate in field or division exchanges. For example, RCE's accompany field SE's on account visits or aid the field SE's with on-site assistance. This gives the RCE an opportunity to interface with customers directly and gain field experience. RCE's also can spend time working at a division in order to understand the divisions concerns, to form important divisional contacts, and to participate in lab support.

A rather recent addition of an activity is the writing of Application Notes. The Response Center distributes Application Notes and commonly received Questions and Answers on a regular basis. (This will be discussed further in the last section of this paper.) Depending on

targeted needs and engineers specific specialties, time is allotted for working on these Notes.

Software Support Engineering Projects

In the past two years, HP Response Center's have provided remote support services to HP product users. Additionally, we have asked, how can we improve (and expand) our services, productivity and performance? The Response Center is continually working towards defining and answering that question. Both the Support Operations group and the Software Support Engineering Organization (SSE) are developing programs dedicated to improving support. While the Support Operations group is responsible for the day to day functioning of the RC, the SSE organization is primarily responsible for providing technical assistance, RCE productivity improvements, and new program development and implementation. The remainder of this paper will address some of the programs and areas that the SSE organization is involved in.

Examples of programs that the SSE organization has played a vital role in are SIT, the Patch project, SE Assist, the Application Note project, and the Product Life Cycle. Each of these topics will be expanded upon.

The SIT was first implemented by jointly staffing it with engineers from the SSE and Support Operations organizations while definition and management of it resided with the SSE group. One of the resources relied heavily upon for "interrupt" calls, is analyzing memory dumps. In the past, paper dumps were often requested from customers in order to isolate the cause of the interrupt but now dumps can be analyzed in a much more timely fashion by using the online Interactive Dump Analyzer tool (IDAT).

Today, because of the increased number of MPE internal and IDAT-trained engineers, it is no longer necessary to read the same number of paper dumps, thus decreasing the time necessary to effectively solve an interrupt call. Currently, the SSE group is involved in working with the HP Knowledge Labs, a division dedicated to developing tools for the RC and field, in defining and testing intelligent "expert" systems (utilizing artificial intelligence technology) for use as a dump analyzer.

Another area that the SSE group is involved in is patch distribution. Today, when it is determined that a software "fix" (i.e., software subsystem or module replacements) for a problem exists, a SSE member transmits a patch to the customer system. While maintaining the day to day operation of transmitting patches the SSE group is also investigating a patch process which will allow for central coordination and proactive distribution of patches as well as improving the methods for data collection and tracking of installed patches.

Along with increasing the range of services offered to customers, the RC is also becoming involved in offering assistance to the field. By making the resources of the RC available to field engineers, ultimately your Account Team will be better able to assist you as well. An example of field assistance is the SE Assist program.

SE Assist is the process where field engineers contact the RC instead of calling directly to the factory. The intent of the program is to leverage off the broad knowledge accumulated at the RC and to free up the divisions to work on the problems that require divisional resources. SE Assist was first implemented with the Commercial Systems Division (MPE support) and currently the feasibility of extending it to other divisions is being investigated.

Another function the SSE organization provides is the management of the Application Note and RC Q&A program. Application Notes are intended to assist in the management or usage of HP products. The need for Notes is determined through analysis of telephone inquiries where the volume of calls received indicates a need for addition to or consolidation of the available documentation. The RC Q&A sheets address commonly received questions at the RC. This project is just the first step in the arena of knowledge distribution. Gathering information and distributing it to the field and factories comprises the other areas of the arena.

The RC is also becoming increasingly involved in the Product Life Cycle. The RC can offer a number of benefits to HP and HP customers by becoming involved in a product prior to its release date. RCE experts, or "Product Champions", can provide secondary high level support to customers and work with the field in resolving product related problems. The Product Champions can track product information such as call volume and frequency of calls, support issues, product problems, etc. They can then provide the divisions with information regarding the product quality. In addition, the SSE group is involved in working with the divisions in defining a Spectrum Support Strategy.

The RC is a key player in future support issues because not only is it the hub for common problem diagnostics, but it is also involved in preventive support services both internally to HP (such as the Product Life Cycle) and externally (such as the Application Notes distributed directly to customers).

Most importantly, the RC is an organization dedicated to the future.

THE VICTORIAN H.P.3000 USERS' GROUP

"HELLO" - An unfriendly greeting or an offer of seduction?

Peter R. Hill,
Megatec Pty. Ltd.,
2 Brunswick Road,
Mitcham, Victoria, 3121.
Australia.

Introduction

Various members of the Megatec Team have been interested and involved in HP3000 security and simplicity of use since 1981.

This interest was initiated by a customer commissioning Megatec to design and write a Menu and Security System. Once this task was successfully completed, we came to realise that the majority of HP3000 users would have a requirement for a similar System. Consequently, Megatec negotiated the rights to purchase the System that had just been completed for the client and used this as a basis for its own package system, which has since been christened SMASH and sold to a large number of HP3000 users.

In the five plus years since the first Menu and Security System was developed, the Staff at Megatec have devoted untold hours to investigation, discussion, design and development in this area.

The following presentation is an attempt to bring together some of the basic principles involved in the area of HP3000 security and ease of use and to open up ideas which may result in further discussion and further improvements.

The aim of this paper is to present some ideas about the need for a more friendly HP3000 and the need for a more secure HP3000. Our title is obviously aimed to gain the maximum attention - but it is also an accurate summary of the two areas of interest that I hope to cover. A past generation of HP3000 users considered the HELLO command to be positively friendly. The new generation consider it to be bordering on hostile; while the past generation now worry about whether this same greeting is an invitation to seduce the computer and expose all.

Many papers and articles have been written that discuss at length the topic of security in relation to the prevention of unauthorised access to the System by "experts". (An "expert" is anyone over the age of five capable of scoring more than 30,000 points on the latest intergalactic video game). These people need only a colon and a few helpful error messages to realise that a far more challenging "game" of "let's crack MPE" is available. This paper does not attempt to directly cover the topic of preventing "experts" (with access to MPE) from trying to crack the System.

Our area of interest is that of providing Users with SIMPLE but SECURE access to the HP3000.

Perhaps we need to take a look back over the history of computer processing so that we can understand why there is now a need for "a better way".

At this stage of computer usage security was not a problem. The computer room door could be locked and access was limited to very few people. Telephone lines were still being used for talking to other people!

But then the erosion began - television sets with keyboards started to appear and the computer became "interactive". It was all rather clumsy at first - I recall the female operators of an early IBM CICS System who used to chant "Enter one, Knit two" as they waited for the computer to be "interactive".

In the early 1960's the words "User Friendly" had never been heard of. Those of us who were busy stuffing punch cards into huge hoppers did not want the computer to be friendly - that would have been considered a threat to our unique and mysterious position in society. (The casual mention at a party that you were "in computers" was worth a lot of points - now you are lucky to get a drink!)

Communication with computers was via Job Streams on cards. Then followed the storing of these streams on disk while data moved to magnetic tapes through Key to Disk or Key to Tape machines.

The Users of this era really had not changed much. Usage was still confined to computer types or key punch operators. But the signs were obvious and it wasn't long before terminals were introduced to people who previously had no direct interaction with computers.

Airline reservations were now being made directly through a terminal. A single interactive computer system replacing a single manual system.

The next step was the sharing of a machine. Users were still operating only one system through their terminal, but other Users in other places were operating a different system on the same computer. The computer now had to become more friendly and, in 1974, "HELLO" sounded pretty friendly: so we taught our Users to type "HELLO" commands and "RUN" commands.

Thereafter followed UDCs, then rudimentary Menu choices which offered various options within a single system.

The next step was multiple systems being offered to individual Users. But the Users tended to be the same people - order entry personnel, credit controllers and stores clerks. These people had come to grips with "HELLO", had welcomed UDCs and had absolutely no interest in indulging in the security video game or industrial espionage.

Then with the eighties all Hell broke loose! More and more "Utility Systems" became available: Spread Sheets, Word Processing, Memo Makers, Mail, Telex and so on. With these systems came a new breed of User - the impatient User. The HELLO command is no longer friendly, a UDC is downright obstructive even if it does result in a home-made menu.

The suppliers of micro computers have long since recognised this and have tackled the problem. They realise that their sales depend upon that first tentative use of the computer. Hewlett-Packard themselves have recognised the problem and tackled it on the HP150 micro. Why else would they go to the trouble of providing a "touch screen" facility, and the "PAM" menu if the world would have been just as happy with HELLO.

When we were involved in the HP PRODUCTIVITY 84 Exhibition, our stand in Sydney was alongside the HP150 stand. One of the evenings had been put aside for the Managing Director's Dinner. The idea being that the MDs and Senior Executives of HP clients and prospects were invited to come along to the display at the Hilton, have a pre-dinner drink while they wandered around and then have dinner in one of the function rooms. I stayed on our stand in the hope of an eleventh hour sale. All the visitors of the day had long since gone and the exhibition was very quiet. Then, I noticed a rather corpulent gentleman, definitely an MD, with a scotch in one hand casually wandering onto the HP150 stand (which was then unmanned). He approached an inviting 150 and after a furtive glance in each direction, to ensure that he was not about to make a public fool of himself, he prodded a finger at the screen. A smile of satisfaction spread over his face as the 150 moved into its demo routine - at his command. We had a convert!

These are the people we now have to cater for on the HP3000.

The situation that we now find in most HP3000 installations is either the continuing use of UDC commands as an entry into a menu that is part of the particular system chosen, or the use of a more universal menu system (HPMENU, SECURITY/3000) under which hang the various system choices available. Neither of these situations is satisfactory as our new breed of "impatient User" has still the hurdle of a HELLO command to overcome. In addition the system is still left exposed as MPE is open to anyone looking for a challenge.

So what are the real requirements :

1. The simplest possible entry into the system.

2. The quickest possible entry into the system.

3. A secure system.

4. The provision of information on the above to ensure continuity.

Perhaps we should look at each one of these in turn :

The SIMPLEST Possible Entry into the System

The majority of you must be thinking: "What could be more simple than a HELLO command?" To us self-trained typists who have been bashing keyboards for almost ten years now, a logon at high speed consists of about twenty-five characters sprinkled with about twelve back spaces! What could be simpler - or more impressive?!

What we need to emphasise is that we are no longer dealing with Users who have been exposed to computers for many years. We are now dealing with complete novices who are intimidated by the terminal and who only need one excuse not to use it. The simplest entry into the system is either to have a menu permanently displayed ready for use, or to have a menu displayed as soon as a terminal is switched on.

The QUICKEST Possible Entry into the System

The terminal and computer are only a means to an end for a User. The Users objective is to create a sales budget, write a memo, produce a report or make a simple enquiry.

There is no greater "turn-off" than having to master a language to converse with an operating system when all you really want to do is find out how much a customer owes your company.

Users should not have to talk to the operating system, they should be allowed to choose a function and then start work. Menus should be set up with the most commonly chosen routines featuring as the first choices. If there have to be delays between the choice of a function and its availability for use, then the system should indicate to the User that something is in fact happening.

A SECURE System

This is where we seem to have contention. Simplicity and speed seem to conflict with security. Having disposed of the HELLO command and any dialogue with MPE, we now have to provide some security system to ensure that the various Users only have access to information and systems for which they are authorised. However, by removing any contact with MPE we have in fact overcome the major problem. The ultimate video game is simply not available.

Now we only need to keep the unauthorised from restricted areas. Passwords still seem to be the most flexible way of doing this. Our initial Menu Screen needs to provide a way of establishing who is trying to use the computer, and subsequently, what access this User has. In our Security and Menu System we have chosen to use a combination of identification and password.

In keeping with our requirement for quick access this combination can be limited to three initials and a password from one to seven digits/characters long. But here we have a weakness, people tell other people their passwords - or worse stick them onto the terminal.

This initial access can lead to each subsequent menu being dynamically modified so that only the appropriate functions are displayed to the Users. (What can't be seen can't be of interest!)

People are the Weakest Link in any Security Scheme

One of the users of our Menu and Security System came up with quite a good idea to police the use of passwords on their own System.

As part of the on-going security the System can check the age of any passwords that are used. Assuming that the age of a password should not exceed thirty days (although this period naturally should be parameter controlled), then at a given number of days prior to the expiry of the password, the System can start to warn the user each time he uses his password that the password is about to expire. The purpose of this warning is to prompt the user to change his own password and thereby minimize the risk of the password becoming well-known.

Should the user ignore these prompts for action, then the System can invalidate the use of that password once the specified period has elapsed. This then means that the user concerned can no longer gain access to the password protected functions and will, in fact, have to go to some higher authority in order to be reinstated as a user and gain a new password. Naturally, it is hoped that this minor inconvenience will ensure that users simply change their own passwords at frequent intervals.

VESOFT went one step better in an attempt to make password security more secure. The Security/3000 product holds multiple passwords for each User built up on a profile approach. Each User is asked a series of personal questions. The answers to these questions become the Users passwords. The User is asked one of these questions at random when access is attempted.

Needless to say some allowance has to be made for the careless User who gains access to a secure area of the system then wanders off leaving the "gate open" to anyone. Timeouts can be the answer here. A control that times out after a set number of seconds, (we have found thirty to be about right), and returns to the Main Menu minimizes the danger. Sadly we do not know of a way to monitor usage and access once individual programs are being used. Perhaps Timeouts should be included in all sensitive programs.

So far we have concentrated on "screening" Users who are attempting to establish their credentials to use the System. Before the days of computers we used to lock our filing cabinets, then lock the office doors at the end of the day. Now we leave terminals switched on and ready to use - like open filing cabinets.

So perhaps we should have TIMELOCKS as part of our security. This approach is more interested in the Device than the User and simply prohibits any use outside of specified hours.

Leaving a terminal "open" is like leaving your car with the keys in the ignition and the engine running!

The Provision of INFORMATION about Users, Devices, Functions and Security

Given that a system can control Users, Devices and Functions on an HP3000 then we have an opportunity to collect information on the usage of the computer and its functions. Normal MPE usage does not provide good information on what exactly your computer is being used for. A good Security and Menu System can provide this.

Having gathered this information we can use it to ensure that we continue to provide the correct balance of SECURITY, SIMPLICITY and SPEED to Users of the HP3000.

Conclusion

Given that more and more people are gaining access to computers on a daily basis and that the use of micro computers seems to have actually increased the use of larger interactive machines, rather than decreased it, then it is our opinion that the need for simple secure and speedy access to HP3000 computers will continue for a number of years.

We also believe that we have only started to scratch the surface in this area and that there will be a number of dramatic improvements in the years to come. Certainly, the area of security for proprietary software is one which is, at this stage, not adequately catered for on the HP3000. However, in the meantime, we hope that we have provided sufficient stimulus for you to at least be aware of the necessity for good software to handle the areas of security, ease of use and speedy operation, and that as a result of your interest in this area, we can sell more copies of our product and consequently spend more time and money on further research and development in our particular area of interest.

COMPUTER SECURITY AND LEGAL ISSUES

ISAAC BLAKE
Hewlett-Packard
Western Response Center
3300 Scott Boulevard
Santa Clara, CA 95054

INTRODUCTION

Concern over computer security is increasing on a daily basis. Business are finding out that they are losing thousands of dollars to computer criminals. Unfortunately, many businesses invest in computer security after they have been victimized and not before. They are learning, like the rest of us, that the days of not securing your house, car, and business are gone. Because not taking steps in protecting these assets, will make them easier targets for the would be criminals. This paper will attempt to cover ways to secure your computer system, and reduce the chance of being a target.

WHAT IS COMPUTER CRIME?

It is important first to realize that these are crimes, not a joke. For too long, we have chuckled over teenagers causing havoc by cracking into computer systems, or enjoyed watching the "fun" of overcoming computer systems like in the movie "War Games". Until recently, these were viewed as harmless pranks, or headaches. But businesses are losing millions of dollars each year to hackers stealing time from computers, pirates cracking access codes to use long distance lines free, and by criminals using unauthorized credit card numbers to purchase items. The end result, is that us, the consumer must pay for these acts.

Most states now have added sections to their criminal laws dealing specifically with computer related crimes. You should find out what laws are established for your particular area. The laws are usually grouped into the following areas:

Trespassing

This can be either physical or remote access to your system. If a person accidently accesses your system, and doesn't immediately log off, they are guilty of trespassing. If the person knowingly accesses your system without your permission, or continues to access your system, it is a misdemeanor.

Theft

There are several areas of theft, there is the traditional taking of software and/or hardware, but also includes theft of information and theft of services. Service related thefts are items such as moonlighting using company equipment, using timesharing facilities that you are not entitled to, unauthorized access to data/tele communications.

Fraud and Embezzlement

These are the crimes that you usually see in the papers. About people who alter computer programs for their own benefit.

Tampering and Sabotage

Any changing of software or data is usually a felony. This can be such actions as changing customer data, leaving obscene messages, or logic bombs.

Civil actions

Most actions in this area are the invasion of privacy, but can also be breeches of contract such as non-disclosure and non-competition agreements.

Most state, county, and local police departments have started their own bureaus that work on computer related crimes. You should contact your local law enforcement agencies to determine if they have such a bureau.

WHO DOES IT?

As with all crimes, the victims wonder why they have to protect their property, and why the police don't spend more time catching the criminals, rather than teaching people how not to be victims. The answer is simple, since criminals don't go around with "I'm a criminal" tatooed on their forehead it's hard to identify them, where it's easy to identify a potential victim.

The computer criminal can be anyone, a Chairman of the Board, data entry clerk, cleaning crew, ex-employee, teenager, a competing business, the list goes on and on. These people can cover a wide range, from a hard core criminal who's trying to steal millions, to those who don't realize what they are doing is wrong.

THE BEST DEFENSE IS A GOOD OFFENSE

As with any basic crime prevention, the idea is to make your site, like a bank, car, home, or business, so difficult to get into, that the criminal will bypass you for an easier target.

The foundation for this is a security plan that's integrated into your business and operations. There are many ways to develop such a plan, go through a EDP audit, hiring a computer security consultant, or doing it yourself. The important thing is to develop and have a plan, rather than being without one.

YOUR SECURITY PLAN

First you must identify what areas need to be secured. This can be types of data or programs, backup tapes, physical access to the computer system, etc. Once identified, then classify them into levels like public, company use only, restricted, and confidential.

Then determine who needs to have access to the area. Develop a threat potential and action plan for each person. For example, what needs to be done if a data entry clerk leaves verses a system manager. Often when a person leaves a company, they could still gain access to the system, even a year later, because their logon and passwords are not changed or removed.

Look at putting functions into seperate compartments, along with dividing responsibities and authority in different areas. I.E. operations, system management, program management, data management, and communications.

Make sure to evaluate two major threat areas, internal verses external. The external threats are from persons outside your company, such as ex-employees, hackers, competing business, etc. However, the greatest threat will be from the internal user. This can be anyone who has legitimate access to your system, like the data processing staff, consultants, and other employees.

Your plan should also include non-computer related areas; personnel, policies & procedures, building security, employee and non-employee identification.

Develop hiring practices; do background checks, inspecting all references, and the signing of agreements such as a code of ethics, non-disclosure, and non-competition agreements.

COMMON WEAK POINTS IN A SYSTEMS SECURITY

MPE has very good security, if you take the time to set it up correctly. Most security breeches on the 3000 are due to human failure, rather than design failures. Working at HP's response center, I have the opportunity to dial into dozens of different systems each week. It never ceases to amaze me the number of systems that have no security or default passwords on their systems, such as the default passwords for the TELESUP account.

This leads me into the first weak point of security, being predictable. Make sure that you change and/or add passwords to common accounts like TELESUP, PACHnnnn, SUPPORT, SYS, HPPLnn, HPOFFICE, ITF3000, VESOFT, REGO, etc.

Next, identify users and accounts that have SM, PM, or OP capabilities. If a user has ANY of these capabilities they can totally compromise your system in a short period of time. Within this, identify any programs that require PM capabiltity to run and make sure that they are not released and have execute access only.

Setup a program that is run as part of your system logon UDC. This program can enforce ANY security specifications that you setup. Such program should notify you if there is a security breech and keep a log.

Speaking about logging, turn on all of the system logging. MPE system logging provides alot of valuable data about who did what, when. You can then store the logfiles to tape using either :STORE or LOGSNAP from the Telesup tape.

As far as creating users and accounts, make sure if at all possible, to give every user a unique logon and password. Trying to track down who did something when 20 people could log on as USER.APPL is difficult. If you have to use a common logon (I.E. MANAGER.SYS, OPERATOR.SYS, etc), make sure that the people use a different job or session name.

Guard your dial up lines. If your shop is not a 24 hour, 7 day a week shop, a person could dial into your system and it would be a couple of days before you found out what happened. If you don't need the dialup line, unplug it or down the device(s) until you do need it.

Protect your tapes. If a person has access to your backup tapes they can get any information about your system. They can use FCOPY to get your password, or take the tapes offsite and :RESTORE your files. Make sure to enact a policy on possession of tapes, and removing them from your site.

Another area to consider is data encryption. There are several ways to encrypt data, programs, and communications to ensure privacy. Surprisingly, there has been several cases reported of persons hooking into telephone lines to get information. If you encrypt your sensitive data, this would reduce the chance of a person using something like QUERY or DBDRIVER to look at your data.

Eariler I covered that for Trespassing, a person has to know that they are not suppose to be on the system. An easy way to handle this is with either the system message catalog or a :WELCOME message. The follwing message is an example of such a message:

```
**************************************************************
*                    Welcome to MY SYSTEM                    *
**************************************************************
*  This is a private system operated for XYZ Company         *
*  Business ONLY!  Authorization from XYZ management is      *
*  required use this system.  Use by unauthorized persons    *
*  is prohibited and may result in prosecution.              *
**************************************************************
```

WHEN YOUR PLAN IS FINISHED

It is important to identify and layout your security system first. Once done, you can then go out and look at the numerous security systems available. Publications like Interact, The Chronicle, SuperGroup, etc, have page after page of advertisements of vendors dealing with security. Make sure the system meets your needs, rather than you meeting the security system needs.

Once you start pulling the pieces together to form your security system, have yourself and a few others attempt to crack the system. This sounds like encouraging people to do what you don't want them to do, however if the system is sound, it will stand it. Remember, the criminal who's has his sights on you will try anything. An example is a very good security system I was evaluating once, it did a great job of enforcing the security I desired, but I was able to breech security by logging on as a job from my terminal (instead of :HELLO USER.ACCT, I did a :JOB USER.ACCT).

Don't think that once your security is in place, that is all you have to do. Security, like system management, is an ongoing process. You will have the constant maintenance of userid's and passwords. Most of all is the inspection of the logs and the security systems performance. Look at the military, police, and private security layouts, they all include human monitoring no matter how sophisticated the security design.

WHAT TO DO IF SECURITY IS BREECHED

You must accept that no matter how secure you make the system, at some point it will be violated. The reason to install security is to reduce the chance and minimize the loss of a breech. Hopefully your system was designed with alot of traps, mines, and logging to help you detect problems. Once you have detected a problem, try to identify the basics, like who did what when and where. Your plan should of included an action plan covering most possible occurances.

The basic duty is to collect evidence. This will be the listing of your logs (both system and security) and any additional supporting documentation. If the violation involved the changing of data, make copies of the altered and normal data. In a case of a theft, identify the items, information, or services stolen and their appropriate value. A very important point in the collection and preservation of evidence, is to secure the evidence in a safe place and maintain a chain of evidence.

If you do not specifically know who did the violation, then keep the knowledge of the violation to a "need to know" basis. This is to allow you time to gather more information and watch for clues. However, if you know, or at least feel you know, who did the violation, question them. You unlike the Police, can question them without having to give the person their rights, and then you can testify in court. A police officer must give the person their Miranda rights (right to remain silent, attorney, etc...) before any questioning.

Bottom line is to PROSECUTE, whether criminally and/or civilally. Only prosecution will deter these activities. Further by gaining convictions, other businesses will be able to identify these criminals in their background checks. This will keep the problem from passing from one business to another.

CONCLUSIONS

Although not deliberate, most computer sites victimize themselves by not taking the time to include security into their computer operations. Often security is viewed the same as documentation, something that's needed, but you never seem to get around to doing it. Further many businesses will spend thousands of dollars in physical security to protect their building and assets, but very little on computer related security. However if you take the time to evaluate and implement a security system, you

will significantly reduce the chance of your business becoming a victim and suffering the loss associated with it.

BIOGRAPHY

Isaac Blake is currently a Senior Support Engineer for Hewlett-Packard at the Western Response Center. He joined HP in 1985 and his experience covers 15 years in the computer industry. The last nine years was working on HP3000 systems mainly as a system manager. He is also a reserve police officer who has dealt with investigation and prosecution of computer related crimes.

SECURITY CONCERNS AND SOLUTIONS

JANINE FIRPO
OPERATIONS CONTROL SYSTEMS
560 SAN ANTONIO ROAD
PALO ALTO, CA 94306

To those of us in the data processing profession, the issue of security has become almost commonplace. There is no longer any question regarding the importance of implementing security procedures in your data center. The focus has shifted from "should we use security" to "how do we maximize security".

As I am sure you are aware, there are a plethora of publications that address security. Furthermore, there are security conferences and security organizations in abundance. Unfortunately, however, very few of these sources provide information specific to the HP 3000 user community. Most of the information is still too generalized to provide in-depth insight into a particular environment.

It is my goal to discuss security as it relates to the HP community. We will begin with a cursory look at the security provisions available through MPE and other HP utilities. For many users, these measures are not sufficient to ensure a secure environment, because knowledge of the loopholes and bypass techniques have become available to too many programmers and other technical specialists through time. As a result, something more is required. The question of what this something more consists of and how you can obtain and utilize it constitutes the remainder of this presentation.

SECURITY FEATURES OF MPE

As I mentioned, there are a variety of methods available through MPE to increase security. These methods run the gambit from password requirements through capability assignments to enforced menu restrictions.

To demonstrate the security mechanisms available, let's describe the first in a series of pathways through which the typical user must pass to gain entrance to your system.

LOGON SECRECY

Our user, let's call him George, must logon to the system. Although you can set password restrictions at several levels including account, group, and user, the logon sequence remains the weakest link in your security setup because its success depends on secrecy. Unfortunately, it's also one of the most important links because further security checks, such as file security, use information that is established at logon time to perform validation processing.

There are two main reasons to establish passwords. First, it's important to verify a user and second, we must authorize his access to the system. Since success depends upon secrecy and secrecy is often at a premium in most work places, when George signs onto the HP 3000, we cannot ensure that the account he is attempting to access is his own. This illustrates the first major gap in MPE security. Greater control should be established over verification and authorization.

UDC LOGONS

While George signs on as a session, one of the tasks he can perform is the streaming of jobs, which may or may not exist in his logon account. How can we ensure that George is not submitting a critical job? One effective method would be to restrict George's access to the system from the onset by implementing an OPTION LOGON NOBREAK UDC. In this case when George logs on, an application program will immediately execute which restricts his access to only a limited number of options. For example, the logon could drop George into a menu which allows him to access EDITOR, E-MAIL, and Accounts Receivable only. Remember, the NOBREAK option is important. Otherwise, George could hit the break key, type ABORT and access MPE.

Although the OPTION LOGON method can be extremely effective for some users, if George is a sophisticated user, this method could prove to be too restrictive. In the latter case, the security manager has another option. He could globally restrict the use of certain MPE commands, by changing the contents of their associated UDC. For example, the STREAM UDC could be modified to include only a comment line. Thus, when George attempts to STREAM a job, he will be reminded of his inability to perform this function. Aside from its obvious limitations, such a control is also easily circumvented. In general, it is much wiser to restrict sophisticated users through the use of existing MPE file and account security features.

CAPABILITY SETS

Each user defined to MPE has a set of capabilities associated with their account, group and user names. In order to perform a specific function, George must have adequate capabilities at all of these levels. If he fails at any level, George will not be able to perform the anticipated action.

Some capabilities within MPE provide far greater leverage than others. Among these are AM (Account Manager), which allows a user complete freedom within his logon account. SM (System Manager), which allows the user access to and control of the entire system. PM (Privileged Mode), which not only allows access to privileged data files, but could also provide a user with SM capabilities. This can be accomplished via a well-known loophole in DEBUG. A final high powered capability is OP (System Supervisor), which allows the user to change

any file since it allows the user to perform STORE and RESTORE commands on any file in the system. Thus, the data security officer should only give OP capabilities to users who are trusted as much as system managers. Or such users should be restricted from STORE and RESTORE.

Each of these capabilities, in varying degrees, allow users to access confidential information. Thus, users with these capabilities are able to create great damage to your software and data, either intentionally or unintentionally. Therefore, when the System Manager creates accounts, groups, and users, he should be very selective when deciding which users are assigned to these capabilities. It is also wise to verify that the original capability sets have not been modified. When such caution is taken, it can be assumed that a user without AM, SM, PM, or OP can produce very little harm. Right? Wrong, because what is it that protects the accounts with AM, SM, PM, and OP capabilities? Passwords.

But we already discovered the security flaw associated with password requirements. So, even with all of the positive elements inherent with MPE requirements, you are still leaving your system open to considerable risk. As you can see we have hit upon a Catch-22 when attempting to secure the system. Each solution points to another method and eventually returns us to passwords and - secrecy. Thus a second MPE security hole has been identified.

FILE ACCESS

Another level of MPE security is file security, which is probably the most sophisticated approach. For that very reason, it is also the least used and least understood security system provided by MPE. It is possible to build a "security matrix" for each file, which describes what classes of users can read, write, append, execute and/or lock a file. A similar "security matrix" is also available at both the group and account levels. Much like capability access, a user must pass all three file security levels before a file can be accessed. Through file security, it is possible to allow some users complete access to a file, while other users have restricted access. This is accomplished by setting greater restrictions at each level.

A major breach of this system is the RELEASE command. When a file is released, ANYBODY can do ANYTHING to the file while it is in the RELEASED state. This means read it, write to it, or even purge it. To further complicate matters, RELEASED files are seldom SECURED. Thus the file remains open to tampering. Commonly used reporting procedures do not help because they don't flag RELEASED files, so it is difficult to isolate security breaches.

All is not lost. There are three methods to minimize the amount of RELEASED files. First, make it as easy as possible for users to

create their files for unrestricted access by others. Build accounts with read, write, execute, append, and lock access to ANY user. There are still checks available at the group and file levels. If necessary, restrict specific files at one of these levels. The ALTSEC command (Alter Security) can be used to protect individual files. Second, place passwords on accounts, groups, and users and lockwords on your files. You must be aware, however, that these passwords and lockwords can be common knowledge amongst any users with access to the file, because they are usually embedded in your jobstreams. Thus, yet another MPE security gap has been isolated. Third, an undocumented feature of the FOPEN intrinsic allows a file to be opened to read access IGNORING ALL SECURITY. The program that opens this file requires PM capability. What this means is the file can be specially accessed by FOPEN, but non-PM users will be locked out.

SECURITY CONCERNS

It should be evident to you that regardless of the significant number of security features within MPE, they will probably fall quite short of meeting your security requirements. Not only are there gaping holes within the features I mentioned, there are many areas that MPE simply does not address. For example, it is fear of being identified as a security violator that results in violation attempts occurring across telephone lines, or during off hours. Thus, it might be useful to restrict users to a particular port or time of day and deny all other access.

MPE does not even begin to address this issue. Another vital requirement is documentation of blatant violations. MPE currently provides some violation information. Invalid attempts are logged to the console. Unfortunately they look just like all of the other console messages. Violation messages are also sent to MPE log files where they are quickly lost in an avalanche of data.

An additional security risk arises from a regularly overlooked source - the logged on terminal. Referring to George again, suppose he logged onto his highly restricted account. Since George knows all the passwords and has the required capabilities, he was allowed access to some critical accounts and confidential information. If George gets side-tracked, he might walk away from his terminal without signing his account off. At this juncture, ANYONE could access the confidential files and potentially reek havoc within the system. This is a very definite risk and one that MPE is incapable of monitoring. Although screens may be cleared, one keyboard stroke reactivates the screen display.

ADVANCED SECURITY FEATURES

Well, now that we have established many of the inadequacies in MPE security, what improvement steps can be taken?

The answer is two-fold. First you must analyze the goals of your security. Be specific. What exactly do you want to accomplish? Second, you should determine the depth of reach of that security. How many of your accounts do you want it to encompass?

For most installations, the primary goal is to identify and authenticate individual users. As I mentioned, this requires greater control and more specialization than MPE allows. How can increased security be accomplished at this level? There are several methods available to achieve this primary goal. Depending upon the depth of control you wish to achieve, these features can be used singly or in conjunction with other features.

ACCOUNTABILITY

A very effective tool is accountability. It has been found that an employee who is held personally accountable for his logon account, is less likely to divulge information required for access. The employee can be held responsible by insisting that he regulate and modify his password on a regular basis. Some firms even define security responsibilities as part of employee job descriptions and include an evaluation of compliance in the review process.

PERSONAL IDENTIFICATION

To further reduce the risk of illegal entry, a second goal is to create additional access conditions. It is often wise to have users supply additional passwords or answer specific questions. If questions are used they should require a complex answer which is unique to each user, yet easily remembered. An example of a poor question would be "What is the color of your eyes?". There are a limited number of common answers to this question.

A better example might be "What is your mother's maiden name?" or "What high school did you attend?".

For authentication purposes, you will probably want some users to encounter more access obstacles than other users. For example, your payroll accountant may require access to the system between 8:00 a.m. to 5:00 p.m. on weekdays, but various programmers may work odd hours problem solving. The latter group would mandate freer access. Therefore, when you envision a security program it should have the ability to treat users to varying degrees of security.

REMOTE ACCESS

As a third goal, you should consider the security of your dial-in ports. One very effective method of securing these ports is the use of dial-back modems. However, there are two serious concerns present regarding these devices. First, they are expensive. Second, it is

possible to bypass the security via the call-forwarding service AT&T so happily provides. Another solution involves coupling a port password requirement with port access restrictions, such as those just described in regard to user access.

The elimination of risks from unattended terminals is another goal worth mentioning. Without some mechanism for verifying that unused ports are signed-off automatically, all of the time and thought that was invested in security will be circumvented. An AUTOLOGOFF device is a very important consideration.

REPLACE SECRECY WITH RESPONSIBILITY

If these verification and authentication tactics were introduced in your environment, you would be replacing the dubious dependence upon secrecy with quantifiable personal responsibility. You would be replacing an open-ended system with one to which access is denied during the most vulnerable time periods. You should be replacing expensive dial-in port security with less expensive equally valuable controls. In short you would be filling many of the gaps left by MPE security with solid solutions.

SECURITY GOALS

Obviously, the elements presented exist as a security package. The package can either be one that is created internally or one purchased from a software vendor. When discussing security software, the only goals necessary are not those addressing the immediate security issue. Additional considerations include cost effectiveness, installation requirements, overhead, and ease of maintenance. Simplified installation can also be an obvious plus. More importantly, these goals are easy to achieve. However, it's very important to remember that pre-installation preparation can take considerable effort. The more time spent planning how to configure security solutions, the easier the transition will be.

USER ACCEPTANCE

One final goal that I would like to mention at this time is the importance of considering your user community. No security implementation will be successful if the user community is unresponsive to change. Firms have experienced everything from strong resistance to actual sabotage.

I am referring primarily to technical experts who attempt to circumvent the security measures. Nobody wants to feel that their performance will be hindered because they are being kept from resources they need. Therefore, a security installation is best handled as a psychological implementation as well as a technical one.

Among the groups to be included in the installation process are the systems software, applications, operations, and auditing departments. An excellent way to draw the user community to your side is to implement the project in phases concurrent with training. Ensure that, from the user point of view, each step will be easy to use and have minimal impact.

CENTRALIZED vs. DISTRIBUTED

As I mentioned earlier, there is another set of necessary definitions. In addition to establishing the goals security should address, the scope of the project must be drawn. Basically, there are two extremes possible with a lot of gray area in between. For simplicity's sake, this discussion will only address the extremes. The first of which is known as the centralized approach. In this context a limited group of individuals will provide control over all security changes. Other users would have to obtain approval from this group before access was allowed. For an installation, whose primary security goals apply to a limited subset of total resources, a centralized approach is most advantageous. For example, where the primary goal is security of the SYS account only, a centralized approach is the optimal solution. Only a limited group should have access to this account. The benefits of the centralized system include organizational and procedural simplicity coupled with lower overhead. A minimal negative reaction from the user group would ensue because many system users would not even be affected by the security changes. Remember, however, that this approach is inflexible. Since only a few people have access to critical pieces of information, if another individual requires such data, it can prove cumbersome or difficult to obtain.

The other extreme is referred to as distributed or decentralized. In this approach, both security and security control extend over the entire system. Every aspect is secured and entry throughout requires predefined access procedures. An example of this expansive approach is the personal responsibility for passwords, which I described earlier. For this reason, a decentralized system is more flexible than the centralized approach. However, a security manager is required to oversee the scattered control and to provide assistance when problems are encountered. The major benefit of a decentralized system is the accountability it can provide. If all users are defined to the system, it is much easier to isolate illegal entries or violations.

AUDIT CONCERNS

Thus, we arrive at a very important security consideration - auditability. Although many data professionals are aware of the problems inherent in insecure environments, this represents an easily ignored concern. For the most part, actual cases of sabotage or fraud are scattered and difficult to document.

No corporation wants to believe such a violation will happen to them. Most companies, which are becoming security conscience, are doing so because their auditors require it. Notice I did not say request, I said require. Therefore, one of the main ingredients to keep in mind when you consider a security system is the auditing capabilities it provides. You should request complete violation reports at several levels. Have the auditors make suggestions and recommendations. If one of your goals is to appease the auditors, their input in software selection and installation will be invaluable.

SUMMARY

To summarize then, MPE provides a variety of security features. These include passwords, lockwords, logon UDCs, capabilities, and file security. However, due to either their inherent nature, such as the secrecy requirement for passwords, or due to the loopholes that have been discovered, such as the ability to access SM through DEBUG, MPE security is simply not sufficient for many HP 3000 users. As a result, many companies have sought additional security provisions through software currently available from third-party vendors.

Some of the features desired from such software are password management, access control, autologoff capabilities, and audit reports.

In order to determine how to best utilize these additional features within each environment, you must define your goals, determine the benefits to be derived, and determine the approach you wish to take -be that centralized or decentralized.

It is imperative that the executive staff be strongly behind the security commitment. It is also important to draw the auditors into the decision making process, because to a large degree, they are the beneficiaries of security. Finally, it is vital that the user community be introduced to the product slowly and thoroughly.

4GL APPLICATION DEVELOPMENT GUIDELINES

MARC PRALY

COGNOS INCORPORATED
3755 RIVERSIDE DRIVE
OTTAWA ONTARIO CANADA
K1G 3N3

Along with the evolution of application development and fourth generation languages comes great benefits. For example, claims that 4GL systems are developed in hours not months or years. Claims that 4GL systems greatly increase programmer productivity and greatly reduce maintenance, ensure more control and user satisfaction. In general, 4GL means developers can get results quickly and really meet the users's needs.

But, you ask: "How do I get these benefits from a 4GL?"
or: "What's the best way to design a system using a 4GL?"
or: "Can I use the same methodology?"
or: "Where do I start?"

This is how I do it.

To get the most from a 4GL, you must know what it can do for you and what your users really need. Then take an active role in the entire development process. The project manager's role and the approach to the development process are what these Application Development Guidelines are all about.

These Guidelines simplify the traditional development process and make the process adaptable and flexible. It is not like traditional methodologies that follow a rigid series of steps and produce 'unnatural' results. These Guidelines describe the components or activities that lead to a successful 4GL application. At the same time, the Guidelines tell you which actions give you the 4GL benefits. The Guidelines apply to large and small applications. You just tailor the Guidelines to suit your application's environment. Because the Guidelines adapt to different situations, they make the benefits of a "4GL world" attainable.

Before you can start, there are 7 major principles that must be understood and fulfilled to enjoy the benefits of a 4GL application. These are the 7 MUSTS that earned a star on my 7-STAR checklist. The 7 points from the checklist are then turned into an approach for a specific project.

If you carry out each principle in the 7-STAR checklist, your 4GL is likely to bring you the benefits you expect.

THE 7-STAR CHECKLIST

The 7-STAR checklist details the main principles of 4GL application development. The 7 STARS include: modelling, functional prototyping, controlled iterations development, the phased approach, packaging the prototype, systems and programming standards and general plans.

STAR NUMBER 1 MODELLING involves the definition of the relationship between the data analysis and the business functions.

STAR NUMBER 2 FUNCTIONAL PROTOTYPING manifests or translates the users' requirements and acts as a means to convey the system back to the users.

STAR NUMBER 3 CONTROLLED ITERATIONS DEVELOPMENT deals with the evolution of the functional prototype and beta testing.

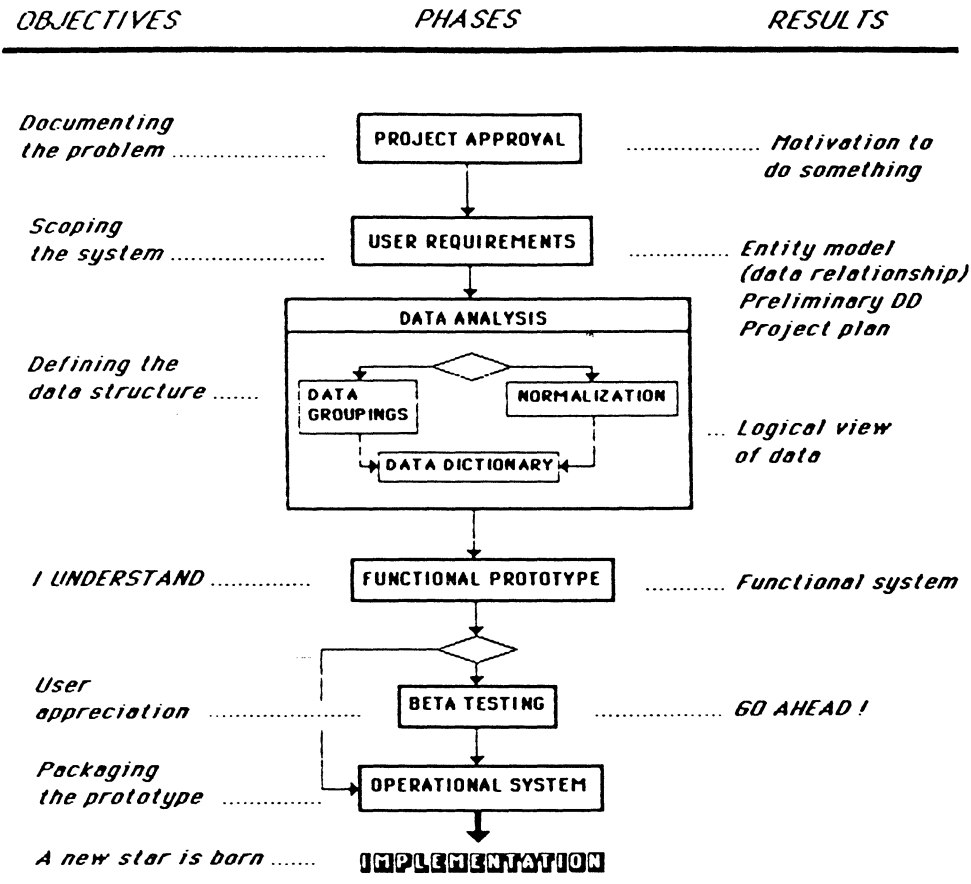
STAR NUMBER 4 The PHASED APPROACH deals with developing a system in phases in conjunction with the separate activities of the development process.

STAR NUMBER 5 PACKAGING THE PROTOTYPE covers the remaining functions to be developed, discusses optimization, system-wide testing and production system preparation.

STAR NUMBER 6 SYSTEMS AND PROGRAMMING STANDARDS deals with menus, screens, reports and naming conventions. It also discusses environmental factors like: development, testing and production. Lastly, it covers documentation and communications.

STAR NUMBER 7 GENERAL PLANS is the control over the project in terms of prototyping, testing, conversion and implementation. These controls must be specified early in the development process, in a Project Master Plan that includes: getting approval for the project by defining the current system, company objectives, the users' requirements and the data and documents required to fulfill the objectives. Then the master plan summarizes the system overview, outlines a plan and approach and discusses project controls like status reports, PROTOTYPE reviews and requests.

HOW TO APPLY THE 7-STAR PRINCIPLES AND DEFINE AN APPROACH



1 PROJECT APPROVAL

Getting approval for a project means documenting the requests and reviewing the objectives. The changing role of the project manager requires an understanding of the organization and management's point of view. The new project manager must also identify and organize the users in preparation for prototyping.

The manager's preliminary analysis summarizes the basic requirements and constraints or impact on the organization, and gives an overview of the system, whether it is manual or automated. Then the feasibility of the project can be assessed and an estimate can be made of the time and resources required.

The manager must also review and receive approval for the project. It must be consistent with the corporate philosophy and objectives. The role of the project manager includes setting up a user committee, getting a commitment from management, opening communication chanels and defining the project and team structure.

2 USER REQUIREMENTS

First, determine the current business functions, study the flow of data of the existing system and identify key users.

Go to the users and document their requirements. Make a summary list that describes the scope of your system and meets the users' needs as well. The important difference is that the manager can and must talk to the users to benefit from user satisfaction.

In determining the scope of the system, define the entities and basic data relationships for the entity model. The entity model is the graphic representation of the entities. An entity is a 'thing' (like a customer or a product) that represents data elements. The entity model is a logical view of the organization of the data. Then define the elements that make up a preliminary data dictionary. This results from documenting requirements with the users.

Document the technical requirements including hardware and software. Review system and programming standards and identify external interfaces. Then list the requirements for support and conversion. With the information the project team has gathered, they can now produce a project master plan and make a presentation to the user committee which helps carry the project onto the next phase.

3 DATA ANALYSIS

Data analysis is defining the data structure in detail. At this point in time, there are two possibilities: First, a good entity model has already been defined and accepted by the users. In this case, you must create the data groupings, define keys and carry out optimization and validation against the users' requirements.

'Data groupings' refers to the distribution of elements in the entity model. Data groupings are efficient if you have good access to and contact with the users.

The second possibility exists when you don't have good access to users or the entity model has not been clearly defined and cannot be used. You must then proceed with normalization. The basic steps are: 1. List all data elements; 2. Identify primary keys; 3. Separate repeating groups of elements; 4. Analyze "data-to-key" dependency; 5. Draw a relationship diagram. Then optimize and validate the grouping results against the users' requirements. The normalization process helps meet 2 objectives: it helps obtain detailed knowledge of the data and it helps to define the data groupings that will contain non-redundant data. The other objective of Data Analysis is that you can almost finalize the data dictionary.

4 FUNCTIONAL PROTOTYPE

The functional prototype is the result of merging the way the persons are organized and the way the data are organized. The objective is for the user to understand and confirm his requirements. This step involves the physical database design and the conceptual design. The conceptual design includes a review of the business functions and relates the business functions to the physical/logical data model. The system architecture (the system of menus and screens) is the result of the above. At the same time, create the development, testing, production and conversion environments.

A functioning prototype is then developed. It consists of menus, functions, and on-line help. At this point, you must win the confidence of the user with a functioning protoype and supply evidence of support like on-line help. This prototype must be developed very quickly to maintain the users' interest and must be a clear reflection of the users' requirements. The idea of producing a prototype is to translate "words" into "code" without detailed specifications.

The iterative development process then starts and includes demonstrations, reviews, collection of user feedback and program changes. You must avoid too many iterations and agree with the users in advance on the number of iterations. The project manager plays an important role because he is the only one who has control. Try to develop the "WE" approach, a critical aspect in prototyping.

If you decide to conduct a beta test, you can prepare, at the same time, the plans to implement the beta test. Complete all forms and the procedures guide which describes the purpose and function of the system, not just an operator's tutorial.

Finally, when you review the functional prototype with the user committee you must confirm the commitment for the beta test. Do not forget to carry out the system testing.

5 BETA TESTING

Beta testing requires the users to try the system' and give their approval to implement the production version of the system. The beta test includes: the hardware (if this is the case) and the software installation process, data conversion, user training, and security reviews. You may also want to finalize the forms and procedures. Performance is analyzed, a mechanism is set up to resolve problems, route change requests and make provisions for new system releases.

6 OPERATIONAL SYSTEM

The operational system is a packaged version of the prototype. First, review the prototype and documentation for the remaining functions. Build external interfaces and develop secondary system functions such as any remaining reports, security (if not done before), back-up and recovery. System test all modules, do volume testing and tuning and plan your installations (and why not use the Beta implementation plan).

7 IMPLEMENTATION

With careful attention to the users' requirements and needs throughout the project, user acceptance and satisfaction is assured and a new star is born the moment the system is implemented.

The operational system demands attention to the production environment, conversion, back-up and recovery procedures, system start-up and deliver, resolving problems and documenting change requests.

SUMMARY OF GUIDELINES AND BENEFITS

3GL methodologies had their problems. Thanks to 4GL we can now be more flexible and define an approach that really suits the size and environment of a project. But let us not forget about our mistakes nor forget to carry with us what we liked yesterday. Live on sound principles and become a successful system developer. What counts after all are: AN EFFICIENT SYSTEM and USER SATISFACTION.

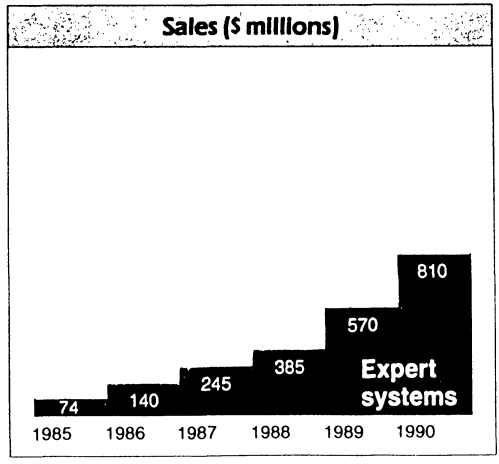
ARTIFICIAL INTELLIGENCE - NO LONGER A RESEARCH PROJECT

J. CHASE McEVOY, JR. & SUZANNE M. SPITZER
McEVOY, COOPER & CO., A PROFESSIONAL CORPORATION
6200 SAVOY DRIVE, SUITE 550
HOUSTON, TX U.S.A. 77036-3370

INTRODUCTION

As recently as 1980, expert system research was largely confined to a few university research laboratories. Today, the United States, England, Japan and the European Economic Community are all in the process of launching major research programs to develop and implement expert systems in the near future. Many Fortune 500 corporations are assembling AI departments; venture capitalists are rushing to invest in entrepreneurial expert system companies; and expert system technology is well on its way to commercial success.

The graph below illustrates the anticipated growth in sales in the expert system industry according to DM Data, Inc. (Scottsdale, AZ). The market is projected to explode from an estimated $140 million this year to some $810 million in the next four years. This paper attempts to examine the nature of expert systems, their historical development, avenues to their attainment, and, their benefits and limitations.



Source: *AI Trends '86*, DM Data Inc

EXPERT SYSTEMS: WHAT ARE THEY?

An expert system is a computer system encoded with human knowledge and expertise. Its goal is to solve problems at an expert level of performance. A typical expert system consists of the following four areas:

- A knowledge base containing both declarative (facts about objects, events and situations) and procedural knowledge (information about courses of action) and heuristics (rules of thumb). Possibly the most prevalent form of knowledge representation use is the rule-based system.

- An inference engine, which consists of the reasoning process. It determines which rules are to be invoked, accesses the appropriate rules in the knowledge base, executes the rules and determines when an acceptable solution has been reached. Common inference methods used are backward- and forward-chaining.

- The user interface is the component of an expert system that communicates with the user. A user may want to ask the system to explain its reasoning for example, or the system may request additional information about the problem. Most user interfaces make use of natural language processing techniques.

- Development and maintenance aids are the utilities and tools needed to test and maintain the system. These tools generally include editors that allow users to enter and change the knowledge base and debuggers to trace the reasoning process.

Expert systems is just one area that makes up the science of artificial intelligence. Other areas include natural language processing, robotics, speech recognition, vision, and theorem proving. Each area is not totally separate unto itself. Each area tends to utilize information gained from the other areas.

HOW DID THEY EVOLVE?

Artificial intelligence, as a science, has been growing as a body of knowledge since the arrival of the first digital computer in the early 1940's. As a science, it was formalized, however, in 1956 at the Dartmouth Conference in New Hampshire. There, leaders representing sciences from several disciplines (mathematics, neurology, psychology, electrical engineering, among others) met. The common thread that connected the interests of this diverse group was the way in which they used computers to conduct their research and simulate various aspects of human intelligence.

Expert system development did not emerge as an entity until the 1970's. In the 1960's, AI scientists tried to simulate the complicated processes of thinking by finding general methods for solving broad classes of problems. However, despite some interesting progress, this strategy produced no breakthroughs. In the 1970's, AI scientists began to realize that the problem solving power of a program came from the knowledge it possesses. This realization led to the development of systems that were expert in some narrow problem area. These systems became known as expert systems.

EXAMPLES OF EXPERT SYSTEMS

The first large expert system to perform at the level of a human expert and to provide users with an explanation of its reasoning was MYCIN. MYCIN was designed in the early 1970's at Stanford Medical Center to help medical specialists diagnose infectious blood diseases and to prescribe antibiotics for them. It was a pioneering expert system and took years of painstaking work to make it operational. Although not in use presently because of its cost to run, MYCIN can perform many tasks that are still beyond the reach of conventional software. It contains the expertise of some of the foremost experts in the field of infectious blood diseases. It uses this expertise to guide its operators to the most reasonable conclusions and then recommend the best alternative treatments for the problems it diagnoses. It communicates in English not "computerese." It has been the benchmark of several generations of successful expert systems, including programs that can be used to develop other expert systems. Some successors of MYCIN are:

- EMYCIN (1974) or "empty" MYCIN. EMYCIN is a tool for building MYCIN-like consultation systems. It is a knowledge system without any domain knowledge.

- PUFF (1976) and ONCOCIN (1980) were developed by attaching new sets of knowledge rules to EMYCIN. PUFF diagnoses the presence and severity of lung disease in a patient by interpreting measurements from respiratory tests administered in a laboratory. ONCOCIN assists physicians in treating and managing cancer patients undergoing chemotherapy experiments.

Examples of some expert systems that are paying their way and that are currrently in commercial use, are as follows:

- XCON (R1) - is used by Digital Equipment Corporation to find the best layout and part configuration for the VAX-11/780 system it sells, given each customers situation and hardware needs. It contains the knowledge from hundreds of skilled DEC technicians. It can now lay out any VAX system better than the best technician. Savings are estimated at $18-20 million per year in manufacturing costs.

- DENDRAL - developed by Nobel Prize winner Joshua Lederberg at Stanford, is used daily by chemists all over the country to discover the molecular structure of unknown organic compounds.

- Delta/Cats - built by General Electric to troubleshoot malfunctions in diesel locomotive engines

- Sophie - developed at MIT, trains electronic engineering students at MIT in designing and troubleshooting electronic circuits

- ISIS - developed at Carnegie-Mellon University is used by Westinghouse to schedule the most efficient use of its job shop and to manage its job shop projects.

SUCCESSFUL PROBLEM SOLVING AREAS

Expert systems have been most successful in the following categories of problem solving: interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction, and control. These categories generally involve problem domains that are: solvable, expertise available, limited in scope, cost effective, and easily maintained.

With regard to industry applications, the medical domain seems to be number one in expert system development with chemistry coming in a close second. Other industries where expert systems are continuing to be developed are: agriculture, computer systems, geology, electronics, engineering, information management, law, manufacturing, mathematics, meteorology, physics, process control, space technology, military science. To date, expert systems related to business management and finance have been few and limited. This area seems now to be receiving more attention and the attention it so very much demands.

WAYS TO ATTAIN AN EXPERT SYSTEM

Expert systems are no longer a tool of the "well funded." New and improved technology along with decreasing hardware and software costs are making the application of expert system development and/or usage in firms more feasible. Outlined below are several methods in which firms can partake of this exciting technology.

- Custom development route. To follow this route an organization needs access to a talented staff of AI professionals or a consulting firm with experience in knowledge engineering and development. Few people outside universities and specialized AI companies are capable of creating expert systems from scratch and many cannot justify the expense of an outside consultant to do the work. Costs may average $150,000-$200,000 per year.

- Semi-custom development route. Following this route, the organization starts with a commercially available expert system shell and fits that generic program to its specific needs by building a base of knowledge around the shell. Harvey Newquist, III, DM Data, Inc., maintains these shells can cut system development time by one-third to one-half. They typically cost between $10,000-$80,000. These shells prepackage an expert system's inference engine, freeing the knowledge engineer from the burden of creating this structure from raw code. To develop a complete expert system, the knowledge engineer need only add a specific knowledge base to the generic shell structure. Most shells are designed for implementation under a LISP environment on specialized hardware from DEC, Symbolics, Inc., Xerox Corp., Texas Instruments., Lisp Machine, Inc. and other players in the AI market. The following are some top development tools for these machines.

- The Knowledge Engineering System (KES) from Software Architecture and Engineering
- The Automated Reasoning Tool (ART) from Inference Corp.
- S.1 from Teknowledge, Inc.
- Knowledge Craft from Carnegie Group, Inc.
- The Knowledge Engineering Environment (KEE) from Intellicorp

. Package route. During 1985, the first large-scale "off-the-shelf," "ready-to-run" applications came to market. It is expected that these applications will go even further than the shell market has gone toward attracting businesses to develop expert systems. They will attract business users because of the great savings in development costs and time that they make possible. Initial purchase costs are high ($30,000-$100,000) but these applications require minimum involvement by the firm's data processing staff. End users are responsible for configuring the system with parameters relating to their environment and thereafter operating the system. Vendors typically provide all the necessary support and maintenance.

The following are some expert system packages that recently became available to assist for the planning and controlling of finances:

- The Financial Advisor from Paladian Software. This systems advises executives on capital planning for corporations.

- Planpower from Applied Expert Systems, Inc. PlanPower is designed to assist financial planners in creating financial plans for individuals. It has a 6,000 rule base and contains knowledge of about 200 types of investment strategies.

. Microbased shells and packages. Microbased shells and tools provide companies with an attractive route into commercial applications of expert systems. Managers do not need a mini, mainframe, or special LISP machines to explore these type of expert system applications. Commercially available shells typically support 250-2,000 rules at prices that start at a couple of hundred dollars. The summer of 1984 saw the introduction of several sophisticated IBM-PC based expert system shells and a number of AI companies have since ported their software to the micros. Examples of some vendors and their products in this category are:

SHELLS
- Expert Ease from Human Edge
- Personal Consultant from Texas Instruments
- Exsys from Exsys, Inc.
- KES from Software Architecture and Engineering
- M.1 from Teknowledge, Inc.
- Expert Edge from Human Edge
- Timm-PC from General Research Corp.

APPLICATIONS
- Planman from Sterling Wentworth Corp. This system is a financial planning system targeted at tax advisors.
- Diagnostic Troubleshooter from Technology Services, Inc. is a system targeted at the semiconductor industry.

BENEFITS AND LIMITATIONS OF EXPERT SYSTEMS

With all new technology, there are benefits and limitations in its use. Some of these benefits and limitations can be summarized as follows:

BENEFITS
- Expert systems can assist individuals in analyzing problems and making decisions. They can assist greatly in the productivity of problem solving as an intelligent job aid.

- Expert systems can produce more consistent, reproducible results on occasions where information must flow constantly. Expert systems are not subject to human traits such as fatigue, emotional strain and the like.

- Expert systems can provide novices with an aid that will help them to learn about a problem domain and to think the way more experienced professionals do. Transferring knowledge from one human to another is a laborious, lengthy and expensive process.

- Encoded knowledge is permanent. Human expertise can fade and the expert(s) can die. Information once acquired by the expert system is, under normal operating conditions, around forever.

- Artificial expertise is easier to document. There is a straight-forward mapping between the knowledge and the manner in which it is represented.

LIMITATIONS

- Expert systems may make mistakes. While conventional programs are designed to produce the correct answer every time, expert systems are designed after experts, usually producing correct answers but sometimes producing incorrect ones.

- People are creative and innovative. Human experts handle unexpected events using imaginative approaches to a new problems, including drawing analogies from different problem domains. Expert systems have little success doing this.

- Expert systems require continual maintenance due to the non-static nature of knowledge.

- Human experts have common sense knowledge. Because of the enormous quantity of common sense knowledge, there is no easy way to build this into an intelligent program.

- Expert systems are only as good as the expert(s). Some experts are more "expert" than others.

Despite the limitations that exist in the development and use of expert systems at this time, expert systems can still offer tremendous potential to most organizations when used as a consultant or aid to either an expert or novice in some problems area. They function best in problems areas which are: solvable, well-defined, have practical importance, no common sense is needed to solve the problem, experts exist in the area, the problem is cost effective to codify and can be reduced to a small number of rules.

CONCLUSIONS

The expert system is the most practical application to date that has come out of the AI research labs and the one that has had the widest impact on business and industry. Despite their limitations, expert systems are here to stay. With improvements in hardware technology and software, the field of expert systems is within reach of most businesses.

If you are in middle or upper management in a corporation or owner of a business, you need to become knowledgeable about the activities in this field. Start experimenting with the inexpensive shells available on the microcomputer and try to keep current with new developments in your area. The infiltration of this technology in organizations is already underway. It will continue through the 1990's as computer-based intelligence moves into every nook and cranny of the organization.

The time is NOW to rethink the way in which your organization handles its business and to monitor closely the literature covering this field. Organizations that are "cognizant" of the application of expert systems to areas of their business are going to be able to utilize this new technology to gain increases in productivity and significant competitive advantages.

REFERENCES

Cronin, Beverly, "Micro-based Systems Pave Low-Cost Route to Commercial AI," Computerworld, January 13, 1986, p. 54

Davis, James R., "Custom-Developed Expert Systems Offer Strategic But Costly Alternative," Computerworld, January 13, 1986, p. 52

Harmon, Paul and David King, "Artificial Intelligence in Business: Expert Systems," John Wiley & Sons, Inc., New York, 1985

Mishkoff, Henri C., "Understanding Artificial Intelligence." Texas Instruments Information Publishing Center, 1985

Newquist, Harvey P., III, Editor "AI Trends '86 - A Comprehensive Annual Report on the Artificial Intelligence Industry," DM Data, Inc., 1986

------, "Expert Systems - The Promise of a Smart Machine," Computerworld, January 13, 1986, p. 44

Pfau, Daniel R., and Barry A. Zack, "Understanding Expert System Shells," Computerworld, February 19, 1986

Van Horn, Mike, "Understanding Expert Systems, An In-Depth Guide to the New Generation of "Smart" Computer Software," Bantam Books, 1986

Waterman, Donald Arthur, "A Guide to Expert Systems," Addision-Wesley Publishing Company, Inc., 1986

Weiss, Henry Fersko, "The Intelligent Computer," Personal Computing, October, 1985, p. 62-73

------, "Expert Systems Decision Making Power," Personal Computing, November, 1985, p. 97-105

Winston, Patrick H. and Karl A. Prendergast (editors), "The AI Business: Commercial Uses of Artificial Intelligence." The MIT Press, Cambridge, Massachusetts, 1985

COMPUTER INTEGRATED MANUFACTURING FOR EXECUTIVES

TERRY H. FLOYD BLANKET SOLUTIONS 22607 SMOKEY HILL KATY, TX USA 77450

INTRODUCTION

Planning and organizing for CIM implementation is a much larger task than implementing, installing, and using other systems, even closed loop Manufacturing Resource Planning (MRPII). In total concept, factory automation is a diverse set of individual decision support systems. Management is now challenged with adapting to using these automated techniques and integrating them into the factory of the future. Non-manufacturing companies, institutions, and society at large can benefit from the progress being made by CIM pioneers.

This paper examines some of the CIM applications and discusses a project approach to implementation. The key issue in both aspects is vendor cooperation because the project spans all departments in the organization and many different types of hardware are already in place.

CIM APPLICATIONS

Presentations about computers in manufacturing concentrate on specific applications or their integration because all of the data is filtered and passed upward through networks. The goal is to produce information for management decisions. As the integration progresses, the automation of business functions through knowledge bases, inference processing, expert systems, and other artificial intelligence techniques will do for the knowledge worker what traditional CIM applications do for the shop floor and engineering departments.

Attendance was low for the HP1000/9000 technical presentations at last year's Interex Conference in Washington, D.C. Nonetheless, some very interesting and exciting ideas were delivered. The emphasis on HP3000 applications that stress business applications drew most of the participants who probably thought "scientific" applications had little relevance for them. They would have been surprised by many of the ideas in the so-called technical papers.

Some papers were more hardware oriented such as the following: "Using the HP1000 to Automate a Machining and Gauging Process" by C. Pappagianis and "Using an HP1000 A-Series Computer as an SNA Gateway" by Reid MacGuidwin. Others explained software, like "A Test Program Development System based on the HP9000 Family of Computers" by Jerry C. Merritt, "Real Time Analysis under EDS" by Dan Schneberk, and "HPTECHWRITER, Integrated Text and Graphics for the HP9000 Series." These and several others would have been of general interest to executives responsible for white collar automation.

Manufacturing automation was directly addressed by numerous authors whose knowledge of business goals and project management bridged the gap between "scientific" and "data" processing. Excellent examples included K. Benson's "Real Time Management and Control of Factory Operations", "Incoming Material Management and Vendor Quality Control" by G.R. Nevogt and G.L. McCrory, "CAR/1000 Computer Aided Calibration and Recall System" by Thomas Barrett, and "Good Laboratory Practices and How They Can be Supported by a Laboratory Information Management System" by Tom Boyer and Kazmer Latven. There were also presentations on communications and networking.

One of the points of this paper is that executives interested in office productivity improvement can learn the latest concepts from practitioners of engineering and production disciplines. The HP1000/9000 technical presentations at the HP/Interex conferences are a good place to see practical applications of these concepts.

REALIZING THE CIM DREAM

The complexity of the project and the committment required to implement may surprise those who are not familiar with Computer Integrated Manufacturing. Today's large factories have hundreds of opportunites for interfacing existing stand-alone systems into the MRPII (corporate) system. Choosing the CIM candidates and managing for standardization requires upper level supervision and control.

Total CIM will take a long time and initial success is important. Detailed concept analysis and planning are critical. The expense of large scale automation of factories has produced excellent guidelines for others to follow and improve upon. The amount of preparation is immense, spanning the organization from marketing to engineering to finance.

Detailed knowledge of the products and product lines, their life cycles, mix, volumes, and similarities to future products must be collected and prepared to see where alternatives for automation will have the best payback. Inventory components and process routings are studied to estimate which resources will be used. Performance measurement of existing methods of utilizing workforce and machines, and plans for future capacity must be considered.

Relationships with vendors is the most important aspect of CIM in more ways than one. Quality from vendors supplying raw materials for production is an area often addressed by CIM projects. The working partnership with the various suppliers of automation tools and technology is as important to the project team. Frequently, integration is an inventive process of solutions worked out between the customer (your company?) and the vendor's technicians or designers.

But, the elegant technical solution is misdirected and futile unless management has chosen to proceed with cost reduction AND improvements in production quality and flexibility. Labor and material cost improvements are possible through standard MRPII systems with decision support output. CIM can improve the

PROCESS of manufacturing and collect MRP data sooner and more accurately.

THE PLAN

An overview concept definition includes production decisions like just-in-time flow lines and Flexible Machine Centers, yields, and plant layout. These are some of the physical and site analysis and design aspects. The information flow, the organization and skills of personnel, and business plans must be documented. Training and education needs are assessed. Many charts and graphs are the product of the acquisition and analysis of the state of the existing systems and best guesses about the future.

A schedule for short term implementation and long term strategy could be expected after a 3 to 6 month study. At the conclusion, resources required and possible problems needing special attention should be estimated. The cooperation of all areas must be assured. Lack of participation of one department in a project involving integration can stall the implementation. A project team with responsibility for all information flow from capture to presentation is needed.

The project team ideally would be led by an individual with knowledge of manyfacturing from engineering to cost accounting, information systems, finance, marketing, and a reputation among vendors. Such a rare person must be a manager of exceptional ability. Outside consultants, systems integrators with subcontractors, analysts, programmers, technicians, design engineers, auditors, accountants, and many others must be organized to work together. Goals and milestones must be reported to upper management, problems have to be understood and explained, and resource allocations justified.

CONCLUSION

Only the best people available can be expected to produce a working pilot system for the total integration effort. The model set by the initial sub-project will guide future implementations. The people will be trained here for the next generation's improvements. Read and learn from those who have had their hands on similar projects and Plan, Plan, Plan.

In Search Of The Software Transistor

Tim Chase

Corporate Computer Systems, Inc.
33 West Main Street
Holmdel, New Jersey 07733
U.S.A.

(201)946-3800

642672 CCSHOLM

"In the beginning the computer was invented to solve the problem. What seems to have happened is that the computer has become the problem. So now the question is, what can we invent to ..."

- Robert M. Baer
The Digital Villain

In Search Of The Software Transistor

Despite advertising to the contrary, programming is still done pretty much the same way in the 1980's as it was done in the 1950's. There have been some changes, but they are nowhere near the order of magnitude of change which has been brought about in hardware by technological advancements like the transistor.

This paper presents a sketch of some important current software development techniques including forth generation languages, expert systems and ADA. It discusses some of the problems which must yet be solved if the future of programming is to be a bright one. Finally, this paper makes some predictions about where programming is going in the near future. Although optimism springs eternal, it is concluded that the discovery of the true "Software Transistor" is still a long way off.

About Predicting

The theme of this conference is "Focus On The Future." With such a theme, it would appear appropriate to take a chance and make some predictions about what will be happening to software development in the near future. This is a dangerous game, especially when the predictions are made about the time frame which will (hopefully) include the life of the author. If we were to predict for the year 3000 our reputations for soothsayers would remain unsullied for the remainder of our lives. Undaunted, we will attempt to sketch a brief picture of what we think is around the immediate corner for software development in general and programming languages in particular.

By their vary nature, programmers tend to be optimistic creatures. This

was noted by Frederick Brooks in his, by now classic, book The Mythical Man-month. Brooks explains programmer optimism by saying that perhaps there is a natural selection process by which the frustrations of the job drive away all but the most optimistic. Whatever the reason, the trade is populated with optimists who survive mentally by believing that the project is really 95% finished or that this bug is the last one in the system. Predictions by optimists (especially those trying to get research grants) are bound to be tainted.

Marvin Minsky, a popular M.I.T researcher, in Artificial Intelligence was quoted in the November 20th, 1970 issue of Life Magazine (one of the US' better technical journals) as saying:

> "In from three to eight years we will have a machine with the general intelligence of an average human being. I mean a machine that will be able to read Shakespeare, grease a car, play office politics, tell a joke, have a fight. At that point the machine will begin to educate itself with fantastic speed. In a few months it will be at genius level and a few months after that its powers will be incalculable."

Poor Marvin, he committed the double error of being an optimistic programmer (a-hem, researcher) and predicting within his own life span. The point of this all is that programmers are often the ones who are making predictions about programming and computer science. This usually means that things are predicted to be much rosier than they really are.

What we will offer here is a slightly pessimistic prediction of the near future, but since we, ourselves, are programmers, the prediction will actually be somewhat optimistic. We hope that the two forces will cancel out and the result will be realistic.

History teaches...

Before looking into the future, it is often helpful to look into the past if only to discover that looking into the past is not all that helpful. Fortunately, for computer historians, computer science is quite young. We don't have to find fossilized printouts in order to get insight into the dark ages of data processing. Most people refer to "generations" of computer hardware. Although this is often just a marketing technique (any given vendor is always working on the "next generation") it is sometimes useful to contemplate the generations of computer hardware:

1. Electromechanical/vacuum tube computers. These were the first. They were large, unreliable, slow and often doubled as space heaters.

2. Transistorized computers. IBM's 7090 was one of the first, and some wistfully think one of the best transistorized computers.

3. Integrated circuit computers. Smaller parts made for logically larger computers.

4. VLSI (Very Large Scale Integrated) computers. More (less) of the same.

5. Computers from Japan, Inc.

The fifth generation computers haven't been born yet regardless of what vendors are saying. Most American Universities writing grant proposals feel as if the Japanese are on the brink of the fifth generation and that the US will lose its dominance in computer science unless more money is spent for research.

As luck would have it, there also appears to be similar number of generations in computer software. This is especially obvious to all those folks selling forth generation languages. There is little connection between the generations of hardware and the generations of software other than faster computers can do more computing. It appears to be a fact that advances in software always require more computing.

As we see it, the five language generations are:

1. Ones and zeroes. Really the old days. This is where Grace Hopper got her start.

2. Assembly language. This includes macro languages, linkers and the like. It is amazing how many programmers still feel that there is something noble about assembly language.

3. So-called high level languages. These include FORTRAN, BASIC, C, PASCAL, PL/1, LISP, COBOL and your favorite.

4. Programming environments. These are integrated facilities which combine languages, data bases, and screen facilities. They claim to enable programmers to develop prototype and final applications quickly.

5. What ever Japan, Inc. picks for the fifth generation. More seriously, the fifth generation appears to be expert systems with a side order of nonprocedural programming. This is different from programming languages in the classical sense, as we shall see.

By looking at this very brief history and by observing where we stand right now we may conclude some interesting things. The single most interesting conclusion we can make is that that hardware is far and away outpacing software in terms of progress. In the world of hardware, significant advances have been made just about every 10 years. These advances have led us from computers which filled rooms to computers which fill thimbles yet perform faster, cheaper, better, etc. The important thing to note is that there have been orders of magnitude improvements made in hardware development which come at regular intervals and are related to improvements in basic technology.

Software, unfortunately, is another story. If you include FORTRAN in the third generation of software language development then you find that we entered that generation on November 10th, 1954! On that date, a document titled "PRELIMINARY REPORT, Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN" was published by the Programming Research Group, Applied Science Division of IBM. The amazing thing is that the first computers had only come into being around 1948. This means that about six to eight years after the first generation of computers, we were already into what we now consider the third generation of programming languages! Couple this with the fact that we think we are currently in the fourth generation and you have the basis for a depressing hint of what might come.

Granted, FORTRAN does not embody all that is true and beautiful in current modern programming languages. The point we are making here is not that language development stopped in 1954, but rather that the changes which have come to programming have been small and have not even come close to having the impact on throughput that corresponding changes in hardware have had. We realize that some readers will respond violently to these charges; that there is a favorite feature of a favorite language which is being maligned here. To this we ask that you stop and consider the difference between a computer constructed from relays and a Motorola 68000. No programming language improvement comes anywhere near that level of change.

Why is there such a difference between hardware and software?

This is an important question. In order to answer it we must first begin to insult hardware developers. If you look at the changes in hardware development you notice one significant thing. The software model of computers has not changed much since the Beginning Of Time. By software model we mean how the "inside" of the computer is organized; the part the programmer sees. Again, we expect that there are some who will argue, but when you get right down to it computers have remained much the same since the beginning. What has changed with the computer generations is the technical implementation. Take the venerable IBM 370 as an example. It would be possible to implement a 370 in vacuum tubes. Clearly you might need Niagara Falls to cool it, and the G.N.P of a medium sized Latin American country to pay for it, but it could be done. Likewise, a 370 could be built using transistors and other discrete components. Finally, a 370 could be built from VLSI parts. In fact, it probably would only take one VLSI part. What we would see across the different implementations would be a vast range of performance with the vacuum tube 370 hopefully at the low end of the scale and the VLSI at the high end.

These so-called "technology remaps" have been used by computer vendors throughout the years to offer faster computers which still run the same software. The important point to remember, then, is that the "stuff" that computers are made from has been changing but the design has remained steadfastly the same. When a new computer is announced, we all ask the same questions (how many registers, how many CPU's, etc). We are never surprised with the answers because the architecture is always pretty much as we expected. (It is interesting to speculate how well a

really different computer would sell. Imagine you get the first look at a new computer design and find it resembles a fish tank filled with a rose colored jelly with wires sticking out from it and nobody you have working for you has the slightest idea how to get accounts receivable running on it. How many would you buy? With economics as the master, perhaps we are getting exactly what we are asking for.)

So, hardware has the benefit of physics behind it. The hardware boys are innovative, sure, but they don't have to find vastly different organizational approaches to improve their product. A pipe line here, a parallel processor there and a heavy dose of solid state physics accounts for the orders of magnitude in hardware improvements.

Now, how about software? Well, software is a tough one. This is because programming is very much akin to thinking. Programming is problem solving. In a very real sense, programming is us. The difficulty is that it is hard to do a technology remap of our own brains. The implementation of the programming "machine" has remained constant over the last 40 years. It still remains "liveware." The problems associated with programming significant programs are problems which have faced mankind for ages. They are human organizational problems. How do you organize a number of people so that they are all working toward a common goal? This is especially difficult if the goal is getting a computer to do something.

What is programming and why is it so hard?

One of the problems facing program developers is that programming is difficult, yet the popular concept of computers (from numerous Charlie Chaplin ads) is that they are easy to use. It may be true that computers are easy to use, but it is also true that they are difficult to program. This difficulty stems from the fact that the physical act of programming represents only a small part of getting a program out of a customer's head and into a computer.

Programming is much more than writing COBOL statements. A large portion of any job is spent in planning what the program will do. Frederick Brooks says that at least one third of a project is spent in planning and only about one sixth is spent in actually writing code. Our own experience indicates that this is quite true. Further, as planning progresses, the thing being planned often becomes so complex that no one fully understands it any more.

The software developer wants to develop functional requirements which are detailed so that he knows exactly what is going to be built. The finished documents are often beyond the understanding of users or customers who are forced to sign off on them in order to begin development. Time allocated to testing is often used up by development which results from customers finally getting to try the system. The relationship between developer and consumer is often ruined by mismatched anticipation levels. Even with lengthy requirements documents, the customer often does not get what he wants.

In short, software development is a dirty difficult business. Regard-

less what the data sheets say, it is hard to write good programs which meet the needs and anticipations of users.

Our conclusions for the current state of computer science is that things have not changed all that much since the early days of programming. The big changes have come from the hardware side of the house -- no one has, as of yet, discovered the software transistor.

What does future hold in store?

Hold on. This is where we start predicting. Software development has not changed significantly since the beginning. We don't see big changes in the near future. What we do predict is that programming computers will not get easier -- using computers for some, however, will get much easier. There is a very important distinction between using computers and programming computers.

If things continue as they are now, we see a sort of class structure developing. In H. G. Wells "The Time Machine" the world is peopled with two classes: the Eloi and the Morlocks. The Eloi are forever young and beautiful. They live lives of complete leisure while the Morlocks toil beneath the ground tending the machines which make the world work so ideally for the Eloi. Of course, in the end, the hero discovers that the Eloi are actually raised like cattle for the Morlocks to eat.

Except for the culinary twist, we see much the same for computers. There will be the Eloi who work with increasingly sophisticated packages designed to enable them to use the computer without a great deal of effort. One of the technologies which will make this possible will most likely be what we now term "expert systems". Expert systems are a form of "declarative" or "nonprocedural" programming brought to you by the folks in the artificial intelligence labs. (Remember Marvin Minsky?)

The basic goal of nonprocedural programming is simple: tell the computer facts about the problem, toss in a few rules relevant to the solution and the computer does the rest. The Japanese in their Fifth-Generation project have (according to some reports) selected a programming language called PROLOG as the base for nonprocedural computing.

Nonprocedural programming is a good technique but it is not without problems. PROLOG is a good example. Most would agree that PROLOG is a nonprocedural language and for small programs it does, in fact, appear to do just what is asked for. PROLOG allows the programmer to enter facts and rules and then ask questions about the data PROLOG "understands." The PROLOG system searches the facts and rules to derive an answer to the programmer's question. For small problems, PROLOG does not need any procedural input from the programmer and demonstrations are quite impressive. But, for interestingly large programs, PROLOG grinds to a crawl. This is not too surprising because declarative languages spend most of their time searching the solution spaces defined by the facts and rules. The only way in which they may be speeded up, short of faster hardware, is to introduce (you guessed it) procedural programming to encode heuristics in order to trim the search space down to size.

Others, in fact, have basic doubts about the whole concept of non-procedural programming. As Jean Sammet pointed out way back in 1969 in her book "Programming languages: History and Fundamentals", the concept of nonproceduralness is really a very relative term which changes with the state of the programming art. To an assembly language programmer a statement such as

```
X = A + B * C
```

is nonprocedural. After all, we did not tell the compiler how to calculate the expression, only that we wanted to calculate it and where we wanted the results to end up. If you really understand the inner workings of a language like PROLOG (and you'd better if you're going to write any industrial-strength applications) then it becomes procedural. But, of course, it is not a very good procedural language.

If the Eloi use the expert systems who is going to build them? The Morlocks are the builders and they are faced with a double whammey. First, they must code the basic core of the expert system. To mystify the art, the basic core program is often called the "inference engine." The bad news is that the inference engines are "old fashioned" procedural programs with all of their associated problems (hard to write). Worse than that, expert systems introduce a new kind of programming called Knowledge Engineering (KE for short). If you think classic programmers have a bad time of it, wait until you hear what KE's do for a living.

It appears that expert systems are well suited for "consultation" programs. This is where the Eloi user sits down and chats with the computer to get some advice on what to do in a given situation. The example everyone sites is the MYCIN program developed at Stanford University in the 1970's. Until MYCIN, most expert systems spent their days trying to beat humans at chess or tic-tac-toe. MYCIN was the first serious expert system. Its job is to act as a consultant giving advice on the diagnosis and treatment of bacterial blood infections (we're not talking pawn to king four here). Now, you might ask how did MYCIN get its smarts about blood? The answer lies in the KE. The knowledge engineer's job is to sit down with experts, to pick their brains and then to encode the expert's problem solving techniques into a data structure. The resulting "knowledge base" is the brains behind the expert system. If writing good programs is kind of hard, then knowledge engineering is down right difficult! For certain it is not something that the Eloi are going to be able to do on their days off.

As knowledge bases grow so does the potential complexity of the computer's responses. It is currently difficult to fully test and debug conventional computer programs. In the future it will be even harder to debug expert systems. In their most gross form expert systems are collections of facts and rules. Are the rules right? Are there enough of them? Do some contradict others? If expert systems are built which approach the complexity some computer scientists say we can expect in the near future, we should not be surprised at hearing something like the dialog Arthur C. Clark wrote for 2001: A Space Odyssey. In one scene space man Dave Bowman is locked out of the spacecraft by HAL the onboard computer (obviously a PROLOG-based expert system):

Bowman: Open the pod-bay doors, please, HAL. Hello, HAL, do you read me?

HAL: Affirmative, Dave. I read you -- This mission is too important for me to allow you to jeopardize it.

HAL has reasoned that the only way in which he (she?) can complete his mission in space is by killing the crew. It's perfectly clear to HAL even if it isn't clear to the crew. In the end, it is a set of conflicting rules in HAL's programming which drives the computer into an electronic psychosis and gives Dave a one-way ticket to the infinite. We predict that large expert systems will be plagued with the same HAL-like problems for some time to come in the future.

But what about the Morlocks? They reap none of the benefits of the Eloi when it comes to programming ease. This means that even in the future someone will still have to bang the bits. Expert systems may become great at diagnosing diseases, but ask them to write a conventional program and they'll call for a urinanalysis. Thus we see programming remaining a job which will have to be done by humans for some time yet to come.

The final thing to remember about expert systems is that there is nothing magic about them. The concept of an expert system is "just another" programming technique. It makes some problems easier to solve, but the results gotten by expert systems may be obtained by conventional programming techniques. Often those selling expert systems lose track of this fact.

In the near future, we predict that expert systems will fail to be commercial successes until their developers understand the consumers are interested in solutions and not techniques. Joe user wants a system to do a particular job. He doesn't care how its implemented only that it does what it is supposed to do and that it doesn't cost an arm and a leg. Currently, expert systems appear to be a technology in search of an application.

Well, how about ADA?

If the expert system isn't the software transistor is there anything around the corner which might be? Sadly, we don't see it. There is, of course, work being done on programming languages with one current result being ADA. ADA brings smiles to the faces of a good number of people. In fact, just the mention of ADA during a presentation (with an appropriate roll of the eyes to the ceiling) is guaranteed to get a laugh. Seriously though, ADA does contain some important features which will be needed for a well-focused programming future.

Starting with the worst, ADA's least attractive feature is its size. This stems primarily from the fact that members of committees which design languages have never developed an effective argument against the statement "put the feature in -- if programmers don't like it, they don't have to use it." ADA, and its associated environments, are large

enough that there will be local experts in the language. People will be skilled in ADA task management, but won't be so hot on ADA I/O. This will be something we'll just have to live with.

Better ADA features include the attempt to make a really portable language. Languages like C have been touted as being portable, but, in fact, much of the portability found in C is a result of the careful use of features by the programmer. ADA's portability comes from within. Portability will be extremely important in the future. This is because systems built for the Eloi will be expensive to develop and it will be important to amortize that cost over a large number of installations. To have a package which runs on many machines will be a requirement.

If language efforts like ADA are making important contributions to program portability, then they are also making contributions toward people portability. People portability? People portability is being able to get your programmers to easily migrate from one computer to another. UNIX and C have gone a long way to make portable people a reality. If software and programming environments move easily from computer to computer, then computer systems will tend to look more or less the same. "If it's UNIX I can make it work" is something we have heard UNIX programmers say. We will be hearing more of this in the future.

Portability will be a good thing for programmers and computer customers of the future, but perhaps not such a good thing for computer vendors. If everyone has the same operating system (UNIX?), then computer customers will no longer be held to a given vendor. Customers will be able to "shop" for solutions and buy the most bits for the buck. Vendors will no longer be able to count on the captive customer for their computer sales. They will have to compete through raw horse power or intangibles like support or service.

We will be getting a glimpse of this when HP starts shipping the Spectrum computer line. The technical computer version of the Spectrum machine will be a UNIX box. This will mean that it will compete with all the other UNIX boxes out there. It will either have to be a barn burner or potential customers will have to believe in HP service, support, etc., etc. This is dangerous for computer manufacturers, especially for those who don't make their own chips.

In the future, computers will come to be considered "delivery vehicles" for well known applications. Customers will know the accounting package they want to purchase, and they will shop around for the appropriate delivery vehicle for that package. The number of registers, whether or not the machine is RISC will be unimportant. How well the chosen application runs will be the only question. Users will have the ultimate benchmark -- the actual system they want to run.

ADA and other language systems, as opposed to compilers, will also aid in some of the organizational problems facing programmers. ADA compilers will maintain application data bases allowing routines to be compiled within the context of a given intended usage. This enables the compiler to make more checks to insure that subroutines are called correctly and that parameters are passed as required.

If all of this sounds like Big Brother, you're right. In the future, we predict that much of the romance of programming will be gone. Many of today's software gurus pride themselves in being nonconformists; working odd hours and subsisting on peanutbutter-cheese-cracker sandwiches. "No neckties for me, no sir!" ADA (or at least the intention of ADA) is the beginning of the end for the happy hirsute hacker. Building for the Eloi will require legions of Morlocks and legions require order not anarchy. Programming as a means of self-expression will begin to fade as the programming languages and tools start to insure that you have to play it by the rules. Hackers may hate this, but like the Great American Cowboy, they will have to make way for Big Business. Managers need more control over projects and completed software will have to be easily maintained. Remember, portability will mean that software will have an extended life cycle.

Finally, we are beginning to see techniques and tools emerge which are short of flash but long on usefulness. They address the design and support phases of software development. It is said that the cobbler's children often run bare foot. This is certainly true for programmers. It seems as if programmers are often the last to benefit from computerization.

Additional work must also be done in software prototyping in order to avoid lengthy prose descriptions of what systems will be like. Wouldn't it be much nicer for developers and customers alike if they could sit down at a computer and watch a prototype of the application execute? One terminal session is worth a thousand pages of typed description. Some 4th generation languages provide features such as this. We expect to see more in the future.

Software change control systems are in use now. We see improvements in them and the integration of program development subsystems. Perhaps expert systems will help us stay on track when managing our time and our work load as programmers, designers and debuggers. In the past we have devoted much of our time to the development of the ideal programming language. Now we are starting to realize that there may be equally important uses for the computer in other phases of the program life cycle.

And in conclusion....

Future users of computers will have a field day. They will be freed from the nuts and bolts of programming, even if they are restricted in what they can use the computer for. Well defined applications will be easily performed by expert systems in areas which will likely surprise us.

There will, however, probably be even more need for classic programming in the future. For those who choose to do this work, we just don't see the software transistor waiting in the immediate future. The problems of program development are profound and are inexorably intertwined with being human. Tools are under development which will make life a little easier for those who will do "real" programming. Software portability, programmer portability and Big Brother programming environments are all steps in the right direction. We think, though, that the best we can

hope for is a slow and steady sequence of small steps toward Every Programmer's Dream -- to put himself out of work.

LOGIC PROGRAMMING AND EXPERT SYSTEMS

Shawn Brayman

Brant Computer Services Limited
Burlington, Ontario

Contents

1.0 Introduction

This paper is intended to provide an executive summary or overview of Artificial Intelligence (AI) in general and Logic Programming in particular. I have attempted to provide enough detail and examples to make the paper meaningful while at the same time keeping it geared to an introductory level.

The first sections will be a short overview of AI and a capsule history of the field. In section three we will take a quick look at AI programming, some myths and some observations about this new tool. Included in this discussion will be a mention of the pros and cons of LISP versus PROLOG as the two primary development tools in the AI world.

In section 4 we will take a look at the knowledge acquisition process and try and differentiate between the role of a systems analyst and a knowledge engineer. Section 5 reviews a few of the specific expert systems projects Brant is directly or indirectly involved with and the final few sections discuss the AI market place, and HP's positioning in that market.

2.0 Overview of Artificial Intelligence

Artificial Intelligence is not a new field; it has been around since the 1950's when people like Dr. Marvin Minsky helped found the first Artificial Intelligence Laboratory at MIT. Since that time, thousands of researchers in dozens of universities have added to the research effort. Although the field is not new, what is new is the perception of commercial readiness of certain aspects of the technology, namely expert systems.

For the first decade, the research appears to have been down what proved to be a blind alley. The effort was based around the attempt to create a hardware/software "thinking machine". It was felt that once we discovered how a person thinks, we could put this "thinking algorithm" into a computer. We could then provide it with information and it could "think" out the answer. The search was on for a general problem solver, a thinking machine. The search was unsuccessful.

By the mid 1960's, recognizing that the attempt to create a general problem solver was not going to be successful in the short term, researchers tried a new tack - to try and create a program that could emulate a human expert in a limited domain of knowledge. The first major expert systems were born in the late 60's.

DENDRAL was an expert system designed to help determine the structure of chemical compounds based upon analysis of the components of the molecule. MACSYMA was a second expert system started shortly afterwards that was designed to solve

symbolic mathematical problems, much as we did in Algebra back in high school. In both cases, however, the difficulty of the problems is substantially greater than a high school level. Both of these expert systems are in routine use today.

In discussions of AI, much confusion tends to arise as a result of the various aspects of the field; in the same way that the question "What is computer programming?" could be met with answers ranging from machine-level programs to applications programs, with a world of possibilities in between. In the field of AI it is further complicated by the fact that the linguists, the psychologists, the philosophers and the computer scientists, all working in AI, all have different perspectives. Some of the application areas under study are:

- expert systems
- natural language understanding
- automatic programming
- learning systems
- perception and vision recognition
- robotics and more

When we discuss whether or not any of the existing AI systems are truly "intelligent", all of the experts, the AI gurus, disagree. Some feel that we can already emulate intuition and human intelligence in some areas. Others, like Minsky, feel we are probably fifty years from machine intelligence and must first teach machines things like common sense and a sense of humour.

I will not take sides in the argument, nor do I feel it is that important. Most of us have enough trouble determining if there is intelligence in many people, let alone machines. What is important is that whether intelligent or not, we are developing a new style of software that in some ways emulates human "thinking". There appears to be little doubt that the impact of this new type of software will be substantial.

Now let's take a look at some specifics about AI programming.

3.0 AI Programming

Although it is obvious that there must be differences in AI programming (else why the hype?), there are obviously as many myths about AI. In this section of the paper, we will initially discuss several basic issues concerning AI. From there we will go on to discuss some specifics of AI programming and how it is advantageous.

First let me emphasize that AI programming is, most importantly, programming. You have a language, usually either LISP or PROLOG, and you develop programs. The programs we

will be discussing are intended to solve certain types of problems - problems that are usually solved by someone in your organization with specific types of expertise; an expert in his field.

Second, although AI is often called fifth generation and PROLOG and LISP specifically called fifth-generation languages, this is a bit of a misnomer. LISP was invented in the 1950's, around the same time as FORTRAN. It processes symbols rather than data , but other than this qualitative difference, they are both of the same basic "generation" of programming tools. PROLOG was developed in the 1970's and may be more equivalent to a 4GL like PowerHouse or SPEEDWARE. Symbolic processing is not designed to replace 4GL's.

In fact, one of Brant's current projects is the development of a Personal Financial Planning System for a client in Toronto. This system is actually composed of two parts: a conventional 4GL model developed using SPEEDWARE and an Expert System Financial Advisor that incorporates the financial planning expert's knowledge on how best to manipulate the model. We are in effect applying the strong suits of a 4GL language to those aspects of the problem which are algorithmic in nature, and using Brant's MPROLOG product for the expertise aspects of the system.

Third, let me point out that any program that can be written in MPROLOG or a 5GL can be written in a traditional language like COBOL. In fact, MPROLOG is written in the "C" language on the HP3000. With this in mind it stands to reason that anything written in MPROLOG on the HP3000 could be written in "C".

A good example is the Canadian Forestry Service department which is involved in forest resource management. Over the course of fifteen years, a comprehensive program was developed in FORTRAN that was designed to provide on-line real time advice to individuals trying to manage forests, deal with fires, etc. As the system evolved, it became more and more complex, alterations and their ramifications more difficult to predict and maintenance was a costly part of the budget. The decision was made to rewrite the system in PROLOG, and in so doing to be able to "sidestep" the logic ordering of the program and to deal with the relationships or rules.

Most AI programs are a series of rules and facts expressing the relationship between any number of things in our problem domain. The rules express these relationships in simple forms:

Rule 1: The computer won't work if
there is no power.

Rule 2: The computer won't work if

the power supply is shot.

Rule 3: There is no power if
the on/off switch is turned off.

Rule 4: There is no power if
the cord is not plugged in.

Rule 5: There is no power if
the building's power is out.

In creating a diagnostic program to determine why our computer won't work, we put in a series of rules that outline relationships. MPROLOG will then sort and order these relationships - much as you might structure them in an ordinary program.

Let's take a look at some specific examples.

3.1 Dealing with Complex Relationships

In an effort to outline a few of the areas where AI techniques and tools are advantageous, we will look at a few examples. First let us consider a situation where a relationship exists between several conditions. The example we will use is that of a program to determine if an individual is eligible to receive a personal loan. Let us assume that the factors that are considered in a loan application are the amount of the loan, monthly payments, income of the individual, security of the loan and stability of the individual.

In MPROLOG we may have:

```
eligible (name, amount, payment, income, security,
          stability) if
capable_payments (amount, payment, income) and
loan size_OK (amount, security) and
stability_OK (stability).
```

Each of the relationships to determine stability and the capability to make payments may in turn be a function of other factors or rules. We may have a query on our program of the sort:

```
?eligible(Shawn, 5000, 300, 2000, 0, none).
```

This query is asking if Shawn is eligible for a loan of $5000.00 with payments of $300.00 per month if he makes $2000.00 per month, has no security for the loan and is an unstable sort.

At this stage, we have seen very little that couldn't be easily accomplished with any other languages. Now let's try a

few other inquiries.

```
?eligible(WHO, 20000, __, __, __, __).
```

This inquiry would give us a response of anyone who is eligible for a $20000.00 loan. Another example query might be:

```
?eligible(Shawn, HOWMUCH, 300, 2000, 0, none).
```

This inquiry would tell us how much of a loan Shawn would be eligible for given that he only wanted to pay $300.00 per month and the other personal information is the same.

The important thing to recognize is that once the relationship has been defined, each of the different types of inquiries all use the same code - instead of having to write a routine to calculate the size of loans people are eligible for and a routine to list groups of people eligible for certain types of loans, or just to confirm if someone is eligible or not.

In effect, every argument in a relationship may or may not be defined. This means that achieving the same degree of flexibility in a traditional program would require 2^N routines. Obviously not every real world situation requires this degree of flexibility, but given a complex situation, the advantages become clear. In our example above, 64 separate routines or procedures would be required to accomplish the same effect as our one relationship in MPROLOG. In other words, you must think out all 64 routines.

3.2 Dealing with Complex Inter-Relationships

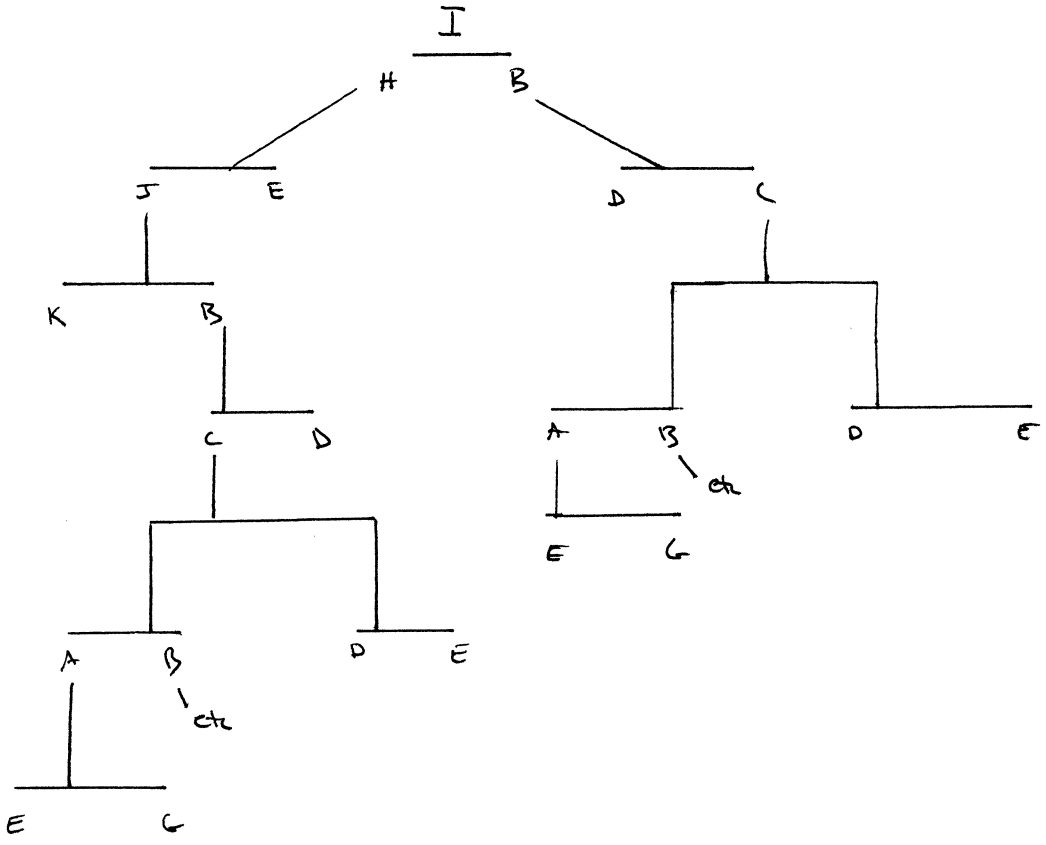
A second aspect of the strength of AI languages is the way in which we can handle a series of complex inter-relationships. Let's take a look at another example.

Consider the following relationships:

```
C is true if A is true and B is true or
C is true if D is true and E is true
A is true if E is true and G is true
B is true if C is true and D is true
J is true if B is true and K is true
H is true if J is true and E is true
I is true if H is true and B is true
```

Given that D, E and K are true, is I true?

In a traditional problem-solving style, we would work out the relationships and develop some form of "tree-like" structure such as:



We would then encode this "solution structure" into our program in a series of structured IF/THEN's, LOOPS or whatever appeared necessary to best address our problem.

Now consider the problems we may encounter if we increase the number of relationships (we only have seven listed), if a relationship needs to be removed (say the relationship of H to J and E) or we need to be able to solve to find any of the ten conditions at any time. As we can perceive, with any of these scenarios, the complexity of our program would increase dramatically and problems of maintenance would increase proportionately.

To write the same program in MPROLOG, we would have:

```
c if a and b.
c if d and e.
a if e and g.
b if c and d.
j if b and k.
h if j and e.
i if h and b.
```

Our inquiry would take the form:

```
 d.
 e.
 k.
?i.
YES
```

To add new relationships or remove them, we simply add or delete that line of code. The order of the relationships does not matter, nor which condition we are searching for. In effect, we define the problem as opposed to the solution. We let MPROLOG automatically create our "tree-structure" and work out the inter-relationships. We do not need to worry as we add and delete new relationships.

3.3 How do I know what is happening?

One of the obvious things about an AI program is that it is designed to automate an area of expertise that is not "obvious" in content, else we wouldn't call the person who does that job an expert. With this in mind and given the examples we outlined above where a small piece of code can be used 64 different ways, how does a programmer know what his or her program will do? The answer is, he doesn't.

AI programs are designed to address real-world problems with non-obvious answers, so we must expect that once an AI program reaches a certain level of complexity, your programmers will not know what the system will do. Experience with some of our

own projects has demonstrated that with as few as half a dozen relationships and a few variations of each, you will be suprised at some of the ways that your system will react.

This raises two key issues concerning AI programming. First, you ask your program "WHY?" Unlike traditional programs where you know it is doing a process or procedure because you told it exactly what to do, in our AI systems we don't know. The solution is that your expert system can be coded in a manner so as to allow you to ask WHY at any stage, and the system will outline the rules that it used to reach its conclusion.

In our Expert System Manager for instance, you may tell the system that your HP3000 won't work. It may respond with: "Check to see if the on/off switch is turned off". If you ask it "WHY?" it would respond:

Because - Rule 1: The computer won't work if there is no power, and

Rule 3: There is no power if the on/off switch is turned off.

A second benefit of this ability is that it exemplifies the value of AI as a training and educational tool. The fact is that when your company encodes the expertise of someone in an AI system, this expert system can be used both in production and in training less experienced people.

3.4 LISP versus MPROLOG

In Brant we still have "discussions" among many of our people as to the advantages and disadvantages of COBOL versus fourth-generation languages. A similar argument exists in the AI world between LISP and PROLOG users.

People wishing to develop applications in AI have three choices: one of the two base languages, LISP or PROLOG, or what are called expert system shells. Expert system shells are partial expert systems already in place, where you drop in the specific rules you want and have a working expert system in less time and at less cost for man hours than with the base language.

According to the March issue of a newsletter entitled Artificial Intelligence Markets (AIM) published in the United States, "several high end language vendors reported the return of customers who went off to buy expert system development tools to AI language products because they found the tools inadequate for their needs. Since sixty to seventy percent of high end AI language customers are Fortune 1000 firms or the federal government (as opposed to universities and AI companies), this is not a trend to sneeze at."

Accordingly, we will leave this discussion focussed on the two AI languages and leave discussions of the tools to others. In a tutorial on PROLOG presented at the International Joint Conference on Artificial Intelligence last August, Dr. William Kornfeld pointed out several "cultural" and practical differences between LISP and PROLOG, namely that LISP is all-inclusive, has complex semantics and hence a large manual that can be confusing. PROLOG strives for simplicity, doing a number of things very well, such as symbolic structure manipulation and processing facts, and consists of a few well-chosen constructs. In a practical sense, LISP has a few features PROLOG does not, like destructive assignment and structure manipulation, iteration with DO's or LOOP's and global variables. PROLOG has logical variables and efficient procedure calls that make the programs clearer and easier to understand, at the expense of some generality.

Again, according to the AIM newsletter, "We think LISP will evolve into a systems manager language - system management functions will be in LISP, and they will be layered between the operating system and the applications. PROLOG and other languages (including conventional languages) will serve as secondary, application languages which will be called from LISP."

By no means is this intended to resolve any debate, but rather it is intended to outline some of the issues and where things appear to be heading. Brant at this time is only involved in application work with MPROLOG and therefore has a built-in bias.

4.0 The Knowledge Acquisition Process

An important aspect, actually a critical aspect, of developing an expert system is the process of knowledge engineering. This is, in effect, the new name that we give to systems analysts to confuse things. A knowledge engineer plays the same role as a systems analyst, with a few important differences.

A knowledge engineer is an individual who attempts to discover the "rules" that a human expert makes to reach conclusions in problem solving. A traditional systems analyst may ask a user specific questions about an application and hope to get a reasonable answer. Knowledge engineers are warned on the other hand not to expect a reasonable answer and in fact not to take what the expert says at face value.

Research has uncovered a phenomenon known as the "Expert Paradox": the better your expert, the worse they will be at describing how they make decisions. It sounds like a joke, but unfortunately it is true.

It appears that our "experts" use what can be called "compiled expertise", where they may take dozens of specific rules or conditions and relationships which they look for in a specific instance, and compile them all down to one specific rule. It is the knowledge engineer's task to "decompile" this expertise to a form that is usable in our automated systems. The problem is that when asked, your expert will not recognize that some of his "rules" are actually compilations and so will provide the knowledge engineers with inaccurate or misleading information.

To select your knowledge engineer, look for an analyst who has shown more than a moderate level of paranoia or disbelief, is related directly or indirectly to Sherlock Holmes and is as tenacious as they come.

There are many good books describing knowledge engineering and expert systems, a few of which are listed in the bibliography of this paper.

5.0 Expert Systems

Having belaboured what AI is all about, some aspects of how it operates and some discussion of the tools we can use, it's about time we got down to why and where you use it. Why develop an expert system and how is it different?

Rather than try to provide some infallible rules of thumb, I will review briefly two projects currently underway at Brant and a few we have been discussing with clients and other prospects. I will attempt to outline why an expert system appeared necessary and to what point the project has evolved.

5.1 Personal Financial Planning System

Brant was approached in December of 1985 by one of the top financial planners in the Toronto area in conjunction with a Canadian insurance company to create a personal financial planning system. The system was intended to allow the insurance company to provide a standard high level of personal financial planning to the customers as part of their service. The software would ensure certain levels of depth and professionalism in the analysis.

After meetings with the financial planner, it was discoverd that there were two aspects to the system. First, the planner had developed a financial model that generated cash flow projections for the individual to and after retirement. The model was comprehensive and "algorithmic" in nature, so could easily be developed using a conventional language. We selected SPEEDWARE, a 4GL that would allow us to deliver the solution on either HP3000's or Vectras with no problem.

The second aspect of the system involved the manner in which the financial planner manipulated the variables in the model. This would include factors like retirement age, return on investment, risk tolerance, portfolio arrangement and much more. We discovered that the planner used such considerations as "Who appeared to be the decision maker on investments?", "Who is the primary breadwinner?", "Who appears more knowledgeable?" and a myriad of other factors in determining a family's overall risk tolerance on investments. Decisions as to whether the client would react more favourably to retiring later at a certain income level instead of retiring when he wanted to with less money, were all decisions made from a large number of what we called "soft facts".

This part of the system has been developed in the MPROLOG language, a product developed by Logicware Inc. in Toronto and distributed on the HP3000 by Brant. Like SPEEDWARE, this language is available on the HP3000 and the Vectra, allowing us to deliver to total system on both micros and 3000's. This Planning Advisor is the "expert system" that has been developed by Brant to complete our Financial Planning System.

5.2 Expert System Manager

Brant currently has eight HP3000 computers in our various offices with operators and system managers in the necessary offices. In a number of instances we have had to bring a senior systems manager to a specific project, leaving more junior support people in charge of a facility.

When certain problems arose (as they inevitably do), the operators in an office would call their systems manager who was somewhere across the country, and resolve the problem through a long-distance dialogue. After witnessing the process of question and answer strategies to resolve problems remotely, we concluded that this was an ideal example of an "expert system" with a live expert. As an internal project, we undertook the development of a limited expert system manager for the HP3000.

There is a separate paper in these proceedings that specifically addresses this project, written by Ross Hopmans and Gil Harrison of Brant Computer.

5.3 Other Expert System Projects

At this time (June, 1986), Brant is involved in a number of major conversion projects. One aspect of the application of AI that we are currently studying is its use in writing software program conversion systems.

When we convert one version of COBOL on a foreign architecture

to the HP3000, we are in effect going from one language to another, or more accurately between two versions of the same language. In so doing, we are "processing symbols" or in this case, our program syntax.

Although we have not yet implemented any conversion tools, discussions with Logicware have indicated the achievability of some distinct advantages using this approach, based on their own development on convertors between different versions of PROLOG.

Another area where a number of feasibility expert system prototypes have been developed is in the Risk Assessment area. Specifically, some of the major banks in Canada have been looking at commercial loan risk assessment systems that would allow some degree of consistency in the processing of loan applications. The prototype captures many of the factors that the banks' top loans evaluators use in assessing an applicant, including management competence, business environment and marketing ability, as well as standard items like the balance sheet. For large commercial loans in excess of $5 million, this process also ensures that the branch will address all issues prior to referring the application to head office.

There are hundreds of existing applications already in use and, as we are all aware, substantial amounts of effort going on behind the scenes. With all the information that has been published, I will not prolong the discussion on this point.

6.0 The AI Marketplace

Of interest to all of us must be the rate at which this technology is impacting the marketplace and the "real world". The two major indicators that are available from various market research studies on AI are the dollar volumes of products on an annual basis, and the number of "units" that are delivered, given that the unit price of AI products will be dramatically declining much as is true for micro-electronics in general.

In many areas, we can expect that "intelligence" will be built into other products and will not be covered by the studies outlined herein. As an example, if intelligence is built into an electrocardiogram, it would not be included in the AI marketplace, but rather is a byproduct. The magnitude of this secondary market is difficult to discern at this time.

According to a report by Frost & Sullivan printed in the January, 1985 issue of Computing Canada, entitled "Artificial Intelligence Products", the hardware, software and services marketplace for Artificial Intelligence nearly doubled from $181 million in 1984 to $342 million in 1985. It is expected that it will double again by 1987 to $665 million and will

reach a total of $1.6 billion by the end of the decade.

Of this $1.6 billion, 50% will be software products, 30% hardware and the balance services. The software component will be comprised of 20% natural language software that will be popular on mainframes and personal computers, due to the higher proportion of novice users. Expert systems are expected to comprise 35% of the total software sales, 31% will be AI languages and the balance will be AI applications.

More important than the growth in dollars of the marketplace is the fact that as hardware prices continue to drop, more units can be expected to be delivered, leading to a projection that "the AI market will multiply by a factor of 40 between 1984 and 1989".

According to a projection on AI languages specifically by Artificial Intelligence Markets, between 1985 and 1990 we can anticipate a growth in sales from $32 million currently to $226 million in 1989. This agrees relatively closely with the Frost & Sullivan study (see Table 1). During the same period, the number of shipments of AI language packages is expected to grow from 5,000 in 1985 to 50,000 per year by 1990.

Frost & Sullivan

	1985	1990	
Expert Systems		280	
Natural Languages		160	
AI Languages		248	
Applications		112	
Total Software	137	800	484%
Hardware	85	480	464%
Services	120	320	167%
Total AI	342	1600	368%

Table 1: Growth in mil's of the AI Marketplace, 1985 to 1990

7.0 HP and the AI Way

In this paper for the Hewlett-Packard users group, we would be remiss if we did not do a quick review of what is happening in the HP world in particular.

To begin with, let me say that HP probably made the most impressive showing in last year's International Joint Conference on Artifical Intellegence held in Los Angeles. From a totally non-participatory role the previous year, HP became one of the top two or three vendors with their new HP9000 Series 300 AI development environment.

This hardware and software environment, running under HP-UX, amazed many attendees at the conference with its quality, thoroughness and presentation. HP released the new 9000 system with a full LISP development environment complete with expert system shells, tools and more. By any standards the offering was impressive and HP's commitment to the AI field notable. Word is that a PROLOG language will be implemented in conjunction with a Third Party from Switzerland some time in the next year.

On the HP3000 there has been a noticeable lack of attention to the AI field, primarily because most people saw the potential for AI in the engineering and technical environment as opposed to the commercial world. For the past few years, the only AI tool available was a version of LISP from Robelle in British Columbia that provided a learning tool, but which was not robust enough for a proper production environment. I strongly recommend that anyone interested contact the people at Robelle.

Because of the lack of tools and rumblings of interest from commercial users, Brant entered into an agreement with Logicware Inc. to port their MPROLOG development environment to the HP3000. At the time of this paper, the porting project has been underway for two months, and the full interpreter is expected to be available by the time of the IUG. Because MPROLOG runs on the Vectra, application code will be transportable between IBM-PCs and the HP3000. Brant has been able to commence work on our expert systems prior to having the interpreter on the HP3000. MPROLOG will be written in "C" on the HP3000.

Of specific interest to Brant has been the role of AI on Spectrum, and although definitive statements have not been forthcoming from HP, the rumour mill is churning up good things. In the July issue of High Technology magazine, Ira Goldstein, director of the Distributed Computing Center at HP's labs in Palo Alto, was quoted as saying: "You want machines that can do symbolic computing, but not at the expense of conventional computing...and that's where HP's forthcoming Spectrum line is a step in the right direction.

The new computers will have a large address space and a key requirement for knowledge-intensive AI programs as well as a large number of storage registers, which can hold functions commonly used by the LISP language. Perhaps most important, the Spectrum computers are designed to support coprocessors of different types. Many observers believe that the hybrid machine of the future will have general purpose and LISP microprocessors working together to run (respectfully) the numeric and symbolic portions of mixed applications!"

Hardly a lot to go on, but based on the smiles of many of the people in the HP labs we feel optimistic that HP's new precision architecture will lead the way for those interested on this new technology.

8.0 Conclusion

Its hard to apply a moral to this story, but I do hope this paper has been successful in providing those people who have had their interest in AI tweaked with a little more insight into some of the issues in this area.

This paper has been geared for those new to the field and thus will not address some major areas in the detail. For others with more specific interests or inquires, we at Brant would be more than happy to talk with you.