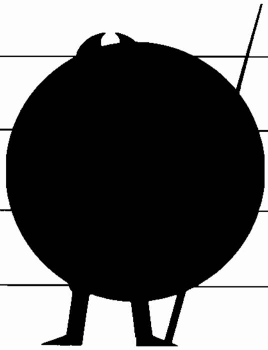
PROCEEDINGS

Interex HP 3000 Madrid Conference '86

Madrid, Spain, March 10 to 14, 1986
Hotel Meliá Castilla

PROCEEDINGS

Interex HP 3000 Madrid Conference '86



Madrid, Spain, March 10 to 14, 1986
Hotel Meliá Castilla

COMMITTEES

Organizing Committee

Jesús Parada	- Chairman
Marta Jáudenes	- Secretary
Eduardo López-Lozano	- Treasurer
Antonio López-Maya	- Public Relations

Technical Program Committee

Jesús García Catalán
Jesús Parada
Luis Baroja

Vendor Shows Committee

Jelle Grim	
Tim cullis	
Ma. Antonia Marín	- System Manager

Professional Congress Organizer

Ana María Eichler	- Creatur

VIAJES CREATUR
Profesor Waksman,11
28036 MADRID

SPANUG
Génova 17
28004 MADRID

CATEGORIZED INDEX OF SPEAKERS

AL - Advanced Languages and Development Tools

SD - System Development

SM - System Management

OP - Operation Management

VS - Various

ADDITIONAL HEWLETT-PACKARD PAPERS

ALPHABETICAL INDEX OF SPEAKERS

Interconnecting Heterogeneous Computer Systems

Hassan Alam
Network Specialist
International Region

Table of Contents

Section 1
Overview

Section 2
Open Systems Interconnect Model

Section 3
Advantage of Layering

Section 4
International Standards

Section 5
Picking A Standard

Section 6
Designing a System

Table of Contents

Section 7
Summary

Overview

SECTION 1

1.1 Abstract

The rise in mini and micro computers led to proliferation of computer systems. With these inexpensive computer systems each work group, and often person, could afford his own computational facility. This proliferation of computer systems also increased the need for exchange of data between computers, and the sharing of expensive resources. To address this need computer vendors produced their own proprietary computer networking capability. Examples of this are HP's DS network which pioneered distributed processing, and IBM's Binary Synchronous network, which was primarily hierarchical.

This approach worked well up to a point. It, however, had three limitations:

1) Proprietary networks did not communicate to other proprietary networks. Thus if users bought computers from more than one vendor, inter-vendor communication was difficult at best.

2) Once the network was implemented, it was expensive to rewire it as the computational need of the user changed.

3) The proprietary nature of the networks precluded incorporation of new networking technology.

To ameliorate this problem HP has adopted the International Standards Organization (ISO) Open Systems Interconnect (OSI) model which allowed customers to get the benefit from their network. In contrast IBM adopted its System Network Architecture (SNA), a closed architecture which obsoleted its older BSC network.

This paper outlines a method of designing computer networks based on the ISO OSI model. It explains the ISO model and develops criteria for selecting different hardware and software components from the model to implement an efficient, flexible, and cost effective network based on computational and data traffic needs. The paper shows how a network designed with such principles can allow heterogeneous communications, changes in topology, and upgrading to newer technologies.

1.2 Organization of Paper

This paper addesses the need for developing networks which provide the benefits stated above. Section one briefly explains the International Standard Organization Open Systems Interconnect model. Section two examines the architecture of the OSI model, and the benefits of layering in providing modularity, and expandability. In section three the advantages of using standard communication protocols are explained, and this is expanded in section four which develops a model for selecting different communications protocols. Sections five and six sum up with a case study of networking a multinational company, and a summary of the paper.

Open Systems Interconnect Model

SECTION 2

With the advent of mini and personal computers, came lower prices and the proliferation of computers. These computers were distributed in many sites, and a need arose to connect the computers together. The International Standards Organization (ISO) developed the Open Systems Interconnect (OSI) model for connecting computers to provide four services. They are:

- Share information. Computer networks allows companies to share and distribute information over the entire company, even if the company is geographically dispersed

- High Reliability. Computer networks allow higher reliability by providing back up data and processing power. If a disc drive or processor fails others are available to allow computation.

- Load Sharing. Computer networks allow computation to occur at the site of the data, thus allowing only pertinent transactions to be transmitted, and lowering communications cost.

- Communications Network. Computer networks allow electronic mail between members of the company.

This section examines the International System Organization (ISO) Open Systems Interconnect (OSI) model for implementing computer networks which provide the above mentioned functionality. The objectives of the layered model, and the functions of each layer are explained.

2.1 OSI Model

The ISO committee designed the OSI model in seven layers. With each layer the committee intended :

- Clearly defined layers.

- Each layer has a specific distinct task.

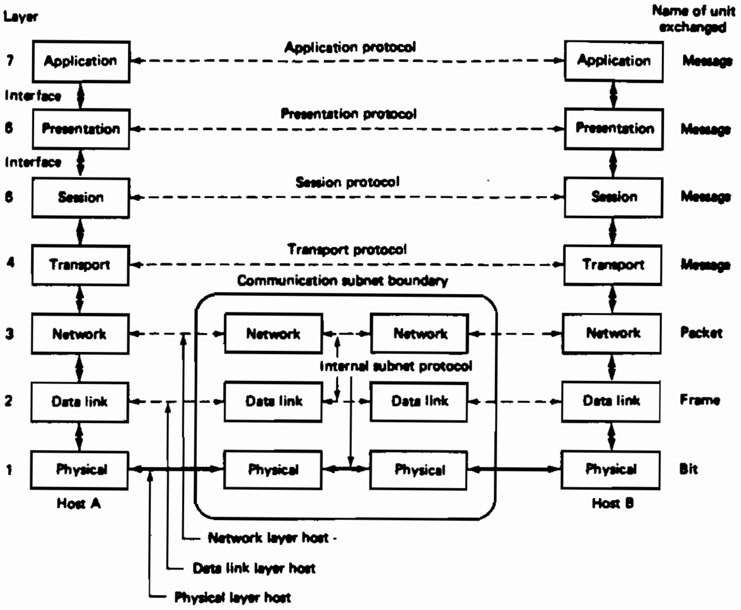
- Modularity in decomposing the network.

The OSI model above has seven layers. Each layer communicates with the layer above and below The functions of the layers are:

- Layer 1: The Physical Layer. The bottom layer is the physical layer which transmits raw bits over a communications channel. The layer defines the medium, the voltage, the length of electrical signal. The layer defines mechanical, electrical, and procedural interfaces.

- Layer 2: The Data Link Layer. The data link interprets the signals transmitted on the physical link, and ensures accurate transmission and receipt of data through error correction or retransmission. For instance a noise burst can destroy data. The data link protocol requests retransmission and acknowledges the data received.

Open System Interconnect Model



OSI Model

- Layer 3: The Network Layer. The network layer controls the routing of data. It accepts the data from a source converts them into packets and directs them to the right destination.

- Layer 4: The Transport Layer. The transport layer accepts data from the session layer and ensures the data arrives at the destination in the correct sequence. This is specially important in a host computer which establishes multiple sessions with other computers.

- Layer 5: The Session Layer. The session layer provides the user an interface into the network. It establishes a connection between a user on one computer with a user or service on another computer. This connection is known as the session. The session layer handles authentication, and communication protocols employed in the lower layers.

- Layer 6: Presentation Layer. The presentation layer presents the data transfered by the network in an understandable format. This is specially important when the two computers involved employ different internal data representations (eg 7 bit and 8 bit characters). The layers also handle data compression and encryption for speed and security.

- Layer 7: The Application Layer The application layer defines the topmost layer of the OSI model. This layer is defined by the users of the computers. The actual format of the layers depend on the application of the network, and the people implementing them. For instance, electronic mail may use message passing, while distributed data bases may use remote data base access, and distributed processing may use inter-process communication.

This layering allows building of modular networks, which the user can modify easily to adapt to new computational needs and technology.

Advantage of Layering

SECTION 3

3.1 Modularity

Since the layers in the OSI model pass well defined messages to the ones above and below, the OSI model allows for modular decomposition of computer network commnunications. This modular decomposability allows for individual, or group, of layers to be replaced easily by other layers providing the same functionality of the layers replaced. This ability to replace layers makes changing of network topology, communications protocol, and technology with minimum effect to other parts of the network and the computer system. This section describes how modification of networks can be effected.

3.2 Changing Your Network

Firstly, modularity is useful in changing computer networks. For instance, suppose a user connects two computer system using point to point synchronous connection on HDLC protocol. This line communicates at 56KBPS, and adequately supports the communications need for the customer. As the users computational needs grow, he purchases another computer and connects this to one of the other computers This creates a problem since the user has to use one of his computers as an intermediate node between the other two. This puts an extra communication burden on the intermediate computer. This is further complicated because the users' communication needs increase.

To resolve this problem, the user switches from a point to point network to a 802.3 local area network. With this the user replaces the binary sychronous cable with a coaxial cable, and the HDLC protocol with the 802.3 protocol.

3.3 Expanding the Network

Let us examine the case if the customer needs to expand his network. Suppose the user wishes to add a computer to another site at a different location. The user then needs to exchange data between his current computer facility and the new computer facility. The user can designate a computer at each site to act as a communications gateway and use an dial up line to connect the two computer sites on an "as required" basis.

To connect these two sites the user adds lower level HDLC communication protocol over a twisted pair. This protocol works in conjunction with the existing upper layers or networking software, allowing the user to utilize existing network software over both the local area and the wide area network. In this manner, from the user's point of view, the local and the wide area computers reside on the same network, the only difference being the speed of the line.

3.4 Adopting New Technology

While the above examples illustrate addition, modification and substitution of the lower layers of the network, network users can modify other layer with a similar approach. The user can expand this approach to encompass new network technologies. For instance if the computer user decides to adopt a mail network based on X.400, he can simply pass messages through the existing lower six layers to implement a new mail system. Similarly if the user increases his data traffic between his remote computer sites, and a new technology, such as optical fiber, offers him a higher data throughput, he can replace his twisted pair copper line with an optical fiber line.

3.5 Advantages of Layering

Thus a layered networking protocol provides a number of advantages:

1) Easy modification of network to suit actual data communications needs. Since the network is layered, each layer can be replaced without major impact to the other layers.

2) Easy addition to exiting network to expand coverage. Additional links or services can be added to the exiting network, thus providing more functionality when it is needed.

3) Easy incorporation of new networking technology. As new software and hardware technologies become available, they can be added to the network without losing the existing investment in the network.

4) Protection of investment. Because the network can be modified incrementally, existing investment in software and hardware need not be scrapped to either change the network, or adopt newer technology. This reduction of switching cost ensures networks can be cost effectively developed for customer needs.

International Standards

SECTION 4

Once the user adopts the OSI layered communication protocol for networking, the user needs to decide which specific protocol to implement. Many vendors offer their own protocol; some of the vendors implement an OSI like layered communication protocol for their network. This approach allows users to connect, to a large degree, computers manufactured by the same computer vendor. However, many users apply computers manufactured by different computer vendor for different task. If the user should wish to connect these computers from multiple vendors in a homogeneous information management system, he will often find the different vendors do not provide the ability to connect to computers manufactured by another vendor. The user then is left to his own resources to pass information between these various computers.

4.1 Advantage of Standards

Adopting networking standards relieves this problem in three way, firstly, through a standard set of protocols, computers manufactured by different vendors can communicate with each other. Secondly, economies of scale ensure cheaper and more reliable networks, and thirdly, international standards tend to exist longer than single vendor protocol, and thereby offer protection to the users investment.

4.2 Mulitivendor Connectivity

Adoption of international standards for all communication provides the user with one immediate benefit - computers purchased from different vendors can communicate using the same protocol. While a vendor proprietary protocol may allow more efficient communication between some of the machines of a given vendor, communications between different vendors, and even different product lines of the same vendor (IBM series 36 and Series 38) are more difficult. Adoption of internationally agreed upon protocol standards (such as GM's MAP) allow users with more connectivity between heterogeneous computers.

4.3 Product Longevity

Another benefit of adopting standard communication protocol is that, the communication protocol tends to exist longer than a protocol supported by one vendor. This is because typically many vendors will support an international standard, and products communicating via the protocol will exist for a time longer than products supported by a single vendor. This is important from a user's financial point of view, as he can purchase communication products to supporting his existing application programs for a longer time than application products using vendor proprietary products.

4.4 Lower Cost

Because multiple vendors support international standards, products supporting these standard protocols are developed by a relatively large group of companies. This results in competition between the suppliers and a lower cost product for the user. Also, because standard communication protocols are supported by multiple vendors, the number of products available increases, the number of units in use increases, resulting in more variety and lower cost due to economies of scale.

4.5 Tradeoff for Standards

While following a standard provides many benefits for the computer user, typically the standard lag the leading edge technology by a few years. This implies that customers needing to use the latest technology cannot be guaranteed of an international standard with the above mentioned benefits. For instance, fiber optic communication has no established protocol which looks like becoming an international standard.

Similarly, if a user wishes to provide the fastest communication between two specific systems, he may wish to bypass a standard protocol and develop a protocol which he can fine tune to provide the best performance for his specific application.

The user has to evaluate whether these advantages outweigh the benefits of standardization and low cost. In systems like real time fire control on military aircraft, the user may indeed decide a high performance proprietary protocol provides the technological edge to ensure survival of the aircraft. However, the user should realize, that developing proprietary protocol will be more expensive and more time consuming than using standard protocol. In the majority of the cases the user will opt for standard protocols.

Picking A Standard

SECTION 5

Once the user commits to the OSI model for his computer communication, the question of what standards to use for his application arises. HP aggregates the OSI model into three layers, and picks standard based on these three layer. This makes choosing standards much simpler with three layers, and the network does not lose much flexibility.

Starting from the bottom the layers are the link, incorporating layers 1 and 2 of the OSI model, the transport, incorporating layers three and four of the OSI model, and the application incorporating layers 5 through 7 of the OSI model.

5.1 Linking the Systems

To establish the computer network, the user must first connect the two systems together. This is the link between the computer systems. It incorporates the lower two level of the OSI model. The user needs to ask himself a number of questions when deciding the type of link over which he will connect his computers.

1) Where does he plan to place his computers. The geography of his computer network determines some of the options he has for connecting his computers. If all his computers are in the same building within 1.5 KM of each other, he can use IEEE 802.3 local area network. If they are in separate nearby building he can use IEEE 802.4 broadband LAN. The user can also use point to point connections, phone switches, and X.25 based data switches for local connections. If distances are larger telephone lines, leased lines, public X.25 networks, and satellites may provide the appropriate connections.

2) What are the performance requirements of the network. In other words what applications does the user expect to run over the network. If the applications require high data transfer, LANS would be appropriate for local connection, and fiber optic and satellite for remote connection. If the user expects rapid response, a satellite connection would cause too much delay.

3) How many systems does the user intend to connect in the network. If the user intends to connect few systems, point to point connections may be the most effective method of connecting the systems. If he wishes to connect many systems, LANs, X.25 switches, and PBX switches (for local connections) may be necessary. LANs, X.25, and PBX switches allow multiple systems to coexist on the same network.

4) Price. How much is the user willing to pay for the network. The higher the performance and functionality, typically, the higher the price.

The user needs to carefully analyze his current and future data communications requirements to develop cost effective connections between his computer systems using the above mentioned guidelines.

5.2 Transporting the Data

After the user establishes a connection between his computer systems, he needs to transport his data between the systems so that the receiving computer can meaningfully interpret the data sent by the transmitting computer.

Luckily the choices here are less. The user needs to determine reliability of the link he has chosen and adopt a transport protocol which provides the most throughput. For instance an 802.3 LAN provides more reliable data than a disc drive. Unfortunately there is only one major standard available today - Transport Control Protocol/Internet Protocol (TCP/IP). TCP/IP was designed for relatively unreliable communications links, and thus provides much higher error checking and correction than is necessary with some of the modern links.

However, many vendors have adopted TCP/IP and it is key in connecting multi-vendor computer. Thus till a better standard is designed by ISO (eg TP4), TCP/IP is the transport protocol of choice.

5.3 Network Applications

After the user develops a method for transferring data reliably from one computer to another, the next task is to pick a set of applications which provide high degree of functionality. This layer covers layers 5, 6 and 7 of the OSI model. This layer provides the functionality of communicating between the computer systems. Functions provided include:

- Inter Process Communication
- Network File Transfer
- Remote Terminal Capability
- Remote Process Management
- Remote Data Base Access
- Network Management

Many implementations of the application layer offer variations of these services. This layer is offered both in vendor developed packages, and in packages based on internationally developed standards. In picking the set of protocols in this layer the user has to keep two thing is mind:

1) Are the protocol based on ISO. That is can the protocols communicate using layers 1-4 of standard international protocols such as TCP/IP X.25, 802.3 etc. If this is not true then the protocols cannot provide communications between multiple vendors. If the protocol is based on international standard, the user can either buy implementations of the protocols on different vendors or, in the case of computer vendor developed protocols, implement these protocols on machines of another vendor.

2) Do these protocols allow modular decomposition. In other works can these protocols allow applications developed on one machine to be distributed over multiple machines. Global address spaces and interprocess communication protocols allow for this expansion capability. Inter-process communication tends to be easier to implement than global address spaces. With

IPC, the user can develop his application program in modules. As the user needs more computation power, he can easily move modules to other computers on the same network, and thereby increase his computational capability by utilizing multiple processors.

A number of international protocols are being developed for different application. They include the MAP protocol for manufacturing environment, the TOP protocol for the engineering environment, and the X. 400 protocol for the electronic mail environment. In addition vendors such as HP, DEC have developed application protocols based on the OSI model.

Designing a System

SECTION 6

The previous sections described the OSI model, the advantanges of designing a network based on this model, and some guidelines for selecting layers of these models. This section integrates the previous sections by applying the principles developed before to a hypothetical company which needs distributed computation. The section describes the company's computational needs, its geography, and then designs a network to serve the company

6.1 Company Computing Need

Let us assume we are dealing with a company named ABC. ABC designs, manufactures and sells state-of-the-art widgits. To operate efficiently ABC divides its operations into four functional area, Engineering, Manufacturing, Marketing and Sales, and Administration. Below are described the operations and computational needs of each functional area.

- Engineering. The engineering organization designs the widgits with powerful Computer Aided Design (CAD) workstations. These engineers need to share design between themselves to efficiently design the widgits. They also need access to expensive plotters and disc drives to plot and store the designs. The engineers also need input from the marketing organization for new designs. They also would like to send the designs to the manufacturing operation quickly so that designs can be manufactured and sold before the competition's widgits.

- Manufacturing. To keep costs down ABC company manufactures its widgits in a totally automated factory. The orders are received from the sales force to a materials management computer. This computer automatically plans production based on demand, and orders raw materials and manages inventory. The orders are passed on to factory floor computers which run the production process. Typically one production line is operated by multiple computers. These computers build products based on designs sent to them by the CAD computers in the engineering operation.

- Marketing and Sales. Marketing and sales are two different functions in ABC's marketing organization. The marketing organization performs traditional marketing function of merchandising, forecasting, and product management. To accomplish this efficiently ABC has provided its marketing staff with personal computers which support forecasting, data base management, word processing and graphics. The marketing staff shares common data among its staff. The sales organization was provided with portable computers. The sales force determines price quotes with these computers and enters orders to the factory by these computers.

- Administration ABC employs a relatively small administrative staff. Their main functions consists of accounting, personnel, and report generation. They use a minicomputer, and need data from all other operations on revenues, expenses, personnel, and capital outlays.

- Network File Transfer. This function is needed to transfer files and share information between users on different computers.
- Remote Data Base Access. This function provides access to data on remote computers.
- Virtual Terminal. This allows users of one system to initiate sessions on a remote computer.

6.3.2 Transport Protocol

The standard industry transport protocol is TCP/IP. Company ABC picks TCP/IP since this allows them the most flexibility in choosing the upper level application software, and lower level network link.

6.3.3 Link Technology

ABC needs a number of communication links. Each of these links is for a different communication need.

- Broadband LAN. The manufacturing environment needs a deterministic real time network which allows high speed communication between production line computers. for this purpose a broad band 802.4 LAN is appropriate. This LAN is extended to connect the manufacturing planning computer to the production computers. The 802.4 LAN also reaches the next building where the CAD computer reside. It connects to the engineering computer network via a gateway.
- Baseband LAN. For the engineering CAD stations a 802.3 baseband LAN provides the same high performance at a lower price than the broadband LAN. The workstations connect over this 10 MBPS LAN to each other and a dedicated server which supports high speed disc drives, printers, and plotters.
- Work Group Thin Lan. For the marketing organization, a local high speed network is necessary. The network implemented by ABC supports workgroups of four to six people. The network uses a small minicomputer which double as a file, print, and plot server. Eighty per cent of group communication is among themselves. Twenty per cent is to other group. To accomplish this ABC employs two LANs. The first is for the group. It is a thin 802.3 based 10 MBPS coaxial cable to connect the workgroup to each other. Each work group LAN in turn connects to a campus wide thick 802.3 coaxial cable. This backbone coaxial cable allows the different groups to communicate among themselves.
- PBX. The minicomputer supporting the administrative staff also connects to the backbone computer. The administrative group supports a large number of terminals. These terminals connect to the mini computer via a telephone switch. The switch allows support of a large number of terminal which all do not need to connect to the CPU at the same time.
- X.25. The sales offices around the continent are connected to head quarters via a public X.25 network. The X.25 network allows flexible configuration, and a demand based network. ABC is charged for actual data transfer between its remote computer systems.
- Point to Point Connection. The two main sites of ABC (marketing & administration and engineering & manufacturing) transmit large amounts of data to each other. For this purpose, ABC uses a direct telephone line.

6.2 Company Geography

ABC is arranged in three groups. The Engineering organization and manufacturing organization are situated in adjacent buildings in one locations. Marketing and administration are located in another two adjacent buildings in another city, and the sales offices are spread all over the continent.

- Engineering and Manufacturing. ABC's engineering and manufacturing facilities are located in an adjacent building. The engineers each have design workstations, and share a number of high speed disk drives, printers, and plotters. The manufacturing floor is organized around production lines. Each line is controlled by computers operating a number of PLCs. The production line computers themselves are scheduled by a production planning computer.

- Marketing and Administration.. The Marketing and administration groups are situated in one building. The marketing organization is grouped into 4 to 6 man units. Each person in the unit works on PC and needs to share his information with others in his workgroup. Occasionally a person in one group needs to share information with people in other groups. The administrative organization employs a central computer to manage corporate wide data bases. The people in administrative organization use intelligent terminals to establish sessions on the computer. The computer is maintained in a central computer room.

- Sales Office. ABCs sales offices are scattered all over the continent. Each sales office has a small computer with a group of PCs. The main function of the computers is to generate quotes, log orders, and keep track of customers. The sales office receives new parts lists every week, and transmits orders every hour on an as-needed basis.

6.3 Picking The Network

Given ABC's computational needs, communications traffic, and geography, the next task is to pick the application software, the transport protocol, the connection technology. The application software is determined by the functionality required from the network, the transport is determined by performance and error recovery concerns, and link technology is determined by the geography of the systems.

6.3.1 Application Software

ABC needs to perform a number of functions over the network. In the engineering environment it needs to pass files between the engineers and access data bases remotely. In the manufacturing environment the different computers need to pass information to each other in real time to inform the other computers on process control. In the marketing organization, the PC users need terminal access to the mini computers, and file transfer capability. The sales office needs batch file transfer capability, to HQ to receive updates and send orders. All functional groups need a mail capability to pass messages. Below are a list of standard protocols and services ABC implemented on its network.

- MAP. For the manufacturing operation the protocol to follow the Manufacturing Automation Protocol (MAP) being developed. This protocol is being developed for real time computer networks on the factory floor. This provides remote terminal and file transfer capability.

- X.400. For the corporate wide mail network, X.400 is the networking protocol being developed by the CCITT for electronic mail. This packages the messages from any computer and routes it to another computer.

Summary

SECTION 7

From the study of ABC corporation we saw how a company wide network can be designed using the ISO OSI model. The different parts of the organization need different communications capability. This is done by using the layered architecture and employing a multitude networking products based on the OSI model. This allows ABC to:

- Share information between users.
- Provide higher reliability of information available.
- Share information between different computers.
- Communicate via electronic mail.

The OSI structure ensures the ABC has the capability to:

- Connect computers from multiple vendors.
- Design a flexible network with multiple technologies.
- Upgrade and modify the network easily.
- Protect ABCs investment in the computer network.

7.1 Biography

The author, Hassan Alam, has been with HP for two years. He is a network specialist for HP's international region. He has had work experience designing WANG's I/O system, and managing ROLM's military computer products. Hassan Alam holds a Bachelor's degree in computer science, a Bachelor's degree in Electrical Engineering from the Massachusetts Institute of Technology, and an MBA from Stanford University.

X.25: WHAT TO DO AFTER THE NETWORK IS IN PLACE

Stephen J. Coya
MCI Digital Information Services, Washington, D.C., USA

Summary

Many organizations are discovering the advantages of using X.25 packet switching networks to support their data communication requirements. However, it's not until the network is installed that the real "fun" begins: connecting all the hosts and getting them to talk to each other. This is especially true when hosts from different vendors are to be connected to the network. While the OSI model provides a good theoretical framework, many vendors have implemented this model in different ways. As a result, diverse, and sometimes incompatible, protocols must be accommodated and coerced into cooperation before they can communicate.

This paper will review the problems (and solutions) encountered connecting HP3000s, HP1000s, and DEC VAX 11/785s together over a nationwide private X.25 network. It will touch on the physical connections to the packet switch (OSI Level 1), a special Transport Layer protocol that had to be developed (OSI Level 4), and some of the problems encountered with DSN/DS (combination of OSI Levels 3 and 4) and what we had to do to make it all work.

Introduction

We knew from the start it would be interesting. Many companies have made the transition to X.25 networks with relative ease, but this can often be traced to systems where only one vendor host is involved. For example, a network consisting of HP3000s and the selection of a network switch vendor that supports such an environment. The MCI Mail system, an electronic mail system with hardcopy support, was designed around the use of VAX hosts for the mail application and HP3000s to process and print hardcopy traffic on 2680A Laser printers, and we knew the hosts would be connected to the network switches (PSNs) at 56 kilobits per second (kbps). We drew network (cloud) and system diagrams, examined technical specifications for the hosts and the switches, realized what could and could not work, and reached for the bottle of aspirin!

I should mention that there were time constraints involved that did not permit extensive experimentation or research, and the capabilities that exist on today's equipment did not exist three years ago. The system, designed in the latter part of 1982 was built in six months during 1983. Even after details were worked out in joint design meetings, special plugs and cables had to be developed on the fly as we tried to meet our target date, which we did.

Transport Layer (OSI Level 4)

The first "Gotcha" was getting information from the VAX hosts to the HP3000s. All VAX hosts were located in a central facility while the HP3000 based print sites were set up across the country. Tape transfers were out of the question, and the old standby, RJE, did not fit the overall system design. When DSN/DS was initially developed, it was for point to point communications between HP equipment. It was expanded to support X.25 interfaces, but still required the use of DS as a transport layer protocol. So one still had to use DS to transfer files to an HP3000. Well, DS was never implemented on VAX equipment, and we needed something that would talk straight X.25 with the VAXen and DS with the HP3000s. Enter the HP1000.

The HP1000 was inserted into the network diagrams between the VAX hosts and the HP3000s (we also started using pencils instead of pens to draw the diagrams). We used two LAPB modem cards and two physical connections to the packet network. One modem card was set up to use straight X.25 (DSN/X.25) to communicate with the VAX hosts. The second card was configured to use DS (DSN/1000-IV) to communicate with the HP3000s. Simple? Not quite. While the HP1000 could use DS to communicate with the HP3000, there was no transport layer protocol to communicate with the VAX.

The solution was to design a Simple File Transfer Protocol (SFTP) to accommodate the need for a transport layer. This protocol was implemented on the VAX equipment and on the HP1000s. As its name implies, it is a simple protocol: no CRCs, just simple checksumming and byte counts. It was also implemented on an IBM 4341 which was connected to the network and served the application as the accounting and invoicing host.

So, we had finally identified the host equipment that would be connected to the packet network and had designed a protocol that would provide the OSI level 4 Transport Level between the VAX and HP1000 equipment. Back we go to the basics, OSI Level 1, and connect the equipment to the packet switches.

Physical Layer (OSI Level 1)

OSI Level 1 pertains to physical connections, which pins are used for what signals, and the electrical levels of those signals. Unfortunately, there are a number of different standards that have been recommended and implemented: RS232c, RS449/RS422, and V.35 were the ones we had to work with (See figure on last page).

The packet switch nodes (PSN) supported RS449 physical connections. The switches did not supply a clocking signal, and the hosts were to be connected directly to the PSNs (no modems) at 56 kbps. I mention clocking because at that time neither the HP1000, HP3000, or the VAX could supply clocking at 56 kbps (if they could we didn't know about it). So we had to connect three different hosts that support three different interfaces to a packet switch that supported only one of the three.

The HP1000 was the easiest as it uses RS449 as a physical interface. All we had to do was insert a RS449 Synchronous Modem Eliminator (SME) between the HP1000 and the PSN to provide clocking at 56 kbps. Since then, we have been successful in operating without an SME as the HP1000 will now supply the clocking signal.

The HP3000, however, uses V.35 as its physical interface. To establish this connection, a V.35 cable from the INP is used which terminates in a Winchester type connector. The next link is a cable which terminates on one end with a Winchester connector (connected to the HP3000 INP cable), and on the other end with a DB-37 connector. The DB-37 connector is wired to conform to the RS449 specification, the electrical characteristics of which conform to RS422 (RS422 is the electrical specification for signals carried on an RS449 interface). The DB-37 connector is then plugged into a SME. From the SME comes a cable to a special V.35/RS422 convertor that is plugged directly into the HP3000's port on the PSN. Two down, one to go.

The VAX supports RS232 connections, which is rated at 19.2 kbps up to 75 feet. However, the interface from the VAX to the packet switch goes through a KMS11 processor which has a rated speed of 56 kbps. A shielded cable is used to connect the KMS11 to a RS232/RS449 SME which is then connected to the packet switch.

So we now have all the hosts connected to the network, transport layers are in place, and we can transfer files where they need to go. All done? Not yet - the system must be operated and strange things can happen!

To Flush or not

Our network supports data packets of up to 1024 bytes in length. When data is to be transmitted from the HP3000 over the packet network, the DS protocol sets up a buffer containing four 1024 byte packets. The packets are sent one at a time to the local packet switch which acknowledges receipt of each packet. After the fourth packet is transmitted and the local PSN acknowledges receipt, the buffer is flushed and the next group of 1024 byte packets are placed in the buffer. This looks acceptable on paper, but in practice caused some problems.

As the packets traveled through the network on their way to the destination PSN (the switch connected to the host to receive and process the data from the HP3000; in this case, another HP3000), a network problem caused a RESET to be generated by a packet switch. This RESET packet is returned through the network to the originating HP3000, essentially stating that something bad happened and requesting that the packet be retransmitted. If the reset is received by the HP3000 before the fourth packet has been acknowledged, the packet is located in the buffer and retransmitted.

However, if the reset is received after the fourth packet has been acknowledged, the buffer has been flushed and the packet is not available to be retransmitted. When this happens, DS just sits there wondering what to do. Recovery from this state required bringing down the DS connection, bringing it back up, and starting the transmission all over again.

The problem was reported to HP, and our SE captured numerous dumps to substantiate our claim, especially after HP informed us that 1) they couldn't reproduce it and 2) the local PSN had acknowledged receipt of the fourth packet, so it was a network problem, not an HP problem. That they couldn't reproduce it was understandable; they'd need to replicate our system switches, configuration, traffic patterns, line speeds, etc. Their second claim was a little hard to live with as it wasn't a "network" problem, but a SYSTEM problem. HP's position was understandable albeit cavalier. By the way, the occurrence of a RESET packet is part of the CCITT X.25 recommendations.

The solution? You won't believe it. We received a DS patch from HP that essentially said "If network=01 (BBN), don't flush the buffer until acknowledgment received from destination PSN."

Error Checking and DS

As I mentioned earlier, DS was written originally as a point-to-point transport agent for the HP3000s. It was later modified to support X.25 connections as a short term solution until a better defined interface was developed. It should be noted that error checking in DS is only done at the packet level (not message level) with the local PSN.

Lets step back a moment and look at what a packet switch is: a computer. It's a special purpose computer, but a computer none the less. And what makes a computer work? Software. A packet switch contains an operating system, tables, microcode, routing algorithms, etc. Guess what happened when we upgraded the switch hardware and installed a new version of the PSN software designed to run only on the upgraded software.

The new equipment and software was tested for some time in a test mini-network we maintain that replicates our operational system, and we verified its functionality, new capabilities, and features. Unfortunately, there's one thing that cannot be tested in our mini-network: LOAD. The new hardware was installed throughout our network and the new software was propagated over some period of time. Two weeks after full propagation we got burned!

There was an obscure bug in the PSN software release that caused an "overlay" of data, overwriting the data contents of a packet in the memory of the PSN. The PSN then calculated the checksum and sent it on through the network. The "bad" data was not noticed by the HP3000 when received, since error checking is done on a packet level and the checksum for the packet was ok (as it was not calculated until after the overlay), and it was accepted. The "clobbered" message was printed and no error was encountered...but there was an error in the message, one that got through the network unnoticed by any of the software!

The PSN software was backed out and is only now being repropagated as the cause of the problem has been identified and corrected. I mention this here because there IS a deficiency (or hole) with DS and X.25 networks in that these errors are not discovered. What is needed is an end-to-end protocol that can be implemented on both the originator and receiver of the message so that error checking may be done on the message level itself, not just at the packet level which we now see is inadequate. But there is hope.

NS/3000

The latest release of communication software from HP will include, among other features, straight X.25 support; decoupling the requirement of DS as the transport layer protocol, though it will still be available and probably widely used. As an alternative to DS, NS/3000 will support the TCP/IP protocol which includes an end-to-end (host-to-host) checksumming algorithm that can be turned on and off. This is message level checking, not packet level, and data errors can be identified and recovered from by the software.

As TCP/IP has been implemented on VAX equipment (The Wollongong Group), it is feasible to use this protocol to permit VAXen and HP3000s to communicate directly with each other without the HP1000 being in the loop. However, there are costs involved that must be considered before we make the decision.

In our application, removing the HP1000s from the communication path would eliminate a central focal point for real time monitoring and controlling of the hardcopy system. On the other side of the coin, it would remove a potential bottleneck as 24 VAXen now communicate with 20 HP3000s through three HP1000s.

The network impact of removing HP1000s must also be considered as this would dramatically alter the flow of data over the packet switched network as all VAXen and HP3000s would communicate over virtual circuits, increasing the traffic and complexity of the network environment. A number of network topology studies would have to be conducted to anticipate the changes to the network, design host redeployment strategies, and determine the network modifications required to accommodate these changes.

Another hurdle is that DEC has not officially "sanctioned" the Wollongong Group implementation of TCP/IP. Indeed, DEC would prefer your using DECNET in an all VAX environment, just as HP would prefer you using DS in an all HP3000 environment. All this really means is that DEC is not actively marketing TCP/IP, and users must ask about it specifically. Just remember, user needs always outweigh vendor desires, especially since it is the user doing the buying.

And finally, the benefits of implementing TCP/IP must be compared to the costs. Error detection and recovery handling adds to the processing requirements, which is one reason HP will permit this function to be turned on and off (I do not know if the VAX implementation includes this option or not). If the network and transport protocols are solid and the encountered error rate is sufficiently low or within acceptable levels within the current environment, the benefits of introducing end-to-end error checking may not offset the costs incurred in modifying the system to support the protocol. If data integrity is critical and there is a history of damaged or incomplete messages/file transfers, the benefits may outweigh the costs of implementation.

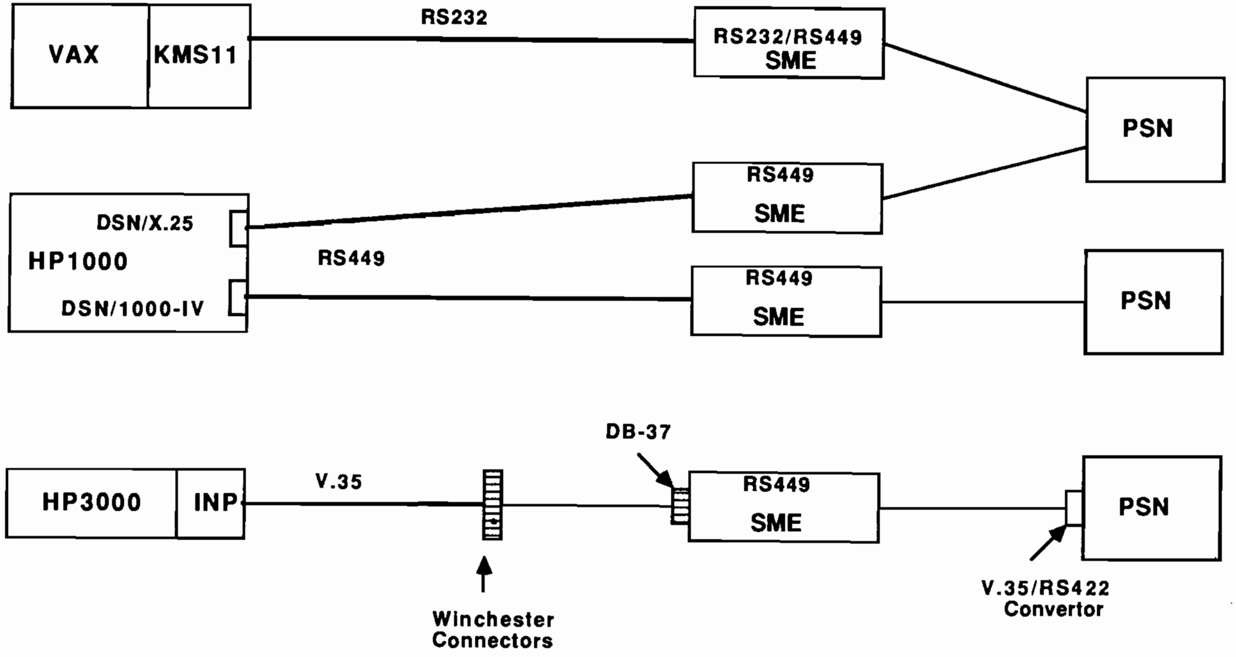
Conclusion

In this paper, I have attempted to provide a little insight (and humor) to the difficulties of bringing up a multi-vendor system on a packet switching network; that there is more to do than wait for the network to be installed before you just plug in the hosts. I included a couple of "horror" stories to illustrate that the job isn't done when everything has been connected and initial communication tests completed, but that it is an on-going effort to operate, improve, and even upgrade the system to support current and future requirements.

MCI believes in open architectures and makes no secret as to how the MCI Mail system is implemented. In fact, a more detailed description of our application and the network was the topic of another paper delivered at the Madrid INTEREX conference. I have limited the length of this paper and presentation so that there will be enough time to answer any questions that might be raised, or to discuss other related topics; including the experiences or plans of other organizations that might be building a system from scratch as we did, or who might be planning a transition to a X.25 network based system.

Biography

Steve Coya has been with MCI Digital Information Services for the past three years. He is the Senior Project Manager for MCI Mail Hardcopy Systems Development, and is also responsible for the overall planning and scheduling of system integration tests and for managing and coordinating the implementation of new software releases into the operational environment.



Physical Connections to the PSN

THE ANATOMY OF AN X.25 BASED EMS APPLICATION
(AND HOW WE PRINT THE MAIL)

Stephen J. Coya
MCI Digital Information Services, Washington, D.C., USA

Summary

This presentation describes the architecture of a multi-vendor electronic mail system (MCI Mail), concentrating on the innovative Hardcopy Distribution System supported by HP1000s and HP3000s with 2680A Laser Printers. The basic system consists of six Digital Equipment Corporation (DEC) VAX clusters and three HP1000s located in a central facility, and a number of HP3000 based print sites located across the United States and Europe. The components of this system are connected by a private Bolt Beranek and Newman (BBN) X.25 packet switching network which uses a combination of 9.6 and 56 kbps lines, largely derived from the MCI Telecommunications Transmission System. There are three operational centers, one of which monitors and controls the hardcopy system exclusively.

This paper will include an overview of the network components and system architecture, but will focus on the transmission and processing of hardcopy bound mail. It will also describe our implementation of a Graphics Design Center which supports the use of letterhead and signature graphics, and will touch on our plans to provide a communications path to support print sites located on foreign public data networks. The scope of this paper does not provide for a detailed description of each and every feature or capability of the electronic mail system.

This paper is divided into four sections covering the Packet Network, the Electronic Mail System, the Hardcopy Distribution System, and the support of "Off-Net" Print Sites. To facilitate the presentation, a "simple" network configuration diagram is presented in Figure 1.

The Packet Network

The packet switching network is made up of five major components: Packet Switch Nodes, PADs, the Network Authentication Server, the Server Maintenance System, and the Network Operations Center. The switching subsystem is made up of packet switch nodes (PSNs) which are interconnected via MCI long-haul microwave or optical fiber trunks operating at 9.6 or 56 kilobits per second.

Packet Switch Nodes

There are over 50 packet switches in our network, each of which can support up to 30 host connections and 14 trunk lines, the total of which is constrained to 44 or less. Each PSN can support a throughput rate of approximately 300 packets per second, counting one for a packet which enters and leaves the PSN. The network interfaces to service hosts and PADs by way of the CCITT X.25 protocol, and utilizes the balanced Link Access Protocol (LAPB) version of CCITT standard High Data Level Link Control protocol (HDLC). Data rates up to 56 kilobits per second are supported for host and PAD access to the PSN.

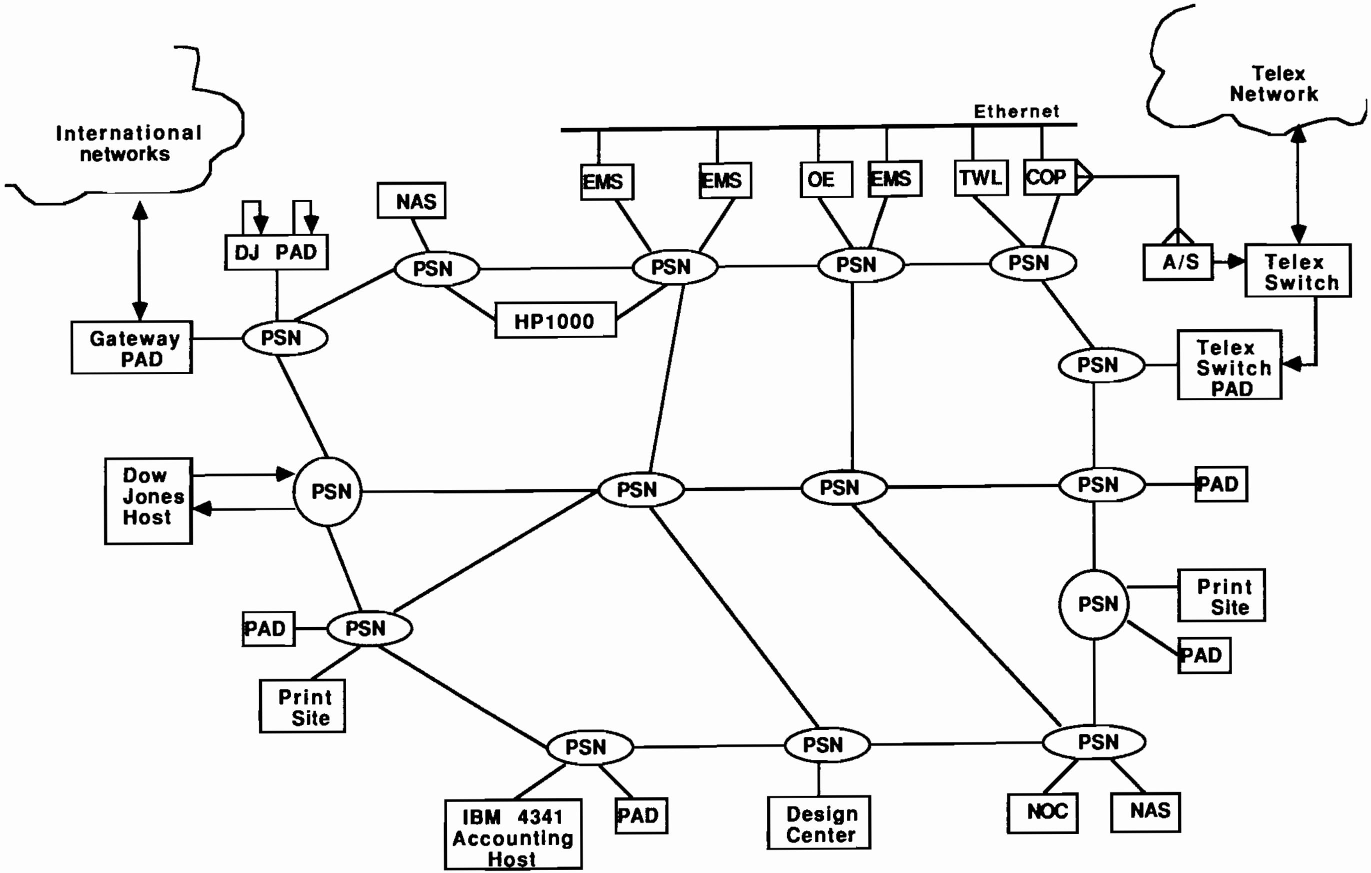


Figure 1: "Simple" Network Diagram

In addition to the link level reliability provided between hosts and PSNs, the inter-PSN protocols in the network permit adaptive alternate routing in the event of network congestion, line or node failure. These features preserve end-to-end virtual circuits even when intermediate nodes or lines fail.

Packet Assembler/Disassembler (PAD)

The next component of the packet network is the PAD. This device supports external access to the electronic mail system from dial-up and hard-wired terminals, Personal Computers, Word Processors with asynchronous dial-up support, Telex devices which use Direct Distance Dialing, and other asynchronous access devices. There are approximately 60 PADs in the network, each of which support up to 64 devices operating at speeds up to 19.2 kbps. Our system currently supports 110-1200 baud asynchronous dial-up, and special hard-wired services at 4800 baud. For our Document Processing service, special provision for the support of word processors operating bisynchronously at 2400 baud is also provided.

Two special PAD variants, the Gateway PAD and Telex Switch PAD, provide special purpose interfacing to domestic and international public packet switch users, the Dow Jones News/Retrieval Service, and the Telex user community.

Network Authentication Server

Access to the electronic mail system is controlled by a special processor called the Network Authentication Server (NAS). There are three NASes in our network to support both load sharing and reliability. Each NAS has a database that contains information for all registered users of the system, and PADS are able to access any NAS for purposes of user authentication. The database contains login names, a one-way encrypted password, the corresponding mailbox identifier, the network address of the host serving the mailbox, and a table of user service privileges.

All users accessing the mail system begin by establishing a connection from their device to a local access PAD. The access PAD prompts the user for username and password. This information is sent, in an encrypted form, to any one of the NAS servers, where the information is checked against the database. Once a user is validated, the NAS provides the access PAD with the network address of the user's mail host. The access PAD then establishes a virtual circuit to that host, and the mail session begins.

The NAS databases are updated daily by means of a Server Maintenance System which runs as an adjunct to the Order Entry System. On a daily basis, changes to the registered user database which must be reflected in the NAS, such as password changes, addition of new users, movement of mailboxes among service hosts, removal of users, and changes in user service privileges are communicated to each NAS using a private application protocol between the host running the Server Maintenance System and each NAS.

Network Operations Center

The monitoring of the network is accomplished by another special processor called the Network Operations Center (NOC) which uses a special interface to the network to access internal processes operating in each PSN. Each PSN reports periodically to the NOC the status of all attached hosts and trunks, throughput, alarms, and abnormal conditions. The NOC is also capable of monitoring the status of all network PADs. Statistics on the use of PAD ports can be collected for analysis.

The NOC is capable of remotely controlling the reloading of operational software into any of the packet switches. Consequently, the propagation of new software or recovery from a node failure is readily accomplished. The NOC can also distribute new releases of the PAD software by downline loading the operational programs through the network.

The Electronic Mail System

A multi-cluster system, situated in a central facility, acts as the focal point for the electronic mail system. Physically, the system is made up of DEC VAX 11/780s and 11/785s which are organized into six clusters of four processors each. Each cluster supports twenty four large (404 mbyte) disks which are sharable among all processors in the cluster. The system supports disk shadowing so that the 24 disks in one cluster are organized as 12 shadow pairs. Any data written to one of a shadow pair is also written to the other. A disk failure does not cause any loss of service and a new disk can be installed without interruption of service.

Each processor is connected to a 10 mbps Ethernet to support high speed transfers both within and between VAX clusters. Each VAX is also connected to a packet switch and it is this connection that permits PAD users to access the mail system and to accommodate communication between these VAXen and with hosts not connected to the Ethernet.

The Mail System

Users may use the electronic mail system either interactively or in batch mode. Interactively, there are two classes of service: basic and advanced. The basic service is menu driven and supports the basic creation, editing, reading and sending of electronic mail to other electronic mailboxes, Telex users, or to one of the print sites for postal or courier delivery. The advanced service is command driven and provides users with more capabilities. All users, basic and advanced, have access to the Dow Jones News/Retrieval service and may exchange messages with the Telex community.

Each mail host has access to a copy of a relational database containing all registered subscribers of the mail system. During message creation when users enter addressee names, they are looked up in the database and any ambiguities or failures to find matches are instantly reported. The service permits a correspondent to be addressed by his formal name, user name, or the unique mailbox identifier.

The batch electronic mail service is requested by the user when initially connecting to the mail system. This service gives access to a subset of the interactive services and is oriented around the requirement to support computer based interfaces to the mail system. The batch service provides an exchange protocol permitting the caller's PC or mainframe to stay in synchrony with the mail service, handshaking at each major step to assure completion and to report any problems in a machine understandable fashion.

Each VAX mail host also has an administrative subsystem that includes an accounting facility which logs information about user sessions and message deliveries (including hardcopy and telex). On a daily basis, every VAX sends this accounting file over the network to an IBM 4341 which functions as the accounting host. The administrative subsystem also maintains logs of system activity, error messages including any user encountered problems that may be detected by either the application software or the operating system. It also accepts daily updates to the user databases produced by the Order Entry system which is described in the next section of this paper.

Order Entry and the COP

There are two "special purpose" VAXen in our central facility. One is the Order Entry (OE) machine. The Order Entry system is the means by which the registered user databases are maintained and it is through this system that user information is added, deleted, or modified. The system provides customer service personnel with full interactive access to the user database permitting the entry and editing of user records. The system also supports interactive access via the network to the accounting and invoicing database maintained on the IBM 4341. The Order Entry system is the originator of all database transactions to the EMS database and, through the Service Maintenance System, to the NAS databases.

The second special purpose VAX is called the COP (which doesn't mean anything...it functions as a traffic COP) whose purpose is to interface the electronic mail system with the Telex community. Every registered user of our mail system has a unique Telex number which is simply the mailbox identifier preceded by the characters 650. A telex sent to one of these 650 numbers comes through the Telex switch which has a connection to the Telex switch PAD mentioned earlier. This PAD is linked to the COP through the packet network and a special process turns the incoming telex message into an electronic mail message and delivers that message to the user's INBOX. In addition to sending messages to other registered users and to the hardcopy system, mail users may also send messages to domestic or international telex machines. In this case, the system reformats the message for injection into the MCI International Telex Store and Forward AUTOSAFE system, which automatically transfers the call to the Telex switch.

The Hardcopy Distribution System

One of the innovative features of the mail system is the ability to send hardcopy or "paper" mail to specific message recipients. The Hardcopy Distribution System is made up of four components: the Hardcopy Relay Agent which runs on the VAX, the HP1000 which serves as an interface and router, the HP3000 based print sites (called Digital Post Offices or DPOs), and the Graphics Design Center which maintains the master graphics database.

If a user wishes to send a hardcopy letter to another registered user, the mail system will look up that recipient's registered address from the EMS database which becomes the mailing address. If the recipient is not a registered user of the service, or the registered recipient is not at the "home" address, the originator has the ability to supply a postal address for the recipient. The registered address of the message originator is used as the return address.

Another special service offered by the mail system is the Telex World Letter (TWL). This service permits Telex users to address and send telex messages from their devices that are to be printed at one of our HP3000 based print sites and delivered by the postal service. This service allows telex users to take advantage of our hardcopy system without being registered as mail users.

Though the message is created on a VAX which serves the user as a mail host, copies need to be transmitted over the packet network to the appropriate print sites, based on the postal code of each recipient, where the copies will be printed on a 2680A Laser printer. This requirement caused some problems initially as HP3000s insist on using DSN/DS as the [OSI model] transport layer protocol and DS was not (and is not) available on VAX equipment. The "normal" methods of transferring data files between VAXen and HP3000s were not acceptable: tape transfers were obviously not appropriate due to time constraints and geographic separation of the DPOs and VAX facility, and RJE sessions did not support the overall design concept of the electronic mail system, and would have been both expensive and inefficient.

The solution was to introduce HP1000s between the VAX and the HP3000. The HP1000 has two modem cards installed, each of which is connected to a PSN. One modem card is used to communicate with the VAX using the the DSN/X.25 communication package and a simple file transfer protocol (SFTP) transport layer developed internally to facilitate file transfers among multi-vendor host computers. The other modem card uses DS/1000-IV to communicate with the various print site based HP3000s, using DSN/DS as the Transport Layer protocol.

The hardcopy system provides the user with the ability to specify that a letterhead is to be printed on the first page of the message, and whether or not a signature graphic is to be printed. Of course, this requires that the user register the letterheads and signatures with the mail system. The mail system will not permit a user to reference a letterhead or signature graphic that is not associated with that user's mail account.

Hardcopy messages can be delivered by one of three methods: the postal service, courier for next day delivery, and courier for Four Hour, same day delivery in some locations. When a user has created the message and posts it, the mail system checks the country code, postal code, and priority for each recipient. If the user has requested a delivery option not available in the recipient's area, the system will not post the message but will warn the user and permit the editing of the envelope.

Hardcopy Relay Agent

After the message has been posted by the user, a copy is delivered to a special mailbox which is serviced by the Hardcopy Relay Agent (HCRA). The HCRA processes only those recipients that are to receive the message in hardcopy form. As a message may contain any number of recipients, because there may be different delivery options specified for the hardcopy recipients, and because the monitoring, tracking, and accounting must be done for each copy, each postal recipient is handled as a separate unit. Based on the combination of a recipient's postal code, country code, and delivery option, the HCRA determines which print site is to receive and print the message.

All hardcopy traffic is transmitted from the Hardcopy Relay Agent to the one of the Hardcopy Distribution System Interfaces (HP1000s), which in turn transmit the print files to the appropriate remote print site. Each print site has a primary HP1000 associated with it, and the VAX will attempt to send all hardcopy messages to the primary HP1000. If the HP1000 is unable to accept the print files, the Hardcopy Relay Agent will send the print files to an alternate HP1000. Each VAX may send hardcopy traffic to any of the HP1000s, and each HP1000 can communicate with any HP3000 print site. Interestingly enough, each HP1000 believes it is the only HP1000 in the network, but what it doesn't know won't hurt it. While the hardcopy environment is capable of being supported by a single HP1000, three are used for backup and load leveling purposes.

The HCRA knows the node names of each of the print sites, but not the network addresses. Instead, each print site is initially associated with one of the HP1000s and the DTE addresses of these HP1000s are known to the VAX. An output queue process exists for each HP1000 in the network. The HCRA takes each message and places it into one of the output queues. Within each output queue, the messages are maintained in priority order.

After the message has been sent from the VAX, acknowledgments are returned from the remote print sites, permitting the HCRA to keep track of successful processing of each hardcopy message. These acknowledgments are used by the HCRA to generate accounting transactions, monitor the status of individual messages, generate return receipt notifications (if requested by the originator) and to maintain the status database of the hardcopy system.

The Hardcopy Distribution System Interface

The Hardcopy Distribution System Interface (HP1000) serves as the "router" of hardcopy traffic and provides operational personnel with an overview of the state of the hardcopy environment.

When the connection to the HP1000 is opened by the VAX, all print files are transmitted in one session over an X.25 virtual circuit using SFTP. The HP1000 is responsible for examining each print file to determine the target remote print site and the priority of the message. The HP1000 is capable of receiving print files from every VAX simultaneously, and places these print files in the appropriate queues, based on the target print site, and the priority of the message. These queues are maintained internally in the memory of the HP1000, and the queues may be viewed or manipulated by authorized operations personnel. It is possible to move the contents of one print site's queue to another, and it is possible to instruct the HP1000 to always send a particular queue to an alternate print site. Manipulating queues only effects the eventual print site that will process and print the letter. The priority of the message is never changed.

When the HP1000 has print files in its queue for a particular print site, a connection to that print site is made, using DS over an X.25 virtual circuit. The HP1000 starts up the receiving process on the remote HP3000 and transfers the contents of the queues in priority order, receiving confirmation of successful transmission from the remote print site for each print file transmitted. When the transmission of all print files in every queue for the print site has been completed, the transfer process is closed down and the connection is cleared.

The processing of acknowledgments from the remote print sites are handled in a similar way. The print site will open a connection, via DS, and transmit all the acknowledgments to the HP1000. The HP1000 places the acknowledgments in the appropriate queues, also maintained in memory, for each VAX. Regardless of the print site that processed the print file, acknowledgments are always returned to the VAX that originally sent it, for monitoring and accounting purposes.

The HP1000 permits the operations staff to monitor the state of the hardcopy environment and provides mechanisms to react to potential problems or operational decisions, such as the rerouting or moving of print files to alternate print sites. Monitors are also used by operations to show the progress of file transmissions between the HP1000 and the EMS hosts and the remote print sites. The monitors notify the operations staff when problems are encountered in transmitting files to either the VAXen or the remote print sites.

Laser Print Sites

The printing of hardcopy messages is performed at the remote print sites. The application software runs on an HP3000 series 40 to which the HP2680A Laser Printer is attached. Print site equipment also includes disk and tape drives and operator terminals. The HP3000 and the Laser Printer are configured with two mbytes of memory each.

The application which runs on the HP3000 has been designed in such a way as to stream line much of the processing and printing activities, minimizing the amount of operator intervention required, providing a real time display of activity and status, and providing the capability to control the flow of messages through the system. There are five major processing modules that handle the messages to be printed, from receipt of the message from the HP1000 through the actual printing and sending of an acknowledgment back to the originating VAX. The design of the print site software is illustrated in Figure 2.

Receipt of Print Files

When the HP1000 has messages in its queues for a particular print site, the HP1000 logs on to that HP3000 and initiates a process to accept the print files. During the transfer, all print files are transmitted in queued priority to the HP3000. The receipt process on the HP3000 notifies the preprocessing module of incoming print files, and sends acknowledgments back to the HP1000 for each print file received. This acknowledgment is not sent until the entire print file has been received and stored on the HP3000.

If additional print files for the HP3000 are received at the HP1000 during the transmission, these files are injected into the queues and transferred during the same session. When the entire transfer is completed, the receipt process is closed down by the HP1000. The application is designed to permit up to eight simultaneous transfers from the HP1000s. The print site operator has the capability to disable any or all of the HP3000 receipt processes to prevent any transfer from the HP1000.

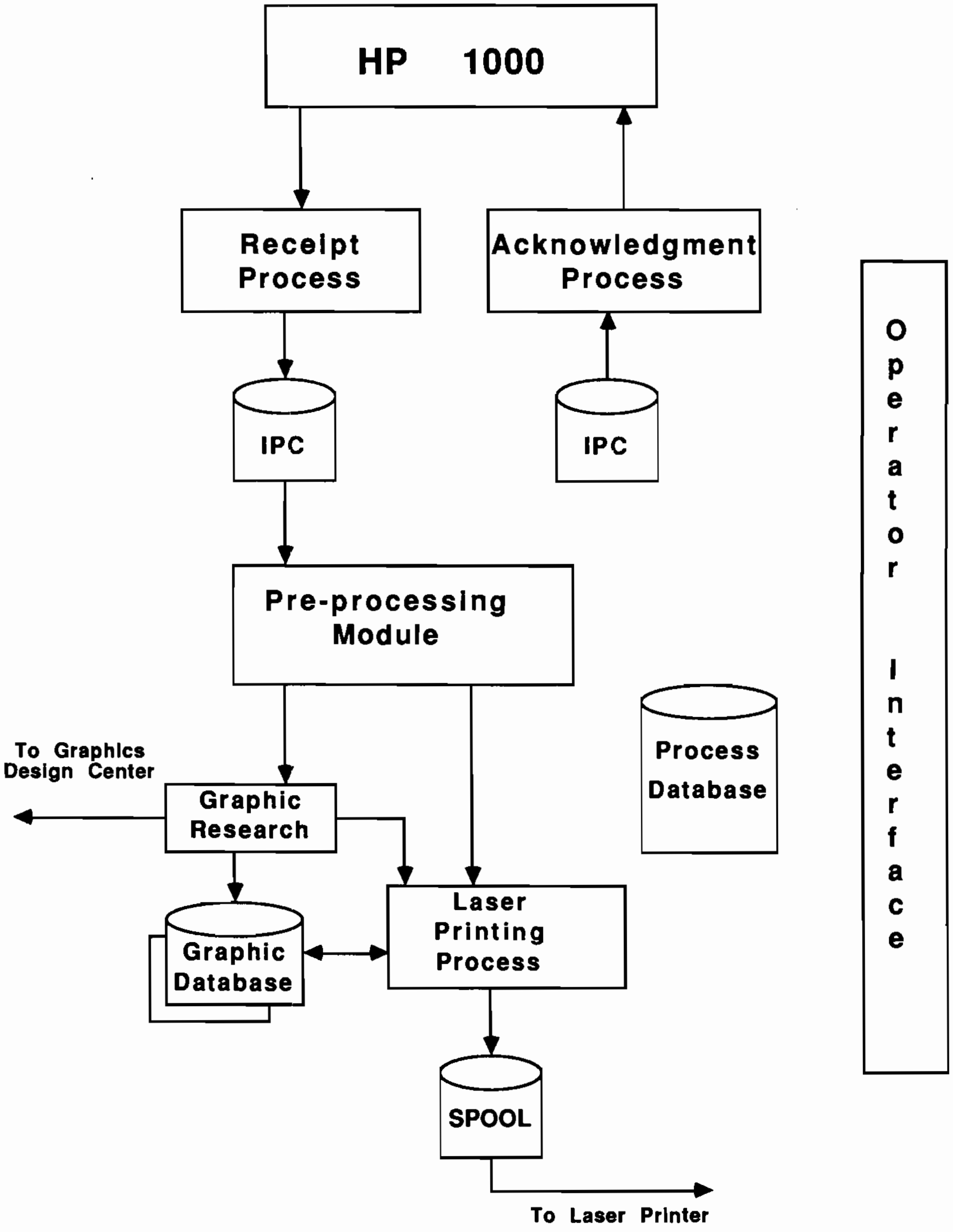


Figure 2: Remote Print Site Design

Pre-processing of Print Files

The pre-processing module of the application examines the print file contents for errors and graphic requirements, reformats the print file for later processing and printing, and places the print file into a prespool file. A prespool file is a collection of print files of the same priority. Prespool files are used to optimize the efficiency and throughput of the laser printer, and also to provide more controls to the remote print site operations personnel.

The pre-processing module will continue to add print files to a particular prespool file until the maximum number of print files has been entered or until the configurable spool timer expires. When the prespool file has been created, the pre-processor notifies the printing module, which takes over the processing responsibility of the prespool file. If necessary, the pre-processor immediately creates a new prespool file and continues processing print files that have been received by the HP3000. The pre-processing module dynamically allocates disk space to store the prespool files. This process prevents the allocation of more storage than is necessary to store the file, and also provides for the storage of very large messages.

Another function of the pre-processing module is to examine the graphic requirements for a given message to determine if the required graphics are in the remote print site's local graphics database. If the graphics are in the local database, processing continues. If the graphics are not in the local database, a message is sent to the graphics retrieval process.

Graphics Retrieval

When the pre-processing module encounters a print file requiring a graphic not stored locally at the print site, the print file is placed in a separate prespool file and the graphics retrieval module is notified that a particular prespool file contains one or more print files requiring a specific graphic. The graphics retrieval module establishes a virtual circuit through the packet switching network to the Graphics Design Center. The print site HP3000 logs onto the Graphic Design Center and performs a remote database access against the master graphics database. When the requested graphic record is located and extracted, it is stored in the local database. The graphic retrieval module then informs the printing module directly that the prespool file is ready for printing. If for any reason the requested graphic is not available, the printing module is notified to print a reject page for that particular prespool file.

The local graphics database is actually two files. The first file contains the unique identifier, margin settings and other data, including pointers to a position in the second file. The second file contains the actual partitioned raster files of all graphics stored at the local site. The local graphic database also contains the date the graphic was last referenced and print site operators have the ability to delete entries from the local database.

Laser Printing of the Print files

The Laser Printing module accesses the prespool files, creates the print spool files and submits them to the HP2680A laser printer. The prespool files are processed and submitted to the laser in priority order. In addition to the printing of text and either letterhead or signature graphics, the laser printing module supports bolding, underscoring, superscripting and subscripting, and the use of headers and footers within the text of the message.

A spool file, containing the information needed to drive the laser printer, is created for every prespool file submitted to the laser printing module. The laser printing module prints a header and trailer page before and after the letters in a prespool file. These pages contain the name of the prespool file, the total number of letters within the prespool file, and the unique identifier assigned to each recipient letter by the HCRA.

Between the prespool header and trailer pages are the letters themselves. Each printed letter is preceded by an address page, which contains the mailing address of the recipient and the originator's return address. Following the last page of each letter is a print control page which contains information about the message just printed. These control pages are maintained at the remote print site for tracing and monitoring purposes, and as required by law.

To facilitate the handling and monitoring of priority mail, and to meet the requirements of the courier company, each priority message is assigned a bill of lading number as it is received at the HP3000. Each remote print site has a unique location code (three characters), and this location code becomes the first three characters of the bill of lading number. The bill of lading number appears beneath the recipient mailing address on the address page. For overnight or courier mail, a destination airport code is appended to the bill of lading number.

This module will also print REJECT header and trailer pages if it is not able to print a letter. Between the reject header and trailer pages will be a control page stating the reason for the reject (for example, if no graphic was found by the graphic retrieval process). Any "problem" messages will result in a REJECT header page followed by a control sheet indicating the problem, followed by the trailer page.

Acknowledgments

Throughout the processing of a print file, entries are made to a process database, including the status returned from the Laser Printing module. This information is included in the acknowledgment sent from the HP3000, through the HP1000, and back to the originating EMS host VAX. The acknowledgments require remote print site personnel to assert an accounting code and cause the acknowledgments to be transmitted. The accounting code is used to indicate whether the letter is billable or not. Acknowledgments are generated for a given prespool file. The operator may set a global accounting code for each print file within the prespool file, though they do have the ability to modify the accounting codes for particular print files.

Receipt of a successful printing and billable acknowledgment by the EMS host VAX causes the accounting transaction record to be generated and, if requested by the message originator, will cause a return receipt notification to be generated and posted.

Remote site personnel also have the option of sending back an acknowledgment that essentially requests the VAX to retransmit the print file to an alternate print site. This capability facilitates the processing of user mis-addressed mail which resulted in the print file being transmitted to the default printer.

Acknowledgments are sent back to a primary HP1000, as configured at each remote print site. If the network connection cannot be made to the primary HP1000, or the HP1000 cannot accept the acknowledgments, this module will attempt to send the acknowledgments to an alternate HP1000. Should the module determine that is is unable to send the acknowledgments to any of the HP1000s, a warning message is printed and displayed on the HP3000 status screen. In any event, these acknowledgments are maintained in a queue until successfully transmitted.

Operator Interface

The operator interface module controls the hardcopy application software on the HP3000 and continually displays the "state of the world" when not being used by remote site personnel. The operator interface module consists of four separate screens (Command, Status, Acknowledgment Processing, and Report Generation) which are used by the site personnel to monitor the application and hardcopy processing activities, or to process operator commands and requests.

The command screen is used by the remote print site operators to control the application and perform operational tasks. This screen is used to start and stop the entire application, pause or resume certain processes, check and delete graphics from the local database, reprint letters received, and to provide control over the acceptance of new print units and the transmission of acknowledgments.

The Command Screen will also display information on the screen, either by prespool file or print file, and permits detailed examination of the displayed information. The command screen also simplifies such tasks as performing maintenance dumps (system backups), the ability to change the information or status of an acknowledgment and the ability to retransmit an acknowledgment.

As many of the command screen options are very powerful, some capabilities require the operators to enter their personal operator code before the requested process is begun. Some of the capabilities require the password of the remote print site supervisor.

The Status Screen is a real time display which is continually updated as it receives status information from the various modules. The status screen will display which connections to the HP1000s are open, which are currently active, which pre-processing components are active. If active, the name of the prespool files are displayed with the number of letters currently contained by the prespool file. The Status Screen also displays which prespool files are being processed by the laser printing process, along with the total number of pages that have been printed.

A table in the corner of the status screen displays, by priority, a historical record of the number of letters that have been received, printed, rejected, and acknowledged. The table also shows how many graphic retrieval requests have been made and completed. These counters are maintained over time, even if the application is stopped and restarted. Through the Command Screen, the supervisor has the ability to reset all or some of the counters.

The Acknowledgment Processing Screen is the interface used by the operators in generating and transmitting the acknowledgments. The operator is required to enter the prespool file name, enter the global accounting code, change the status for any number of print units within a prespool file, and request that the acknowledgments be transmitted. To facilitate the process, this screen displays all valid accounting codes and validates the operator's entry.

The Report Generation Screen, as its name implies, is used to generate reports which are printed on the laser printer. The operator may request a report and specify how the information to be displayed is to be sorted and the time period the report is to cover. The report generation screen is also used to generate the courier manifests which include the bill of lading numbers. This manifest accompanies the letters to be delivered by the courier.

The Graphics Design Center

The Graphic Design Center, an HP3000 series 64, is the central repository for all graphic information which may be used at the remote laser print sites. This includes both letterhead and signature graphics. An HP26096A Digital Camera System is used to optically convert these letterheads and signatures into a digital dot-bit format for electronic transmission and reproduction. Storage, retrieval, maintenance, and transmission facilities are included within the Design Center to allow access to the registered graphics from all laser print sites.

Each graphic is stored in the graphics database at the Design Center and associated with a unique graphic identifier. This identifier is assigned by a module of the Order Entry system. Once the graphics have been created and entered into the master graphic database, the graphic identifiers are added to the user's EMS database record. At this point, these graphics may be referenced by the user when creating a message to be printed at a remote laser print site.

Each mail account has a default letterhead. If only one letterhead is registered, it is the default. If more than one letterhead is registered, the user specifies which letterhead is to be the default. An account may also have more than one signature registered, but it is not necessary to designate one as the default signature. The user assigns a name to each letterhead and signature and references them by their assigned names. The mail system will substitute the actual graphic identifier associated with the named graphic when the message is posted by the user.

When creating a message, the user specifies which graphic is to be used by providing the name of the graphic in the handling field of the message. If no specific reference is entered, the defaults are used. If a letterhead has not been registered, there exists a system default letterhead which appears on the laser printed message. There is no system default signature. If no letterhead is desired, the user may request a "BLANK" letterhead.

All graphic information is stored in an IMAGE database. In addition to the graphic identification number, the database contains internal information such as the submission date, the creation date, margin defaults, and the last access date.

Inter-Network Hardcopy Support

As system usage increased, it was noted that more and more users were sending hardcopy messages to recipients in foreign countries. As this percentage grew, we were soon faced with the demand for print sites located outside of the United States, specifically in Europe, to support time critical processing and delivery of hardcopy messages. A print site was established in Belgium, connected by a private leased line to one of our domestic switches, and plans were developed to establish additional print sites in other foreign locations. The use of dedicated international circuits to link foreign print sites to the domestic U.S. system is expensive, however, and we were strongly motivated to make use of sharable public packet net systems as an alternative means of supporting these remote facilities.

These "off-net" print sites must still function as Digital Post Offices, receiving hardcopy traffic, initiating graphic research requests to the Graphics Design Center, and sending acknowledgments back to the originating EMS host VAX located on the private network. In essence, a link had to be established between our private network and a public network.

Fortunately, our organization had just introduced a public packet switching network (MCI DataTransport) and had established an X.75 Gateway connection to our international packet switching network (MCII IMPACS) which already had connections to a number of other public data networks. All that remained was to link our private network and our public packet networks.

X.75 only supports connections between public data networks, not private networks, so X.25 links had to be established between the our network and the public data network. This was accomplished by adding a physical connection from the Graphics Design Center and from one of the HP1000s to the public network switch (See Figure 3). From the public network, traffic (print files) will pass through X.75 gateways to other public networks, and from there to the final "destination" network. Once a print file reaches the destination network, it will be delivered by the network to the print site host at the destination DTE address.

An off-net print site must be connected to our public packet switching network or to a network that can be reached by the public network through a series of X.75 connections between consenting networks. All hardcopy traffic destined for one of the off-net print sites is sent by the Hardcopy Relay Agent to the inter-network HP1000 whose network print site link (to HP3000s) is connected to the public network. The inter-network HP1000 maintains a table of all off-net print sites, their Data Network Identification Code (DNIC), which identifies the destination network, and the DTE address.

The VAX-based Hardcopy Relay Agent still refers to the print site by its node name only and does not need to know on which network the print site is located or its DTE address. The destination print site is determined by the country code and postal code of the recipient's mailing address, though in the case of the off-net print site it will only key off of the country code. If the destination print site is off-net, the Hardcopy Relay Agent will transmit the print file to the inter-network HP1000.

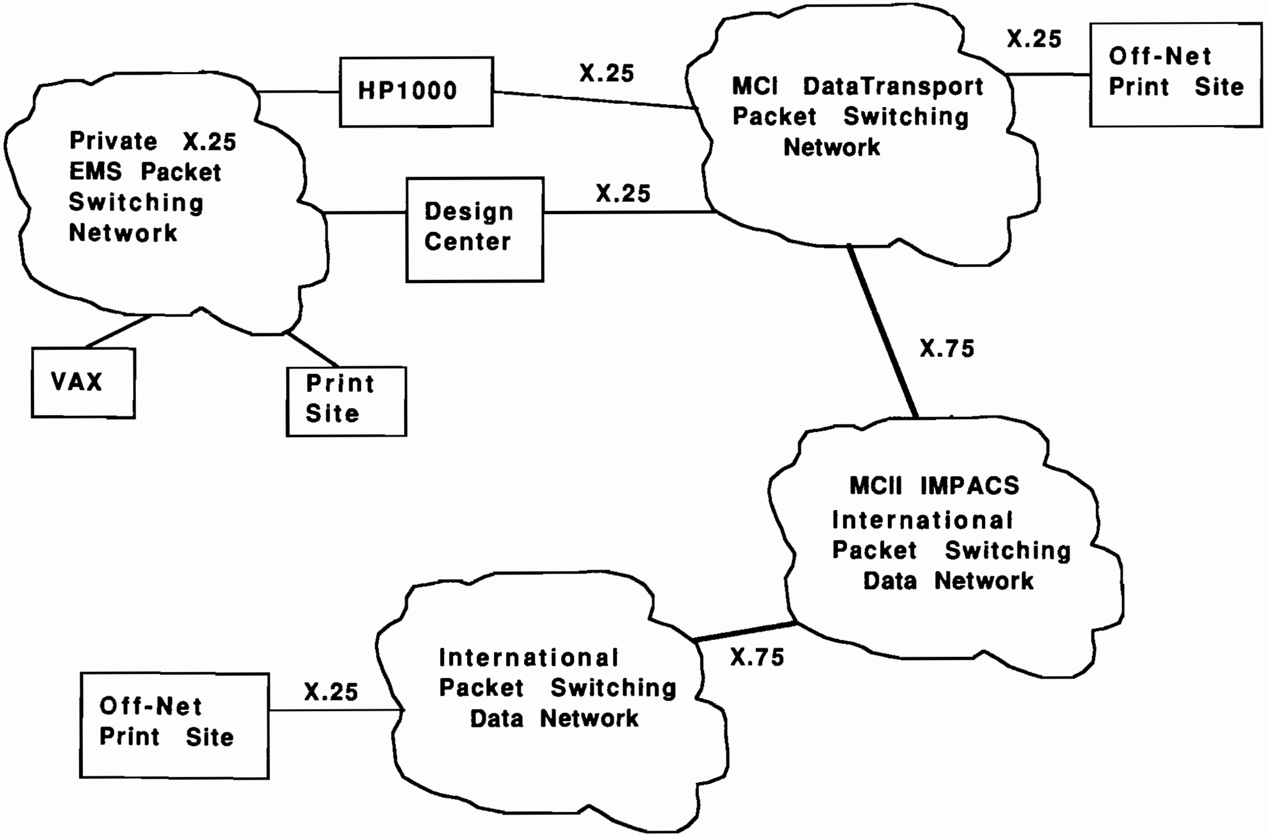

Private X.25 EMS Packet Switching Network
HP1000
X.25
Design Center
X.25
MCI DataTransport Packet Switching Network
X.25
Off-Net Print Site
VAX
Print Site
X.75
MCII IMPACS International Packet Switching Data Network
X.75
International Packet Switching Data Network
X.25
Off-Net Print Site

Figure 3 : Off-Net Print Site Architecture

From the inter-network HP1000, a virtual circuit is established through our public data network, out the X.75 gateway(s) to the public DTE address of the off-net print site. Once the connection is established, the print files are transferred to the off-net print site for processing and printing. When the print site processing is completed, acknowledgments from the off-net print site follow the same path back through the X.75 gateway(s) to the inter-network HP1000. The HP1000 then establishes a connection to the appropriate VAX and transmits the acknowledgments.

If an off-net print site receives a hardcopy message requiring a graphic not available in the print site's graphic database, a Graphic Research Request is initiated by the print site. This results in a connection being established from the off-net print site on the "destination" network through the X.75 gateways(s) to the Graphics Design Center which is now connected to the public data network as well as our private network. Once the connection is established, the off-net print site performs a remote database access against the master database on the Graphic Design Center, retrieves the necessary graphic information, and closes down the connection. The graphic information is then placed into the off-net print site's local graphic database for future use.

Mail Control

There are three operational centers supporting the mail application and the network. Once of these, the Mail Control Center, is responsible for monitoring and controlling the Hardcopy Distribution System. A central facility is organized around a set of data terminals which are used to access special software running on the VAX EMS hosts and on the HP1000s. Mail Control personnel have special privileges that permit them to access these programs through the network from the data terminals connected to PADs.

When connected to the VAX Master Node (where the software is located), Mail Control personnel are able to start or stop the Hardcopy Relay Agent process on all or individual VAX hosts. They are also capable of viewing the hardcopy process log files (and any error messages about problems in processing hardcopy mail) and maintaining the postal code routing databases. The software includes report and query options to display all pending messages (those sent out but awaiting acknowledgments from the print sites) and all queued messages, on a host by host basis.

When connected to an HP1000, Mail Control personnel can monitor and control the various VAX and HP3000 queues maintained in memory. This includes the moving of print files from one print site's queues to another, entering instructions that will automatically reroute traffic from one print site to an alternate print site, and stopping the flow of traffic to individual HP3000s or VAXen. The software on the HP1000 permits the addition or deletion of hardcopy hosts, changing node names or DTE addresses, and closing down all packet network links.

A data terminal is connected (again via a PAD) to each HP1000 and functions as a monitor, tracking the progress of file transmissions to and from the HP1000. The software running this monitor displays inverse video error messages and "beeps" whenever a problem is encountered establishing a virtual circuit to either a VAX or HP3000.

Conclusion

This paper has explored the basic architecture of most of the components of our mail system in general, and the hardcopy system in particular. It focused more on WHAT is done rather than HOW it is done. There are a number of features and capabilities that were not mentioned at all, such as our Custom Mail product and Response Plus services to support large volume mailings of hardcopy messages. Even though some of the tools were described, there was no mentioned of the operational aspects of supporting the hardcopy system, how and why these tools are needed and used, or what is involved in digitizing customer letterheads and signatures. Unfortunately, time and size limitations prohibit a more detailed explanation.

The MCI Mail system integrates a broad range of technologies and vendor products into a coherent collection of practical and innovative services. The system described in this paper has been in operation since 1983. A number of implementation details have changed since them as we learned from experience about operating the system and supporting new and "unique" client requirements, but the basic architecture has remained stable.

Biography

Steve Coya has been with MCI Digital Information Services for the past three years. He is the Senior Project Manager for MCI Mail Hardcopy Systems Development, and is also responsible for the overall planning and scheduling of system integration tests and for managing and coordinating the implementation of new software releases into the operational environment.

PACKET SWITCHED NETWORKS - THE FUTURE OF DATA-COMMUNICATIONS ?

Joerg Groessler
Joerg Groessler GmbH, Berlin, West Germany

Summary

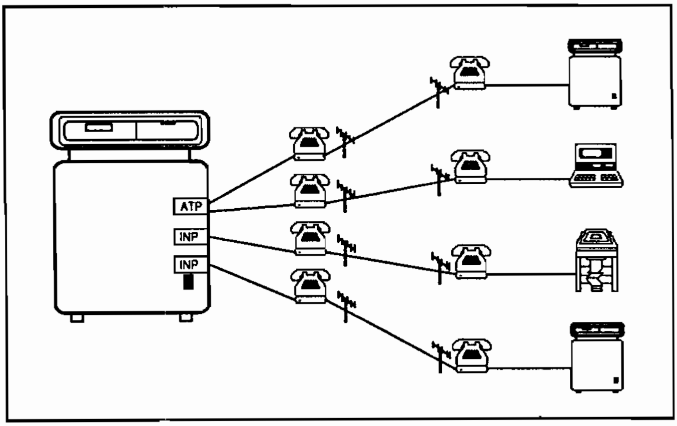
Packet Switched Networks (PSN) provide data communication on the basis of an international standard data communication protocol (X.25). In addition to fail-proof data transfer it offers a method of establishing logical rather than physical connections.

Since PSN has been introduced, efforts have been undertaken to create new standards for all kinds of tasks in data communications (terminal access, file transfer, remote job entry) using X.25 as their basic transportation method. The first result was PAD (packet assembly and disassembly) which meanwhile is supported on all major computer systems worldwide (including HP3000). Other standards are still discussed or in the status of a draft (e.g. file transfer).

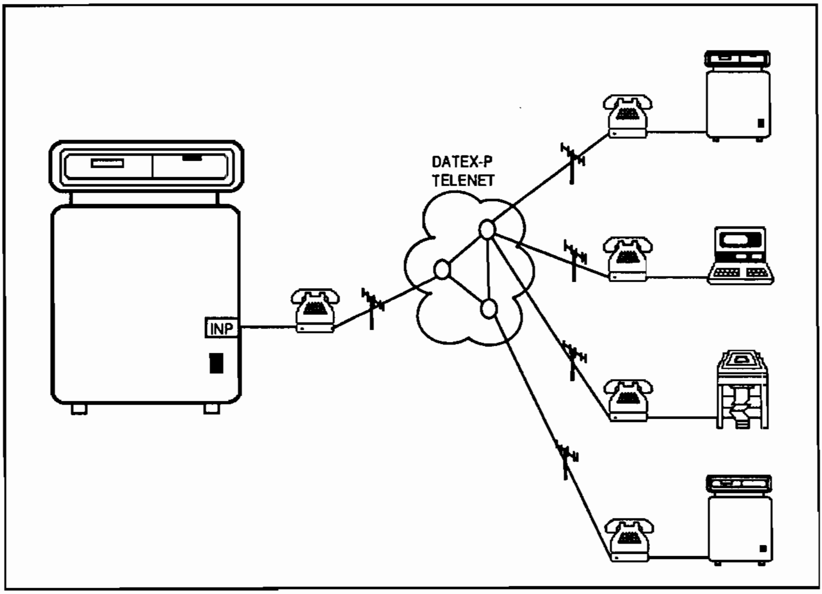
After some time of confusion (basically about handling and pricing) users start to understand that PSN opens them a world where various computer systems made by different vendors can talk to each other on a very high level of communication (something like DS working on all kinds of systems). Big companies solve their problem of communication between different computer systems. Other Companies start to provide services (like database access, electronic mail) which can be used by everybody having PSN access.

1. Basic Structure of PSN

In traditional data communication enviroments, data terminals (which stands for either a real terminal or a host computer) are connected via telephone lines. Modems are required to transform the digital signals which have a theoretically unlimited bandwidth to analog signals with the bandwidth of a telephone line which is app. 3 Khz. Connections are done either by simply dialing (using a handset) or by establishing a permanent connection (leased line).



In Packet Switched Networks computers take care of establishing a connection between one data terminal and another. The data terminal is connected via a modem to the exchange computer (mostly using leased lines). Data is transmitted no longer in a steady flow but in portions of so called packets. Each packet is provided with an address, which makes it possible to have more than one logical connection open at a time (using the address in the packet the exchange computer knows which logical connection part of the data belongs to).



2. Standards involved in Packet Switched Networks:

X.3 Packet Assembly and Disassembly Unit (PAD) which is used in a PSN enviroment for an asynchronous data terminal. A virtual terminal which can be controlled (speed, XON/XOFF, echo etc.) by standard functions (e.g. escape sequences) regardless to what host computer it is connected.

X.21 Interface between data terminals and data communication units (modems) for synchronous transmission within public networks.

X.21 bis as X.21 but using the V-Series of modems (e.g. V.22)

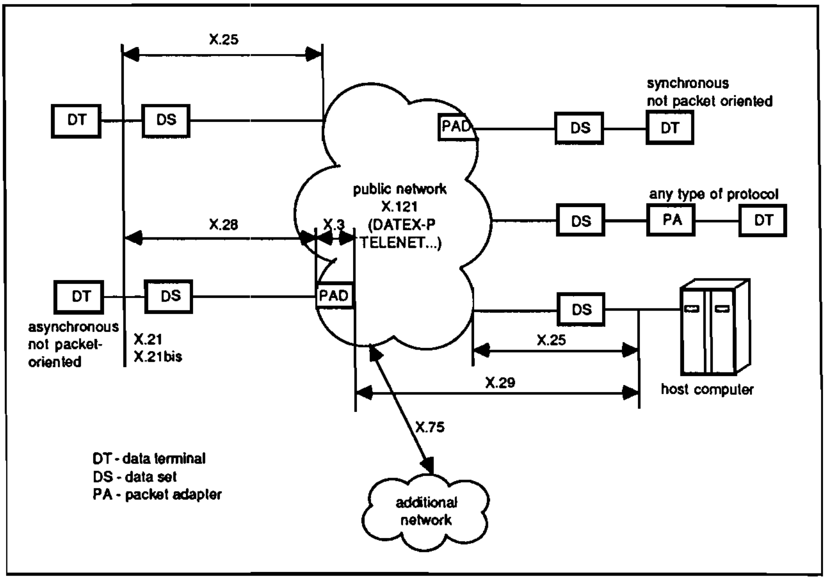
X.25 Interface between data terminals and data communication unit for terminals using packet switched networks.

X.28 Interface between a data terminal and a data communication unit for an asynchronous terminal using a PAD unit to access the packet switched network within the same country. This standard can also be applied to local PAD units which may be a program in a mainframe or a PC.

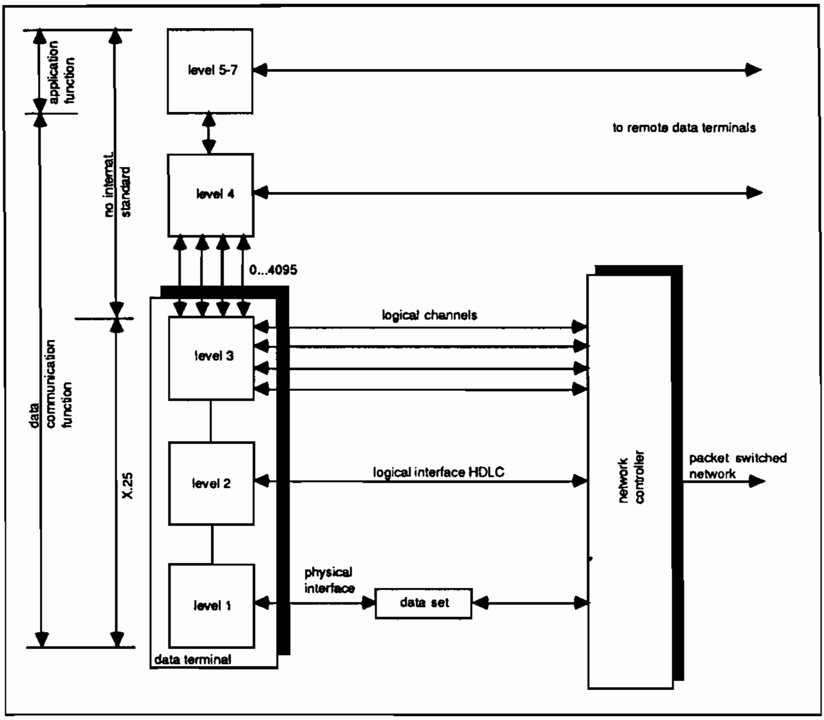
X.29 Method of exchange of data and control information between a PAD unit and a data terminal working in packet mode (which typically is the host computer).

X.75 Communications interface between different packet switched networks.

X.121 International numbering scheme for public packet switched networks.

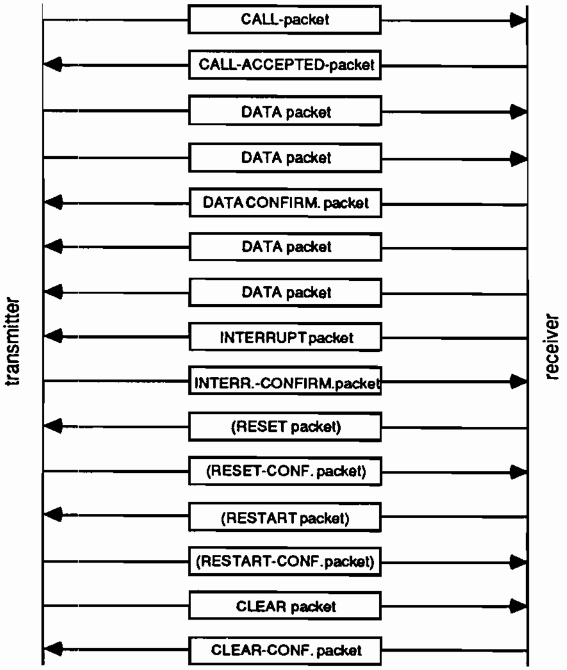


3. X.25 Characteristics

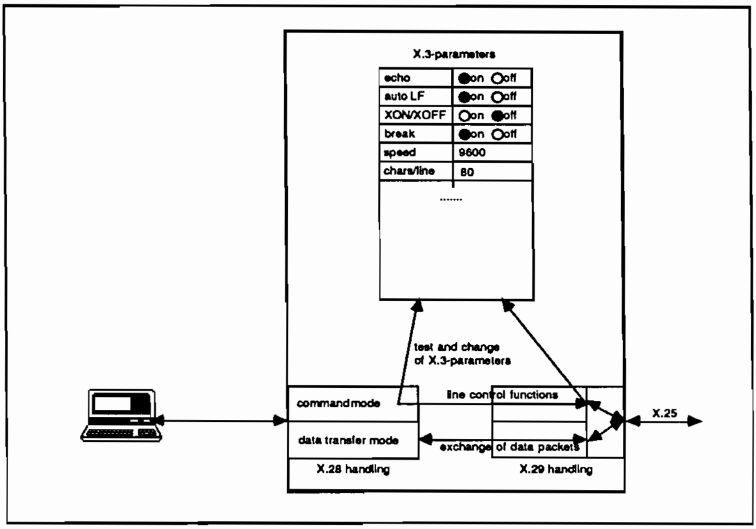


These are the main characteristics for packet switched networks:

- Automatic dialing with various options (collect call, "closed user group" (access only possible for a certain class of users), "call user datafield" (additional information about the kind of connection)
- error proof data transfer using HDLC protocol on level 2
- flow control on level 3 (packet level)
- interrupt as a bypass in the event of e.g. erromous flow control
- various levels of error handling (soft, hard etc.) using RESTART and RESET packets



PAD: a standard data terminal; a set of parameters (speed, echo on/off etc) can be tested and manipulated by both end user and host computer. So far this is the only available standard for levels 4 to 7.

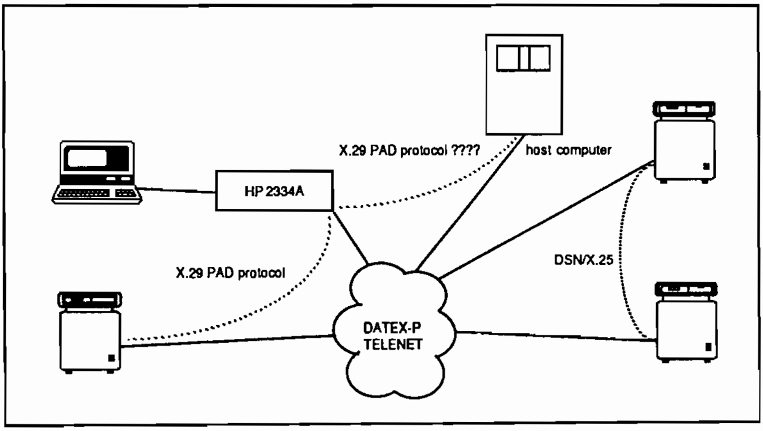


X.3-parameters

echo	● on ○ off
auto LF	● on ○ off
XON/XOFF	○ on ● off
break	● on ○ off
speed	9600
chars/line	80
....	

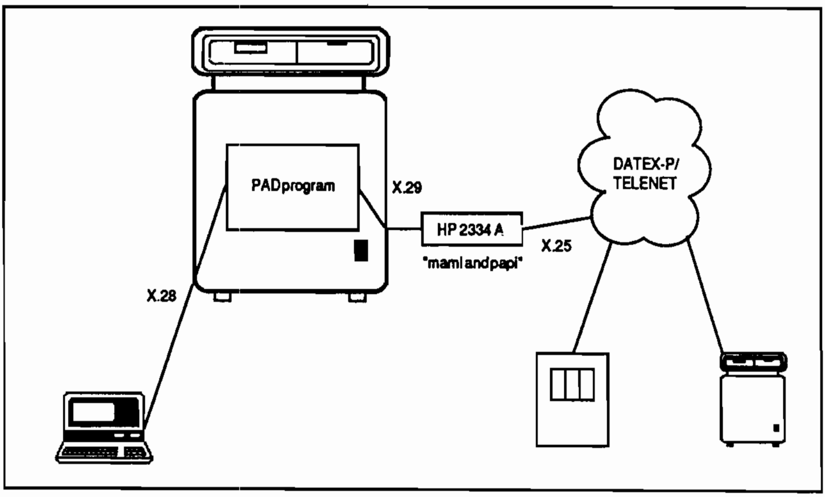
4. HP3000 and PSN

Available for HP3000: DSN/X.25 (using an HP-specific protocol on levels 4 to 7), X.29 PAD protocol which allows to run PAD sessions in MPE

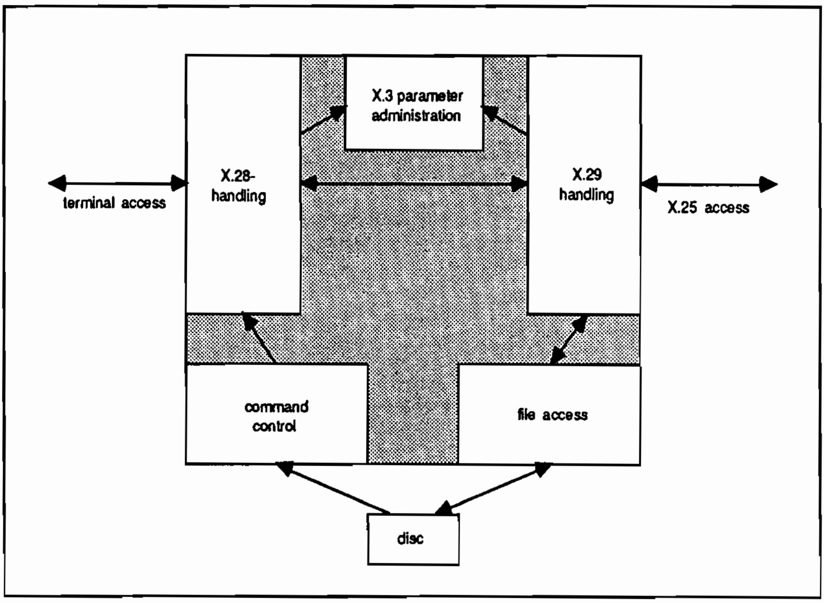


5. Structure of a PAD Program

A PAD program would enable access to any host computers which support the X.29 PAD protocol.



Introducing special stuctures a PAD program could be used for much more than just running a session on a remote computer: As long as there is no international standard for file transfer, PAD could perform this function temporarily. Another interesting feature would be to 'predefine' PAD sessions so that they can be run in a job stream.



Biography

Joerg Groessler is founder and general manager of the Joerg Groessler GmbH company in Berlin, West Germany. Since 1980 he designs and develops software tools for HP3000 and works as a consultant in special technical issues (e.g. systems performance, special capabilities). After having got in touch with an HP computer back in 1971 (HP2114) he now has over 10 years of experience in HP3000 programming.

SYNCHRONOUS CAUSES AND EFFECTS

SALLIE KAY STODGHILL
AMFAC DISTRIBUTION CORPORATION
81 BLUE RAVINE ROAD
FOLSOM, CALIFORNIA 95630
916-985-5000

JACK HYMER
HEWLETT-PACKARD
15815 SE 37TH ST.
BELLEVUE, WASHINGTON 98006
206-643-4000

****

The synchronous communications used for high speed, long distance data communications are adversely affected by imperfect transmission channels. These channels, made up of modems or their equivalent (DCE) and transmission facilities (leased lines, dial-up lines, microwave links, unloaded metallic lines, fiber-optics, etc) introduce sporadic errors which are usually detected by the communications protocol in use, causing re-transmission of at least the block in error. These channels also introduce delays in the transmission of data, which, though they may be quite small, prevent full utilization of the apparent channel speed.

This paper examines the causes of these errors and delays, their measurement and their effects on point-to-point communications links. MODEM test and selection criteria are presented with emphasis on multiple parametric testing.

OUTLINE

Section I. Throughput over point-to-point links

- A. Definition of throughput
- B. Factors affecting throughput
  1. MODEM (DCE) speed
  2. Link error rate
  3. MODEM turnaround time
  4. Block size
  5. Protocol
     - a. Half-duplex protocol
     - b. Full-duplex protocol
     - c. Protocol overhead characters
  6. Path length
  7. CPU/Interface servicing time
- C. Calculation of throughput
- D. Results

Section II. MODEM (DCE) test and selection

- A. MODEM economics
- B. Measurement of MODEM quality
  1. Live link testing
  2. Impairment distribution on US Bell System
  3. Simulated line testing
- C. Comparative analysis
- D. Results

Section I.

A. Definition of throughput

$$\text{Throughput} = \frac{\text{Number of bits correctly transmitted}}{\text{Time to correctly transmit the bits}}$$

The unit of measure used to express throughput here will be bits-per-second, abbreviated BPS.

B. Factors affecting throughput

1. MODEM (DCE) speed

In the U.S. synchronous MODEMS (Data Communications Equipment - DCE) are available at speeds ranging from 1200 to 230,400 BPS. MODEM equivalents (CSU/DSU or ISU) are available for use on strictly digital facilities at speeds of 2400, 4800, 9600, 19200 (with duo-plexer) and 56000 BPS.

Devices which attach to these DCE, namely Intelligent Network Processors (INPs), Synchronous Single Line Controllers (SSLCs), cluster controllers and multiplexers, etc. normally receive bit timing information from the DCE (i.e. when to send a bit or when a bit may be sampled for received data) as opposed to asynchronous equipment where the transmitting and receiving devices (Data Terminal Equipment - DTE) pace the communications rate based on internal clocking.

As a parameter taken alone, the faster the DCE speed, the higher the throughput in direct proportion.

2. Link error rate

The imperfection of a data communications link is expressed as:

$$\text{Error rate} = \frac{\text{Bits in error}}{\text{Bits transmitted}}$$

or

$$\text{Error rate} = \frac{\text{Blocks in error}}{\text{Blocks transmitted}}$$

The most common error rate abbreviations are BER for the bit error rate and BLER for the block error rate where a block is normally 1000 bits.

To measure the error rate of a channel, the DTE at the ends of the point-to-point link are replaced by Bit Error Rate Test set (BERTs). A BERT is capable of simultaneously transmitting and receiving a pseudo-random bit stream (PRBS) of fixed length, usually 63, 511, 2047 or 4095 bits. Longer tests are accomplished by simply repeating the fixed length PRBS. Receiver synchronization takes place in a period set by the binary root of the PRBS, (6 bit-times for 63-bit PRBS, 9 bit-times for 511-bit PRBS, 11 bit-times for 2047-bit PRBS, etc.) so it is not necessary to be terribly precise about starting the test at both ends of the line at exactly the same time.

BERTs may be set to transfer a fixed number of bits (1000 bits - 10^3, 10,000 bits - 10^4, 100,000 bits - 10^5, etc. to 1,000,000,000 bits - 10^9), may be set to transmit for a fixed period (5, 10 and 15 minutes are commonly used) or may be set to transmit continuously. By using a fixed number of bits, the error rate may be expressed independently of the line rate and test duration.

The link error rate is due to the combined imperfections in the communications facility and the DCE connected to the facility. Impairments which affect the error rate are composed of two types: Steady State and Transient.

Steady State Impairments

- Attenuation (Amplitude) Distortion
- Background Noise
- Frequency Shift (Offset)
- Envelope (Delay) Distortion
- Phase Jitter
- Non-Linear Distortion

Transient Impairments

- Impulse Noise
- Gain Hits
- Phase Hits
- Dropouts

Steady state impairments appear as random errors in error rate testing while transient impairments show up as bursts of errors.

The causes, measurement and acceptable limits of error rates and line impairments are covered in the Bell System technical publications 41004 through 41009 and in the Hewlett-Packard manual "Data Communications Testing", part number 5952-4973 chapters 2 and 3.

3. MODEM turnaround

On all half-duplex (HDX, two way non-simultaneous) links and on full-duplex (FDX, two way simultaneous) links (point-to-point as well as multipoint) some time is required for the receiving DCE to synchronize with the transmitting DCE. To restrain the transmitting DTE from sending data during this synchronizing (training) period, the transmitting DCE provides a delay between the time the transmitting DTE turns Request-To-Send (RTS) ON and the time when the transmitting DCE turns Clear-To-Send (CTS) ON. This time period varies from about 7 milliseconds on short, slow speed circuits to over three seconds on long, high-speed circuits. Common values fall in the range of 7ms (ATTIS Model 201C FDX private line, switched carrier), 12-15 ms (fast-poll/ fast-train modems), 50 ms (ATTIS Model 208B with "50" switch pushed in on short dial-up lines) to 148-150 ms (ATTIS Models 201C and 208B dial-up lines with normal settings).

On FDX channels the RTS-CTS delay time will be incurred only at link establishment if one selects the constant-carrier mode for the DCE. Then, for the purposes of this discussion, the turnaround time may be considered to be 0.

Improper configuration of the MODEM and/or Communications controller (INP or SSLC) transmission mode may adversely affect throughput by causing the modems to re-synchronize on each and every transmission when in fact this is not required. For example, on a full-duplex circuit, if the MODEM is strapped for switched carrier and the INP or SSLC is set to Transmission Mode= 1 (TM=1 under CSDEVICES) each transmission will be subject to an RTS/CTS delay which is neither necessary or desirable.

4. Block Length

As the block length of the transmitted data block is increased, the number of protocol overhead characters becomes a proportionally smaller fraction of the overall block transmitted. However, as the block length is extended, the probability that an error will occur is increased.

Block length is a parameter of throughput over which an HP3000 user has some control. The parameter "Preferred Buffer Size" used to configure the communications controller sets the default block length (excluding protocol characters) in words, with maximum sizes of 1024 words (2048 bytes) on the INP and 4095 words (8190 bytes) on the SSLC. Communications subsystems may override the default settings as follows:

a. RJE

RJLINE — MAXRPB parameter sets number of records per block to be transferred. Size of block is the size of the records times the number of records per block.

RJIN — COMPRESS parameter is used to prevent the transmission of EBCDIC blanks or ASCII spaces within each record.

TRUNCATE parameter is used to prevent the transmission of EBCDIC blanks or ASCII spaces at the end of each record.

RJOUT — OUTSIZE parameter sets the length of the data records to be received.

b. Bisync DS — Configuring the monitor, IODSO as subtype 0 will cause DS to transmit data in uncompressed format while subtype 1 will cause DS to compress transmitted data. Subtype 1 is recommended below 56000 BPS.

DSLINE — LINEBUF parameter sets the maximum size of the transmitted data block in the range of 304 to 1024 words (608 to 2048 bytes) if an INP is being used or 304 to 4095 words (608 to 8190 bytes) if an SSLC is being used.

COMP parameter overrides the system configured default turning compression on.

NOCOMP parameter overrides the system configuration turning compression off.

c. X.25 DS — Configuring the monitor, IODSXO as subtype 0 causes DS to transmit uncompressed data while subtype 1 causes DS to transmit compressed data. Subtype 1 is recommended below 56000 BPS.

NETCONF Line Characteristics Table: The PACKET SIZE parameter set the maximum number of data bytes in a packet in the range of 32 to 1024 bytes.

d. MTS

The maximum number of characters to be transmitted in one write is 4096 (SSLC only).

The maximum number of characters to be received in one read is 2048.

Writes to peripheral devices attached to MTS terminals should be treated very carefully. Since the attached device may use a transfer rate that is lower than the communications line rate, checking transfer status after writing each record should be avoided because the status won't be available until after the transfer to the peripheral has been completed or interrupted and the status won't be returned to the user's program until the group/device is next polled. It is faster but slightly less secure to write several records in a block (programmer is controlling block length here) and then checking transfer status.

Other subsytems (Bisync/SDLC, IMF, MRJE, NRJE and SNA/IMF override the default buffer size parameter but are dependent on the host/FEP (Front End Processor) configuration parameters.

5. Protocol Dependencies

a. Half-duplex protocols.

Half-duplex protocols require an acknowledgment block to be returned for each data block transmitted. Only the data block in error will be re-transmitted although there may be some additional protocol overhead when data blocks are not perceived to be in error (i.e. when the acknowledgment is withheld or is lost). The time required for these relatively infrequent occasions will not be a part of this paper.

Two examples of half duplex protocol are Bisync (BSC - Binary Synchronous Communications), used on any type of communications facility, and SDLC (Synchronous Data Link Control), used on multipoint facilities (for example SDLC/IMF, SNA/IMF and NRJE).

b. Full-duplex protocols.

Full-duplex protocols require positive acknowledgments only when the transmit window size is reached. Negative acknowledgments indicate the number of the frame received in error (or not received at all) and require the re-transmission of not only the frame in error but also each frame transmitted after the frame in error. On paths with little delay this normally involves only the transmission of 2 frames (the frame in error and the frame following) but in paths with large delays it may be necessary to transmit 3 or 4 or more frames to correct the error and continue the transmission.

Examples of full duplex protocol are HDLC and its subsets SDLC and LAP/ LAP-B (used for X.25).

It should be further noted that higher levels in the communications subsystem may degrade throughput even more by requiring end-to-end acknowledgments for each packet. DS/X.25 uses the "D" bit ON, requiring an end-to-end packet acknowledgment when used with Public Data Networks (PDNs).

c. Protocol overhead

The addition of protocol characters for error detection, addressing, control information, etc. adversely affects throughput. For Bisync, approximately 8 characters are added per block (4 sync characters, STX, ETB/ETX, 2 block check characters). HDLC adds between 6 bytes (2 flags, address octet, control octet, 2 frame check octets) and 7 bytes (2 control octets are used with window sizes between 8 and 127) plus bit-stuffing bits depending on the content of the data.

The exact number of sync characters sent in Bisync can be obtained from the CSTRACE Information Display in the DOPTIONS bits 14:2 as follows:

0= Send 4 Sync bytes
1= Send 8 Sync bytes
2= Send 12 Sync bytes
3= Send 16 Sync bytes

Higher levels in the full-duplex protocols add additional overhead in the for of message headers for each level, the content and length of which are beyond the scope of this paper.

6. Path length

Signal propagation through free space is approximately 186,000 miles (300,000,000 meters) per second or, inversely, 5.4 microseconds per mile (3.3 microseconds per kilometer). Since not all of the communications path passes through free space, a longer transit time is imposed on signals. A common value used for propagation is 1 millisecond per 100 miles of actual circuit path (not straight line mileage) which is about double the free space transit time.

When a satellite is encountered in a communications path, an additional delay of 250 to 300 milliseconds (earth-station to earth-station) transit time must be added to the overall delay due to circuit delay.

7. CPU/Interface servicing time

The time required for servicing (generating an acknowledgment or starting the next transmission) in the CPU/Communications Controller may be quite variable. Interface response time is small compared to the delays introduced by the modems and line paths and will be ignored here. CPU response time is dependent on parameters outside of the scope of this paper and will also be ignored.

C. Calculation of throughput

A model for the throughput of a link including the first 6 items above becomes:

$$\text{Throughput} = \frac{I * L * (1-P)}{((L+O) * T/S) + D) * (1-P) + (N * P)}$$

where : D = Delay between block transmissions
I = number of Information bits per character
L = Length of data block in characters
N = Number of blocks to be re-sent on error
O = number of Overhead characters per block
P = Probability of error in a block
S = Modem speed in BPS
T = Total number of bits per character

Assumptions: Lost blocks and lost acknowledgments are ignored.
Errors are single bit errors (worst case)

Evaluations:

Probability of errors in a block

$$P = 1 - (1 - E)^{((I + O) * T)}$$

where E = Link error rate

Values often used for E are:

Analog lines 10^{-5}
Digital Lines 10^{-6}

For existing lines the actual value of E may be measured with a Bit Error Rate Test set as described above.

An alternate method of obtaining the probability of error when the error rate of a dedicated link is not known is as follows:

1. :SHOWCOM NN;RESET at location A

2. Send 1000 fixed length blocks with the communications subsystem at hand from location A to location B.

3. :SHOWCOM NN;ERRORS at location A. The probability of error on the link in the direction from location A to location B is:

$$P = \frac{\text{Retransmissions}}{\text{Messages Sent}}$$

as long as there are no response timeouts indicated.

Delay between blocks

The delay between blocks is the sum of:

1. The propagation delay from source to destination
2. The RTS/CTS delay of the destination DCE
3. The time required to send the acknowledgment which is:

$$D = \frac{X * T}{S}$$

where:

T = Total bits per character
X = Number of characters in the acknowledgment block
S= Line speed in BPS

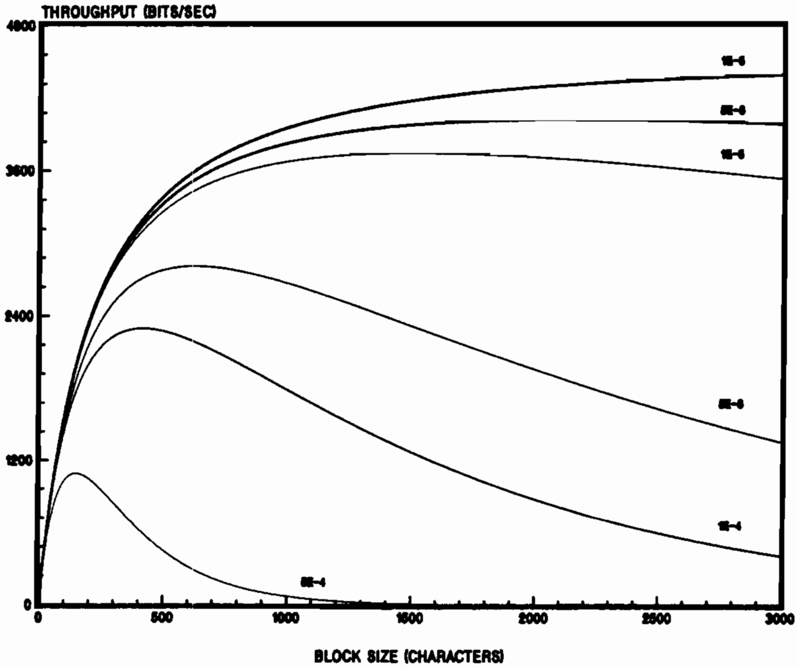
4. The RTS/CTS delay at the source DCE

D. Results

Results are presented below choosing block size as the independent variable because block size is the parameter most easily controlled by a computer user.

Figure 1 Throughput vs Line Error Rate
Figure 2 Throughput vs Modem/Line Type (Error Rate 1E-5)
Figure 3 Throughput vs Modem/Line Type (Error Rate 5E-5)
Figure 4 Throughput vs Propagation
Figure 5 Throughput vs Protocol

THROUGHPUT VS LINE ERROR RATE



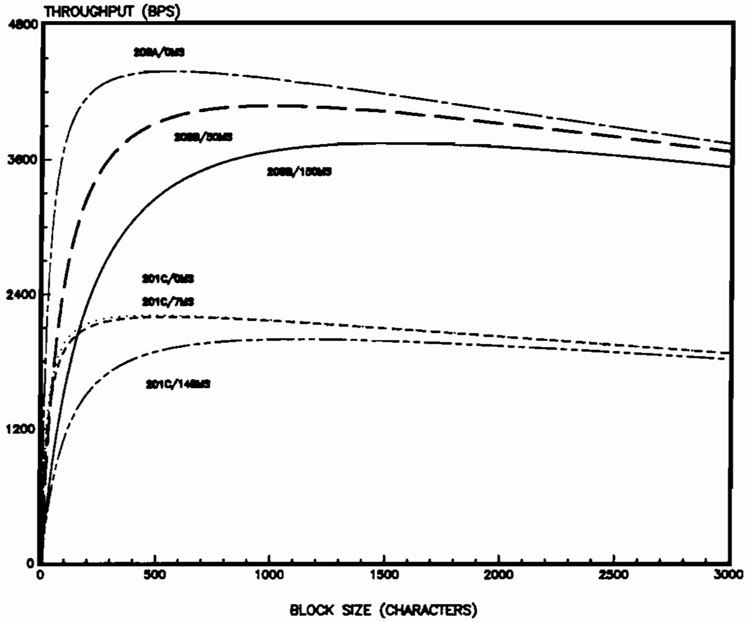
TEST CONDITIONS:

Data Bits/Char	8
Total Bits/Char	8
Overhead Char/Block	6
Modem Speed	4800 bits/sec
RTS/CTS Delay	150 MS
# Blocks Resent on Error	1
Length of Circuit	1000 Miles
Length of ACK Block	6

Figure 1

THROUGHPUT VS MODEM/LINE TYPE

ERROR RATE 1 IN 10^{-5}



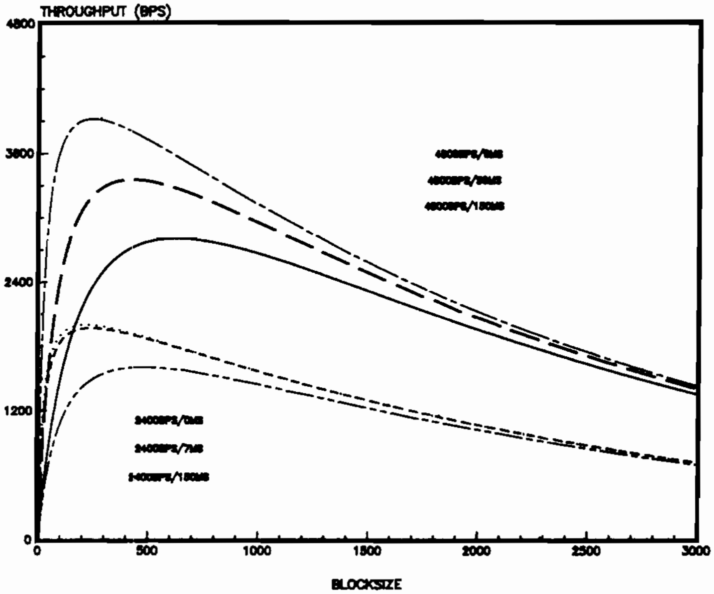
TEST CONDITIONS: (Error Rate 1E-5)

Data Bits/Char	8
Total Bits/Char	8
Overhead Char/Block	8
Blocks to Resend	1
Length of Circuit	1000 Miles
Length of ACK	6 Char

Figure 2

THROUGHPUT VS MODEM/LINE TYPE

ERROR RATE 5 IN 10^{-5}

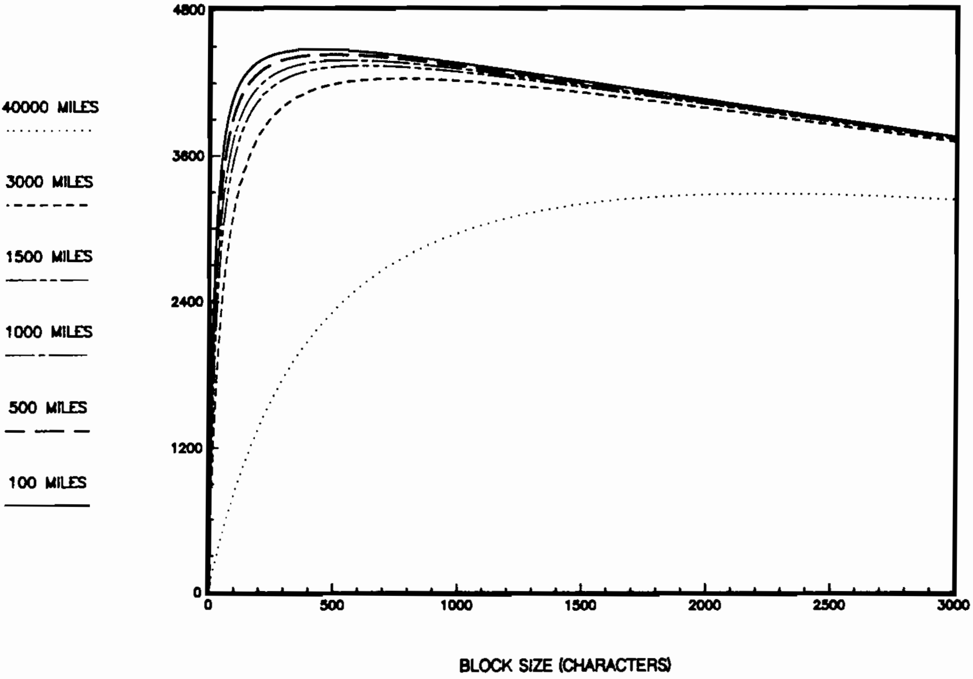


TEST CONDITIONS: (Error Rate 5E-5)

# Data Bits/Char	8
Total Bits/Char	8
#Overhead Char/Block	8
#Blocks to Resend	1
Length of Circuit	1000 Miles
Length of ACK	6 Char

Figure 3

THROUGHPUT VS PROPAGATION

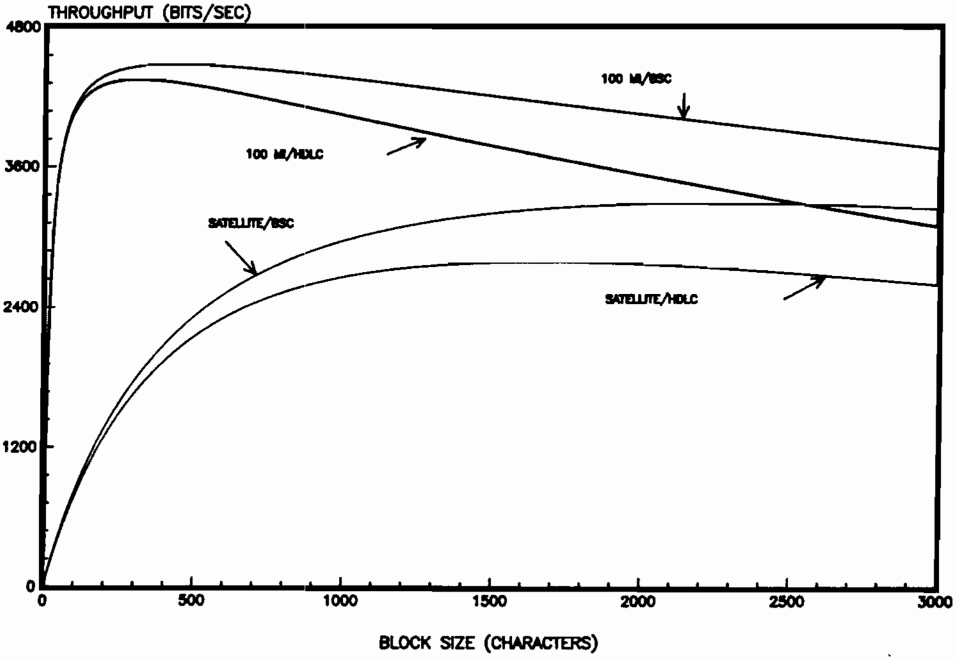


TEST CONDITIONS:

Error Rate	1E-5
Data Bits/Char	8
Total Bits/Char	8
Overhead Char/Block	8
Blocks to Resend	1
Length of ACK	6 Char
Speed	4800 Bits/Sec
RTS/CTS	0 MS

Figure 4

THROUGHPUT VS PROTOCOL



TEST CONDITIONS:

Line Speed	4800 Bits/Sec
RTS/CTS	0 MS
Length of ACK	8 Char
Data Bits/Char	8
Total Bits/Char	8
Error Rate	1E--5
Overhead Char/Block	8 Char

Figure 5

Section II.

A. MODEM Economics

Currently available synchronous modems vary widely in price (from under $300.00 to over $15,000.00), in speed (1200 to 56,000 BPS), in features (no frills to modems with internal 16-bit processors, network management, built-in diagnostic test equipment, keyboards, displays .. even modems that know their own serial numbers) and in support (send it back to the factory, spare-in-the-air, on-site same day service).

A method of normalization is suggested here: Select a minimum set of required features and compute **THROUGHPUT vs. PRICE**.

As can be seen from the graphs in section I, the error rate of the DCE and line combination has a pronounced effect on throughput (Figure 1). If a half-duplex MODEM is required, synchronization/training time becomes a factor; however, this is a figure which may be obtained directly from the manufacturers data sheet.

Error rate, on the other hand, while it may be mentioned, is not usually accompanied by much supporting data. Asking the vendor "What were the measurement conditions - signal-to-noise ratio, amplitude distortion, phase jitter, etc.?" or "Were the impairments used in testing this MODEM applied one at a time or if applied in combination, what were the combinations?" usually elicits a vague response of "Hmmm, uh, I guess I'd have to call the factory for that information but our company tests our MODEMs real well," if there is any response at all.

In addition to the MODEM error rate question above, how does the communications facility measure up? Will a conditioned line be required? Should a digital facility be considered?

The answers to these questions can be obtained by a judicious choice of how to test the candidate MODEMs.

B. Measuring Modem Quality

1. Live link testing

Probably the simplest method of determining the quality of a particular model of MODEM is to connect the candidate MODEM pair to an existing line and comparing the throughput of the candidate pair with that of the pair previously in use by applying one of the error rate test methods noted above.

This method has severe limitations in that it tests the candidate MODEM pair under only one set of line conditions, namely that set of conditions existing at the time of the test on the live circuit.

How can we tell how the MODEM will react under other conditions on other lines?

2. Impairment Distribution

Studies performed on the U.S. Bell telephone system between 1959 and 1970 (1970 is the latest survey for which results have been published) have provided information on the distribution of impairments to be found on a very large number of lines. A summary of these studies is included as Figure 6.

	PERCENT OF LINES									
IMPAIRMENT	10	20	30	40	50	60	70	80	90	95
ATTENUATION	C4	C2	C1	C1	C1	C1	C1	UN	UN*	
DELAY	C4	C2	C2	C2	C1	C1	C1	C1	C1	C1
SIGNAL/NOISE	43	41	40	39	38	36	34	33	28	27
FREQ. SHIFT	0	0	0	0	.1	.2	.4	.7	1.1	2
PHASE JITTER	2	3	3.5	3.8	4	5	6	7	8	9
2ND HARMONIC	47	44	42	41	39	37	36	34	31	29
3RD HARMONIC	45	44	43	41	39	36	35	34	32	31*

*UNCONDITIONED LINE LIMIT

Figure 6

In the first two rows of Figure 6, the designations C1, C2, C4 and UN refer to the leased line conditioning specifications. A summary of these specifications is provided as Figure 7.

	CONDITIONING LEVEL			
	UNCONDITIONED	C1	C2	C4
FREQ RANGE	300-3000Hz	3000-3000Hz	300-3000Hz	300-3200Hz
RESPONSE RANGE / DB VAR.	300-3000 / -3 TO +12 500-2500 / -2 TO +8	300-2700 / -2 TO +6 1000-2400 / -1 TO +3 300-3000 / -3 TO +12	300-3000 / -2 TO +6 500-2800 / -1 TO +3	300-3200 / -2 TO +6 500-3000 / -2 TO +3
DELAY DIST.(uS) VALUE / FREQ	<1750uS / 800-2600Hz.	<1000uS / 1000-2400Hz <1750uS / 800-2600Hz.	<500uS / 1000-2600 Hz <1500uS / 600-2600Hz <3800uS / 500-2800 Hz	<300uS / 1000-2600Hz <500uS / 800-2600Hz <1500uS / 600-3000Hz <3000uS / 500-3000Hz
IMPULSES	15 COUNTS / 15 MIN.	15 COUNTS / 15 MIN.	15 COUNTS / 15 MIN.	15 COUNTS / 15 MIN.

Figure 7

Please note that in reading Figure 6, the entries represent that a line in a particular percentile column will have impairments no greater than the amount shown.

The problem now is to find a hundred or so lines to sample that fall into the summary chart within the percentage of confidence that we'd like to have in our choice of MODEM and then test our candidate MODEM pair on each of these lines.

Fortunately there is a better solution and it might be designated ' multiple parameter simulation ' for want of a shorter name.

3. Simulation testing

The simulation testing method is quite simple. Only three pieces of equipment are required:

1. A candidate MODEM
2. A Bit Error Rate Test set (BERT)
3. A line simulator.

The test procedure consists of completing the following steps:

1. Connect the output of the candidate MODEM to the input of the line simulator.
2. Connect the input of the candidate MODEM to the output of the line simulator.
3. Connect the BERT to the candidate MODEM.
4. Power up all of the equipment.
5. **SET UP THE LINE SIMULATOR FOR THE APPROPRIATE TEST CONDITIONS**.
6. Start the BERT.
7. Record the results.
8. Repeat steps 5, 6 and 7 a few hundred times.

The key to success here, of course, is knowing how to set up the line simulator.

C. Comparative testing

Examining the ranges of the parameters in Figures 6 and 7 and translating them into combinations useful for the limited number of tests which can be run economically is not difficult. Combinations of impairments might be chosen so that the tests run are statistically representative of the universe of actual circuits but that is another thing that is beyond the scope of this paper. Let us choose a set of tests that are useful in COMPARING the performance of our candidate MODEMS under varying line conditions by subjecting them to test conditions which are likely to occur on a large percentage of available lines.

The error rates determined in these tests will lead to the

THROUGHPUT VS PRICE

used to select the most economic unit.

Figure 8 shows a mapping of the digits 0 through 9 to the quantized impairment levels which were not exceeded in 95% of the lines tested in the 1969/1970 U.S. Bell System survey.

	RANDOM NUMBER									
IMPAIRMENT	0	1	2	3	4	5	6	7	8	9
ATTENUATION	C4	C2	C1	C1	C1	C1	C1	UN	UN*	UN
DELAY	C4	C2	C2	C2	C1	C1	C1	C1	C1	C1
SIGNAL/NOISE	43	41	40	39	38	36	34	33	28	27
FREQ. SHIFT	0	0	0	0	.1	.2	.4	.7	1.1	2
PHASE JITTER	2	3	3.5	3.8	4	5	6	7	8	9
2ND HARMONIC	47	44	42	41	39	37	36	34	31	29
3RD HARMONIC	45	44	43	41	39	36	35	34	32	31*
IMPULSE NOISE	-18	-16	-14	-12	-10	-8	-6	-4	-2	0

*UNCONDITIONED LINE LIMIT

Figure 8

To determine the parametric combinations necessary to set up the line simulator simply generate a series of random numbers of N digits (one digit for each type of impairment covered by the line simulator) and apply each of the digits to a corresponding parameter type and level.

For example, if the random number is **42633409** the line simulator setup might be:

digit	parameter	level
4	Attenuation Distortion	C1
2	Envelope (delay) Distortion	C2
6	Signal-to noise ratio	34 dB
3	Frequency Shift (offset)	0 Hz.
3	Phase Jitter	3.5 Degrees
4	Second Harmonic Distortion	-39 dB
0	Third Harmonic Distortion	-45 dB
9	Impulse Noise	0 dB.

Impulse noise repetition rate should be set to create 15 counts per 15 minutes of test duration.

If a bit error rate test of 10^6 bits is to be performed on a 4800 BPS MODEM each test setup will require about 3.5 minutes (10^6 bits / 4800 bits per second) to complete. Allowing 2.5 minutes to note the end of test, record the result and set up the next test, shows that 100 test will take about 10 hours. If you were to test all of the 8 impairment types with the ten levels shown in Figure 8, the time required would be approximately 190 years neglecting time out for coffee breaks.

D. Presenting the results

Since each of the 100 or so tests performed above has an individual result, comparing candidate MODEMs on a test-by-test basis can still be difficult. However, if one simply multiplies the number of bits per test by the number of test performed, subtracts the total number of errors and divides by the number of tests times the time per test, the results will be:

$$\text{Throughput} = \frac{\text{Number of bits correctly transmitted}}{\text{Time to correctly transmit the bits}}$$

It is now a simple matter to **COMPARE** candidate MODEMs using

THROUGHPUT VS PRICE

****

Biography

Sallie Kay Stodghill has been with AMFAC Distribution for just over 1 year during which she has participated in the design of AMFAC's network of over 100 HP3000 computers. She is currently in the process of installing and managing this network. A graduate of Mills College in Oakland, California, with degrees in mathematics and computer science, Mrs. Stodghill has been employed by both Hewlett-Packard and Tandem Computer Co. in the field of data communications for .... years.

Jack Hymer has been employed by Hewlett-Packard for 10 years in the Bellevue, Washington Sales and Service office. He is currently a Network Consultant. Mr. Hymer graduated from the University of Washington in 1973 with a BSEE after working as a communications technician for 9 years.

NETWORK DESIGN FOR A DISTRIBUTOR

JACK HYMER
NETWORK CONSULTANT
HEWLETT-PACKARD
15815 S.E. 37TH STREET
BELLEVUE, WA 98006
(206) 643-4000

****

This paper describes the business needs analysis, traffic study, network design and alternative options analysis used at AMFAC Distribution to consolidate three separate operating divisions' data communications requirements in one network of approximately 120 machines.

Bisync autodial DS, X.25 private network and terminal/multiplexer solutions integration methods are presented. Network management and operation problems are discussed. Use of commercially available network design software in the design of this network is discussed. Analog and digital cost models are covered.

OUTLINE

Section I. The Opportunity

A. AMFAC Distribution
   1. Operating Divisions Organizations
   2. Business Volumes
   3. Geographic Distribution
B. Applications to be Supported
   1. SFD (Systems for Distributors)
   2. MTI (Management Technology Incorporated)
   3. ADI (American Data Industries)
   4. HPDESKMANAGER
C. Information Flow and Traffic Volumes
   1. Electric
   2. Industrial and Plumbing (I&P)
   3. Health Care
D. Restraints and Requirements

Section II. Alternatives

A. Topologies
   1. Central CPU(s) with point-to-point multiplexer links
   2. Central CPU(s) with multipoint links
   3. X.25 Public and Private Networks
   4. Dialup

B. Response Time Modeling
    1. Connections Software
        a. Input data
        b. Tariffs
        c. Output data
    2. Results
C. Considerations
    1. Analog vs Digital lines
    2. Private vs Public X.25 Networks

Section III. Recommendations

A. General
B Electric
C. I&P
D. Health Care

Section IV. Network Management

A. Available Types and Selection
    1. Analog Lines
    2. Digital Lines
    3. Multiplexers
    4. X.25 Switches

Section V. Summary

****

Section I. The Opportunity

A. AMFAC Distribution Group

AMFAC, Incorporated is a New York stock exchange listed diversified service company with interests in wholesale distribution, food processing, retailing, hotels and resorts, sugar and land development and management.

AMFAC Distribution Group is the largest of the AMFAC, Incorporated core business, generating revenues of more than 1.3 billion dollars in 1985 from three specialized segments of the US wholesaling sector. AMFAC Distribution Group is located in Folsom California, 15 miles east of Sacramento.

AMFAC Electric Supply has 87 branches in 16 states marketing a full line of wire, cable, conduit lighting, switchgear and other electrical products to the residential and commercial construction industry, dealers, utilities and government customers. AMFAC Electric ranks fourth in sales within the US electric wholesaling industry.

The AMFAC Electric Supply business is organized as 7 regions located in California(2), Hawaii, Washington, Utah and Texas(2), serving branch locations in up to 5 states from each region office.

AMFAC Industrial and Plumbing Supply (I & P) serves 24 states through 156 branches distributing pipe, valves, fittings, plumbing supplies, building materials, fluid power and industrial supplies to construction, utility industrial and government customers. AMFAC I & P has the largest geographical coverage of any US mechanical supplies distributor.

The AMFAC I & P business is organized as 5 regions located in California(2), Colorado, Texas and Washington serving 18 geographical Market Areas composed of branches located in up to 4 states per Market Area.

AMFAC Health Care serves 47 states through 57 branches distributing ethical pharmaceuticals and selected over-the-counter items to independent drug stores, retail drug chains and hospital pharmacies.

The AMFAC Health Care business is organized as 4 regional accounting centers located in California, Indiana, Texas and Washington serving the individual branch offices and reporting to Folsom.

B. Application Support

Application software system selected by AMFAC for distribution control were Hewlett-Packard's Systems For Distributors (SFD) for the Mechanical division, Management Technology Incorporated (MTI) for the Electric Division and American Data Industries (ADI) for the Health Care division. These applications differ in their terminal handling characteristics in that character mode (Health Care's ADI software), V/Plus Block mode (Health Care's ADI software and Electric's MTI software) and Data Entry Library (DEL) Block mode (Mechanical's SFD and Electric's MTI software) are used.

The significant limitation in the terminal handling methods to be used is that DEL Block mode is not supported in an X.25/Packet Assembler/Disassembler (PAD) environment, eliminating X.25 as a consideration in terminal connection for the Mechanical and Electric divisions.

Analysis of the actual screens used by the three applications showed CPU character counts ranging from 4 to 7055 characters and CPU input character counts of 1 to 609 characters.

As CPU output character counts exceed about 400 characters, delays encountered in the communications link can cause perceptible screen-writing time degradation. Above about 2000 characters these delays may render screen-writing time unacceptable.

For example, a 2000 character screen requires 2000/80=25 ENQ/ACK handshakes. At 9600 BPS direct connect this screen would require:

$$\frac{\text{2000 data characters + 25 ENQs + 25 ACKs}}{\text{960 characters/second}} = \frac{2050}{960} = 2.14 \text{ seconds}$$

For a single user, a statistical time division multiplexer (STATMUX) with a terrestrial 9600 BPS composite link rate and no ENQ/ACK spoofing generates a maximum screen-writing time of approximately:

$$T(in) = \frac{2025}{960} = 2.11 \text{ sec}$$

Time to put characters into CPU side MUX at 9600 BPS.

+

$$T(xmit) = \frac{2300}{1200} = 1.92 \text{ sec}$$

Time to transmit characters to remote MUX with 10 overhead characters per MUX frame.

+

$$T(out) = \frac{2025}{960} = 2.11 \text{ sec}$$

Time to move characters from remote MUX to remote terminal at 9600 BPS.

+

$$T(acki) = \frac{25}{960} = 0.03 \text{ sec}$$

Time to move ACK from remote terminal to remote MUX

+

$$T(ackx) = \frac{275}{1200} = 0.23 \text{ sec}$$

Time to move ACKs from remote MUX to CPU MUX with 10 overhead characters per frame

+

$$T(acko) = \frac{25}{960} = 0.03 \text{ sec}$$

Time to move ACKs from CPU mux to CPU

$= 6.41$ seconds

While use of a STATMUX with a terrestrial 9600 BPS composite link rate and ENQ/ACK spoofing might theoretically produce a screen-writing time close to that of the direct attached terminal with one user, measurements indicated a screen-writing time of 3.1 to 4.4 seconds could be expected with real multiplexers, modems and lines.

Due to the large screen requirements, a primary design goal was to implement terminal links with minimum delays.

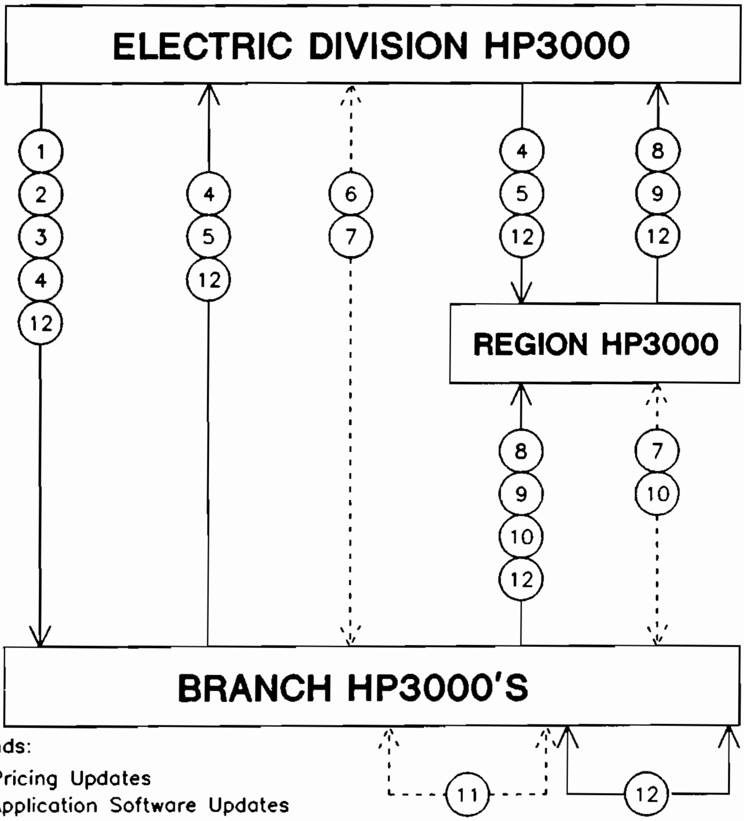
AMFAC expressed interest in using HPDESKMANAGER in the Distribution Group, creating an all-systems interconnection requirement to be considered during the network design process.

C. Information Flow and Traffic Volumes.

Figure 1 is a pictorial summary of the communications requirements for AMFAC Electric Supply division.

Figure 2 is a pictorial summary of the communications requirements for AMFAC I & P division.

Figure 3 is a pictorial summary of the communications requirements for AMFAC Health Care division.



Legends:

1 Pricing Updates
2 Application Software Updates
3 Conversion Files
4 Consolidated Purchasing
5 EIC Summary
6 Central Item Number Assignment
7 Stock Status Inquiries
8 General Ledger Account Balance Summary
9 Summary Database
10 Regional Accounting
11 Inter-Branch Ordering
12 Electronic Mail

——— Batch File Transfer
- - - - - - Interactive

Figure 1

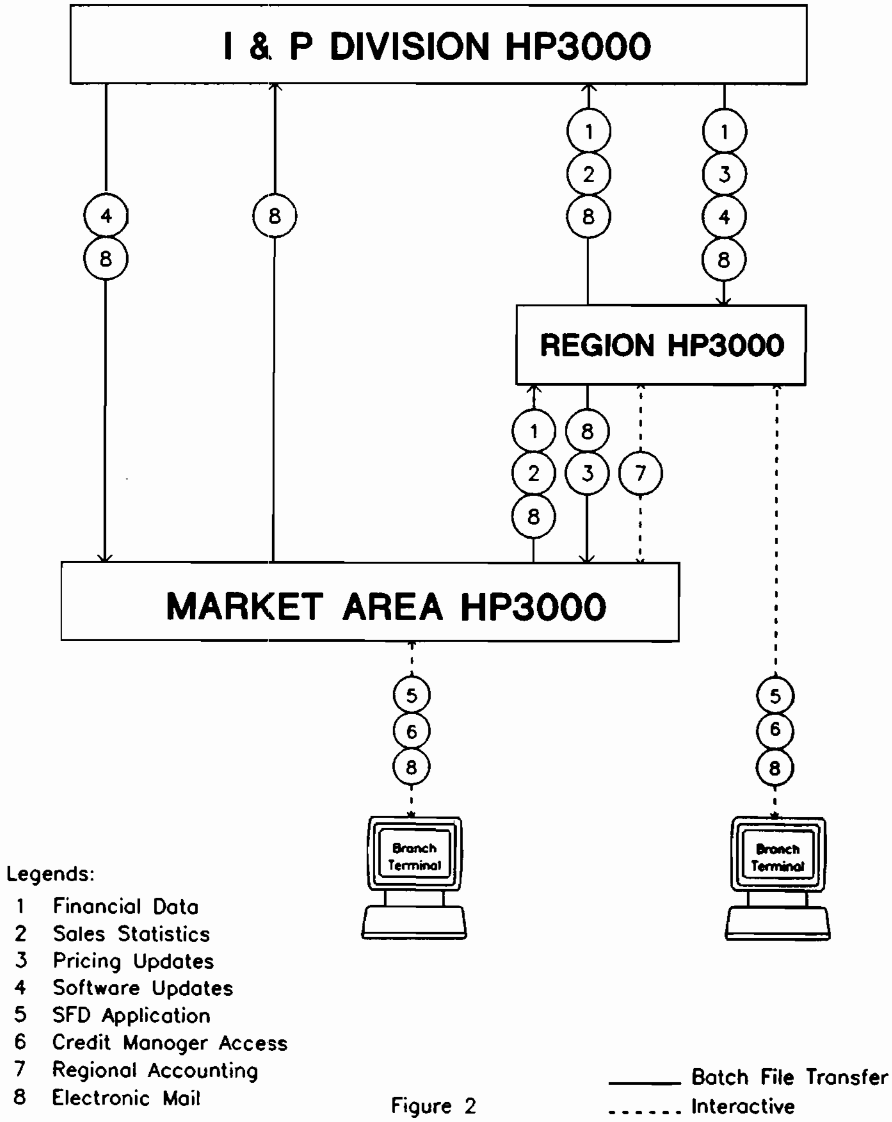
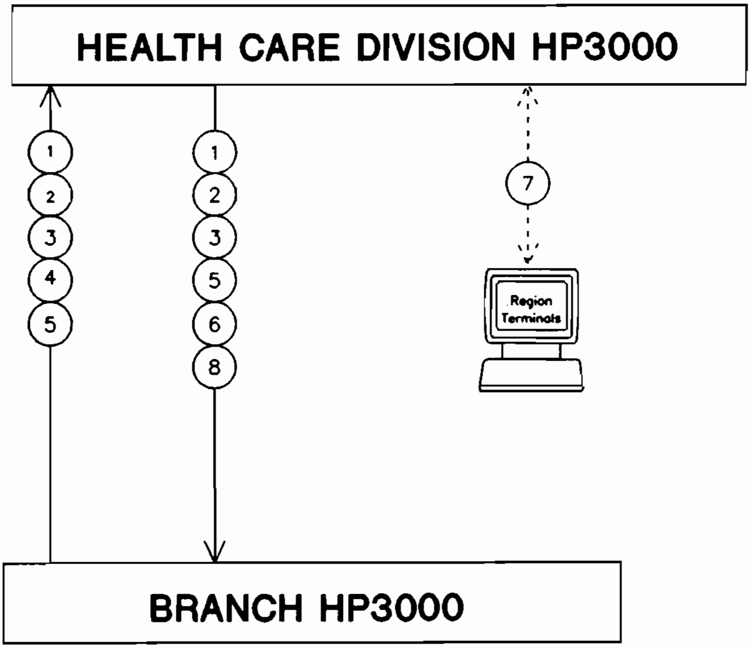

I & P DIVISION HP3000
REGION HP3000
MARKET AREA HP3000
Branch Terminal
Branch Terminal
Legends:
1 Financial Data
2 Sales Statistics
3 Pricing Updates
4 Software Updates
5 SFD Application
6 Credit Manager Access
7 Regional Accounting
8 Electronic Mail
Batch File Transfer
Interactive

Figure 2



Legends:

1 Centralized Payables
2 Centralized Purchasing
3 Marketing Data
4 Journal Entries
5 Inventory Management
6 Software Updates
7 Regional Accounting
8 Electronic Mail

Figure 3

______ Batch File Transfer
- - - - - - Interactive

Traffic volumes were collected for the information flow paths shown in the figures (where available) and were estimated where data was not available. collection techniques varied form obtaining AMFAC-prepared detailed information, to interview and data collection via prepared forms, to character counting with an HP4951A data analyzer on active applications terminals. Traffic volume information is quite extensive and will not be presented here unless pertinent to the network design activity being discussed.

D. Restraints and Requirements.

As in any real network, several restraints and requirements were imposed on the network design by the customer, AMFAC Distribution, Inc.
These restraints/requirements were:

1. Industrial and Plumbing
   Restraints:
   a. All CPU locations were pre-determined.
   b. Region, Market Area and Branch connectivity were predetermined.

   These restraints limited the application of network design algorithms which produce minimum cost networks.

   Requirements:
   a. Terminal response time less than 3 seconds average, less than 9 seconds 95th percentile.
   b. Batch communications complete overnight.

2. Electric
   Restraints:
   a. On topology - minimal within a region. Some small, terminal only branches were pre-determined.
   b. Region locations are pre-determined.

   Requirements:
   a. Terminal response time less than 3 seconds average, less than 9 seconds 95th percentile.
   b. Branches should have the capability to do inter-branch ordering
   c. Batch traffic completes overnight.

3. Health Care
   Restraints:
   a. CPU per branch.
   b. Region accounting locations pre-determined (no CPUs).

   Requirements:
   a. Batch traffic completes overnight.

Section II.

A. Topologies

Topology considerations were quite simple.

I & P:

I & P restraints and requirements dictated a terminal network with point-to-point lines and multiplexers for a supported configuration with minimal link delays and Region-to-Division batch links (dialup bisync DS).Many Region and Market Area HP3000's were co-located dictating a direct connection for DS between these machines. The only topology consideration, then, was the Region-to-Market Area cases where the Market Area HP3000 and the Region HP3000 were not co-located. The interactive regional accounting traffic dictated a leased line between these sites but the volume did not support 24 hour connectivitiy. A design goal of facility sharing between I & P and Electric networks was developed to permit I & P use of existing Electric leased facilities where possible to permit low volume interactive traffic for I & P regional accounting.

Health Care:

Health Care requirements and restraints dictated a dialup network using bisync autodial DS at all branch locations. The traffic analysis study indicated that communications for the largest branch would require 92 minutes per day and the smallest branch 21 minutes per day at 4800 BPS, half-duplex, 150 ms. turnaround delay or 143 and 33 minutes respectively at 2400 BPS, full-duplex, no turnaround delay on lines with an error rate of 1 in 10^5 bits. Daily and monthly data transfer costs were determined based on 11 PM to 8 AM AT&T direct dialing rates to do comparative costing of the two speed choices. Results were $7268 per month at 2400 BPS and $4699 at 4800 BPS, a difference of $2569 per month. For the 41 sites considered, a modem cost differential of $1500 per site could thus be accommodated for a 24 month purchase price payback from line cost savings. (24*$2569/41=$1503). A 4800 BPS Bell 208B compatible MODEM and Bell 801C compatible autodialer that met this cost criteria were selected and, to further increase savings, WATS lines were installed and 50 ms. turnaround was selected at central site MODEMS to be used with sites within 100 miles per the Bell PUB41211 paragraph 2.2 recommendation.

To verify the estimated transfer times for the largest branch, multiple tests at 2400 and 4800 BPS were performed with actual data. Results agreed to within 4 percent at 240 BPS and were exact at 4800 BPS.

Electric:

For the Electric division, 4 topologies were considered:

1. 1 or 2 Central CPUs within a region (large HP3000/Series 48 or small HP3000/Series 68) with point-to-point terminal multiplexer links to branch locations.

2. 1 or 2 Central CPUs within a region (large HP3000/Series 48 or small HP3000/Series 68) with multidrop links to terminal locations.

3. Distributed HP3000/Series 37s at branches, HP3000/Series 42 or 48 at Region locations. Non-cpu branches (twigs) connected via point-to-point multiplexer links. Fully interconnect CPUs within a region via private or public X.25 networks to limit INP requirements at each site.

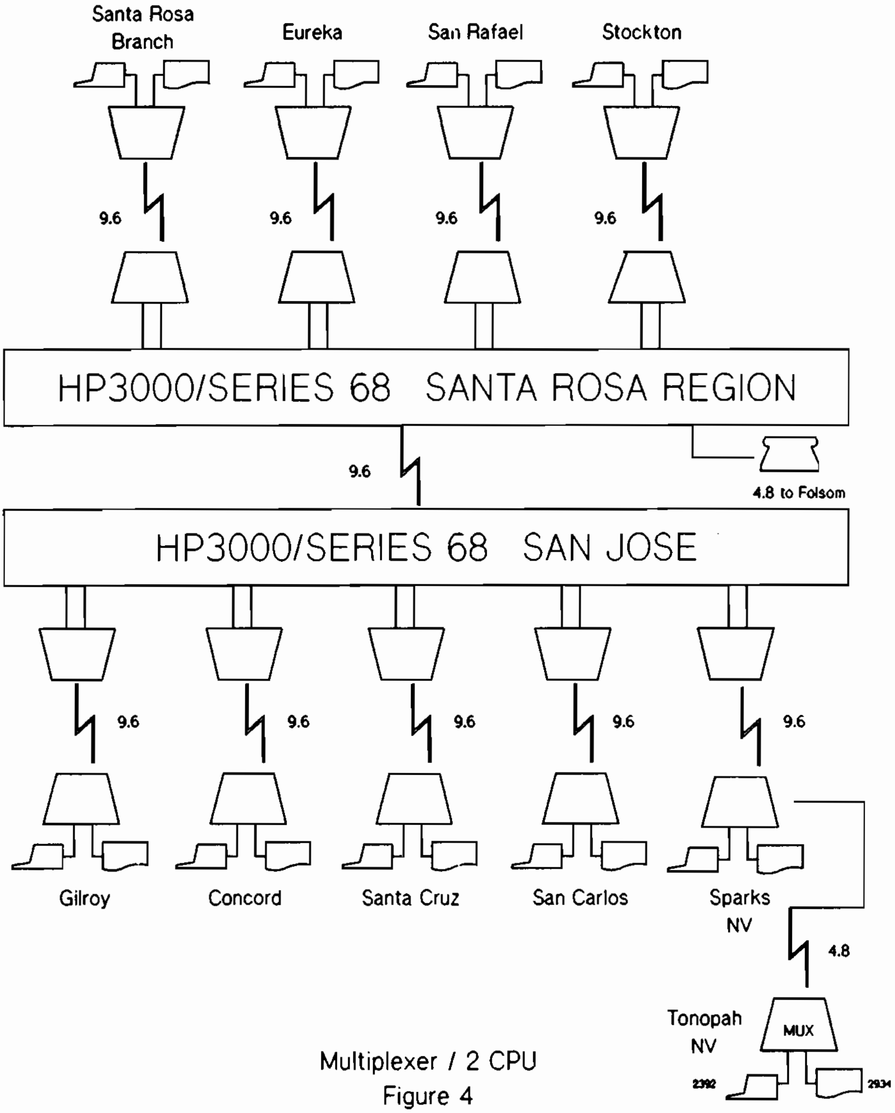
4. Autodial DS between systems.

Item 4 was discarded immediately due to the 1 to 2 minute delay in establishing connection with another branch plus the CPU overhead associated with session establishment and deletion.

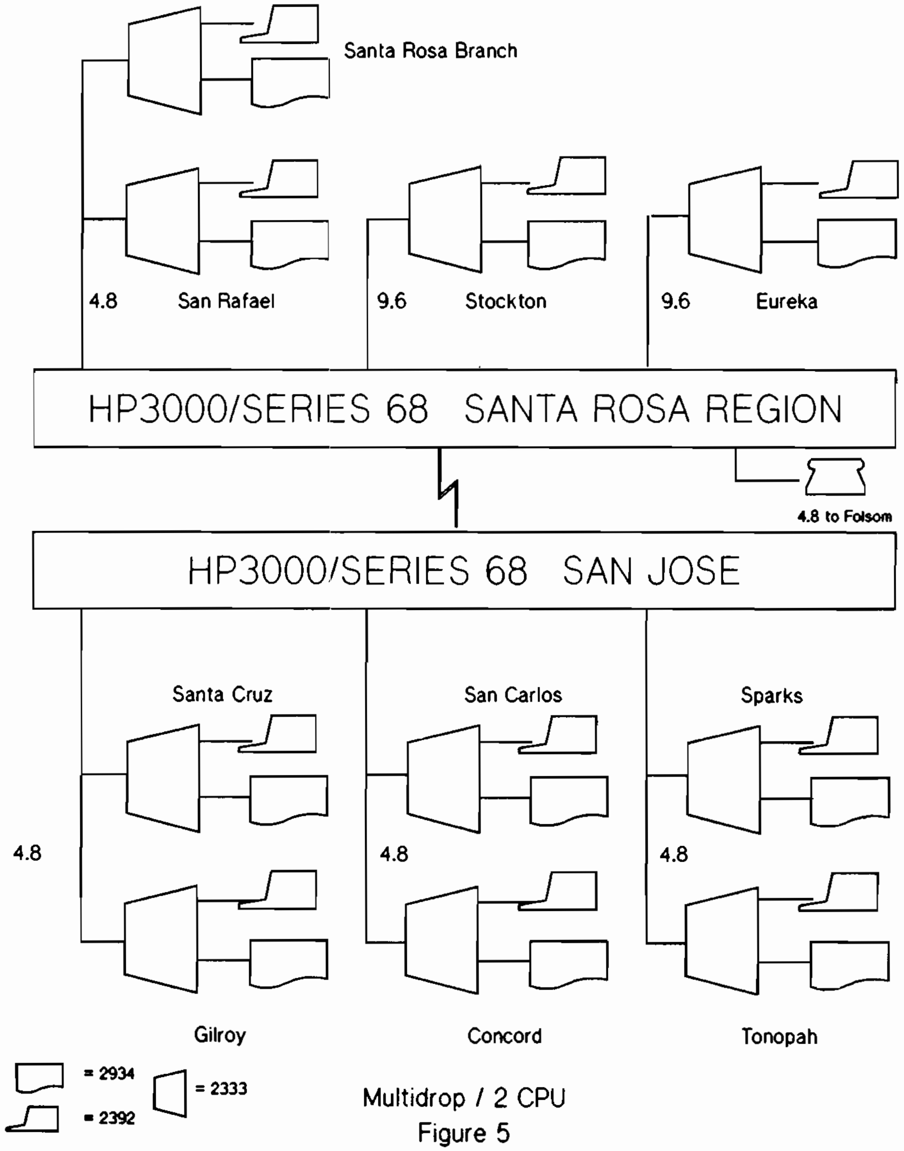
Topologies 1, 2 and 3 were then modeled for response time and cost using the techniques described in Section IIB. All three topologies met the 3 second average and 9 second 95th percentile response time requirements with an assumed host response time of 2 seconds with exponential variation. Topologies 1 and 2 failed to meet the response time objectives when remote site printing requirements were imposed on the links.

More importantly, and contrary to the modeler's intuition, the cost comparison of these 3 topologies revealed that a private X.25 network with analog links was 29.65% less expensive than a point-to-point terminal/multiplexer network with 2 central CPUs and 24.26% less expensive than a multidrop network with 2 central CPUs.

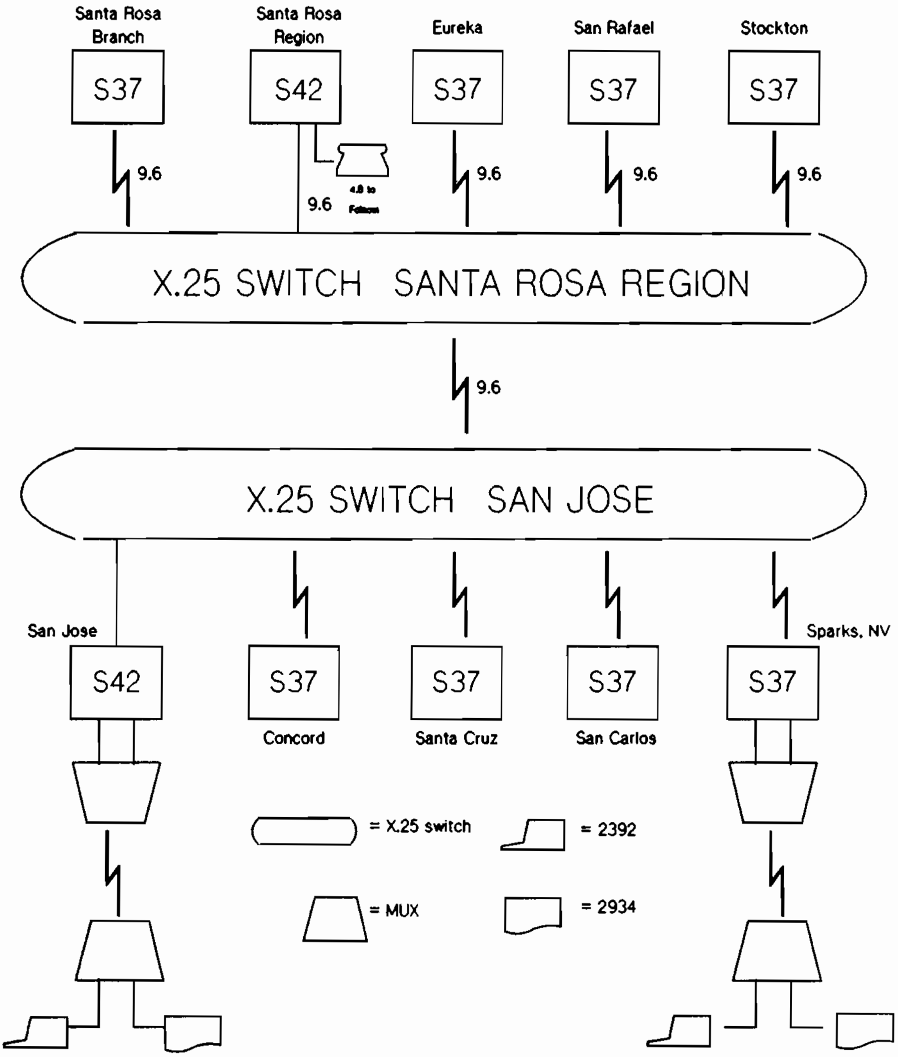
Diagrams of the actual topologies modeled are included as Figures 4, 5 and 6.



Multiplexer / 2 CPU
Figure 4



Multidrop / 2 CPU
Figure 5



X.25 Private Network
Figure 6

B. Response Time Modeling.

Of the tools available for network design and analysis, two were considered applicable at AMFAC: MNDS (Multipoint Network Design Software) from Connections, Inc. and MIND (Modular Interactive Network Designer) from CONTEL Information Systems. MNDS was selected on a cost, speed and ease of use basis. MNDS runs on an IBM PC/XT and was considered suitable for long-term use at AMFAC designing, pricing and verifying networks of the AMFAC region size.

1. CONNECTIONS Software.

MNDS provides the user with the capability of performing 2 primary functions:

1. Predictive performance analysis of multipoint networks. Point-to-point links are simply multipoint links with only 2 points.

2. Topological optimization and alternative price comparison for data communications.

MNDS uses the Esau-Williams algorithm for network optimization which produces a minimum cost conectivity but is not exact (i.e. link drops/adds may produce a lower cost solution).

A. Input Data.

Data required to model a network is provided to MNDS in the following categories:

1. The message file (MF) contains the details of each of the applications which operate in the network. MNDS can consider up to 50 separate application types.

2. The protocol file (PF) is already prepared with 4 default protocols. If these are acceptable, the designer only has to review them and save them in the network database. If the designer is using non-default or customized protocols to do performance analysis (such as when DS or MTS are considered), then the PF must be completed for each of them.

3. The network file (NF) is used to enter the design default values. These are not necessary if the designer is only doing optimum routing and pricing. The NF is also not needed if the designer chooses to enter all of the performance criteria directly into the response model (RM).

4. The site file (SF) contains the specifics of each physical location in the network. MNDS can accommodate up to 200 locations, 10 host computer sites and 20 concentrator/multiplexer sites. A total of 230 sites in each unique network can be handled.

5. The traffic file (TF) contains the number of times each application occurs at each sit in the site file (SF) during the period specified by the designer.

B. Tariffs.

MNDS uses FCC tariffs 9,10 and 11 for pricing. FCC tariff 9 covers Interoffice channels, tariff 10 covers the location of offices and services for serving central offices and tariff 11 covers access rates on a state average basis. Since a large portion of AMFAC's network is located in California, a California specific pricing diskette was obtained for intra-LATA and inter-rate-center channel pricing.

C. Output.

MNDS has 4 modeling programs:

1. Response Model (RM) allows calculation of response time and line utilization for a selected set of network characteristics.

2. Distribution Model (DM) graphically depicts the RM results.

3. Link Model (LM) provides the least cost network layouts and pricing.

4. Operator model (OM) approximates the number of terminals, operators and telephone trunks required at a site.

2. Results.

Modeling was performed on 4 branch sizes in the Electric Supply division's Northern California Region for multipoint and point-to-point multiplexer lines considering each MTI application without screen changes (i.e. data and separator characters only). The results were used to determine a network topology for the Electric supply division. These results showed that neither multipoint (of even 1 drop) or point-to-point multiplexer links could meet the stated goal of 3 seconds average response time when printing was taking place.

C. Considerations.

Of the many items considered during the network design, probably the most taxing areas centered around using digital or analog communications links for leased lines and whether to use public or private x.25 networks for CPU interconnection.

1. Analog vs Digital lines.

Analog lines have the following benefits:

a. Widespread availability
b. Modem availability from 1200 to 19200 BPS
c. Well developed testing methods
d. Moderate costs
e. MODEM based network management systems are available.

Analog lines have the following disadvantages:

a. Comparatively high error rates (1 in 10^5 bits worst case, 1 in 10^6 bits expected with some currently available MODEMS)
b. High speed MODEMS have relatively high cost
c. Timing considerations in large networks may be difficult.

Digital lines have the following benefits:

a. Low error rates. Do not confuse the expected error rate of 1 in 10^7 bits with the expected 99.5% error-free seconds which can be translated into a bit error rate of .5 in 10^2 or 5 in 10^3 which is not very good.

b. Low DCE (Data Communications Equipment) cost.

c. Automatic circuit rerouting to alternate path.

Digital lines have the following disadvantages:

a. Higher cost than analog lines. Cost analysis of the model region and of the entire network revealed that on the average a digital line costs 1.92 times as much as an analog line. Comparisons were also done in California alone using both the AT&T and CPUC pricing which showed that digital cost 1.85 times as much as analog.

b. Limited availability.

c. Limited variety of DCE speed. Present offerings are 2.4, 4.8 9.6 and 56000 BPS.

d. Limited customer testing options.

e. Limited network management options.

For cost comparison a 9600 BPS circuit was chosen. Costs were:

DIGITAL	ANALOG
2ea CSU/DSU @ $800 = $1,600	2ea MODEM @ $5,900 = $11,800
line at $960 / month	line at $500 / month

Neglecting the error rate differences, it can be seen that the analog investment can be paid off in ($11,800-$1600) / ($960-$500) = 23 months. On a 5-year cost of ownership basis the difference is:

```
 Digital=$1,600 + (60*$960) = $59,200
-Analog=$11,800 + (60*$500) = $41,800
                              -------
                              $17,400
```

Amplifying factors such as the non-availability of digital service at many of the sites in the network and the long digital installation lead times plus the lack of network management options for digital solidified the choice of analog circuits for AMFAC's network.

2. Private vs X.25 networks.

Comparison of the cost of public vs private X.25 networks was relatively straightforward. Using public data networks (GTE TELENET chosen for comparison) the cost of CPU interconnect for Electric and I & P was $154,209 per month. Acquisition cost of the private X.25 network was computed at $261,000 for switches plus monthly MODEM and line lease costs of $81,877 (AT&T Total Service Offering). Comparison shows a payback of:

$261,000/ ($154,209-$81,877) = 4 months!!

III. Recommendations.

Recommendations to Amfac took the following forms:

A. General:

1. Use analog lease lines.
2. Use analog MODEMS with network management capability (Codex 2600 series)
3. Use multiplexers with ENQ/ACK spoofing, with a buffering scheme which will allow termtype 19 remote spooled printers to be used and with HP line, page and user-controlled block mode support (Codex 6002 series)

B. Electric Supply Division:

1. Use an X.25 Private network for CPU interconnect.

C. I & P:

1. Use a Point-to-point multiplexer terminal network.
2. Connect CPU's via leased facilities using x.25 DS and share facilities with Electric when possible.
3. Use time division multiplexing (TDM) for minimum delay times to build multidrop-like lines to minimize mileage costs. (4 and 6 channel TDMs are available as a low cost option to the Codex 2600 series MODEMS)

D. Health Care:

1. Use dialup bisync DS at 4800 BPS.
2. Use OUTWATS lines
3. Use Codex 5208R MODEMS and 2207 autocall units

E. Independent Verification

Toward the end of the network design process AMFAC Distribution group secured the services of Network Strategies Inc. (NSI) to perform 6 tasks:

1. Review all documentation relative to the network design
2. Resolve open issues and interview AMFAC staff
3. Develop network requirements
4. Develop Major network architecture components
5. Develop detailed architecture report
6. Develop a 2-year phased implementation plan

The results of NSI's first 5 tasks is presented in Figure 7. Examination of the the least cost solution column shows agreement with the HP designed solution.

Architecture Cost Comparison

		Dial	Leased Line	Public X.25	Private X.25	StatMux (Leased Line)	Least Cost
Electric	Host-Host	2	Too Few INP Slots	3	1	Too Few INP Slots	Private X.25
	Host-Terminal (Twig)	3	2	No PAD Support	No PAD Support	1	StatMux
I&P	Host-Host	4	2	3	1	No Collocation of Hosts	Private X.25
	Host-Terminal	3	2	No PAD Support	No PAD Support	1 (6002)	StatMux
Health Care	Host-Host	1	(High)	(High)	(High)	(High)	Dial ex. where collocated w/ other sol'ns
	Host-Terminal (Regions)	4	3	5	1 (PAD)	2	PAD to X.25 Switch
Consolidation Problems		None	Can't Handle Electric	Can't Handle I&P Terminals	Can't Handle I&P Terminals	Can't Handle Electric	
Consolidation Configuration		All Dial	Need X.25 For Electric	Need StatMux For I&P Terminals	Need StatMux For I&P Terminals	Need X.25 For Electric	

Figure 7

Section IV Network Management

A. Available types

1. Analog lines

Analog MODEMS are available which provide network management by using part of the available channel bandwidth for an independent communications path (side channel). These systems provide error statistics, modem configuration control, individual modem identification, set-point alarming, management reporting, etc. Features vary from front panel access to multiple node, multiple operator control and costs range from a few hundred dollars per MODEM to over $100,000 per system. The Codex 2600 Series MODEMS support Codex Distributed Network Control System (DNCS).

2. Digital lines.

Only two systems were evaluated for digital link network management systems. One system, AT&T's DATAPHONE II LEVEL 4 provided most of the features noted above but was limited to a non-distributed controlling environment. An alternative offering, Customer Test Service, available from AT&T on a line-by-line basis and requiring only a terminal and 1200 BPS MODEM for access was evaluated but proved to be extremely limited in its testing and problem resolution capacity. Since the anticipated direction was to have an all analog network, these two solutions were not exhaustively evaluated.

3. Multiplexers.

The multiplexers chosen for the AMFAC network have a control port from which the multiplexer may be configured and statistics obtained. The minimum requirement is to set the control port baud rate via front panel switches. Remote access to multiple units is accomplished via a Telematics programmable selector switch and a 1200BPS dialup modem.

4. X.25 Switches

Dynapac Model 8 and 12 X.25 switches chosen for this network based on HP hardware and software support availability for the Model 8 and the connectivity and throughput of the Model 12. Access to these devices for configuration and statistics will be vial a PAD at Folsom.

Section V Summary

During 1985 86 CPU and 56 terminal/multiplexer sites were installed. Current HP activities include the completion of the design of the network management. AMFAC activities include continued installation of sites and implementation of an X.25 pilot region.

****

Jack Hymer has been with Hewlett-Packard for 10 years and is currently engaged in network design activities for HP customers as a Network Consultant. Mr.Hymer obtained a BSEE degree from the University of Washington in 1973 after working as a communications technician for 6 years.

NETWORK SUPPORT: THE MULTIVENDOR SERVICE DILEMMA

Deborah Nelson
Hewlett-Packard Co.

SUMMARY

When a network is malfunctioning, efficient problem diagnosis and resolution is the immediate goal. However, since most networks are a combination of computer and communication vendors' equipment, problem escalation procedures become unclear. Fingerpointing among vendors is counterproductive to restoring the network to normal operation. Service vendors need to recognize that to provide comprehensive support, they must integrate cooperative problem resolution into their escalation process. However, putting in place daily operating procedures between worldwide service organizations is nontrivial. The user organization also has a responsibility to provide an internal operating structure which will accomplish efficient problem resolution. A united effort of vendor and user organizations is the key to effective network support.

INTRODUCTION

The multivendor environment is a fact of life in data communications. The multitude of computation and communication products and technologies, specialized applications, and the linking of departmental automation points have contributed to this situation. As networks become more and more predominant, and therefore critical to business applications, achieving efficient problem resolution within the multivendor environment becomes imperative.

Providing effective service in this context produces a challenge for both vendor and user organizations. Just as there are standards for communication between vendors providing a common methodology for transferring information, a standard mode of operation should be established for problem resolution among vendors.

To successfully support networking customers through a multivendor service methodology, each vendor must examine their own service structure and define internal service requirements. In addition, each customer needs to understand their role and responsibilities in successful ongoing network operations and support.

SITUATION ANALYSIS

Many vendor support strategies are oriented to a homogeneous environment, which is defined as one where they provide most, if not all, of the products, and therefore service. For the customer it is difficult to find the best communications solution if a company is restrained to one vendor as one vendor will not usually provide all possible data communication and computation products. So instead, an applicable communications solution is designed and implemented with the ongoing support tackled on a per incident basis.

The vendors, being responsible solely on a product by product basis under traditional service contracts, leave a large part of the initial problem diagnosis on the customer's shoulders. The customer calls in the service personnel from each vendor separately to work on the problem. If a solution is not forthcoming, the customer may ask all vendors involved to work together on the problem, and then has the challenge of attempting to schedule a mutual time. Once assembled, this situation can result in the customer refereeing the vendors' service efforts with each vendor proclaiming innocence and the problem still existing. Service vendors must begin to take a broader view of service and incorporate dealing with other service organizations into their normal escalation procedures.

CERTIFICATION

As a basis for a successful working relationship between vendors, the performance of the vendor interconnection should be characterized. Both customers and vendors should understand which vendor products work together within a specified environment. In addition to providing a basis for mutual support, this would provide customers with increased latitude in designing communication solutions to fit their application.

One way to achieve this is a program of cooperative testing between two or more vendors to provide compatibility assurance to their customers. As vendors cannot test each of the multitude of hardware and software configurations, a subset must be defined for the testing. This subset should be as representative as possible of the most likely application environments.

Often termed **certification testing**, this process confirms that within a defined environment the tested products operate in their specified manner. For customers, this provides confidence in the proper operation of the certified products when used within the specifications. In addition, during the problem diagnosis stages, the interconnection of these products could be supported by both vendors. Thus certification sets the stage for successful support of the network.

Products could be chosen for the certification based on demand from mutual customers and fit within each vendors offering. For example, a computer vendor may wish to certify several modem models from different vendors in order to be able to support their customers in the widest range of communication solutions. In summary, the certification process offers customers a very flexible and robust range of communications solutions, plus the added assurance that the configurations have been tested and are supported by both vendors.

JOINT MAINTENANCE

The next step in providing a multivendor service methodology is to set up a formal relationship between vendor service organizations. Under this arrangement, the vendors will not physically service each other's equipment, but will integrate the other vendor(s) into their escalation procedure. To be successful this relationship must be an explicit process, clearly understood, and a part of each organization's daily operating process.

A successful service relationship can be built between vendors with similar organizational structures. For example, both vendors can have a centralized support facility as the initial customer contact. This would allow customers a choice of initiating contact with either service organization and provide a centralized contact between organizations. While keeping the customer informed of the progress, the vendors could exchange information regarding a problem on a timely basis. The methodology and type of information exchanged would be described in a joint support operating plan which is created by and shared between both organizations.

A centralized vendor contact is only part of the picture. Local cooperation between vendors is also critical to a customer's success. Once a problem has been escalated to onsite assistance, it may be necessary to have intervendor relations to either work jointly at the customer site, or to initiate a service call and communicate problem status. The local joint support effort will be based on a preestablished protocol and commitment between vendors. As a result of the direct vendor interaction, the customer experiences a more efficient and less frustrating problem resolution process.

VENDOR RESPONSIBILITIES

The service vendor's responsibility is two-fold: to deliver effective network problem resolution and to keep the customer informed and involved in the problem resolution process. To accomplish this, the vendor first must implement joint operating agreements with representative vendors. In implementing this, the vendor must realize that the certified configurations will most likely vary from country to country, dependent on local communications restrictions and popularity. The local vendor representatives should be included in the initial support planning process before network installation, if the customer wishes to take advantage of the cooperative vendor relations.

Another way to help ensure a customer's success, is to hold regular reviews to clarify vendor and customer expectations, performance requirements, and outline expansion plans. At this time, the customer can bring the vendor up to date on related network activities and the vendor can work with the customer to plan for changes to the network, such as major software revisions. This will give the customer added control of his own network environment. The vendor can also help the customer identify internal operating procedures to facilitate timely problem resolution.

CUSTOMER RESPONSIBILITY

Customers have the responsibility to structure his internal operating procedures to accomplish effective problem resolution in partnership with the various service organizations and to facilitate communication among network service vendors.

The structure of an internal operations staff will have an obvious effect on timely problem resolution. Even under a program of cooperative vendor relationships, the customer organization must do the best job possible of initial problem description and diagnosis. If the symptoms are described correctly up front, this will hasten any problem resolution process. The customer organization should have an identified person to "own" the problem, thus avoiding additional time delays and internal confusion. Good network documentation can preclude frustration in the event of a knowledgeable staff member's absence.

A critical factor in managing a network is the coordination of changes and expansion. The customer, as the focal point for all vendor activities, must ensure that the vendors are aware of simultaneous or related activities, such as node or product additions.

And finally, active participation in periodic reviews with all service vendors provides a formal process for the clarification of expectations, review of past performance, and an arena to discuss future changes.

CONCLUSION

As the number and size of networks increase, accomplishing effective problem resolution within the multivendor environment becomes critical. Vendors must expand their traditional view of service to include cooperative vendor relationships. The success of these relationships is based on the development of an explicit problem escalation process integrated into each organization's daily operating procedures. A program of certification testing can provide the foundation for joint maintenance. Certification will also provide compatibility assurance to customers and broaden the available range of supported communication solutions. In addition, the customer's internal operations must be structured to facilitate the communication and problem resolution process. The mutual efforts of vendor and customer organizations can effectively put an end to the multivendor service dilemma of network support.

Deborah Nelson is presently Network Support Product Manager for the Product Support Division at Hewlett-Packard in Cupertino, California. In this capacity she is responsible for the development and marketing of network maintenance and installation products. Deborah joined HP in 1982 and holds a Bachelor of Science degree in Industrial Engineering and Management Science from Northwestern University.

Lynn Barnes
Hewlett-Packard Cupertino
U.S.A.

NOTE: See page 583.

Luc Beersmans
I.C. Systems n.v.
Belgium

"The Logical Life in your Data Sets".

NOTE: Because of reasons out of the hand of the Host Committee, this paper will not be published in the Conference Proceedings.

TurboIMAGE INTERNAL FILE STRUCTURE

Doris Chen
Hewlett-Packard
Cupertino, California, USA

Introduction

Through the last decade, many people have developed interest and expertise in IMAGE. Thousands of programs and products have been written using IMAGE databases. With the release of TurboIMAGE, an enhanced version of IMAGE, the internal file structure has changed. To upgrade to TurboIMAGE, databases must be converted. The purpose of this paper is to provide IMAGE experts a better understanding of the new file structure. Any privileged mode programs dependent on IMAGE internal file structures may require modifications to run with TurboIMAGE. The complete TurboIMAGE file structure is presented here. The differences from IMAGE are discussed. Some examples will be given during the live presentation. Note: Some information in this paper will also be included in the U-MIT System Tables Reference Manual.

TurboIMAGE vs IMAGE

TurboIMAGE has relaxed many IMAGE limitations (refer to the TurbolMAGE Reference Manual for the new limitations). These are the reasons for changing the database data structure. A TurboIMAGE database consists of 3 types of files: a root file, master data sets, and/or detail data sets (same as IMAGE). Master data sets must be converted due to the change of the detail chain count in the master set entries. It was increased from a single word (65K entries in a detail chain) to a double word. The root file must be converted due to the following changes:

a. The maximum number of items in a database was increased from 255 to 1023. An item number can no longer be stored in a byte, it requires a word. Thus, the Item Table and Data Set Control Blocks were changed.

b. The maximum number of data sets in a database was increased from 99 to 199. The Set Table and Data Set Control Blocks were changed to accommodate additional data sets in the database and additional fields in the data set. (A field is an item defined in the data set).

c. All major tables, i.e. the Item Table, the Set Table, and Data Set Control Blocks, have been moved to begin at the record boundry for easy access.

d. Some information has been added to the root file information (label 0) and the database global information (record 0) for general housekeeping purposes.

e. Two tables were added: Item/Set Maps and the Device Class Table.

Throughout this paper, a '+' next to a variable indicates the value or the meaning of the variable is new or has been changed for TurboIMAGE.

Root File Structure

The database root file is an MPE file, with a file code of -400, created by DBSCHEMA. It consists of information about the database: the password table, item table, set table, etc. The record size is 128 words, fixed, binary format with a file system blocking factor of 1. The size of the file depends on the number of data items and data sets defined in the database.

Following is the general format of the root file. The detailed format of each table in the root file follows the general format.

```
             ________________________
            |                        |
   LABEL 0  | ROOTFILE INFORMATION   |
            |_______________128_words|
            |                        |
         1  | PASSWORD TABLE         |
            |________________________|
            |                        |
         2  | PASSWORD TABLE (CONT.) |
            |________________________|
            |                        |
         3  | ITEM R/W TABLE         |
            |________________________|
            .                        .
            .________________________.
            |                        |
         .  | SET R/W TABLE          |
            |________________________|

             ________________________
            |                        |
  RECORD 0  | DATABASE GLOBAL INFO   |
            |_______________128_words|
            | ITEM MAP               |
         1  |------------------------|
            |_SET_MAP________________|
            |                        |
         2  | ITEM TABLE             |
         .  | (variable size)        |
         .  |________________________|
         .  | SET TABLE              |
         .  | (variable size)        |
         .  |________________________|
         .  | DATA SET CONTROL BLOCKS|
         .  |                        |
         .  |        (DSCB)          |
         .  |                        |
         .  |    (variable size)     |
         .  |________________________|
         .  | DEVICE CLASS TABLE     |
         .  | (variable size)        |
         .  |________________________|
```

Root File Information (Label 0)

```
                                                             %
          ___________________________________________
WORD 0   |_RL'CONDITION____(rootfile_condition)___|   0
     1   |_RL'DATE_________(creation_date)________|   1
     2   | RL'TIME         (creation time)        |   2
     3   |________________________________________|   3
     4   |_RL'EVEROPEN____________________________|   4
     5   |_RL'COLDLOADID___(cold_load_id)_________|   5
     6   |_RL'USERCOUNT___________________________|   6
     7   |_RL'DBG'DST______(DBG_DST_number)_______|   7
     8   | RL'LOGID        (log id for            |  10
     .   |                  transaction logging)  |   .
    11   |________________________________________|  13
    12   | RL'LOGPASS      (log id password)      |  14
     .   |                                        |   .
    15   |________________________________________|  17
    16   |_RL'FLAGS________(database_flags)_______|  20
    17   |_RL'STORDATE_____(DBSTORE_date)_________|  21
    18   | RL'STORTIME     (DBSTORE time)         |  22
    19   |________________________________________|  23
    20   |_RL'BUFSPECCOUNT_(buffer_spec_count)____|  24
    21   |_RL'ILRCREATEDATE_(date_ILR_log_created)_|  25
    22   | RL'ILRCREATETIME (time ILR log created) |  26
    23   |________________________________________|  27
    24   |_RL'ILRLASTDATE__(last_log_access_date)_|  30
    25   | RL'ILRLASTTIME  (last log access time) |  31
    26   |________________________________________|  32
    27   |_RL'RBPREDATE___(previous_rollback_date)|  33
    28   | RL'RBPRETIME   (previous rollback time)|  34
    29   |________________________________________|  35
    30   |_RL'RBDATE_______(rollback_date)________|  36
    31   | RL'RBTIME       (rollback time)        |  37
    32   |________________________________________|  40
    33   |_RESERVED_______________________________|  41
    34   |_RL'LANGUAGE'ID__(language_id)__________|  42
    35   | RL'LANG'MNEMONIC (language mnemonic)   |  43
     .   |                                        |   .
    42   |________________________________________|  52
    43   |_RESERVED_FOR_DBCONV____________________|  53
    44   |                                        |  54
     .   . RESERVED  FOR  FUTURE  USE             .   .
    63   |________________________________________|  77
    64   | RL'MAINTWORD    (database maintenance  | 100
    65   |                  word)                 | 101
    66   |                                        | 102
    67   |________________________________________| 103
    68   | RL'BUFFERSPECS  (buffer specifications)| 104
         |                                        |   .
    to   |                                        |   .
         |                                        |   .
   127   |________________________________________| 177
```

RL'CONDITION (IN ASCII):

TurboIMAGE uses 2 ASCII characters to indicate the condition of the database. Following are the condition codes:

```
JB      - The database has not been created yet.
FW      - The database is OK.
RM      - Random Modification. The database is being modified with
          output defer mode.
MC      - Maintenance Create. The database is being created by DBUTIL.
ME      - Maintenance Erase. The database is being erased by DBUTIL.
IL      - ILR is in progress (DBOPEN).
IE      - ILR enable is in progress (DBUTIL).
ID      - ILR disable is in progress (DBUTIL).
CN  (+) - Conversion by DBCONV is in progress.  The process can NOT
          be continued if DBCONV is interrupted.
CA  (+) - Conversion by DBCONV is in progress.  The process can be
          continued if DBCONV is interrupted.
MV  (+) - Data set move is in progress (DBUTIL).
```

RL'DATE: Root file creation date*. Its format is:

```
 _0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15
|year_______________|day_of_year_______________|
```

RL'TIME: Root file creation time*. Its format is:

```
 _0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15
|hour_________________|minutes_________________|
|seconds______________|tenth_of_seconds________|
```

RL'EVEROPEN: This field is no longer used under IMAGE B and TurboIMAGE.

RL'COLDLOADID: MPE system cold load id. This id and the RL'USERCOUNT are used to determine if there was a system failure while the database was in used.

RL'USERCOUNT: The number of users currently accessing the database.

RL'DBG'DST (+): The DBG (Data Base Global Control Block) DST number. When the database is opened by the first user, this number is stored here. Subsequent users can find the database DBG easily.

RL'FLAGS:

bit 0	- RECOVERY	Default is NO (0)
1	- LOGGING	Default is NO (0)
2	- ACCESS	Default is YES (1)
3	- DUMPING	Default is NO (0)
4	- AUTO DEFER (+)	Default is NO (0)
5-6	- SUBSYSTEM ACCESS	Default is R/W (00)

```
     7 - ILR                Default is NO  (0)
     8 - ROLLBACK           Default is NO  (0)
     9 - Reserved for future use.
    10 - DIRTY FLAG         Default is YES (1).
                            This indicates the database has
                            been modified but not DBSTOREd.
    11 - DBRECOV RESTART (+) Default is NO  (0)
 12-15 - Reserved for future use.
```

Bits 0 to 8 are set by DBUTIL. Bit 10 is set by DBSTORE, DBDELETE, DBPUT, or DBUPDATE. Bit 11 is set by DBRECOV.

RL'STORDATE: Same format as RL'DATE*.

RL'STORTIME: Same format as RL'TIME*.

RL'BUFSPECCOUNT: Maximum number of buffer specifications allowed.

RL'ILRCREATEDATE: Same format as RL'DATE*.

RL'ILRCREATETIME: Same format as RL'TIME*.

RL'ILRLASTDATE: Same format as RL'DATE*.

RL'ILRLASTTIME: Same format as RL'TIME*.

RL'RBPREDATE (+): Same format as RL'DATE*.

RL'RBPRETIME (+): Same format as RL'TIME*.

RL'RBDATE (+): Same format as RL'DATE*.

RL'RBTIME (+): Same format as RL'TIME*.

RL'LANGUAGE'ID: Same format as defined in the system configuration.

RL'LANG'MNEMONIC: Language mnemonic for this database.
Maximum of 16 characters.

RL'MAINTWORD: For data bases with no maintenance word this field has
2 semicolons (';;') and trailing blanks.

RL'BUFFSPECS:

```
            _0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15   %
WD     68  |buffers_for_1 user____|buffers_for_2 users___|  104
       69  |buffers_for_3 users___|buffers_for_4 users___|  105
        .  | etc...                                      |  .
        .  |_____________________________________________|  .
      127  |buffers_for_119 users_|buffers_for_120 users_|  177
```

* All DATE and TIME fields can be formatted (for display purposes) individually by calling the FMTCALENDAR and FMTCLOCK Intrinsics respectively. Both fields can also be formatted at once with the FMTDATE Intrinsic.

Password Table

```
           _LABEL_#1_______________________________  %
 WORD 0  | Password for user class 0               | 0
      1  | (this is a dummy field since user       | 1
      2  |  class 0 is not defined)                | 2
      3  |_________________________________________| 3
      4  | Password for user class 1               | 4
      5  |                                         | 5
      6  |                                         | 6
      7  |_________________________________________| 7
      8  | Password for user class 2               | 10
      9  |                                         | 11
     10  |                                         | 12
     11  |_________________________________________| 13
      .  |                                         | .
      .  .                                         . .
      .  |_________________________________________| .
    124  | Password for user class 31              | 174
    125  |                                         | 175
    126  |                                         | 176
    127  |_________________________________________| 177

           _LABEL_#2_______________________________
      0  | Password for user class 32              | 0
      1  |                                         | 1
      2  |                                         | 2
      3  |_________________________________________| 3
      4  | Password for user class 33              | 4
      5  |                                         | 5
      6  |                                         | 6
      7  |_________________________________________| 7
      8  | Password for user class 34              | 10
      9  |                                         | 11
     10  |                                         | 12
     11  |_________________________________________| 13
      .  |                                         | .
      .  .                                         . .
      .  |_________________________________________| .
    124  | Password for user class 63              | 174
    125  |                                         | 175
    126  |                                         | 176
    127  |_________________________________________| 177
```

The Password Table occupies user labels number 1 and 2. There are four words (8 characters) reserved for each password. The relative position of a password corresponds to the user class number defined in the schema. For user class numbers not defined in the SCHEMA, the four word field is filled with blanks.

Item Read/Write Table

```
          _LABEL_#3_______________________________   % (Octal)
WORD 0   | Item1 read/write bit map               | 0
     .   |                                        | .
     .   |                                        | .
     7   |________________________________________| 7
     8   | Item2 read/write bit map               | 10
     .   |                                        | .
     .   |                                        | .
    15   |________________________________________| 17
    16   | Item3 read/write bit map               | 20
     .   |                                        | .
     .   |                                        | .
   119   |________________________________________| 167
   120   | Item16 read/write bit map              | 170
   121   |                                        | 171
     .   :                                        : .
     .   :                                        : .
     .   :                                        : .
   127   |________________________________________| 177
```

The Item Read/Write Table begins at user label 3. There are eight words for each Item Read/Write bit map. For databases with more than 16 items, the read/write table continues in the next user label. The specific format of this table is explained after the Set Read/Write Table since it is defined the same way. The number of user labels occupied by the Item Read/Write Table depends on the number of data items defined in the schema and can be obtained by rounding up (ceiling) the result of:

Num-of-labels = [(Num-of-items)*8]/128

Since there can only be a maximum of 1023 data items in the schema, the maximum size for this table in user labels would be:

Max-size = [(1023)*8]/128 = 63.93 => 64 labels.

Set Read/Write Table

```
          _LABEL_#?____________________________  % (octal)
  WORD 0 | Set1 read/write bit map              | 0
       1 |                                      | 1
       2 |                                      | 2
       3 |                                      | 3
       4 |                                      | 4
       5 |                                      | 5
       6 |                                      | 6
       7 |______________________________________| 7
       8 | Set2 read/write bit map              | 10
       9 |                                      | 11
       . :                                      : .
       . :                                      : .
       . :                                      : .
      15 |______________________________________| 17
      16 | Set3 read/write bit map              | 20
      17 |                                      | 21
       . :                                      : .
       . :                                      : .
       . :                                      : .
     119 |______________________________________| 167
     120 | Set16 read/write bit map             | 170
     121 |                                      | 171
       . :                                      : .
       . :                                      : .
       . :                                      : .
     127 |______________________________________| 177
```

The Set Read/Write Table begins at a user label boundary following the Item Read/Write Table. There are eight words for each Set Read/Write bit map. For databases with more than 16 data sets, the read/write table continues in the next user label. The specific format of this table is shown in the next page.

The number of user labels occupied by the Set Read/Write Table depends on the number of data sets defined in the schema, and is obtained by rounding up (ceiling) the result of:

Num-of-labels = [(Num-of-sets)*8]/128

Since there can only be a maximum of 199 data sets defined in the schema the maximum size for this table in user labels is:

Max-size = [(199)*8]/128 = 12.44 => 13 labels

Format for the Read/Write bit maps:

The Read/Write Table has an entry for each item/set in the database. Each entry is 8 words long and up to 16 items/sets per record (user label). Within each 8 words, the first 4 words are the flags for the user classes which have read access to the item/set. The second 4 words are the flags for the user classes which have write access to the item/set. The detailed format for an eight word field is shown below.

A. Four words for read access:

```
 0_____________15 16_____________31 32____________47 48___________63
|_word_1_________|_word_2_________|_word_3_________|_word_4_________|
```

4 words represent 64 bits. Bit n represents read access for user class n to the item/set. If bit n is set to 1 then user class n has read access to the item/set. For example, if the word settings are:

```
     word 1   word 2   word 3   word 4
    %000016  %020000  %000410  %001300
```

This means that user classes 12, 13, 14, 18, 39, 44, 54, 56 and 57 have read access to the item/set. If no read/write security is defined at all for the item/set, then all of the read security bits are set to 1 by default.

B. Four words for write access:

```
 0_____________15 16_____________31 32____________47 48___________63
|_word_1_________|_word_2_________|_word_3_________|_word_4_________|
```

Write access flags have the same format as the read access flags. Bit n represents write access for user class n to the item/set. If bit n is set to 1, then user class n has write access to the item/set For example, if the word settings are:

```
     word 1   word 2   word 3   word 4
    %000010  %020000  %000000  %001100
```

This means that the user classes 12, 18, 54 and 57 have write access to the item/set. If no read/write security is defined at all for the item/set, then all of the write security bits are set to 0 by default.

Database Global Information (Record 0)

```
               _RECORD #0_____________________________________  %
    word 0  |_ROOT'DBSTATUS______________________________________| 0
         1  | ROOT'DBNAME                                        | 1
         .  |                                                    | .
         4  |____________________________________________________| 4
         5  |_ROOT'TRLRLGTH___(trailer_area_length)______________| 5
         6  |_ROOT'BUFFLGTH___(buffer_length)____________________| 6
         7  | ROOT'LGTH       (rootfile length)                  | 7
         8  |____________________________________________________| 10
         9  |_ROOT'ITEMCT_____(number_of_items)__________________| 11
        10  |_ROOT'SETCT______(number_of_data_sets)______________| 12
        11  |_ROOT'ITEMPTR____(record_#_of_item_table)___________| 13
        12  |_ROOT'DSETPTR____(record_#_of_set_table)____________| 14
        13  |_ROOT'DSCBPTR____(record_#_of_DSCBs)________________| 15
        14  |_ROOT'DEVICEPTR__(record_#_of_device_class_tbl)_____| 16
        15  |_ROOT'DBGFLAG_______________________________________| 17
        16  | RESERVED        (set to blanks)                    | 20
         .  |                                                    | .
        19  |____________________________________________________| 23
        20  |_NOWOPEN____________________________________________| 24
        21  |_MAXOPEN____________________________________________| 25
        22  |_RR'RESTART'CALENDAR________________________________| 26
        23  | RR'RESTART'TIMESTAMP                               | 27
        24  |____________________________________________________| 30
        25  | RR'RESTART'FNAME                                   | 31
         .  |                                                    | .
        28  |____________________________________________________| 34
        29  | RR'RESTART'FGROUP                                  | 35
         .  |                                                    | .
        32  |____________________________________________________| 40
        33  | RR'RESTART'FACCT                                   | 41
         .  |                                                    | .
        36  |____________________________________________________| 44
        37  | RESERVED        (for future use)                   | 45
         .  |                 (set to binary 0s)                 | .
         .  |                                                    | .
       127  |____________________________________________________| 177
```

```
ROOT'DBSTATUS -    TurboIMAGE:
                               (0:8) 'C' in ASCII.
                               (8:8) Octal 2 (filler).
                   IMAGE (NLS version):
                               (0:8) 'C' in ASCII.
                               (8:8) Octal 1 (filler).
                   IMAGE (pre-NLS version):
                               (0:8) 'B' in ASCII.
                               (8:8) Octal 1 (filler).
```

ROOT'DBNAME: DATABASE name left justified (last 2 chars are blank).

ROOT'TRLRLGTH: The DBG trailer length. This is one of many variables DBOPEN uses to determine the DBG size.

ROOT'BUFFLGTH: The largest buffer size required to accommodate the database blocks.

ROOT'LGTH (+): The size of the root file. This variable was changed from a single word to a double word.

ROOT'ITEMCT: The number of items defined in the database.

ROOT'SETCT: The number of data sets defined in the database.

ROOT'ITEMPTR (+): The Item Table Pointer. It consists of the record number where the Item Table resides.

ROOT'DSETPTR (+): The Data Set Table Pointer. It consists of the record number where the Data Set Table resides.

ROOT'DSCBPTR (+): The DSCB (Data Set Control Blocks) Pointer. It consists of the record number where the DSCB resides.

ROOT'DEVICEPTR (+): The Device Table Pointer. It consists of the record number where the Device Table resides.

NOWOPEN: Number of data sets opened. This field is NOT used in IMAGE B and TurboIMAGE.

MAXOPEN: Maximum number of data sets that can be opened. This field is NOT used in IMAGE B and TurboIMAGE.

ROOT'DBGFLAG (+): 1: Information can fit in the DBG.
0: Information cannot fit in the DBG.

RR'RESTART'FNAME (+): Restart file name for DBRECOV stop/restart.

Item and Set Maps (+)

```
           _RECORD_#1______________________________________   %
 word 0 |                                                |  0
      1 |                                                |  1
      . |                 ITEM MAP                       |  .
      . |                                                |  .
     30 |________________________________________________|  36
     31 |                                                |  37
     32 |                                                |  40
     33 |                                                |  41
      . | Reserved for future use                        |  .
      . |                                                |  .
     62 |                                                |  76
     63 |________________________________________________|  77
     64 |                                                |  100
     65 |                                                |  101
      . |                 SET MAP                        |  .
      . |                                                |  .
     94 |________________________________________________|  136
     95 |                                                |  137
     96 |                                                |  140
     97 |                                                |  141
      . | Reserved for future use                        |  .
      . |                                                |  .
    126 |                                                |  176
    127 |________________________________________________|  177
```

The Item Map occupies words 0 to 30. The Set Map occupies words 64 to 94. These two maps are new in TurboIMAGE. The Item Map is used for searching an item in the Item Table and the Set Map is used for searching a data set in the Set Table.

An item name (or a data set name) is hashed, through the internal hash function, to a double-word value. The final hash value is the modulo for the double-word value by 31 (double-word value MOD 31). All items (or sets) which have the same hash value are chained together. There are total of 31 chains in the Item Table (or Set Table). The Item Map serves as the chain head for each chain in the Item Table and the Set Map serves as the chain head for each chain in the Set Table.

Item Table

```
 bits/    _0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15  %
 word 0 | item-name-1                                    | 0
      1 |                                                | 1
      . |                                                | .
      . |                                                | .
      7 |________________________________________________| 7
      8 |_item-no-of-synonym_____________________________| 10
      9 |_reserved-1_____________|_reserved-2____________| 11
     10 |_item-type______________|_subitem-count_________| 12
     11 |_subitem-length_________|_(not used)____________| 13
     12 | item-name-2                                    | 14
     13 |                                                | 15
      . |                                                |  .
      . |                                                |  .
     19 |________________________________________________| 23
     20 |_item-no-of-synonym_____________________________| 24
     21 |_reserved-1_____________|_reserved-2____________| 25
     22 |_item-type______________|_subitem-count_________| 26
     23 |_subitem-length_________|_(not used)____________| 27
     24 |                                                | 30
      . :                                                :
      . :                                                :
```

The Item Table starts in record 2. Each entry is 12 words long and the length of the table depends on the number of data items defined in the schema. The relative position of an item definition depends on its relative position in the schema.

`item-name:` A data item name, left-justified with trailing blanks.

`item-no-of-synonym (+):` The number of the item whose name has the same hashed result as this one (this is utilized for quick item name searches). This has been changed from a byte to a word.

`item-type:` One of the following: I, J, K, R, X, U, Z, or P

```
                 item-type
                |
     VALUES, 20J2;
               | |subitem-length
               |subitem-count
```

The maximum size for this table is 12*1023 = 12276 words

`Note:` The reserved-1 and reserved-2 fields are the old level numbers for read and write security. Now, these values are always zero.

Set Table

```
bits/     _0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15  %
word 0  | set-name-1                                     | 0
     1  |                                                | 1
     .  |                                                | .
     .  |                                                | .
     6  |                                                | 6
     7  |________________________________________________| 7
     8  |_set-no-of-synonym______|_reserved-1____________| 10
     9  |_reserved-2_____________|_data-set-type_________| 11
    10  | DSCB-pointer                                   | 12
    11  |________________________________________________| 13
    12  | set-name-2                                     | 14
    13  |                                                | 15
     .  |                                                |  .
     .  |                                                |  .
    18  |                                                | 22
    19  |________________________________________________| 23
    20  |_set-no-of-synonym______|_reserved-1____________| 24
    21  |_reserved-2_____________|_data-set-type_________| 25
    22  | DSCB-pointer                                   | 26
    23  |________________________________________________| 27
    24  |                                                | 30
     .  :                                                :
     .  :                                                :
```

The Set Table follows the Item Table. It starts at a record boundry. Each entry is 12 words long. The length of the table depends on the number of data sets defined in the schema. The relative position of a set definition depends on its relative position in the schema.

set-name: A data set name, left-justified and with trailing blanks.

set-no-of-synonym: The number of a data set whose name has the same hashed result as this one (this is utilized for quick set name searches).

data-set-type: One of the following: A, M or D.

DSCB-pointer (+): A pointer to the Data Set Control Block. It has been changed from one word to a double word. The pointer is a word offset from record 0. The DSCB is described on the following pages.

The maximum size for this table is 12*199 = 2388 words.

Note: The reserved-1 and reserved-2 fields are the old level numbers for the read and write access respectively. Since this concept no longer applies, the values are set to zero.

Data Set Control Blocks (DSCB)

```
 _____________________________________________ _
| DATA SET GLOBAL AREA (set 1)                |  }
|      (capacity, lengths, counts, etc)       |  }
|                                    30 wds.  |  }
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|  }
| RECORD DEFINITION TABLE (set 1)             |  }
|        a. ITEM NUMBERS                      |  } DSCB
|        b. ITEM DISPLACEMENT                 |  } set1
|                             fieldcount*2+1  |  }
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|  }
| PATH TABLE (set 1)                          |  }
|      (search item, sort item, etc.)         |  }
|                                pathcount*3  |  }
|_____________________________________________|_ }
| DATA SET GLOBAL AREA (set 2)                |    }
|      (capacity, lengths, counts, etc)       |    }
|                                    30 wds.  |    }
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|    }
| RECORD DEFINITION TABLE (set 2)             |    }
|        a. ITEM NUMBERS                      |    } DSCB
|        b. ITEM DISPLACEMENT                 |    } set2
|                             fieldcount*2+1  |    }
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|    }
| PATH TABLE (set 2)                          |    }
|      (search item, sort item, etc.)         |    }
|                                pathcount*3  |    )
|_____________________________________________|____}
|                                             |
:                                             :
:                                             :
:                                             :
|_____________________________________________|-
| DATA SET GLOBAL AREA (last set)             |  }
|      (capacity, lengths, counts, etc)       |  }
|                                    30 wds.  |  }
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|  }
| RECORD DEFINITION TABLE (last set)          |  }
|        a. ITEM NUMBERS                      |  } DSCB
|        b. ITEM DISPLACEMENT                 |  ) last set
|                             fieldcount*2+1  |  }
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|  }
| PATH TABLE (last set)                       |  }
|      (search item, sort item, etc.)         |  }
|                                pathcount*3  |  }
|_____________________________________________|_ }
```

The DSCBs follow the Set Table in the Root File. There is one DSCB for each data set defined. The function of the DSCB is to define each data set within the data base.

Dsta Set Control Blocks (Global Area)

```
 bit/     _0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15  %
 word 0  | DSCAP           (data set capacity)             | 0
      1  |_________________________________________________| 1
      2  |_DSBLOCKLGTH____(block_length)___________________| 2
      3  |_DSMEDIALGTH____(media_record_length)____________| 3
      4  |_DSENTRYLGTH____(entry_length)___________________| 4
      5  |_DSBLOCKFAC___________|_DSPATHCT_________________| 5
      6  |_DSFIELDCT_______________________________________| 6
      7  |_X|_DSPRIMKEY____________________________________| 7
      8  |_DSPATHPTR_____(offset to path table)____________| 10
      9  | logical end of file                             | 11
     10  |_________________________________________________| 12
     11  | max num of records in set                       | 13
     12  |_________________________________________________| 14
     13  | 17 words of binary zeroes                       | 15
      .  :                                                 : .
     29  |_________________________________________________| 35
```

DSCAP   - data set capacity as reported by DBSCHEMA.

DSBLOCKLGTH - data set block length including the bit map overhead.

DSMEDIALGTH - data set media record length (remember that this length includes the pointer overhead)

DSENTRYLGTH - data set entry length.

DSBLOCKFAC   - data set blocking factor.

DSPATHCT   - data set path count. This is the number of paths that are specified for the data set.

DSFIELDCT(+)- data set field count. This is the number of fields (items) specified for the data set. It has been changed from a byte to a word.

X-DSKEYTYPE - data set key type. If DSKEYTYPE = TRUE then the key is hashed.

DSPRIMKEY   - data set primary path or key. For master data sets, this is the field number of the search item. For detail data sets, this is the path number of the primary path.

DSPATHPTR   - data set path table pointer. This is a Word offset (relative to its own DSCB) to the data set path table which contains an entry for each path defined. It points to 0th entry in the path table, so to get to the first entry the pointer should be incremented by the length of the entry (which currently is 3 words).

Data Set Control Blocks (Item Numbers)

```
         _____________________________
word 0  |_item_num_of_1st_field_______|
     1  |_item_num_of_2nd_field_______|
     2  |_item_num_of_3rd_field_______|
     .  |                             |
     .  :                             :
     .  :                             :
     .  |_____________________________|
     .  |_item_num_of_last_field______|
```

The Item Numbers Table is part of the Record Definition Table in the DSCB. It follows the Global Area of the DSCB. The size of this table (in words) is equal to the number of items in the data set. This table has been changed from a byte array to a word array to support larger item numbers, for example, 300.

The first field is the first item defined in the data set. The last field is the last item defined in the data set.

Data Set Control Blocks (Item Displacement)

```
         ______________________________________
word 0  |_word_offset_to_first_field__________|
     1  |_word_offset_to_second_field_________|
     2  |_word_offset_to_third_field__________|
     .  :                                     :
     .  :                                     :
     .  :                                     :
     .  |_word_offset_to_last_field___________|
        |_length_of_entry_____________________|
```

The Item Displacement is also part of the Record Definition Table in the DSCB. It immediately follows the Item Numbers Table.

The word offset points to the starting location of the field within the media record. Remember that the media record includes the pointer overhead so this offset varies for master and detail data sets. If a master data set has only one path, the word offset for the first field is 11, since there are 11 words of overhead (5 words for the synonym chain pointers and 6 words for the data set chain head). For a detail data set with one path, the overhead is only 4 words.

The 'length-of-entry' is the same as the media record length.

Data Set Control Blocks (Path Table)

```
          _____________________________
word 0 | 1st path definition          |
     1 |                              |
     2 |______________________________|
     3 | 2nd path definition          |
     4 |                              |
     5 |______________________________|
     6 |                              |
     . :                              :
     . :                              :
     . :                              :
     . :                              :
     . :______________________________:
     . | last path definition         |
     . |                              |
     . |______________________________|
```

The Path Table follows the Record Definition Table in the DSCB. There are 3 words (6 bytes) for each path definition. The Path Table for master data sets has a different layout from the Path Table for detail data sets.

```
Master sets:
       Byte  Description
       1-2 : Item number of the search item in the related detail set.
       3-4 : Item number of the sort item in the related detail set.
         5 : Set number of the related detail data set.
         6 : Path number of the corresponding path in the related
             detail data set.

Detail sets:
       Byte  Description
       1-2 : Field number of the search item.
       3-4 : Field number of the sort item.
         5 : Set number of the related master data set.
         6 : Path number of the corresponding path in the related
             master data set.
```

Device Class Table (+)

```
                                             %
word 0 | DevClass-name for set 1   | 0
     1 |                           | 1
     2 |                           | 2
     3 |___________________________| 3
     4 | DevClass-name for set 2   | 4
     5 |                           | 5
     6 |                           | 6
     7 |___________________________| 7
     8 | DevClass-name for set 3   | 10
     9 |                           | 11
    10 |                           | 12
    11 |___________________________| 13
    12 | DevClass-name for set 4   | 14
    13 |                           | 15
    14 |                           | 16
    15 |___________________________| 17
    16 |                           | 20
     . :                           :
     . :                           :
     . :                           :
```

The Device Class Table follows the DSCBs. It begins at a record boundry. There is an entry reserved for each data set defined in the schema. Each entry is 4 words long. It contains the device class name which can be optionally specified for the data set in the schema. For data sets without device class names, the entries will be filled with blanks.

This table is created by DBSCHEMA. DBUTIL uses this information to create the data sets. A data set without a device class defined will be created with "DISC" as the default. Once the database is created, the information in this table will NOT be used again. The new DBUTIL command "MOVE" moves a data set from one device to another. However, it does not update the Device Class Table.

The length of the table depends on the number of data sets defined in the schema.

The maximum size for this table is 4*199 = 796 words.

Data Set Structure

A data set is an MPE file, with a file code of -401, created by DBUTIL. It consists of a user label for the data set information, and many records for the user data. The record size by default is 512 words, fixed, binary format with a file system blocking factor of 1. A record is a TurboIMAGE block. The block size can be defined in the schema to be different from the default by using the TurboIMAGE blocking factor and BLOCKMAX options. A different blocking factor can be specified for each data set. Following is an example of defining a blocking factor:

```
NAME:         SETA, Detail
ENTRY:        item1,
              item2,
              item3,
              itemlast;

CAPACITY:     511(22);
```

In the above example, the data set SETA has a capacity of 511 and a blocking factor of 22, i.e. 22 entries in a TurboIMAGE block. The blocking factor is used to define the block size. If the block size is larger than the default, the BLOCKMAX must be specified before the data set:

```
$CONTROL BLOCKMAX=nnnn
```

where "nnnn" is the maximum block size for the data sets that follow this statement.

Following is the general format of a data set:

```
                  ____________________________
                 |                            |
LABEL     0      |  Data Set Information      |
                 |____________________________|

                 |____________________________|
                 |  User Data  (first block)  |
RECORD    0      |                            |
                 |____________________________|
                 |  User Data  (second block) |
RECORD    1      |                            |
                 |____________________________|
                 :                            :
                 :                            :
                  ____________________________
                 |  User Data  (last block)   |
RECORD    N      |                            |
                 |____________________________|
```

Data Set Information (Label 0)

The Data Set User Label consists of 6 words of information. The information for a master data set is different from a detail data set.

Master:

word 0-1: The record name* of the last entry allowed in the set, i.e. the capacity of the data set.

2-3: The number of free records in the data set. This number is incremented when a record is deleted and decremented when a record is added. To determine the current number of entries in the data set, subtract Word 2-3 (free records) from the data set capacity.

4-5: Not used.

Detail:

Word 0-1: High Water Mark. This is the record name of the highest record that has been used in the detail set. It is updated by DBPUT only. For example, a detail set with 75 entries and a capacity of 100 has a High Water Mark pointing to the 75th entry. The High Water Mark stays the same even after entries are deleted.

2-3: The number of free records in the data set; this is the same as the master data set.

4-5: The delete chain head for the detail set. It points to the record most recently deleted. It has a value of zero if no records have been deleted from the data set.

* A record name is a double word that consists of a block number (i.e. file system record number) and the slot number in the block. The first 3 bytes is the block number and the last byte is the slot number. A slot is an area in the block allocated for a media record. The first media record in the block is slot number 1. The last media record is slot number N where N is the blocking factor. The first block in the data set is block number 0.

Data Set Blocks

The format of a TurboIMAGE block is the same as described in the IMAGE Reference Manual with the exception of the chain count in the master entries. The chain count has been changed from a single to a double word. The largest number that can be stored in a single word is 65K, thus 65K was the limitation for IMAGE. With a double word, TurboIMAGE does not have this limitation. However, this DOES NOT mean that all entries in the set should be linked in one chain. It would have a negative impact on the performance of chain reads. In fact, it is strongly recommended that all chains be kept short.

Biography

Doris Chen is the R&D project manager for TurboIMAGE, IMAGE, and DBchange. She has been with Hewlett-Packard since 1979.

Doris Chen
Hewlett-Packard Cupertino
U.S.A.

NOTE: See continuation on page 575.

MULTI SYSTEM DATABASES

By Rolf Frydenberg
Hewlett-Packard Norge A/S
Oslo, Norway

SUMMARY

The term Multi System Databases (or MSDB, for short), is used to refer to applications that utilize data from more than one database, located on geographically separated computers.

The paper presents and discusses MSDB, or distributed data sharing systems as they are sometimes referred to. This area is still largely experimental, but a major R & D effort is going on. MSDB will probably become of major importance within the next decade, and at least by the turn of the century.

The first part of this presentation focuses on the end-user needs that make MSDB a future necessity. Distributed data sharing systems based on homogeneous computers and database management systems are discussed, as well as those based on heterogeneous systems.

The second major part of this paper presents two existing (though mostly experimental) MSDB's. We have also included a brief prensentation of HP3000 capabilities in this area.

Finally, we indicate the direction we think this area is moving in, and how we - as users - can prepare for that development.

I. WHAT IS A MULTISYSTEM DATABASE?

Before proceeding to the details of this presentation, I would like to give you a thumbnail sketch of what multisystem databases are. This should bring the area into closer perspective, so that we all know what we are looking at.

A multisystem database is a database that consists of information stored on more than one computer system. The data stored may be of different formats, or the same. And we will also look at any file - flat file, KSAM file, or "true" database - as being a database or part of one; so long as a description of its format is available to us.

The computer systems that store the multisystem database may be identical - e.g. a set of HP3000's - or they may be different - e.g. some PC's, some HP3000's and a non-HP mainframe.

The data communications link between the computers is assumed to work, but we will not specify it in any level of detail: It might be a dial-up asynchronous link, a leased 9600 bps line, an X.25 connection, or an SNA-link. For the sake of this presentation, we will ignore those differences.

Let me also mention that there are two other expressions often applied to multisystem databases: Distributed databases, and distributed data sharing systems. We will treat these expressions as referring to the same genereal concept, though some people do consider at least the term "distributed databases" as being more restrictive that the other two terms.

There are many possible approcaches to distributing a database over a network of computer systems. Four of the main distribution strategies are:

1. **Independent databases;** this means that each computer system has its "own" database(s), controlled fully by this computer system. Each independent database "contributes" some of its data to the overall Multisystem database.
2. **Centralized database with replicated subsets;** this means that there exists one, centrally located, database, which contains all the data. Subsets of

this database are copied to other locations, mainly for higher speed of access to frequently used data. Typically, updates are only accepted at the central site, and transmitted to remote locations in batches.

3. **Horizontal partitioning;** this means that the same record types may exist at many locations, but a specific record is at just one location. A file of orders, for example, could be distributed so that all orders are located at the warehouse that will process them.
4. **Vertical partitioning;** this means that different components of a specific record may be located at different sites. In a vertically partitioned database, some kind of replication may be used, so that all locations may know all customer-numbers and customer-names, but more detailed information is only stored at the site that actually handles this customer.

In many real-life situations the best solution is a combination of strategies. Strategy number 1 - independent databases - is the most common method of handling the integration of previously independent databases. That makes this strategy important now, though not necessarily as important in the future.

Another aspect of multisystem databases is whether they allow purely local users to exist. Local users are those who access data on one specific computer in the network, without knowledge of, or access to, the "complete" mulitsystem database. Some MSDB systems allow this - particularly those systems designed to handle existing databases - whereas others treat all users as global users, and all transactions as global transactions.

Processing global queries (i.e. data retrieval from the multisystem database) is a reasonably straight forward operation. Updates, though are potentially much more complex. There are two reasons for this complexity: If data is replicated, then all copies of the record must be updated; if the record is split between multiple sites, then all sites must update their part of the record. And, of course, the MSDB´s system must keep track of whether the individual updates succeeded, and if they did not, then it must perform whatever backtracking is necessary.

If there is no replication within the multisystem database, then it is possible to restrict all updates to being purely local transactions, i.e. transactions that may be processed completely on a single computer system. This significantly simplifies the manner in which updates are processed. Centralized systems also simplify updating the database, since only the central copy of the data must be updated.

II. THE NEED FOR MULTISYSTEM DATABASES

A large amount of research into the area of multi system databases is currently being carried out all over the world. Is this research just being carried out "for the fun of it", as the business world often accuses the academic community of, or is it something we will really need in the not too distant future?

It is my contention, that multi system databases are something we need, and consequently that it is an important topic for research. And though it may take a number of years before complete, high-performance, easy-to-use systems are available, we need to forge ahead in this research. There may be many intermediate solutions that represent small steps in the right direction.

Since this is not intended as a theoretical paper - I am not a theoretical computer scientist, but rather a practical engineer - let me present the kind of situations where I think multi system databases can represent a solution for users.

Many data processing users currently have more than one computer system. This is true not only of large corporations with central mainframes, but also for medium-sized companies that may have two or more minicomputers; or smaller companies where the computer mix is made up of microcomputers only, or micros combined with minis. In all these cases, corporate data is stored on more than one computer.

These, often diverse, computer systems communicate more and more closely. This need for communication has grown out of the need for access to data through terminal-emulators, but that is only a brief stage in the total development of corporate data communications: As the storage capacity of "non-central" computers (micros and minis) continues to grow rapidly, so the need for accessing this data directly, instead of transferring it to a central site and accessing it there, grows proportionately.

For many users this is already in the process of becoming a problem. The amount of data stored at the decentralized sites is growing so rapidly that it is no longer possible to copy everything to the central site. So far, the solution is to keep "local" data at the decentralized sites, and "common" data at the central site. But this is not a valid solution for the future: Data is a common corporate resource, and timely access to it is getting ever more important.

In addition to the amount of local data increasing at a rapid rate, yet another issue is cropping up: More and more "local" applications also• need access to central data files, so parts of central databases are frequently copied to the local sites. As soon as this data is updated centrally, the local data bases are no longer consistent with the contents of the central data base. Current DBMSs do not have any functions for managing this. Consequently, decentralized sites do not know whether the data they use is valid or not!

The interim "solution" to this problem, is to copy the databases regularly (e.g. once a week), or to collect all updates in a special file, which is transferred at regular intervals to all sites. Either of these methods is useful, but it does not completely solve the problem: It is only immediately after the file transfer that the databases are consistent; as soon as a single update is made to the central data base, consistency is lost.

Another problem is that all updates of such databases must take place centrally: If we allow updates at the local level as well as the central level, consistency is not only lost much faster, we can end up with updated information being replaced by "obsolete" data from the central database at update-time.

One approach to solving this kind of problem is to avoid centralization, and let every local site manage its own data. This means that for data which is really common to all sites, we end up with as many copies of it as we have sites. If the amount of common data is small, this may be acceptable, but in most cases it is not acceptable.

Yet another problem with this distributed approach of data management at each site is the problem of stuctural consistency between databases. Even in very distributed corporations, with high degree of local control of operations, there is a need for communicating information to higher levels of management, where this data is collected into corporate wide data. For this reason, as well as for reasons of accountability and controllability, the data structure used to store one type of data should be consistent across all local sites within the corporation. With complete local control, this will often not be the case.

Perhaps the main reason why HP3000 users are among those who need distributed databases the most, is that the HP3000 is so popular as a departmental computer. For this reason, a large number of the corporations that have HP3000s have more than one such computer. When you add to this the fact that the HP3000 has very good data communications facilities (to other HP3000s particularly, but also to other computers, e.g. IBM mainframes), you get a system that is almost "begging" to be used for the implementation of multi system databases.

III. CURRENT SYSTEMS

The currently existing multisystem databases (MSDBs) are all experimental. But these exipermental or pilot projects at least to some extent indicate the driection in which the world of distributed processing is moving. And even though a number of these projects are based in the academic community, which has not always been known for a commercial orientation, there seems to be a significant amount of realism behind many of the projects.

The selection of systems that we have done for this presentation is somewhat random or haphazard. But they should still represent a cross-section of the types of experimental multisystem databases currently under investigation at research institutions all over the world. Much of this presentation based on data presented at the Second and Third International Symposia on Distributed Databases, held in 1982 and 1984, respectively. The proceedings from these conferences are available in book form (see the chapter on references).

III. A: MULTIBASE

Multibase is a set of programs for accessing data stored in multiple databases. These databases may exist on identical or diverse computers. Multibase is intended as a commercial product, and as such is no longer a "prototype". Multibase is a product of Computer Corporation of America. It is currently only available for IBM mainframes.

There are four main reasons why we have included Multibase in this presentation:

1. Multibase is a "real" product, not just a prototype;
2. Multibase is a typical query-only multisystem database;
3. Multibase allows access to existing databases;
4. Multibase uses a copied catalogue concept.

This makes Multibase an "extreme" multisystem database from one point of view, and it makes it very easy to contrast Multibase with another - and very different - MDBS: POREL. We will have more to say about POREL later; for now let us concentrate on the features and functions of CCA´s MultiBase.

Multibase is a product that allows relational queries to be made that access multiple databases, on multiple computer systems. Queries Multibase are formulated based on an integrated schema, which defines a "virtual" databases (called views) that each may consist of the data in relations stored on one or more of the computers in the network.

Some views may be "simple" in that they access only data stored in one relation, whereas others may be quite complex, and access data in multiple relations, with mapping of data from one field or item to another. Multibase also supports recalculation before integration. An example of this is when data from relation A specifies monetary values in Pesetas, and relation B specifies it in dollars. In the integrated view, such monetary values from relation A are recalculated as dollars before being introduced into the integrated view.

Another feature of Multibase is that it does not require the actual data storage to be relational. There are internal facilities in the system for relational retrieval of data from other types of databases, including hierarchical and network databases. This means that pre-existing databases of almost any type may be accessed from the query facility of Multibase.

A Multibase user has access to a language called DAPLEX for data definition and manipulation. This language is first used to set up the views, through definition of a Global Schema. Subsequent access to the integrated multi system database is though this schema. Additionally, DAPLEX Local Schemas are defined for each actual database to be accessed. These schema map one-to-one to the Local Host schemas, which define each database in the "native" database definition language of the local computer system(s).

The overall organization of Multibase schemas is as illustrated below:

```
                              DAPLEX Global Schema
          _______________________!     !     !     !__________
         !                  __________!     !               !
         !                 !                !               !
DAPLEX Local           DAPLEX Local        ...         DAPLEX Local
 Schema No.1            Schema No.2                     Schema No.N
         !                 !                !               !
         !                 !                !               !
 Host Local             Host Local         ...          Host Local
 Schema No.1            Schema No.2                     Schema No.N
```

Figure 3.1: Multibase schemas.

When a user accesses Multibase, he does so through the Multibase Global Data Manager (GDM). The GDM finds out how to process the query from the Global Schema. The GDM splits up the query into a number of "single-site" queries i.e. queries that can be accessed with knowledge only of one DAPLEX Local Schema. The GDM then forwards these single-site queries, through the communications network, to Local Database Interfaces (LDIs), where the DAPLEX Local Schema is used in order to retrieve the necessary data from the actual database. LDIs then send their results back to the GDM, which performs any required coordination of data, or operations that require data from more than one host computer Finally, the GDM passes the results of the whole query back to the user.

We have illustrated the relationships between the GDM, LDIs, and schemas in the following figure:

```
Schemas used:                                    Information flow:
                                                 DOWN:             UP:
                               END USER
                                  !              Global Query      Result
                                  !
DAPLEX Global                    GDM
                         _________!!!_________   Single-site       Formatted
                        !         !          !   Query             Data
DAPLEX Local          LDI No.1   ...      LDI No.N
                        !                    !   Local Query       "Raw"-data
                        !                    !
Host Local            DBMS No.1           DBMS No.N
```

Figure 3.2: Multibase component interaction.

The Global Data Manager is the central piece of software in Multibase. It contains a number of advanced functions, including a description of each LDI concerning the capabilities - or lack of such - at each site within the network. This ensures that the queries sent to the LDIs only request functions that really are supported by the local Host DBMS. This function should help to keep the amount of work in creating Multibase LDIs for new systems low, which again means that Multibase might quickly become implemented on a number of diverse computers.

Among other GDM functions, which are not unique to Multibase, but rather standard for MSDBs, are global and local query optimization, and an auxiliary database for data needed to coordinate data from different local databases Multibase uses an internal DBMS to access this database in the final stage of preparing the results for the user.

The Multibase Local Database Interface (LDI), which gives DAPLEX support to each local host DBMS, is a relatively simple processor. Its main purpose is the translation of queries from the DAPLEX language to whatever the host supports. As has been noted previously, the GDM has knowledge of which functions are supported by each LDI/Host DBMS combination, which also simplifies the design of each LDI. For a typical DBMS, the effort necessary to implement an LDI is of the order of 3-6 manmonths.

We can probably assume that typical Multibase networks will contain two types of nodes: Those that support a GDM and LDI, and those that only support an LDI. This means that Multibase is not a uniformly distributed system, which may not matter much to actual users.

At the time of writing this paper, the author knew of no sites where Multibase has been implemented. But since the product is now commercially available the first real user experiences should be available soon.

III. B: POREL

POREL represents one of the other extremes of multisystem databases. POREL is a prototype developed at the University of Stuttgart in the Federal Republic of Germany. Among the major differences between POREL and Multibase are:

1. POREL is still only a prototype, not a product;
2. POREL only allows access to databases created with POREL;
3. POREL allows for all types of access, update and add as well as retrieval;
4. POREL uses a distributed calagoue, where some information may be retrieved from other sites;
5. POREL treats all users as "global".

In other words, we are looking at a database system with significant differences as compared to Multibase. Many of these differences, though on the positive side; they are **additional** features, not available in Multibase or comparable systems.

Perhaps the most negative difference, from a user´s point of view, is that POREL requires all databases to be recreated. If data is to be copied over from an existing database, the user will have to develop the necessary programs on his own. This makes POREL primarily useful for dedicated, applications. This clearly contrasts with Multibase, which is primarily intended as an access method to existing databases.

Let us now take a closer look at POREL, which it definitely deserves: It is a very interesting system, and may point the way farther into the future then the much more restricted Multibase-system.

POREL is a distributed database management system developed for a network of interconnected minicomputers. POREL has been implemented on minicomputers with an 16-bit addressing range, and therefore consists - on each computer - as a set of interacting processes. All of POREL has been written in PASCAL. (This should make it possible to transport POREL to the HP3000 and HP1000 systems if desired; though I have not performed any evaluation of this possibility).

For computer-to-computer data communications, POREL uses X.25. A POREL system may be arbitrarily large, but the greater the number of processors, the higher the probability of failure in communications.

One of the most noteworthy features in the design of POREL is the emphasis placed on reliability and error-recovery. POREL is designed not only so that the failure of one node does not bring the network down, but the restart of nodes, including picking up where they were when they failed, has been carefully designed. (This may lead one to suppose that the computers used to implement POREL are unreliable, or that the level of programming performed is not reliable. There is no proof of either of these assumptions, though).

POREL transactions are described through either the use of a special query language (RDBL = Relational Data Base Language) or though special code available to PASCAL programmers (P-RDBL). This PASCAL-support is based on the programmer inserting special statements in his code, which is converted to procedure calls by a pre-compiler.

During the compilation of a PASCAL program with P-RDBL statements, or th compilation of a query stated using regular RDBL, a Network Independent Analysis is performed. Then, after all checks have passed at this level Network Oriented Analysis is performed. At this stage POREL may access nodes in order to retrieve information about data stored there.

When a complete transaction has been analyzed, it may be executed. This cause sub-queries to be sent to all relevant processors, and they are asked to retrieve the specified data. In the case of an update, a two-phase locking strategy is used, whereby first all data is locked (at all involved locations) and then the update is performed through the use of a COMMIT command. This allows for backing out of a partially completed update if one or more node(s) should fail during the update.

All POREL machines keep track of transactions in progress, and can therefore recover from the point where they were, should any kind of failure occur. The machines also keep track of which other systems are UP or DOWN, which helps avoid locking some data entries when it will not be possible to lock all the ones that are needed for the transaction, because one or more of the machines required are down.

POREL also has support for storing parts of a relation on different computers as well as for storing duplicate copies of a relation - or part of one - on separate computers. This will help keep the amount of data communications down during query processing, but it adds a lot of complexity to update processing. If we assume that query is the most common operation, and update is much less frequent, this may not necessarily impact performance of the overall system too much.

In managing multiple copies of a relation, POREL marks one as being primary,and the other ones as secondary. Updates of secondary copies may be delayed until the next time the data in this tuple of the relation is needed, which means that we can postpone some of the update activity until the system has "idle" time. POREL also keeps track of whether there are any outstanding updates to be performed on a secondary copy of a relation, to avoid letting users retrieved not fully updated information.

IV. HP3000 CAPABILITIES

The current capabilities of the HP3000 file and database systems do not include MSDB support. But there are ways of implementing some MSDB type functions. In this chapter we will look at some of these, both implementation of multisystem databases where only HP3000s are involved and where the HP3000 is a node in a non-HP network (in this case, an IBM mainframe oriented SNA-network).

IV.A: HP3000-TO-HP3000 MSDB'S

In order to access one HP3000 from another, the natural data communications product to use is DSN/DS. This product allows two or more HP3000s to be connected together, running either BSC protocol or X.25 over a public packet switched network (PSN). When using X.25 over a PSN, the HP3000 can also communicate with computers from other vendors that support this method of communication.

DSN/DS can also be used to communicate with HP1000 computers, both the BSC protocol and X.25 is supported on these computers. In this presentation we will concentrate on communication between two or more HP3000s all of which DSN/DS/X.25 over a packet switched network.

(HPs NS (Network Services) product, which handles communications over Local Area Network (LAN) can also be used for HP3000-to-HP3000 communication. For the presentation of DS in this paper, NS and DS may be viewed as providing essentially the same services. We will therefore only refer to DS, and it is up to the reader to supply this with a comment of "or NS" in each instance, if he so desires.)

DSN/DS allows a user on one HP3000 to log on to another HP3000, and to run those programs on the remote system which his account- and username give him access to. It is also possible to handle the logon etc. programmatically, so that the user is not aware of the connection. With DS/X.25, one physical line be used to access multiple computers simultaneously, by one or several users.

(X.25 allows up to 4095 simultaneous sessions across one link; no current HP3000 system can handle it, and the amount of data transmitted would be much higher capacities than is currently available: Typical DS/X.25 connections are at 9600 bps, or 1200 characters per second. Should 4095 users share this capacity, they could each send one character every 3.5 seconds!)

One of the many important features of DS, compared to similar offerings from other vendors, is the complete transparency of a DS-connection to software: A simple FILE-equation can be used to signal that a file resides on another computer; the file system will then automatically use Remote File Access (RFA) capabilitites to access the file on the remote system, in a fully transparent manner - as seen from the user and the application program.

The same kind of capability is available for databases, through DS Remot DataBase Access (RDBA). This provides the capability of accessing IMAGE databases on remote systems as if they were locally available.

RFA and RDBA are important capabilitites, but do not really support multi system database access. One of the reasons for this is that they do not allow for processing of queries on the remote computer before passing the result back to the host (or local) computer. For this reason, selecting data on other than an exact key match, will cause too much data to be transmitted over the data communications link.

Let me give a small example: Let us suppose we have an IMAGE master that contains information about customers. The key is probably the customernumber or the customer name. If we want to find the customers who are in the city Madrid (which could be none!), we will have to search serially through the set. In this case, if we use normal RDBA, information about about all the customers would be transmitted, and the application program would have to select the data it actually needs.

(There is only one available method for optimizing this access; that is to read only a few fields from the set, typically the key (customer-number) and the field we are scanning (city). For those that match, we can afterwards do a direct lookup by key to get **all** information about this customer.

In order to support true multi system databases, we will have to develop efficient programs that allow us to pass **a query** to the remote computer, instead of an IMAGE intrinsic call. For relatively simple types of queries, this should not be too difficult. The easy way to do it, is to develop interprocess communication for the standard QUERY/3000 program, and use this as the vehicle for retrieving information from databases on multiple systems.

There is another facility that is needed as well. This is the ability to specify, in a DICTIONARY type of format, where the individual databases that form our overall multisystem database are located. This dictionary must be capable of defining different types of data transformations that should be performed on the local data before it is compared with data from other databases, i.e. before it is integrated into the overall multisystem database.

IV.B. HP3000-TO-IBM MSDB's

Many HP3000 users also have large IBM minframes as their central data processing systems. These users frequently communicate between their HP3000s and the maniframe(s), using either the BSC products (RJE, MRJE, or IMF), of the SNA products (NRJE or SNA IMF). For interactive communications, only two of these products may be used, IMF (for BSC) and SNA IMF. These two products are compatible from the point of view of a user program attempting to access data on the mainframe, since they have exactly the same intrinsics. There is only one slight difference in one of the intrinsics (OPEN3270, which is called only once by a program anyway).

There are three types of multisystem databases that can be created for a mix of HP3000s and IBM mainframes. These are:

1. HP3000 can access IBM, not the other way;
2. IBM can access HP3000, not the other way;
3. Access is allowed both ways.

For approach 1, above, programming need only be done on the HP3000. For access other way, it is necessary to implement programs on both types of computers even though approach 2 does not require access in both directions. This is because IMF is a "one-way" type of product, for HP-to-IBM access, and requires that communication is started from the HP3000.

Currently, there are some users who do interactive access to data stored on IBM mainframes from HP3000s. This is mostly done in a tailored manner where the application contains its own code for the actual data access on the mainframe. To the best of my knowledge, no general system exists. But a general approach has been described at a previous INTEREX conference, by the author of this paper. Below I will try to explain how multisystem database access to an IBM mainframe is possible with the current HP3000-to-IBM mainframe products: IMF and SNA IMF.

The lowest, user-accesible, level of both IMF products is a well-defined set of intrinsics. These intrinsics allow the user-program to emulate all funtions of an IBM327X terminal with 24 lines of 80 characters each (i.e. larger screen sizes and graphics are not supported). By calling these intrinsics, the user-program can access any IBM mainframe database accessible through a user-oriented program (e.g. a query faciliy).

The customized way these intrinsics are currently used is to allow program interactive access to specific programs, which again access data in one or more specific databases. By redesigning these programs, to work with a general query facility on the mainframe, and a data dictionary (which could be on either system) it is possible to allow the user to formulate a query on the HP3000, execute the query on the IBM mainframe, transfer the resulting data to the HP3000 via "screens", and finally reformat and display the data to the end-user. This kind of facility exists today on PCs, generally designed to work with a specific mainframe query-program, often from the same vendor as the PC

This kind of facility exists today on PCs, generally designed to work with a specific mainframe query-program, often from the same vendor as the PC software.

On the HP3000 we can take this approach one step further. We can develope a data dictionary facility on the HP3000 that allows for specification of mainframe **and** HP3000 databases (local and remote), and then allow the user to formulate queries based on all these databases. These queries must then be sp into subqueries, one for each computer, and transmitted to these system, where the database is located for execution.

For this approach to be really useful, it would be an advantage to providea vendor-independent access-method to the mainframe, since HP3000-users have different query-facilities on their IBM mainframes. This is a challenge to HP and to third parties who have HP3000-based query and data dictional products.

Another approach to multisystem databases in a mixed HP-IBM environment, is it implement an IBM mainframe oriented distributed database program, for example Multibase, on the HP3000. This approach would consist of two components: The necessary software for allowing the MDBS to access IMAGE databases (i.e. the LDI level of Multîbase), and the actual software for managing distributed queries (i.e. the GDM facility of Multibase). This, of course, would have to be done by, or in cooperation with, the vendor of the MDBS.

Perhaps one of the first pieces of software that will really help :s to move in this direction is HPSQL - an HP-developed, SQL-compatible query language the HP3000. This product could work as the access method to HP3000-based databases, and would make it easier to ensure compatibility between queries formulated based on HP3000-structured databases, and IBM mainframe database systems.

V. FUTURE TRENDS

In this paper so far, we have looked at the current "future of the art" and at most a few years into the future. Let me now look a little further ahead towards "the shape of things to come".

There are three specific trends that impact multisystem databases, and we will look at each of these in turn. They are all related to standardization, within the following areas:

i. Data communications facilities
ii. Database structures
iii. Database query languages

V.A. TRENDS IN DATA COMMUNICATIONS

There are three trends in data communications worth watching. They are:

i. Protocol standardization, which focuses on the Open System Interconnection (OSI) Model from ISO. The lower three levels of OSI are virtually identical to X.25. (IBMs SNA is another focus for standardization).

ii. Digitalization, which means that future communication services will be provided through all-digital packet- and circuit-switched networks; the modem is on its way out of basic system-to-system communications.

iii.Higher transmission rates, which makes the movement of significant amounts of data through public and private networks feasible without excessive re sponse times.

Taken together, these trends will lead to the availability of high-speed virtually error-free data communications between all points on the globe (or into space, for that matter). In the 1990s the typical packed-switched interface will probably work at speeds up to 1 Mbit per second, rather than the current 9600 bps.

All significant vendors in the 1990 computer business will supply software and hardware to connect their systems to others, all supporting at least the four or five lowest levels of OSI. With standardized LAN facilities in the 10-100 Mbps range, local communication should also be standardized.

V.B. TRENDS IN DATABASE STRUCTURES

Perhaps the most significant trend in database structures of late, is the movement towards relational and semi-relationals structures. Another trend is the emergence and use of system-wide data dictionaries, that can be accessed from a multitude of programming languages, and that can perform additional tasks such as data base creation and restructuring.

Another trend in databases is the growing size - though not necessarily complexity - of databases in common use in businesses. This is a trend that has been made possible though the rapidly falling prices for direct-access storage media. The emergence now of optical-disk systems for archival datastorage, means that in the future "old" data will not be stored offline on magnetic tape (or paper, for that matter) in a vault somewhere, but will be accessible on-line at all times. This will severely impact database size.

Even on single computers, there is a trend towards integrating data from multiple databases into one common database - logically if not physically. This means that some of the basis for implementing distributed databases is already in place.

V.C. TRENDS IN QUERY LANGUAGES

Query languages of one form or another have existed since before the term "database" was invented. Most of the query languages in existence today (including HP QUERY/3000) are more or less ad hoc solutions that have developed slowly over the years, and have a set of functions dictated by combination of history and user demands.

With the trend towards a standardized set of database functions, i.e. the relational model for database structure, a standard for database query is emerging. This is the query language SQL (Structured Query Language, also known as SEQUEL), developed by IBM. Most vendors of relational database systems seem to have jumped on the SQL bandwagon, and sell an SQL-compatible query language together with their DBMS. This is now also true of HP.

More and more of the SQL-compatible query languages support a programmatic interface, so that database access from a program can take place through the same, simple interface that is used by on-line users. This is definitely an advantage, something anybody who has programmed using procedure calls for database access will recognize. This is even more true of programmers with experience from DBMSs such as IMS.

In the rapidly expanding world of multisystem databases, SQL can almost be considered the "de facto" standard for query languages. This trend will continue, and we will also see that SQL-compatible query languages will become available for non-relational (i.e. network or hierarchical) databases.

CONCLUSIONS

This paper has spanned a wide area of topics, and therefore has been unable to cover the subject-matter to any great depth. Additionally, the area of Multisystem Databases is a very large one, and it is growing rapidly. So we have only scratched the surface. But still, we may draw certain conclusions from the content of this paper. These are:

1. Multisystem Databases are possible;
2. Several approaches exist, with different capabilities;
3. The HP3000 datacommunications capabilities make it an interesting computer for implementation of multisystem databases;
4. Several future trends point towards making multisystem databases feasible in the near future.

So, in order to approach the year 2001 in a forward-looking manner, what do we do as HP computer users, need to think of? In this area, at least, we should consider the following:

1. Move towards relational or pseudo-relational database management systems, not just on HP3000´s, but on all our computers;
2. Move towards standardized query/retrieval languages for databases, particularly systems such as HPSQL;
3. Use only internationally standardized data communiation facilities,e.g. X.25 for wide-area datacommunications, and IEEE 802.3 for local area networking.

By following the above guidelines, we should be on our way towards solving the data-processing and data-communications challenges of the next few decades.

REFERENCES

For a more complete description of Multibase, see: "An Overview of Multibase" by T. Landers and R.L. Rosenberg; in H. J. Schneider (ed): Distributed Data Base, North-Holland Publishing Company, Amsterdam, 1982.

This above paper - presented at "the second international symposium on distributed data bases" in Berlin, 1982 - gives a concise description of Multibase, it´s capabilities and functions. For more information on this product, contact Computer Corporation of America (Ann Arbor, Michigan, USA) or their representatives.

For more information about POREL, see: "An Overview of the Architecture of the distributed Data Base System "POREL" by E.J. Neuhold and B. Walter; in the same book as quoted above.

For a more detailed paper on access to on-line IBM applications from an HP3000 using see: "Implementing Distributed Applicaitons in a Mixed HP-IBM Environment" by Rolf Frydenberg; in the proceedings of the HP3000 International Users Group conference, Montreal, 1983; available from INTEREX Santa Clara, California, USA.

Information about IMAGE Remote DataBase Access, and the products IMF and SNA IMF is available from your local HP sales office.

Other books that may be of interest are: James Martin: "Distributed Processing", Prentice-Hall, Princeton, NJ, USA, 1980. F.A. Schreiber and W. Litwin (eds): "Distributed Data Sharing Systems", North-Holland, Amsterdam, The Netherlands, 1984.

BIOGRAPHY

Rolf Frydenberg
has been with Hewlett-Packard Norge A/S for four months. His job is that of an Applications Engineer, specializing in Data Communicating and Programming Languages. Before joining HP last year, Rolf Frydenberg worked as an independent consultant and a software developer. He was one of the principals behind the HP-to-IBM data communications product VTS/IMAS. He has also worked on other types of data communications software.

Rolf Frydenberg has been a frequent speaker at INTEREX international and national conferences since 1982, mostly on topics related to data communications. Rolf Frydenberg has also written articles for trade magazines in the USA and Norway, and is the author of a book on selecting computer systems (in Norwegian).

TRICKS WITH(IN) IMAGE

Jan Janssens
Cobelfret N.V.
Antwerpen, Belgium

Summary

IMAGE-databases are very attractive for storing information: they are very reliable, they are easy to use, they can be accessed by querylanguages and they can be easy managed. Allmost every application on a HP3000, which is very important or deals with a huge amount of data, is forced to use IMAGE for its data-storage. (Of course there are other possibilities, but that seems to be more like re-inventing the wheel).

Unfortunately IMAGE/3000-databases have also some disadvantages: they can become slow if chainlengths or datasetcapacities are growing (10.000 entries is fine, but 1.000.000 entries is something else !!), they don't provide in a indexed access, their structure is sometimes too simple or rigid for the given problem.

This paper describes a method for "inventing tricks" to bypass the problem (which by the way can be applied to all kind of problems) and gives detailed (non-PM!!!) methods to overcome some IMAGE/3000-problems.

1. The "finding tricks"-technology

We will illustrate this technology through the use of the following example:

- A certain bookkeeping application had some databases and some other files in a separate account per company for which bookkeeping was done by a certain group of users. The files were builded by the financial on-line program and formed the input of a complex batchprogram that runned at night.

- If a user wanted to switch to another company, he had to log off and log on in the appropriate account. This involved a lot of overhead, and had to be done for every single inquiry in another account (company).

- The programs could easily be adapted to access the database and files in another account, BUT could not create (:BUILD) the inputfiles in another account (where the batchprogram and on-line programs of other users expected them) unless they should use PM.

1.1 Getting started

The first thing that has to be done is to FORMULATE THE PROBLEM. For the above example this seems to be "building files into another account". This is really THE problem, because if you could build files into another account, the users could "swap" easily from the bookkeeping system of one company to that of another one (reinitialisation + close/open of database and files).

1.2 Think it over

The second step is : "WHY DO WE WANT TO SOLVE THIS PROBLEM?". Why do you want something to do that is very difficult, impossible or has some disadvantages ? In this case the reason is that programs expect to find the files in the "home-account", but that you can't build them from another one.

1.3 Look for "weak" and "strong" points

When we've located the real problem, we can attack it. It is comparable with the situation of an army that has surrounded a fortress and wants to take it over.
One could take the fort by attacking the big entrancedoor with a lot of soldiers. But is very likely that the 'current fortress owners' have made a fine defence system around the door and that the door is made of a very good material. The door is a "strong" point of the problem and its better to avoid it. Perhaps there are more less obvious ways in entering the fortress, but giving the same result. The defence system is perhaps not so strong if you could go in by the system that provides the fortress in fresh water.

In our case the implicit expectation of finding/building files in the homeaccount is the "strong point".
The "weak point" is that it is not necessarily to have the files in a certain account: as long as we are in the possibility to refind all the files that belong to one company, everything is ok!

1.4 Bypass the "weak" point

In the above case the problem can be bypassed by adding a dataset to the company database with a key=FILENAME and a field which has the fully qualified filename (filename + group + account) of the file under which name it was really saved.
All programs must first read in the (already open) database to find the actual filename! They also must do a delete or an update to it when the file is deleted or renamed.

2. Example of an own database system within an IMAGE database

2.1 Definition of the problem

The setup of a database system for a bookkeeping application posed the following problems (after the "think-it-over" phase):

- it must be possible to do extremely fast lookup in very long chains (i.e. give all entries for a big customer from 18 JUNE till 28 JULY)
- it must be possible to locate very fast all records belonging to a certain period (the period was unpredictable)
- the user-input, which consisted sometimes of more than 50 lines, must be registrated in a detail data-set with 3 keys (DOCUMENT, CUSTOMER, FILE), so that "up-to-second" consulting was possible.Allmost all transactions (say 98 %) were done on this data-set.
  Some chains exceeded the IMAGE/3000 upperlimit of 65535 entries.
- a payment of a single customer could cover more than 1500 entries, which must be updated on-line to avoid double usage. It must be possible to interrupt a transaction and to go on with it the next working day.
- the "accounts payable" and "accounts receivable" must be located in separate datasets to speed up batchjobs (serial read) and on-line transactions (reorganisation of the records to minimize DISCIO when consulting via the CUSTOMER-path (a frequent transaction))
- a certain department required such a complex presentation of their "accounts payable and receivable" that it was nearly impossible to do all that work at inquiry-time.(A lot of lookups in other data-sets was needed to define the sortcriteria and presentation lay-out).
  It must also be possible to run their job with a minimum elapsed time.
- it must be possible to use querylanguages on the developed datastructure.

2.2 "Strong" and "weak" points

The "strong" points were:

- the complete system must work with the same efficiency for big and small databases.
- the long chains couldn't be changed into a lot of small ones with each a different key.
- the large transactions couldn't be broken up into small ones.
- if IMAGE was not used then a lot of maintenance-, management- and inquiry-programs needed to be written

The "weak" points were:

1:- all transactions were done in a chronological order: at each moment, input was done in only two (bookkeeping)periods, and the periods changed always in a chronological manner. Consulting was also done in chronological order.

2:- info entered in the system, was never deleted to allow full auditing.(Besides of "clean-up" jobs after some years).

3:- the number of "outstanding records" was only a fraction of the complete history. People wanted to keep track of all bookings for at least 2-3 years (900.000 records), and the "outstandings" covered only 30.000 records.

4:- when entering information in the detail-dataset with 3 keys, most of the time 2 of 3 keys didn't change within the transaction. A lot of transactions also left some keyvalues zero or blank.

5:- the department that required the complex inquiries and jobs was a very important, but a small one.

6:- it was unnecessary and even unwanted to update immediately all payments. As long as one could not pay 2 times the same invoice, everything was ok. By not doing the update immediately users had the time to correct mistakes they encountered when closing their payments at the end of the day.

2.3 Bypassing the "weak" points

2.3.1 Data-structure for "historic" information

Working on the first 2 "weak points" the following design was drawn for the "history" detail-dataset:

```
 Man-Master            Detail              Detail

 KEY                  KEYPOINT           DETAIL-HIST

I---I   first for A I----------I        I----------I
I A I--I----------->I*A 01JAN86I------->I A        I<--01JAN86
I B I  Ilast unl/relI*A 02JAN86I-----I  I B        I
I C I  I----------->I*A 05JAN86I---I I  I A        I
I D I  I            I B 01JAN86I   I I  I C        I
I E I  I            I C 01JAN86I   I I  I D        I
I   I  I            I*A 10JAN86I-I I I  I E        I
I   I  I            I C 10JAN86I I I I->I A        I<--02JAN86
I   I  I            I . .......I   I    I C        I
I   I  I last for A I . .......I   I    I A        I
I   I  I----------->I*A 27FEB86I-I I    I A        I
I   I               I          I I I    I B        I
I   I    last in setI          I I I    I B        I
I---I    ----------->I F 28FEB86I   I    I A        I
                    I          I   I    I C        I<--05JAN86
                    I----------I   I    I C        I
                                   I--->I A        I
                                        I B        I
                                        I A        I<--last
                                        I          I   in set
                                        I----------I
```

- The detail data-set with 3 keys is a stand-alone detail-dataset. This dataset is always filled up with "empty" records. This gives us the opportunity to place the records where we want them by doing an DBUPDATE instead of a DBPUT. To find all records within a given period, we only have to read all records starting with the first record of the given "startdate" till we find a record with a date greater than the "enddate".An index is maintained to keep track of the first record for each date and the last "used" record. Locating ALL records of ALL keys within a given period out of a dataset of more than 1.000.000 entries, is now possible in a fraction of a second!!!!!

- For each of the 3 keys a pointerchain is maintained in the detaildataset. Because no information will be deleted and consulting doesn't need a "backchained read", a "forward"-pointer is enough. The pointerchain gives us the possibility to do a "chained get" a certain record of a chain is read.

- For each key a "KEYPOINT"-data-set is maintained which gives the recordnumber of the first record of that key with a date greater or equal to a given date. This data-set is also a stand-alone detaildataset prefilled with "empty"-records and contains again a pointerchain for each keyvalue. The masterdata-set "KEY" has pointers to the first and last entry for that key in the "KEYPOINT"-data-set. In addition the last "free" entry in KEYPOINT must be kept aside.

  In the KEYPOINT-dataset are pointers written if:

    - the number of records written in DETAIL-HIST, since the last record which has a pointer to it in KEYPOINT, exceeds a given number. The recordcounter and its upperlimit are also stored in the KEY-dataset.

  AND

    - the date differs from the date in the last record for the key in DETAIL-HIST.

  This system allows to define how many "entrypoints" you want to the chain of a given key. The number of entrypoints grows dynamically with the number of entries in the detail-dataset.

- To locate the first entry for a key for a date greater or equal to a given date, the search can be done in the data-set KEYPOINT.
  Once a record is read with a value greater than the given date, one must "backtrack" one record, read in DETAIL-HIST at the location given by the pointer of KEYPOINT and read further in DETAIL-HIST by following the pointers of the appropriate key until a record is read with date greater or equal to the asked startingdate.
  Because KEYPOINT contains less records than DETAIL-HIST and has a greater blocking factor, locating of a record in the chain goes much faster than a chained read in DETAIL-HIST.

- If the number of records fcr a given key in KEYPOINT becomes great, localizing records in DETAIL-HIST will slow down (although it will still be faster than the chained read in DETAIL-HIST).
  To overcome this situation an "unload-reload" of the data-set KEYPOINT can be done, so that records with the same keyvalue become adjacent.
  This gives already a very good improvement by eliminating a lot of disc IO (KEYPOINT can have a blocking factor of 80-100).
  The "unload-reload" can be done by writing a simple program. Because this program can write its records from the end to the beginning (only forward pointers are maintained) and only DBUPDATE's are done, processing of it goes very fast.

- It is also possible to do a "partial unload-reload" : one could reorganize only those entries after the last record of the last "full unload-reload". This action puts all records of the same key together in two blocks instead of one (one "partial reload" after a "full reload").
  Because we have only forward pointers, one must have a pointer to the last "fully unload-reloaded" record in KEYPOINT for each key.
  The effort is minimal, timesavings are high and results can be great: users are often consulting the most recent history and a "full reload" with a few "partial reloads" gives nearly the same effect as a set of "full reloads".
  This concept of "partial reloads" can also be applied to "normal" IMAGE-datasets. It's a pitty that such a smart reload is not yet available.

- By maintaining in the data-set KEY the last record in KEYPOINT which is "fully unloaded-reloaded" another improvement can be done in speeding up consulting: If the given startdate is less than the date given by the last "unloaded-reloaded" record, a binary search can be done in KEYPOINT between the first record for that key and the last "unloaded-reloaded" record of it.

- For 1 of the 3 keys (DOCUMENTNR) a different approach was made because the number of entries was rather small (1 to 100) and all records were always introduced in one single transaction. (This is in fact another "weak" point).
  Only a pointer to the first and last record are provided.

2.3.2 Data-structure for "current" information

Considering "weak points" 3 and 4, a separate dataset (DETAILCUR) is used which holds all outstandings yet in DETAILHIST and all input of the day.

```
  KEY                             DETAILCUR

I-------I    first entry          I-------I
I A     I-I---------------------->I A * c I
I B     I I                       I A * h I
I       I I                       I A * a I
I-------I I  last reorganised entrI A * i I
          I---------------------->I A * n I
          I                       I B     I
          I                       I C     I<-- last unl/rel
          I                       I . ... I
          I  last entr. in HIST.  I A *   I
          I---------------------->I A *   I
          I                       I B     I<-- last hist
          I  last entry           I . ... I
          I---------------------->I A *   I
                                  I B     I
                                  I F     I<-- last used
                                  I       I
                                  I       I
                                  I-------I
```

- For each key the following pointers are provided in the master KE

  - first entry in this dataset for the given key
  - last entry in this dataset for the given key
  - last entry in this dataset for the given key that is already present in DETAILHIST. (lasthistoryponter)
  - last entry in this dataset for the given key that was "unloaded-reloaded"

  In addition a pointer to the last "used" record, the last "fully unloaded-reloaded" and the last record already in DETAILHIST of this dataset itself must be kept aside.
  In the dataset DETAILCUR all entries belonging to the same key are linked together with forward pointers.

  Here we don't need the KEYPOINT-dataset because:

  - most of the time all outstandings are asked (no selection on date)
  - the number of entries is rather small and and the dataset itself can be "unloaded-reloaded" for the most used path (CUSTOMER) in a rather small amount of time. The trick of a "partial unloadreload" can also be used.

- Consulting of "outstandings" can be done in the same manner as explicated above for the KEYPOINT dataset. The trick of consulting with the binary search mechanism only works for the path to which reorganizing was done!!!!

- All input is done to DETAILCUR-dataset, even if it's not "outstanding".
  All those records receive a special code and will be skipped when consulting the outstandings.
  No input is done in the DETAILHIST-dataset. When consulting the history for a given key, one starts reading in the DETAILHIST-dataset untill the end of the chain is reached, reads the record in DETAILCUR indicated by the "lasthistorypointer", and continues reading the next record of the chain.

- Speeding up registration of a booking is done as follows:

  - the pointers for a key are only updated if the key changes in the next record. For most of the bookings our dataset now acted as a detaildataset with only one key.
  - no pointers are created if a key is omitted (say zero or blank).
  - only "DBUPDATE"'s are done instead of DBPUT's.
  - registration is only done in 1 dataset instead of 2.

- Input of the day is posted to the DETAILHIST-dataset by a batch-job that runs overnight. When no "reorganisation" is done the job can start at the record just after the one indicated by the "lasthistorypointer" of this dataset.
  All modifications to already existant records in DETAILCUR are also redone in the DETAILHIST-dataset. Because modifications mostly exist of "payments of outstanding records" and those records are linked together with another "chain", one can easily retrieve the modified records. Other modifications are tracked by writing away the documentnr in a separate dataset.

- The complete system now becomes:

```
   DETAILHIST         KEY1POINT            KEY1             DETAILCUR

   I-------I    5     I-------I    1      I-------I     6   I------I
B->I   12  I<---------I  11  I<--------I-I       I-I------>I  13  I
   I       I    I-----I       I    2    I I       I I       I      I
   I       I    I     I       I<--------I I       I I       I      I
   I       I    I  I--I       I         I I       I I       I      I
B->I       I<---I  I  I       I         I I       I I       I      I
   I       I       I  I       I         I I       I I       I      I
B->I       I<------I  I       I         I I-------I I   7   I      I
   I       I          I       I         I           I------>I      I
   I       I       D  I       I    3    I           I       I      I
   I       I     ---->I       I<--------I           I   8   I      I
   I       I       C  I       I         I           I------>I      I
B->I       I     ---->I-------I         I           I   9   I      I
   I       I                            I           I------>I      I
   I       I    4                       I                   I      I
   I       I<---------------------------I                   I      I
   I       I                                                I      I
B->I       I          KEYPOINT2            KEY2             I      I
   I       I                                                I      I
   I       I    5     I-------I    1      I-------I     6   I      I
   I   12  I<---------I  11  I<--------I-I       I-I------>I  13  I
   I       I          I       I    2    I I       I I       I      I
   I       I          I       I<--------I I       I I       I      I
B->I       I          I       I         I I       I I       I      I
   I       I          I       I         I I       I I       I      I
   I       I       D  I       I         I I       I I       I      I
   I       I     ---->I       I         I I-------I I   7   I      I
   I       I          I       I         I           I------>I      I
   I       I          I       I    3    I           I       I      I
B->I       I       C  I       I<--------I           I   8   I      I
   I       I     ---->I       I         I           I------>I      I
   I       I          I-------I         I           I       I      I
   I       I                            I           I   9   I      I
   I       I    4                       I           I------>I      I
   I       I<---------------------------I                   I      I
B->I       I                                                I      I
   I       I                                                I      I<-G
   I       I                               KEY3             I      I
   I       I   10                         I-------I     6   I      I
   I   12  I<---------------------------I-I       I-I------>I  13  I
   I       I                            I I       I I       I      I
B->I       I                            I I       I I       I      I
   I       I                            I I       I I       I      I<-F
   I       I                            I I       I I   7   I      I
   I       I                            I I       I I------>I      I
   I       I                            I I-------I I       I      I
B->I       I                            I           I   8   I      I
   I       I    4                       I           I------>I      I
   I       I<---------------------------I           I       I      I
   I       I                                        I   9   I      I
A->I       I                                        I------>I      I<-E
   I       I                                                I      I
   I       I                                                I      I
   I-------I                                                I------I
```

1: A pointer to the first record in KEYPOINT.
In KEYPOINT all records of the same key are linked together and are always sorted on date. This is done by the overnight batchjob.

2: A pointer to the last "fully unloaded-reloaded" record in KEYPOINT. This pointer is maintained by the "fully unloadreload" program.

3: A pointer to the last record in KEYPOINT.
A record in KEYPOINT is written if :

- the current pointer = 0 (no entries in history yet for key)

OR

- the number of records written in DETAILHIST since the last record in KEYPOINT exceeds a certain value (both to be registrated in dataset KEY) and the date differs from the date of the last record in DETAILHIST.

4: A pointer to the last record in DETAILHIST.
The pointer is maintained by the overnightjob.

5: For each combination of KEY and DATE : a pointer to the first record in DETAILHIST. Written by the overnightjob.

6: A pointer to the first record of the key in DETAILCUR. Updated by the on-line programs and by the "reorganisation"-job of DETAILCUR.

7: A pointer to the last record of the key in DETAILCUR that is already available in DETAILHIST. Updated by the overnightjob.

8: A pointer to the last record of the key that is "fully unloaded-reloaded".
Updated by the "reorganisationjob".

9: A pointer to the last record of the key in DETAILCUR. This pointer is updated by the on-line programs and by the "reorganisationjob".

10: A pointer to the first record of the key (DOCUMENTNR) in DETAILHIST. Updated by the overnightjob.

11: In KEYPOINT are all records with the same key linked together.

12: In DETAILHIST are all records with the same key linked together.

13: In DETAILCUR are all records with the same key linked together.

A: Pointer to the last used record in DETAILHIST.

B: Pointer to the first record of a given date in DETAILHIST

C: Pointer to the last used record in KEYPOINT.

D: Pointer to the last "fully unloaded-reloaded" record of KEYPOINT.

E: Pointer to the last used record in DETAILCUR.

F: Pointer to the last record of DETAILCUR that is also in DETAILHIST.

G: Pointer to the last "fully unloaded-reloaded" record of DETAILCUR.

2.3.3 Speeding up complex inquiries

To allow fast processing of the jobs of the (small) important department and to solve their complex inquiries the following modifications were done on the above design:

- the complete structure is made in double : one for the important department and one for all the others. This reduces the capacity of all datasets for that department by 3000 % (compared with the other departments).
  The result on the elapsed time of jobs that have to do a serial read is in the same order.

- a separate small file is builded (by a now fast running job) which contains the "outstandings" in a complex presentation-layout. The job runs overnight and at the quiet periods in the day.
  A separate masterdataset is builded which contains pointers to the file for each key and a pointer to the last record of DETAILCUR that is already in the file. When consulting is done on that file, all records of the given key are given, and eventualy additional records in DETAILCUR complete the overview.
  Whenever the jobs runs during the day, records are always added to the file and the appropriate pointers are changed after adding all records to the file. Consulting stays possible when the job is running!!!
  The EOF of the file is reset to zero when the jobs starts overnight.
  A file was chosen because:

  - writing in it goes faster than writing in a database
  - all info is safely placed in the database
  - all info can be reconstructed by rerunning the job
  - reading in it goes faster than reading in a database

2.3.4 Large interruptable transactions

The registration of the complex payments (update of over 1500 entries in a single transaction with the possibility to "undo" it within the same working day was handled in the following way:

- A (central) bitmap for the DETAILCUR-dataset is created. Each bit in it corresponds with an (outstanding) record in DETAILCUR with a relative recordnumber equal to the bitnumber in the bitmap. When the bit = 0, the corresponding record is "free" for payment, otherwise it is "used".

- When the user starts the input of a certain payment, a copy of the bitmap is taken into an XDS (local bitmap).
  When payment references to an already registrated document, the records of it are read in DETAILCUR. If the corresponding bit (in the local bitmap) is zero, one can go on, otherwise the document is already used in another transaction.

The recordnumber of each line in the workfile or scroll is added to the line itself.
No bits are set in the local bitmap.

- When a user terminates a transaction (i.e. payment of a customer) a module is called to set the appropriate bits in the central bitmap.
  (His workfile or scroll will be scanned to gather the right bitnumbers). The results of this module can be:

  - Everything ok: the transaction of the user can now (temporarily) be stored outside the database in a normal MPE-file.
    Further payments on referenced documents in it, are now disabled by the bitmap.
    1500 or more DBUPDATE's are changed to a few discaccesses to UPDATE the bitmap!!!!

  - Double usage of bits : the user entered two times the same document. The workfile or scroll is scanned to retrieve them (relative recordnumber) and a message is given to the user.

  - A bit changed to 1 : another user referenced the same record before this user could registrate his input.

- When a user deletes a transaction the same module is called to reset the referenced bits in the central bitmap to zero. The appropriate records are again free.
  If some of the corresponding bits are already zero, the current bitmap is "corrupt", and must be recovered starting from the MPE-files (the user-input).

- When a user wants to change a transaction, the same is done when deleting one (but the user-info is not cleared to allow modifications).
  When registrating the modified transaction, the bits are resetted
- After validating the complete input (eventually after hours, or even at the end of the day), a job runs to post the user-input in the MPEfiles into the database. This can be done at the quiet periods or even overnight.

2.3.5 Query

The use of querylanguages was rather easy to imply:

- Because all data is located in IMAGE/3000 databases, reporting itself imposes no problem, once the records are retrieved.

- Modules were build to "localate" records in DETAILCUR and DETAILHIST, using the available pointers in the datasets KEY, KEYPOINT, DETAILCUR and DETAILHIST. Those modules build a "selectfile" that holds the relative recordnumbers of the retrieved entries and is used as input for the query-reporter.

3. "Various" tricks

3.1 Speeding up registration in a detaildataset

A lot of transactions work on detaildatasets in the following way:

- If there are entries available in the detaildataset for a given key, they are shown to the user.
- The user can add, delete and insert lines or change existing lines.
- Registration in the database.

Because inserting and deleting records in a chain is a rather difficult thing to achieve within IMAGE, the easiest programming way for this problem is deleting all existing entries and then writing the new records.

However, as long as the keyvalues remain the same the thing can easily speeded up in the following way:

```
clear flag fend
<findfrom,db="BASE",ds="SET",di="ITEM",from="ITEMVALUE">
if $image = 17 then set flag fend
   <getchain,db="BASE",ds="SET",di="ITEM">
   if $image = 15 then set flag fend
   while info available in user workfile or scroll
      move values from workfile or scroll to buffer
      if flag fend
         <putdb,db="BASE",ds="SET">
      else
         <updatedb,db="BASE",ds="SET">
         <getchain,db="BASE",ds="SET",di="ITEM">
         if $image = 15 then set flag fend
      endif
   endwhile
endif
ifnot flag fend
   while $image = 0
   <deletedb,db="BASE",ds="SET">
   <getchain,db="BASE",ds="SET",di="ITEM">
endif
```

Changing information becomes now quicker than introducing new information!!

3.2 Locking in a PH-environment and use of MR-capabilities

To avoid "overlapping updates" on data, there are 2 possibilities:

```
        --> METHOD 1 <--                          --> METHOD 2 <--

 transaction WITHOUT permanent lock  | transaction WITH permanent lock
 ==================================  | ===============================
                                     |
    - get data                       |    - lock data
    - modification by user           |    - get data
    - lock data                      |    - modification by user
    - re-get data                    |    - 'update' data
    - if re-getted data = old data   |    - unlock data
           'update'                  |
      else                           |
           appropriate action or     |
           give message to user      |
      endif                          |
    - unlock data                    |
```

The first method leaves the database free of "locks" until the update really takes place. It assures a minimum of lockingproblems (not avoiding deadlocks, but avoiding that a user has to wait for another one (who is entering his information or is drinking a cup of coffee while he has something locked that prevents the other user to registrate his data).
The disadvantage is that the processing can become difficult or even impossible.

Method 2 has the advantage that it is easy. It even gives no problems if locking can be done on ITEM-level. The problem arises when one has to lock one or more datasets or eventually the complete database. (Remark: locking on dataset-level is required when updating MASTER-datasets).
In a PH-environment where a process stays alive when activating another one (perhaps in in the middle of a transaction that has a lock on the database) this method can not be used and can cause deadlocks (because MR-capability must be used).

The problem can be solved by applying first a "soft lock" and later on a "hard lock". The "soft lock" takes a lock at item-level in a complete separate database. This database consists only of 1 standalone detail-dataset with just 1 field and has no records in it.
When a user wants to add, modify or delete certain data, a lock at itemlevel is taken on the dummy database. The lockdescriptor is composed as follows:

```
   - descriptionidentifier  (i.e. "CUSTOMERNR")  |
   - value (i.e. "62417")                        | --> "CUSTOMERNR62417"
```

Although the modifications itself require locking on several items, datasets or even databases, only one "logical lock" in the dummy database is required.

(Data of given CUSTOMER can be stored in several records in several datasets and even several databases).
"Logical or soft" locks should be defined for the whole application system, and all programs should use them in the following way:

```
         - conditional lock on dummy database at ITEM-level.
         - if lock is not granted
              user-access in application program changed to "READ" as
              long as user works on this data.
           else
              user is allowed to make his modifications
              lock data ("hard lock" as required by IMAGE)
              update data
              unlock data ("hard lock")
              unlock dummy database ("soft lock")
           endif
```

This system has two advantages:

- only specific "working data" is locked, leaving "THE" database free of locks, so that waiting time for obtaining locks is minimalized.
- no complicate processing when doing the "real" update, so that the number of quiet periods on the database can stay at a good level.

Remark: as long as there are no transactions that lock the complete database, or work on more than on database, the "soft-locking" can be done on a dummy dataset of the database itself.

3.3 The use of dummy sorted paths

IMAGE lacks the capability to do an indexed look-up with the possiblity to retrieve the next or prior value for a given item. If all your keys follow a certain pattern (i.e. DATE, DOCUMENTNR ...) and if entries are most of the time entered in "good" sequence, the following approach can be taken:

Suppose a follow-up program for certain stockmarket indexes: each day all indexes are registrated and the program must give a quick overview of the evolution of them, starting with a date greater or equal to a given date:

```
             I------------I           I-------------I
             I DUMMY-KEYS I           I    DATES    I
             I------------I           I-------------I
               |                                |
               |                                |
path sorted    |                                | 1 entry for each
  on DATE      |     I--------------------I     |             date
               |     I   MARKET-INDEXES   I     |
               |     I--------------------I     |
               |     I  .....    .........I     |
               |     I  .....    .........I     |
               |---->I  |        02 JUL 86I     |
                     I  | c      03 JUL 86I     |
                     I  | h      06 JUL 86I <---|
                     I  | a      07 JUL 86I
                     I  | i      08 JUL 86I
                     I  | n      09 JUL 86I
                     I  |                 I
                     I--------------------I
```

DUMMY-KEYS is an automatic master with just one entry. The path to MARKET-INDEXES is sorted on DATE(YMD). DATE is also an automatic master.
All transactions simply work on the dataset MARKETINDEXES. Because most figures are inputted in sequencial order (the input are the figures of the day), there is little overhead due to the sortpath.

To localize the first entry and read the next entries

```
get in dataset DATES with the given date
while no record found
  compute next day
  get record in dataset DATES with computed date
endwhile
read corresponding entry in MARKET-INDEXES for path DATE
reread same entry in MARKET-INDEXES for path DUMMY-KEY
chained read in MARKET-INDEXES for path DUMMY-KEY
```

Remark: the application program has to check if the startdate falls in "a region of values" that is "supported" by the database. Otherwise, locating the first record can take too much time.

3.4 "Bulk"-handling of information

If one has to store a huge amount of userdata in one single transaction, and if re-screening of it must be done very fast, it is impossible to write all information record by record into a database.

However, if the records are compacted to one big record, the improvement can be astonishing ! Suppose you have a workfile or scroll that has to be saved, and which has 200 lines in it, each of 80 characters wide. The maximum entrysize in IMAGE is 2047 words = 4094

bytes. Up to 50 lines of your workfile or scroll will fit in such one big IMAGE-record, and the complete information can be stored in 4 IMAGE-records!
A very easy compression technique like compression of blanks will further reduce the amount of records with a factor 2 (a lot of data has blanks in it) and even reduce CPU-usage by avoiding extra IMAGE-calls. See further to 3.1 to increase speed when writing to a detaildataset.
Retrieving data goes at the same spectacular speed. At our site, storing and restoring information in this manner requires on a loaded machine less than 1 second for a "screen" with 200 lines.

The price that is paid, is that the data is now in its denormalized form. This can be overcome by writing the modified key-values to a message file.
A second process can now in background read the messagefile and reread the saved (denormalized) entries and write them to a second "CLONE"-database in its normalized form.
Programs that work on information "in the way" it was stored, can still work on the fast denormalized database. The others must work on the normalized one.

Biography

Jan Janssens is since 1980 system manager at Cobelfret N.V. - Antwerp, where he worked exclusively with HP3000-computers. He is a civil engineer of the KU-Leuven and followed MBA at the university of Ghent.

TURBOIMAGE RUN-TIME OPTIONS: BALANCING PERFORMANCE WITH DATA BASE INTEGRITY

Author: Peter Kane, Information Technology Group,
Hewlett-Packard, Cupertino, California, U.S.A.

Summary

Along with improvements in the areas of performance and data base limitations, run-time options were added to TurboIMAGE to allow more flexibility between having high performance and high recoverability. A run-time option is an option which can be used without changes made to any application. In the case of TurboIMAGE, all of the options I will discuss can be enabled through DBUTIL. The purpose of this paper is to compare all combinations of these options, considering the performance impact and what recovery is available.

Introduction

In IMAGE/3000, two run-time options, LOGGING and Intrinsic Level Recovery (ILR) are available. A user can enable LOGGING, ILR, both, or neither. There are two differences between these options. The first is that ILR guarantees only physical integrity, i.e. no broken chains, while using logging and DBRECOV guarantees both physical and logical integrity, i.e. only finished transactions appear in the data base after recovery. The second difference is that with ILR, recovery happens automatically on the first DBOPEN of the data base following an interrupted DBPUT or DBDELETE, while using DBRECOV means restoring an old copy of the data base and waiting while all finished transactions in the log file are issued. Whatever the wait on DBRECOV, it is a much less lengthy process than recovery of a data base not using either ILR or logging, which is by DBUNLOAD and DBLOAD (or a third party utility).

Another option which is not a run-time option since it means application changes, is output deferred mode. Output deferred can be enabled in IMAGE/3000 only when the data base is opened in mode 3 (exclusive modify access), and is enabled by calling DBCONTROL with mode 1. Output deferred can therefore be used only in single-user environments. This mode can drop significantly the elapsed and CPU times needed by DBPUT, DBDELETE, and DBUPDATE. The reason is that modified buffers and set labels are not written to the data base until either the buffer is needed to hold a different data block, or else a DBCLOSE mode 1 or 2 is issued. The drawback of output deferred can be inferred by this last point, which is that a system failure occurring in the middle of an output deferred application can leave a badly damaged data base. The usual use for output deferred is in a batch environment, where the data base is stored before running the applications. If the system fails while the applications are running, the stored data base would be restored and the applications would then be restarted. Output deferred mode can be used in an interactive environment, if logging is set up (otherwise a DBUNLOAD and DBLOAD is necessary if the system fails). However, exclusive access is required, which usually eliminates output deferred for consideration in an interactive environment.

The above options allow some flexibility, but some issues have been outstanding:

1. If there are a lot of transactions on a log file, recovery to achieve logical integrity can mean a lot of down-time.

2. In shortening the maintenance cycle so that recovery can take less time, the data base must be stored more often. Users can not issue transactions against the data base during these times.

3. Applications using output deferred mode must be operating exclusively. Therefore batch processing can take longer than necessary.

4. A high volume of modifications can have a major impact on performance.

Therefore two more run-time options have been added to TurboIMAGE as answers to the above issues. These are: AUTODEFER, or multi-user output deferred mode, and ROLLBACK, or Rollback recovery which allows a faster recovery than IMAGE's recovery method (which I refer to as Roll Forward recovery). The options which have been available in IMAGE have all been carried over to TurboIMAGE, giving four different run-time options. These options can be used in seven different combinations to balance performance and integrity. In this paper I will discuss each combination, specifying the advantages and disadvantages and my recommendations for its use.

Combinations Available with TurboIMAGE

The following is a list of the possible combinations:

1. No options used
2. LOGGING enabled
3. AUTODEFER enabled
4. AUTODEFER and LOGGING enabled
5. ILR enabled
6. ILR and LOGGING enabled
7. ROLLBACK enabled

It may noted that some options are not compatible with others, for instance AUTODEFER and ILR are not compatible. It also should be noted that enabling ROLLBACK automatically enables LOGGING and ILR.

Each of these combinations will now be looked at separately.

Combination 1: No options used.

Performance: Modified buffers and labels are written to disc (or cache if caching is enabled and BLOCKONWRITE is set to NO), within the intrinsic which did the modification. This means that modify intensive applications must wait frequently for writes to occur. It also means that buffers are never left dirty after an intrinsic finishes, which has a positive impact on read intensive environments (more on this in the discussion of AUTODEFER).

Integrity: If the system fails during processing, physical corruption may result. If this happens, a DBUNLOAD and DBLOAD (other utilities from third parties may be used instead) is necessary to recover the data base. Note that logical recovery is not possible in this case. If disc caching is used and BLOCKONWRITE is set to NO, multiple corruptions can result from a single system failure.

Advantages: The multi-user, read intensive environment probably sees the best performance with this combination. The overhead of logging and ILR is not seen.

Disadvantages: Very lengthy recovery if a system failure has caused physical corruption.

Combination 1 (continued)

Recommendation: Use for read intensive (approximately 80% reads, 20% modifications) application mixes on non-critical data bases. Modify intensive environments can be improved in performance by using other options.

Combination 2: LOGGING enabled.

Performance: Contrary to what many users believe, logging causes only a slight (ranging from about 3% to 8%) degradation in performance. The higher end of this range is usually seen in the modify intensive environments. The reason the degradation is slight is because with only logging enabled log writes stay in memory until a DBEND or a DBCLOSE is issued, or if the log buffer in memory fills up. Therefore this combination is only slightly lower in performance than using no options at all.

Integrity: Physical integrity can be achieved without using the lengthy process of DBUNLOAD and DBLOAD. Logical integrity is possible if the applications are written using DBBEGINs and DBENDs. By logging to tape, disc failures can be recovered from.

Advantages: Physical recovery is far less lengthy than DBUNLOAD and DBLOAD. User specific data can be obtained from the log file. Logging to tape or to a different disc from the data base can provide recovery from media failures.

Disadvantages: Dedicated tape drive or usage of disc space for log records. To attain logical recovery, applications must have DBBEGIN and DBEND calls to define logical transactions. Periodic down-time is necessary to back up the data base and start a new log cycle.

Recommendation: Probably best use is in read intensive environments where logical transactions have been defined, where logical recovery is desired, and where application performance is more of an issue than down-time required to recover from a failure. Also provides best protection against media failures.

Combination 3: AUTODEFER enabled.

Performance: Usually will prove to allow the best performance, especially in modify intensive environments. Dirty buffers and labels are not flushed to disc (or cache) until DBCLOSE mode 1 or 2, or unless a buffer is needed to hold a different data block from the data base and all other buffers are dirty. From this one can see that TurboIMAGE will try to keep dirty buffers in memory as long as possible, in an effor to eliminate unnecessary writes to disc. This is useful if users are modifying the same buffers over and over again. However, if one user modifies a buffer containing a data block no other user ever accesses, that block may stay in its buffer for a long time. This will mean less buffers for the other users to do reads, which will in turn mean that buffers may be overlayed with other data blocks before the original user is through. This has an impact on the read intensive environment where there are occasional modifications. In the modify intensive

Combination 3 (continued)

Performance: (continued) environments this is not much of an issue because all of the buffers are modified in time.

Integrity: In short, NEVER use AUTODEFER by itself in interactive environments. This is because the user never has any idea whether the modified blocks or labels (which contain data set ends of file, delete chain heads, etc.) have made it to disc until DBCLOSE time. A DBUNLOAD/DBLOAD is necessary to recover anything at all if the system fails while applications are running. AUTODEFER is fine with batch processing, if a store of the data base is done first. Then if the system fails, the data base could be restored and the applications redone.

Advantages: Highest performance in applications which do more than an occasional modification.

Disadvantages: Physical integrity is highly at stake. Logical recovery not possible at all.

Recommendation: Batch processing where applications modify the data base more than occasionally.

Combination 4: AUTODEFER and LOGGING enabled.

Performance: Since the log writes are buffered by MPE until the buffer fills or until DBCLOSE or DBEND, logging adds very little performance overhead in this combination as opposed to having AUTODEFER alone. The performance advantages of AUTODEFER are still realized with this combination.

Integrity: Roll forward recovery is available. Therefore, physical integrity can be achieved, while logical integrity can be achieved if transactions have been defined in the applications using DBBEGINs and DBENDs.

Advantages: High performance in applications which do more than occasional modifications along with a recovery method in case of a system failure. May be the best combination for environments which are CPU bound.

Disadvantages: Roll forward recovery is not the fastest recovery method. Downtime is necessary to back up the data base and start a new log cycle. Dedicated tape drive or disc space is necessary.

Recommendation: Use in interactive environments where application performance is highest concern, and where data base modifications are done more than occasionally.

Combination 5: ILR enabled.

Performance: Performance degradation with modifications, for two reasons. The first is because there are at least two additional writes to disc to update the ILR file for each DBPUT and DBDELETE. The second reason is that all writes to the data base and to the ILR file go through the Serial Write Queue. TurboIMAGE calls FSETMODE to set this file system option if it determines that ILR is enabled. Going through the Serial Write Queue means that writes can not operate concurrently if they are to different discs.

Integrity: Automatic physical recovery on the first DBOPEN of the data base following an interrupted DBPUT or DBDELETE. ILR has been enhanced to redo the interrupted intrinsic rather than rolling it out as in IMAGE. No logical recovery is available. ILR alone does not protect against media failures.

Advantages: Quick physical recovery method, which is automatic. Easy to use.

Disadvantages: Performance degradation for applications using a high number of DBPUTs and DBDELETEs. No logical recovery available. Can not recover from media failures. Not compatible with AUTODEFER.

Recommendation: For environments with a low volume of DBPUTs and DBDELETEs, this is an inexpensive and painless way of insuring physical integrity. For applications without DBBEGINs and DBENDs, this may be more useful than enabling ROLLBACK.

Combination 6: ILR and LOGGING enabled.

Performance: Logging will cause a slight amount of degradation over having ILR alone enabled.

Integrity: Can achieve quick and automatic physical recovery with ILR, and can have logical recovery with DBRECOV. Media failures can be recovered from.

Advantages: Quick physical recovery method. Logical recovery available. Recovery from media failures is available.

Disadvantages: Performance degradation due to ILR during DBPUTs and DBDELETEs. Logging maintenance.

Recommendation: Probably best use is in environments where ILR's automatic recovery is the usual recovery method, and where logging is used to protect from media failures. For environments where logical recovery is possible (DBBEGINs and DBENDs are used), ROLLBACK is probably a better option.

Combination 7: ROLLBACK enabled (ILR and LOGGING will also be enabled).

Performance: This combination has some more degradation over Combination 6. This is because log writes will now be written directly to disc using BLOCKONWRITE instead of being buffered by MPE as in all of the other combinations using logging. Furthermore, since all data base and ILR writes on this option will go through the Serial Write Queue, the log write will have to wait for any or all of the previous writes before it can be written itself. Therefore, DBPUT and DBDELETE should see the most performance degradation from this method.

Integrity: Rollback recovery used for system failure or "soft crash". Roll forward recovery can be used if there has been a media failure or "hard crash". Logical and physical recovery are both available.

Advantages: Quick logical and physical recovery (rollback is much faster than roll forward recovery). Stores of the data base do not have to be taken as often, since the length of the log file is not as great of an issue with rollback.

Disadvantages: Performance degradation, especially on DBPUTs and DBDELETEs. DBUPDATEs also affected.

Recommendation: Use where transactions have been defined and where it is crucial that the data base be available, and logically intact.

Biography

Peter Kane
has been with Hewlett-Packard for the last 3 1/2 years. He is an Online Support Engineer for Data Base products, which includes IMAGE/3000 and TurboIMAGE/3000. He in the past was responsible for SE training on the same products.

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

Orland Larson
Hewlett-Packard Company, Cupertino, California, USA

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 non-relational 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 demonstrat 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.

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 my company have an excessive backlog of applications to be developed, including an invisible backlog?

2. Are we spending too much money developing applications due to the complexities of using non relational systems?

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

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

5. Do our users feel restricted by a non-relational database?

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

7. Is the nature of our 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?

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.

INDEX SEQUENTIAL ACCESS TO IMAGE DATA BASES

Wolfgang Matt
Industrieanlagen-Betriebsgesellschaft mbH,
Einsteinstrasse 20, D-8012 Ottobrunn, West Germany

Summary

IMAGE, though being a very successful data base system, has essential drawbacks: it does not allow access by generic keys and sorted chains show bad performance. Using KSAM in addition to IMAGE solves some problems, but means additional programming and extensive use of system resources. It is also not possible to log and recover IMAGE and KSAM consistently, unless you rewrite the logging software. This paper presents a software designed to enhance IMAGE for indexsequential access i.e. generic keys and sorted chains. The software uses only IMAGE and is itself IMAGE compatible (same database, intrinsics and calling sequences). It does not use privileqded mode.

The user can define indexsequential access paths for any item of master and detail data sets. These access paths are treated like IMAGE chains, but unlike IMAGE chains they can be defined and deleted without unloading the database. Many user applications (e.g. TRANSACT programs) can take advantage of generic keys and sorted chains without program changes, but programmers can take full advantage of the features by calling DBFIND and DBGET with mode parameters not used by IMAGE.

The paper compares the new access method with traditional access methods with regards to features, usability and performance.

Performance of Typical Database Accesses

We consider a typical data base consisting of a CUSTOMER master and an ORDER detail. Both are connected by the search item CUSTNR, the customer number. For the ORDER detail there exists a second search item, the article number ARTNR (connected to an automatic master).
We now consider some typical inquiries to this data base to see how IMAGE performs.

1. Retrieve name and address of a customer with a given CUSTNR. IMAGE does a hashed access which is very efficient.

2. Retrieve the address of a customer with a given name. A serial read has to be done which is very inefficient.

3. Retrieve all orders for a given customer number ordered by article number.
   If we have a sorted chain, retrieval is efficient, but adding and deleting is not. If no sorted chain is defined, an online sort must be programmed which is not efficient.

4. Retrieve all orders for given customer and article number.
   A chained read must be done which may be inefficient if many entries are discarded.

5. Retrieve all customers which buy a given article.
   A chained read on the second path is very efficient.

6. Retrieve all customers which buy a given group of articles identified by the first two characters of the article number. A serial read has to be done which is very inefficient.

It can be seen from these examples that IMAGE performs well when the access is made using exactly one of the fields defined as IMAGE keys. Performance is bad when we need an access by a none key field of a master, by a combination of keys, or by a partial key. Since these questions exist, we have to look for an alternative method of accessing data in an IMAGE data base. This method should be able to retrieve long keys (i.e. the combination of several IMAGE fields) the same way as partial keys (e.g. the first character of a customer's name). This method should also provide the possibility of sorted retrieval without the overhead of sorted chains.

The B-Tree Method

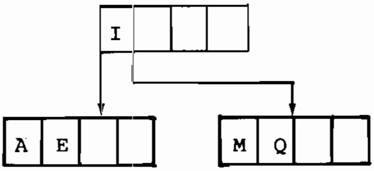
The method which allows partial key access and retrieval in ascending order is called index sequential access. One starts with a given index and reads from there in a logically sequential order. HP has implemented this access method in KSAM but not in IMAGE.

The method used by KSAM and tools of other manufacturers is based on the B-tree principle invented by R. Baier. Keys and data are stored in a special way which minimises disc read and allows fast access to any key. In order to make this method efficient, only pointers to the data are usually stored within the B-tree. This method is ideally applicaple to data stored in IMAGE data bases, since we only have to add a B-tree for the keys. A B-tree consists of blocks containing key values, the associated data pointers and pointers to other blocks.

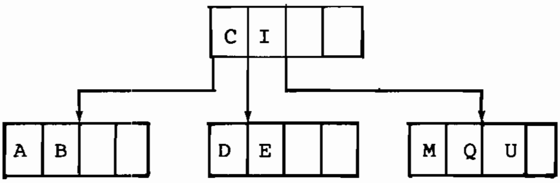
To illustrate the B-tree method we use very small dimensions: a key length of one character and assume that 4 keys fit into one block. We initially want to store the letters A E I M Q U. After storing the first four letters the first block is full

A	E	I	M

To add Q the block must be "splitted":



The new root block contains besides the key I the pointers to the two leave blocks. After adding the key U into the second leave block we want to add the remaining letters of the alphabet. We can add the keys B and C into the first leave block by shifting E but to add D we have to split again:



Using the B-tree we now demonstrate the retrieval of a key e.g. key E. First the root block is read. It does not contain key E, but since E is located in the alphabet between C and I (the two keys in the root block), the pointer is followed and the second leave block retrieved. In this block the key E is found and also the associated data pointer (not shown in the figures above).

Interfacing B-Trees with IMAGE

When we started to design an index sequential access method for IMAGE data bases, we decided not to use one of the following approaches:

1. We did not want to use KSAM. The reason was that standard recovery methods did not allow to keep KSAM and IMAGE files consistent in case of a system failure. Another disadvantage of KSAM is the extensive use of system resources.

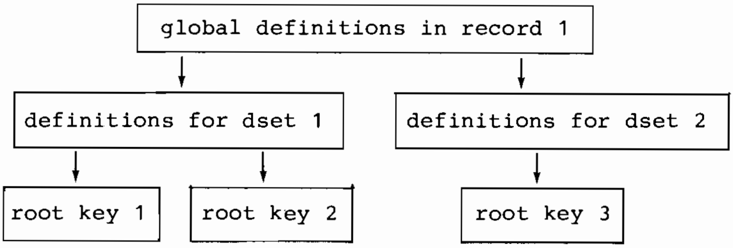
2. We did not want to change the IMAGE source code. This was done to be independent of HP enhancing IMAGE (e.g. TURBO IMAGE).

3. We did not want to use privileged mode. Access to IMAGE is done but solely using the documented IMAGE intrinsics. This makes the software independent of internal IMAGE changes and adds to system security.

For the interface software we defined the following requirements:

1. Indices must be covered by logging

   The B-tree must be stored in a data set within the data base to which it refers. Neither an MPE file nor a seperate data base can provide a consistency between data and keys in case of a system failure. A separate program to reconstruct the B-tree from the data may run for hours, while roll-back recovery can be done within a few minutes using TURBO IMAGE. One extra data set (stand alône detail) is sufficient to store the B-trees of all fields (or combinations of fields) for which index sequential access is defined. The data set also includes the definitions themselves. This leads to a logical structure like this:



2. Indices must be updated automatically

   It would be tedious and prone to error if a programmer had to code a normal DBPUT and then an intrinsic call to update the B-tree, code a DBBEGIN and DBEND around it and also include the index data set in the lock descriptor. We need a "super DBPUT" which does this all automatically. The same is true for DBDELETE and also for DBUPDATE, since we allow the new keys to be updated. Since DBDELETE does not have an argument buffer, the "super DBDELETE" must find out which keys have to be deleted according to the latest DBGET.

3. The calling sequence must be identical

   Though the "super DBPUT" does everything necessary, you do not want to change all programs replacing the call to DBPUT by a new call with additional parameters and pass new parameters from routine to routine. No - the "super DBPUT" must be called DBPUT and it must have identical parameters. Only this garantees existing programs to run without modification, recompilation or patching. The only restriction is that the new intrinsics must reside in an account SL, since they call the HP intrinsics in SL.PUB.SYS.

4. The logic of retrieval must be unchanged

   Though partial key access is an addition to the standard IMAGE retrieval methods, it should be done by the same logic. You want do a DBFIND with the partial key as argument, and a series of DBGET's (mode 5) should retrieve all entries belonging to this partial key. An end-of-chain indicator (condition word 15) should also be returned. But somehow you have to tell the software, how long the partial key is. This is the only information not provided in the standard IMAGE calling sequence. We provide two methods:

   - new values for the mode parameter allow to code the length of the partial key and also allow to specify wether the search should start at the lowest or the highest key which matches the partial key. In the latter case a backward read (mode 6) may be performed.

   - software which cannot use anything but mode 1 for DBFIND (existing programs, RPG, 4GLs) can terminate the argument by a @ . All characters up to the @ are treated as partial key. This is especially convenient when an automatic master is replaced by index sequential access. You do not need to change any of your programs.
     This method also works for 4GLs like TRANSACT. But some 4GLs do not trust IMAGE. They compare the result of DBGET with the argument of DBFIND, which is of course not identical for partial key access.

Biography

Wolfgang Matt holds a PhD in physics. Since 1977 he works with IABG, a company with 1700 employees near Munich. He is head of a group of scientists, consulting HP 3000 users and developping individual software for them. He is the author of SI-IMAGE, a product for index sequential access to IMAGE data bases.

Database Dynamics

F. Alfredo Rego

Adager

Apartado 248
Antigua
Guatemala

The disciplines of Database Dynamics deal with the normalized design, maintenance and orchestration of databases which perform well under heavy-duty use.

In this essay, we see specific examples based on IMAGE, the award-winning database management system built by Hewlett-Packard for the family of HP3000 computers. These examples illustrate fundamental principles which are simple, timely, timeless and, above all, powerful.

Database buzzwords

A database models the dynamic behavior of entities and their relationships by means of *entries*. An entry consists of a key (which uniquely identifies the entity or relationship) and a collection of attributes (which give quality and color to the entity or relationship).

Entities and relationships don't just sit there. They interact with one another and with their environments: Transactions happen which affect (and are affected by) entities and relationships. Such transactions include changes in database structure as well as changes in the meaning and value of the information maintained by the structure.

We cannot store a real entity or a real relationship in a database, just as we cannot store a *real* orchestra in a stereo cassette. At best, we can hope to store a half-decent description or representation which, through the magic of electronics, will play back a reasonably useful likeness. The representation, due to limitations of technology and economics, will consist of a group of values for a relatively small collection of characteristics which, in the case of databases, we call *fields*.

A *dataset* is a homogeneous collection of entries. There are different kinds of datasets, each optimized for a specific access technique. In IMAGE, we like to use *master* datasets to keep entities and *detail* datasets to keep relationships. (IMAGE master datasets come in two flavors: manual masters and automatic masters. Please see the IMAGE reference manual for specific details). Naturally, we may use all kinds of conglomerates of physical datasets (masters and/or details) to represent logical master *or* detail datasets. It all depends on our choices of specialized indexing techniques.

To make an IMAGE database functional, we access its entries in a variety of ways, ranging from serial scans of entire datasets (*the only way to go* in the good old days of batch machines) to hashing and chaining (quite convenient for online applications). Hashing and chaining are techniques based on direct access to specific addresses so that we may jump directly into the entry or entries which interest us without having to wade through millions of irrelevant entries. Please see the IMAGE reference manual for a detailed discussion of hashing and chaining.

The database challenge

Fundamentally, we are interested in two database operations: the addition of new entries and the finding of existing entries (so that we may relate them, report them, update them, or delete them). A Database Management System (DBMS) attempts to help us in the pursuit of these worthy objectives.

Structure vis-a-vis interface

For *efficiency's sake*, a DBMS has some type of internal structure to find and assemble entries. For *convenience's sake*, a DBMS has some type of user interface to create, maintain, and relate entries to produce, somehow, information on a timely basis. The resulting entries which the user "sees" through the interface may be real (if they exist physically in the database) or virtual (if they are the result of relational operations on real or virtual entries).

The user interface serves as an ambassador between the raw bits-and-bytes computer stuff and the human-like specifications of the end-user. A poor interface imposes the restrictions of the structure upon suffering users. A good interface shields users from the structure's shenanigans, while still being able to take full advantage of the structure's properties. A good user interface is as efficient as possible without being obnoxious. An interface knows the internals of the database structures as well as the externals of the user desires, and spends its existence translating back and forth between bits and thoughts. This may very well be the fastest kind of shuttle diplomacy!

Complexity and normalization

Ideally, things should be simple. Unfortunately, though, things *are* complex. But we should avoid *unnecessary* complexity. This is the objective of normalization, in the mathematical sense.

Normalization is the breakdown of seemingly-complex operations into simpler processes. The challenge, at the beginning, is to place the appropriate resources (no more and no less) where they belong, at the appropriate level, at the appropriate place, at the appropriate time. Then, the challenge continues, since we must be able to reallocate resources quickly and effectively to balance the load, at any time, all the time. Normalizing is an ongoing, dynamic activity.

Normalization applies at every level in the global computer hierarchy:

- entry
- dataset
- database
- computer
- node
- network

A normalized structure is open-ended. We can add more elements at any layer without affecting existing systems. We can delete elements from any layer without affecting existing systems which do not access such elements.

Efficiency and normalization

Do we want to favor efficiency in terms of *access* or do we want to favor efficiency in terms of *maintenance*? In general, the higher the degree of normalization (i.e., the finer the splitting into chunks), the higher the communications and coordination costs. Normalization is neither good nor bad. It is simply a method which allows us the freedom to choose our favorite spot in a spectrum (or rainbow), which has highly unnormalized databases at one end and highly normalized databases at the other.

Usually, efficiency in terms of access implies redundancy. But redundancy, in itself, is not bad. It is just more difficult to maintain a bunch of redundant things in perfect synchronization. This is analogous to a one-man band who must play all kinds of disparate instruments in a (more or less) coordinated fashion.

Usually, efficiency in terms of maintenance implies simplicity of roles and a multiplicity of role-players. If we want to change one role, we only have to change one player. But it can be a drag to keep track of thousands of players. This is analogous to those fascinating groups of musicians who play bells, one bell per musician. Each person is a specialist who can only play one note. In terms of maintenance, we can see how difficult it would be to tune a complex instrument during a performance and how easy it would be, on the other hand, to simply exchange a bell which is out of tune.

A super-normalized database contains a large number of small entries, with many instances of key fields distributed over many datasets. Even simple queries may require that we assemble the information from many sources. But we may have a better chance that each of these sources is correct. It is simpler to maintain a "specialist" source up to date than it is to maintain a complex source which tries to keep track of everything at the same time, like a one-man band.

Even though it is paradoxical, our experience shows that normalized databases may actually occupy less total disc space than unnormalized databases. Particularly if the keys are short, which, fortunately, seems to be the case most of the time. For instance, your identity number is probably shorter than your job description. Naturally, we can go to ridiculous extremes and normalize a database to death. We could conceivably chop up the information about an employee in many entries, each containing a single attribute such as name, birth date, salary, and so on. But this would really be splitting hairs! Common sense should prevent us from committing such atrocities, and this is the motivation behind the rules for the fifth normal form: An entry is in fifth normal form when there is nothing significant left to normalize!

Access strategies

In an online database system, we want to get information about given entities and their relationships while somebody waits over the counter or over the telephone. This means that we want to find the entry (or group of entries) of interest to us, among millions of entries, as efficiently as possible. We should design (and periodically tune) our database systems to provide the fastest possible response time for the most important transactions and queries.

Some people have spent endless amounts of time and talent on a fascinating problem: How to minimize the effort required to answer the most infrequently-asked (and most arcane) questions. Other people have invested their time and talent on another problem: How to minimize the effort required to answer the most frequently-asked (arcane or not) questions, while still preserving a reasonably efficient environment for those who must toil, on a daily basis, with the thankless task of feeding and baby-sitting the database.

IMAGE provides two access methods which are optimized for efficiency: hashing and chaining. We may access entities (in master datasets) by means of hashing and we may access relationships (in detail datasets) by means of chains which IMAGE maintains for us as we add or delete detail entries. These are *contents-oriented* access modes (as opposed to *address-oriented* access modes, such as serial or directed).

IMAGE allows us the freedom to go "explorer-like" with sequential and direct access methods. It also allows us the convenience of traveling through "pre-established hubs" by means of techniques such as hashing and paths. We do not have to access anything in a predetermined way. But it is nice to know that we may do so, if we know that a given "routine-route" will get us more quickly to our desired destination. Why wade through swamps if we can use a bridge? Why swim across the Atlantic if we can take the Concorde?

Naturally, we may have valid reasons (usually having to do with convenience, performance, or both) that motivate us to use our own combinations of physical master and detail datasets, with or without physical paths, to model a given collection of entities and/or relationships. Usually, these valid reasons are dictated by our choices of customized indexing techniques which we build on top of IMAGE's intrinsic access modes. (IMAGE itself does not have indexing. IMAGE provides pre-fabricated access methods which allow us to implement all kinds of indexing strategies, according to our pleasure).

Entities, relationships, and keys

In terms of space, an entity may be related to zero, one, or more entities (of its own class or of different classes). In terms of time, these relationships may happen all at once or they may happen one after another, in a strictly sequential fashion. To make things more interesting, some virtuoso relationships may come all at once in an unending sequence of complexities!

A relationship *is* an entity. It all depends on our viewpoint. For example, a marriage is a *relationship* between two people, and a marriage is also the subject of attention of a marriage counselor who treats it as an *entity*. By the same token, an entity *is* a relationship. For example, an individual is an *entity*, and an individual is a *relationship* formed by internal organs, genes, environment, and so on. It is a matter of *convenience* to designate some "thing" as an entity or as a relationship.

Usually, a relationship's key is a concatenated key, composed of a collection of the keys of the related entities. If we can relate the same entities in different ways or under different circumstances (thereby giving rise to several detail entries to represent the different relationships), then each relationship's key must include some additional attribute(s) which define the differences. For example, consider *discretionary* pricing (or *discriminatory*, or whatever you may want to call it). In this case, the price of a product for a customer may depend on the part's supplier, on the customer's rating within the company, on the order date and/or on the ship date, and so on. In other words, the price is an attribute of the relationship among all these entities; the price is not an attribute of the product alone.

IMAGE's implementation highlights

An IMAGE entry (master or detail) models an entity or a relationship with equal ease. The only difference between a master entry and a detail entry is the method of access: master datasets are biased for hashing while detail datasets are biased for chaining.

An IMAGE dataset (master or detail) is a homogeneous collection of entries and an IMAGE database is a homogeneous collection of datasets. Since the fundamental atomic unit is the entry, let's review its main features. An IMAGE entry has:

- A unique identifier (key) for the represented entity or relationship;
- Attributes (if any) which further qualify the characteristics of the entity or relationship.

Please note that a key is simply a field (or a collection of fields) which uniquely identifies an entry. A key does *not* have to be an IMAGE search field. IMAGE search fields are defined only for performance's sake, to allow paths between master and detail datasets. Paths are particularly attractive for online access to fashionable relationships, since paths tell IMAGE to maintain appropriate physical linkages when adding or deleting entries.

Since entities and relationships are equivalent, IMAGE uses the same construct ("entry") to represent either an entity or a relationship. For convenience (and performance) you may want to use master datasets as repositories of entities and detail datasets as repositories of relationships, since masters are biased for hashed access while details are optimized for chained access. This would allow you to pick out the entry that interests you *right now* (by means of hashing into the master) and would display its relationships *right now* (by following chain links in detail datasets, controlled by chain heads in the master entry).

The order of keys and/or attributes in an entity (or in a relationship) is arbitrary. Therefore, the sequence of fields in an IMAGE entry is also arbitrary. To allow for stability within this flexibility, IMAGE provides the *list* construct to map *any* subsets and permutations of key(s) and/or attribute(s) to/from the user's buffers. This permits us to add, delete, or reshuffle fields without the need to recompile *all* the programs which access the affected dataset(s). We must recompile only those programs which explicitly access the affected fields. This gives us a high degree of data independence.

Database Dynamics

The concepts of *Database Dynamics* deal with the orchestration of the transactions which affect (and are affected by) databases. A transaction is something that adds, deletes, or modifies an entry (or a collection of entries, in the case of a complex transaction).

We use *picoseconds* (trillionths of a second) to measure events which we think are super-fast. We use *aeons* (billions of years) to measure events which we think are super-slow. Somewhere in the middle of this wide spectrum we find the events which occupy most of our attention in our daily concerns. By definition, these are the events which are the most useful and interesting. Most IMAGE databases, for instance, keep track of entities and relationships whose event-speed ranges from a "fast" which we can measure in hours to a "slow" which we can measure in years.

The functional dependencies among keys and attributes will tend to show a remarkable stability, particularly if you cluster things around entities and relationships which are obvious to you. For instance, the functional dependency between a personal identification number and the name of a person will probably hold for life. Nevertheless, the particular manners in which people access, combine, manipulate, present, and otherwise massage the data contained in the database will tend to change according to the inevitable shifts in the organization's political winds.

Given these facts of life, it might make more sense to focus our limited energies and resources on the analysis of the most permanent things: entities and their relationships. As a bonus, we find that this entity-relationship approach automatically and conveniently requires very simple interfaces to maintain (and obtain) information using the database.

Naturally, stability should not imply inflexibility. The challenge is to be as stable as possible while still being sufficiently flexible and adaptable to changing environmental conditions. But there should be some back-bone to the whole thing!

A practical database methodology

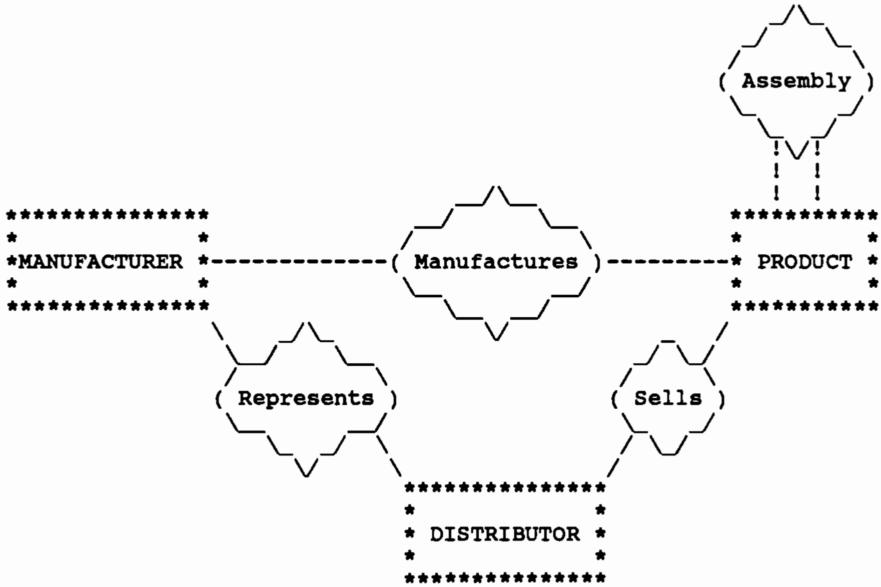
All this nice database theory is certainly a lot of intellectual fun. But you *also* have to address the practicalities! Specifically, you have to remember that your ultimate responsibility is to develop an application system which uses databases only as a means to an end.

Since 1974, I have kept copious notes of theoretical and practical issues which have influenced my failures as well as my successes. The integration of these notes has led me to a practical database methodology whose ideas and steps I consider simple, timely, timeless and, above all, powerful. Here is the outline:

First of all, classify your Entities and your Relationships

Graphics are great for the process of classifying *and* for displaying the resulting classification! I like to use *rectangles* to represent collections of entities, *circles* to represent collections of relationships, and *lines* to make relationships explicit. Since entities and relationships are equivalent, this is a valid choice of geometric figures: After all, a rectangle and a circle are topologically equivalent!

Regardless of the graphics you use to guide your classification, your entities and your relationships will conveniently fall into categories which are obvious to you and to people who are versed in your business. For instance, if you are a manufacturer or a distributor, you could choose something along these lines:



Notice, with pleasure, that this fundamental step of classifying your entities and relationships has all kinds of bonuses. First of all, you will get a clearer picture, in your own mind, of your own system. Later on, you will also see that the resulting IMAGE database(s) will automatically have a clean and elegant design and will be in a very respectable state of normalization.

As an interesting example of a bill-of-materials, modeled with a minimum of database elements, please see the "Assembly" relationship which relates "products" to "products" (or, if you prefer, you may use "parts", or "components", or whatever, instead of "products") so that we may quickly answer either of these questions with equal ease: "Which products do I need to assemble this product?" and "Which products can I assemble with this product?".

Let's see now an example which stresses the importance of the attributes associated with *relationships.* Even though the related *entities* are important in themselves, we will see that shuffling things around a little bit to place the spotlight on the relationships may reward us with pleasant surprises. In this example, we will take attributes which are commonly assigned to entities and we shall assign them to relationships. This way, the entities are free to "wear" different attributes, depending on their relationships, without being stuck with them for life. This is the essence of dynamism, after all! Just for fun, let's study a database model of presidential administrations, on a world-wide scale. Please stretch your mind beyond any parochial limitations:

```
                       ********
                       *      *
                       * YEAR *
                       *      *
                       ********
                         !  !
                         !  !
                         !  !
                        _!__!_
***********        __/          \__        *************
*         *         /            \         *           *
* COUNTRY *        / Presidential \        * PRESIDENT *
*         *-------( Administration )-------*           *
                   \              /        *************
                    \__        __/
                       \_______/
                           !
                           !
                           !
                           !
                       *********
                       *       *
                       * PARTY *
                       *       *
                       *********
```

Notice that this design does not include restrictions such as *citizenship* and *uniqueness.* The same person could be the president of more than one country at the same time and many persons could be simultaneous presidents of the same country (have you heard recently of "presidents in exile"?). You can quickly find an administration by beginning year or by ending year, as well as all current administrations regardless of their beginning. Without any radical changes, this same design could apply to directors of corporations (and would be very useful to trace interlocking boards!)

Time for a little computerese!

Translate your nice graphics to IMAGE's database definition language. Create an IMAGE *schema* which the schema processor (DBSCHEMA) will understand. Rectangles ("collections of entities") translate immediately to master datasets (either manual masters or automatic masters, depending on your orchestration style; don't lose too much sleep on this). Circles ("collections of relationships") translate immediately to detail datasets. Lines which represent obviously "hot" relationships translate immediately into paths (so that we may use IMAGE's hashing and chaining shortcuts to find the entities and relationships which we want at any time). Lines which represent "so-so" relationships are, by definition, not worthy of paths. These lukewarm relationships will rarely pop up in our daily database usage. If they surface every now and again, they will become the subject of serial scans (which are not so bad if you do them only once a month in the middle of the night). If we notice an alarming trend in the rate of serial scans, then we simply add a path. No big deal!

Refine your indexing for performance

IMAGE's *search fields* just happen to be convenient for the sake of IMAGE's hashing algorithm (which converts a data value to an address) and IMAGE's chaining algorithm (which links logical neighbors even when they are millions of entries away from each other). But you can design *any* mathematical mapping of your choice that will convert any data value into a reference to whatever keys you may have defined for IMAGE. Don't stop at the obvious. Let your creativity soar to new heights. IMAGE will be delighted to cooperate with you. If you need some inspiration, simply ask a fellow HP3000 user. You will be amazed at the unbelievable variety of IMAGE indexing techniques in existence today. It is *fun* to do fantastic things with IMAGE.

For instance, you may get sophisticated and decide to use *clusters* of masters and details to index selected entries which reside in specific masters or in specific details. If your indexing incorporates trees and other structures which facilitate keyed sequential access to entries, you may consider stand-alone detail datasets as ideally suited for the storage of such specialized structures. There is no reason to have indexing structures residing *outside* of IMAGE if we can have them as full IMAGE citizens. A stand-alone detail dataset is equivalent to a standard MPE file and offers many additional advantages, such as IMAGE's buffering, backup and protection mechanisms, locking, and remote access.

Choreograph your Transactions

This is the dymanic part! Specify the transactions that will allow you to add, modify, delete, and report these entities and their relationships. Decide whether or not some of these transactions need to be undisturbed by other concurrent transactions. Take advantage of IMAGE's locking to make sure that you achieve a fair compromise between *privacy* and *sharing*.

Perform your Transactions

At your convenience, add, delete, find, modify, relate, and report entries. Do it solo or invite your friends and fellow workers, from the next desk, from the next building, from the next country, or from anywhere in the network. IMAGE is a multi-tasking multi-computer database management system, after all!

A good performance implies the orchestration of a myriad of simple technical details into an impressive, overwhelming presentation. The presentation *is* your application. The technical details are the result of your normalization. Since you carefully allocated the appropriate resources (no more and no less) where they belong, at the appropriate level, at the appropriate place, at the appropriate time, you have a balanced performance.

Tune up your Performance

Balance, though, by its very nature, is a dynamic concept. You cannot just relax and assume that you will never lose it!

As you specify your masters, your details, and your paths, keep in mind that the important question is: "Can you define, redefine or cancel these entities and their relationships at any time during the life of the database?" For performance reasons, you may want to wire some *obvious* relationships *hot* in the database's structure by means of paths. But you do not want to be stuck for life, since some hot relationships may cool off and some sleepers may wake up unexpectedly!

The same questions apply to every component of your database and the same advice applies to every database administrator: Fine tune things in such a way that you reach a reasonable compromise between the *response time* for any of these functions and the *global throughput* for the whole transaction load.

Bravo! You are now a database master, thanks to IMAGE.

Hewlett-Packard's IMAGE database management system has unique mathematical properties which are natural consequences of its original design criteria. These IMAGE properties allow you to model your entities and their relationships in a nicely normalized fashion, without any unnecessary and inconvenient convolutions.

Take advantage of IMAGE's properties. They are sound and they are classic. But they do not have a life of their own. They need you!

Like any fine instrument, IMAGE is there, dormant, waiting for you to wake it up with *your* dynamism. Play it well, with soul. At the beginning, you may want to join a group of fellow enthusiasts, to improve your technique while you develop your style. Eventually, you may want to solo. In any case, happy crescendo!

Appendix A

IMAGE schema for the *Distributor* database mentioned in the *Practical Methodology* section.

```
Begin database DISTR;

<<

NOTES:

 Your imagination and convenience should decide how many (and
 which) attributes to include at the "...".

 The paths are NOT necessary at all, but we include them as
 examples of performance boosters for relationships which
 seem to be "hot and heavy".  You can always take ALL paths
 away, or add OTHER paths at will.  IMAGE does not care: You
 are free to play with the tradeoffs involving time, space
 and the bias on efficiency.

 Capacities can go from one entry to several million entries.
 The help of intelligently-defined paths becomes more obvious
 when you deal with millions of entries.  Toy-like academic
 examples, of course, do not require any overhead in terms of
 structure.

>>

Passwords:

  10  SeeAll;
  <<...>>


Items:

  Manufacturer#,  x6 ;
  Distributor#,   x4 ;
  Product#,       x10;
  assembly,       x10;
  component,      x10;
  name,           x40;
  address1,       x40;
  address2,       x40;
  city,           x30;
  department,     x30;  <<Or "State" or "Province">>
  country,        x30;
  amount,         r4 ;
  production,     j2 ;
  supervision,    j2 ;
  responsibility, j2 ;
  <<...>>
```

```
Sets:

Name:  MANUFACTURER, manual;

Entry:
  Manufacturer# (2),  <<2 paths, to "Manufactures"
                        and "Represents">>
  name,
  address1,
  address2,
  city,
  department,
  country;
  <<...>>
 Capacity: 2000;

Name:  PRODUCT, manual;

Entry:
  Product# (4),  <<4 paths, to "Manufactures", to "Sells",
                   to "Assembly" as a part of another
                   product and to "Assembly" as a product
                   which, itself, has components>>
  name;
  <<...>>
 Capacity: 80000;

Name:  DISTRIBUTOR, manual;

Entry:
  Distributor# (2),  <<2 paths, to "Represents" and "Sells">>
  name,
  address1,
  address2,
  city,
  department,
  country;
  <<...>>
 Capacity: 20000;

Name:  ASSEMBLY, detail;  <<Relates products which are, in turn,
                            parts of other products>>

Entry:
  assembly  (PRODUCT),  <<This search field allows us to find all
                          the products which we need to assemble
                          a given product>>
  component (PRODUCT),  <<This search field allows us to find all
                          the products which we can assemble with
                          a given product>>
  amount,
  Production,           <<Person in charge, for instance>>
  Supervision,          <<Person in charge, for instance>>
  Responsibility;       <<Person in charge, for instance>>
  <<...>>
 Capacity: 150000;

Name:  MANUFACTURES, detail;  <<Relates manufacturers
                                to products>> Entry:
  Manufacturer# (MANUFACTURER),
  Product#      (PRODUCT);
  <<...>>
 Capacity: 2000000;

Name:  REPRESENTS, detail;  <<Relates manufacturers
                              to distributors>> Entry:
  Manufacturer# (MANUFACTURER),
  Distributor#  (DISTRIBUTOR);
  <<...>>
 Capacity: 5000;

Name:  SELLS, detail;  <<Relates distributors to products>>

Entry:
  Distributor#  (DISTRIBUTOR),
  Product#      (PRODUCT);
  <<...>>
 Capacity: 5000;
 End.
```

Appendix B

IMAGE schema for the *Presidential* database mentioned in the *Practical Methodology* section.

```
Begin database CHIEF;

<<

NOTES:

 Your imagination and convenience should decide how many (and
 which) attributes to include at the "...".

 The paths are NOT necessary at all, but we include them as
 examples of performance boosters for relationships which
 seem to be "hot and heavy".  You can always take ALL paths
 away, or add OTHER paths at will.  IMAGE does not care: You
 are free to play with the tradeoffs involving time, space
 and the bias on efficiency.

 Capacities can go from one entry to several million entries.
 The help of intelligently-defined paths becomes more obvious
 when you deal with millions of entries.  Toy-like academic
 examples, of course, do not require any overhead in terms of
 structure.

>>

Passwords:

  35  GuessWho;
  <<...>>


Items:

  Country#,       i  ;
  President#,     i2 ;
  Name,           x50;
  Year,           i2 ;
  Year-In,        i2 ;
  Year-Out,       i2 ;
  party,          x20;
  <<...>>


Sets:

Name:  COUNTRY, manual;

Entry:
  Country# (1),  <<path to "Administration">>
  name;
  <<...>>
 Capacity: 300;

Name:  YEAR, manual;

Entry:
  Year (2);  <<2 paths, to "Administration">>
  <<...>>
 Capacity: 4000;  <<You might want to go back to the Etruscans!>>

Name:  PRESIDENT, manual;

Entry:
  President# (1),  <<path to "Administration">>
  name;
  <<...>>
 Capacity: 20000;

Name:  PARTY, manual;

Entry:
  Party (1);  <<path to "Administration">>
  <<...>>
 Capacity: 1000;

Name:  ADMINISTRATION, detail;  <<Relates countries to presidents
                                  to In and Out years>>

Entry:
  Year-In  (YEAR),  <<This search field allows us to find all
                      the administrations, anywhere in the world,
                      which began in a given year>>

  Year-Out (YEAR),  <<This search field allows us to find all the
                      administrations, anywhere in the world,
                      which ended in a given year.  We can treat
                      Year-out=0 as a current administration>>

  Country# (COUNTRY),

  President# (PRESIDENT),
  Party (Party); <<Who knows?  Presidents might have different
                   parties from one administration to the other.
                   Party is NOT an attribute of PRESIDENTS,
                   as it might appear.  Party IS an attribute of
                   ADMINISTRATIONS!>>

  <<...>>
 Capacity: 150000;

 End.
```

THE 2001 SIREN'S SONG: AI IN THE COMMERCIAL MARKETPLACE

Sharon Bishop
Information Technology Group
Hewlett-Packard
11000 Wolfe Road, Cupertino, Ca., 95014

Summary

This paper gives a brief overview of Artificial Intelligence (AI) and the emerging market parameters that are motivating the use of AI, how AI differs from traditonal programming, examples of tools we take for granted today that are the result of the "fallout" from AI research over the past 20 years, expert systems and HP's activity in AI and finally, some trends and future predictions we might anticipate for this technology.

Introduction

Is Artificial Intelligence the modern-day sea nymph -- part bird and part woman -- of Roman mythology who lured sailors to their death on rocky cliffs by seductive singing? Are we the modern-day sailors snared by the eager contemplation of songs foretelling the day we will have machines that eliminate the need for highly paid human experts, computers that speak and understand speech, robots and vision systems that have the ability to recognize patterns of input stimuli and act accordingly, learning systems that learn from usage patterns and modify themselves consequently, self-programming systems, etc.?

Artificial intelligence, popularly defined as the science of enabling computers to reason, make judgments and even learn, has generated considerable interest in the computer industry over the past year. Optimistic analysts forecast a market growth in excess of $5 billion by 1990. Stories about fifth generation computers and the Japanese effort to build "thinking" computers by 1992 have added urgency to competitive efforts in the U. S. and Europe.

The evolution of AI can be viewed as a two-part process. The first stage of this evolution is the 20-year old ongoing research effort aimed at developing computer systems that can emulate human thought and decision- making processes. The second stage is the newly emerging commercial market for AI-based products.

Hewlett-Packard has, for some time, been developing AI technology at our Corporate Research and Development laboratories. Our first entry into the AI market was presented at the International Joint Conference on Artificial Intelligence, held in Los Angeles during August 1985. Initially, we are addressing the productivity issue, that is, the motivation for power tools for programmers. Other examples of internal development activity using our AI technology include expert systems for natural language understanding, an organic compound analyzer, an I/C photolithography advisor, computer peripheral diagnosticians, and an electrocardiograph advisor. HP views AI not as a revolution, but as part of an evolution toward more powerful and sophisticated computer systems, and we have targeted AI technology to play an important role in our future.

Definition of Artificial Intelligence

Artificial intelligence is a buzzword for a group of technologies It is a multidisciplinary study, utilizing and synthesizing knowledge from several fields including cognitive science, philosophy, sociology, computer science and electrical engineering. It is not in itself a commercial field but a science and a technology. As an academic discipline, it is a collection of concepts and ideas which are appropriate for research, but which cannot be marketed. However, artificial intelligence is the scientific foundation for several growing commercial technologies. AI techniqueь make a variety of AI applications possible, including expert systems, computer-aided instruction, automated programming, natural language understanding, vision systems, voice recognition systems and robotics. The unifying factor in all of these diverse applications is the encapsulation of some knowledge or expertise related to humans.

Goals of Artificial Intelligence

The primary goal of Artificial Intelligence is to make machines smarter. The goal of the basic researcher is to understand what intelligence is, and the goal of the entrepreneur is to make machines more useful. To date, we have used computers to evaluate problems we have already solved, for example, computers give us solutions to models and simulations or they solve algorithms for us. The goal with AI is to move toward having computers participate in the analysis and synthesis of problem-solving -- to help us develop the solutions, rather than simply calculating the results. In it's short history, AI as an academic discipline has developed techniques that support computer systems that do seem to exhibit intelligence. For example, researchers have developed systems that comprehend natural language, systems that can outperform human experts, systems that see and robots that manipulate tools. The caveat of course, is that the current systems are limited and work successfully only in a narrow and specified domain. They cannot respond to unanticipated events in the environment and learn efficiently from experience. Still, even though there is criticism of the current systems and their limitations, they have demonstrated significant capabilities that have captured the imagination and attention of commercial and government institutions. Rapid developments in more powerful computer systems at lower costs have increased the number of applications that are now feasible and economical. Today's challenge for commercial users is to identify practical, worthwhile applications and to develop and market these applications.

AI Systems vs. Traditional Systems

Artificial intelligence development is usually characterized by the use of tools and techniques which facilitate the handling of variable, ill-defined and unspecified programming parameters. While the presence or absence of such tools and techniques does not necessarily make a product "genuine AI", it is clear that AI places new demands on traditional tools and languages and the programmers who use them.

Generally the processing power and memory demands on AI development has led to a dedicated workstation approach. The terms "symbolic processing", "object-oriented programming," "knowledge-representation" and others refer to the process of representing information in a fluid rather than highly structured form. These techniques are often required because in contrast to traditional programming objectives, data must be represented symbolically and related through parameters and conditions in a program similar to the human decision-making process. Languages like LISP, PROLOG and SMALLTALK which are highly interpretive and are function or object-oriented, as opposed to procedurally oriented, lend themselves to incremental programming and rapid prototyping. In contrast, conventional programming techniques are ill-suited to handling uncertain or changing specifications. Virtually all modern programming methodology, such as structured design is targeted to ensure that the implementation follows a fixed specification in a controlled fashion, rather than wandering off in an unpredictable direction. In a well-executed conventional implementation project, a great deal of internal rigidity is built into the system, ensuring its orderly development.

AI applications in general incorporate a knowledge base and separate units of code that operate on the knowledge base, the combination of the two being used to solve a problem. Knowledge used by the system is represented explicitly within the system. In constrast, traditional computer programs state procedures explicity, and knowledge about the domain is implicit in the procedures. This concept of the difference between traditional computer systems and AI systems forms a useful basis to distinguish AI programs from traditional programs. The following table highlights some of the differe~ces between traditional systems and AI systems:

Traditional Computer Systems vs. AI Computer Systems

TRADITIONAL SYSTEMS	AI SYSTEMS
Host/timesharing approach	Workstation/PC approach
Separate programming tools, usually not well integrated	Integrated environments, sophisticated programming tools
Minimal prototyping; systems are hard to modify as the environment changes	Rapid prototyping; systems are easy to update, treat knowledge as data
Tend to degrade with age	Can improve with experience increases; knowledge explicit in system and easily updated
Use compiled-only languages, follow complete predefined step-by-step procedures, algorithms	Use function or object-oriented languages, fluid design, follow a line of reasoning
Process numbers	Process symbols, as well as numbers
Use a "black box" approach	Can explain line of reasoning

The AI Productivity 'Fallout'

It is interesting to note that virtually all the environments we use today in traditional computing environments, whether tightly coupled to proprietary operating systems, or highly portable accross vendors, take for granted many capabilities that are essentially the "fallout" from AI developoment that took place in the AI research laboratories through the 1960's and 1970's. For example, in the 1960's, the first computer time-sharing systems were developed in AI laboratories to addresss a faster means of debugging very complex programs. Word processing was developed in the AI lab in the 1960's and expert systems started in 1965. Bit map displays, the now-popular mouse controls and the whole structure of display windows were also originated during the 1960's as well as object-based programming (as in SMALLTALK) and a number of other productivity tools.

Commercial use of these tools originally was discouraged by the fact that they require large computing and storage capabilities, but in the 1970's improved computer performance at lower cost made it practical to start using these tools in conventional programming environments. For example, there has been a dramatic improvement in recent workstation and interface technology, with great cost/performance benefits for group work. Powerful networked workstations/personal computers with high-resolution graphics are becoming widespread. Today machines offered by vendors such as Hewlett-Packard, IBM, DEC, Xerox, Sun, Texas Instruments and Symbolics support tools and languages such as LISP and PROLOG at modest cost. These machines provide adequate hosts for radically improved programming environments.

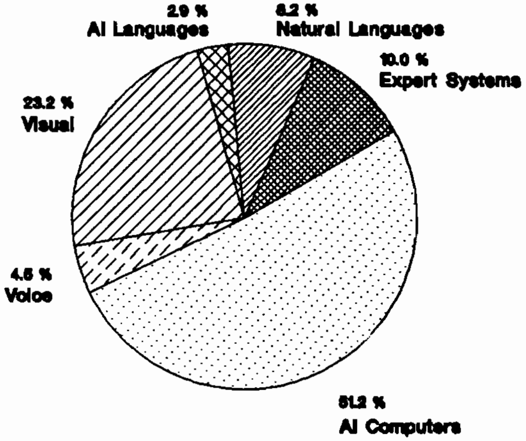
Market Characterization of AI

When looking at the market for Artificial Intelligence, it is important to remember that this market is very young and did not exist in the minds of commercial users even five years ago. There was no market for expert systems or symbolic computing outside the confines of the academic research and development environs. For example, Symbolics, Inc., the leading supplier of LISP machines today, is generally credited for creating a market that nobody knew they wanted; potential users didn't even know what a LISP machine was. Symbolics created a burgeoning industry out of something that researchers did not realize existed, but found to be a very useful tool once they were educated as to its capabilities.

Another point to keep in mind when anticipating the AI market is that you have to view it in terms of commercially viable products that are for sale and can be used by the end-user. There is continuing basic research going on at universities attempting to understand the nature of intelligence and how to replicate the human brain that isn't transferable today into commercial products.

Market segmentation and sizing are two of the hottest topics in AI today. For puposes of this paper, the market will be segmented into the following areas: voice and vision recognition systems, expert systems, natural language understanding and AI computers and languages. Each of these segments is a viable market in and of itself.

THE ARTIFICIAL INTELLIGENCE MARKET
PRODUCT AREAS - 1985



SOURCE: DM Data, Inc.

* AI COMPUTERS

This is the largest segment of the industry because it includes machines that are LISP-oriented (such as those produced by Symbolics, Inc.) and general purpose computers that can also be used for AI applications (such as Hewlett-Packard, Digital Equipment and IBM).

* AI LANGUAGES

AI languages is really a subset of AI computers. This segment includes LISP and PROLOG programming languages and environments for AI applications. LISP is the most popular programming language in the United States. One of the principal features of LISP is that it allows the user to compute with symbolic expressions rather than numbers; that is, bit patterns in a computer's memory and registers can stand for arbitrary symbols, not just those of arithmetic. LISP itself is the programming language often used to develop expert system building tools. PROLOG is the language of choice for AI applications in Europe and Japan. It is based on the idea of representing a program as a set of declarative statements in a form of logic.

* EXPERT SYSTEMS

Expert systems are currently the most visible of the AI technologies because they are the most easily understood in terms of what they actually do: an expert's knowledge in a particular domain is programmed into an expert system and that system emulates his/her behavior. The expert system is composed of two parts: (1) a knowledge base which is the actual information given by the expert and programmed into the system (for example, a doctor who specializes in bone grafts); and (2) an inference engine which is the actual computer program that runs that information, making the correlations and associations between the information that has been programmed in. Expert systems is forecasted to be the largest growing segment of the AI market. More people are aware of its potential capabilities and will plan to utiltize expert systems in order to achieve competitive advantages and profit.

* NATURAL LANGUAGE

Natural language does not encompass voice recognition or actual speaking into a microphone. It is simply communication with the computer or terminal in written conversational style. Natural language systems have to deal with many ambiquities, such as regional slang, and the correct interpretation of a number of variables in sentences. Therefore, it is forecasted that this segment will not grow as rapidly as expert systems. It is expected that natural language will be included in with other packages, such as expert systems and AI hardware. Since natural language still requires the end-user to type, some market analysts forecast the value of this type of interface will be much less than voice recognition.

* VOICE RECOGNITION

Voice recognition tehnology in the context of AI is not to be confused with voice store and forward, which is the recording of a human voice and then replaying it later; or synthesized speech, which is a computer emulating the sound of a voice. It is actually the ability of a computer to respond to vocal commands exactly like it responds to keyboard commands. Utilizing a microphone instead of a keyboard, users would talk to the computer and it would react accordingly. Again, as with natural language, the system has to deal with many ambiguities: speech patterns, pitch, tone and the variability of each word. Most voice-recognition systems today have 1,000 to 2,000 words of memory and are speaker-dependent. There are companies promising to produce voice-recognition typewriters in the near future with memory lists of 10,000 to 15,000 words, however, it is felt that this technology is at least five years from being a truly viable market.

* ARTIFICIAL VISION

In the U.S., artificial vision is mainly associated with the auto industry and robotics. General Motors has holdings in six of the major artificial vision companies. Ford has holding in two others, and Chrysler has holdings it has not disclosed. Quality inspection and areas where it's too difficult for humans to work or the failure differentials are too small for a human to react to or to be able to perceive are a few of the possibilities for this technology. The major research push today is for 3-D vision systems that have acceptable real-time capabilities for commercial exploitation. The market segment is expected to expand and grow rapidly.

THE ARTIFICIAL INTELLIGENCE MARKET
(Millions of Dollars)

MARKET AREA	1985	1986	1987	1988	1989	1990
Expert Systems	74	145	245	385	570	810
Natural Language	59	125	210	320	465	650
Visual Recognition	168	260	370	500	660	840
Voice Recognition	33	65	85	140	200	270
AI Languages	21	35	45	65	80	105
AI Computers	364	510	710	970	1250	1570
Government Contracts*	95	150	150	155	175	200
TOTAL	719	1130	1665	2380	3225	4245

* Not in total; already included in other areas.

SOURCE: DM Data, Inc.

User Needs Characterization

In-house corporate and university research development projects are the largest users of artificial intelligence today. The U.S. government, primarily the military, is the next-largest user.

It is important that we look at the underlying need that is driving the purchase of computing systems that support symbolic computing, that is, that area of computer technology where the aim is to optimize the processing of symbols, representing objects or concepts rather than numerical data or text.

* There is an overwhelming need for a major breakthrough in programmer productivity. Symbolics, Inc. has stated that their customers, with a Symbolics system, claim productivity increases in software development by a factor of between two and 50. Even taking the low end of that range, if a customer can solve a problem in one year instead of two, they can gain a major competitive advantage.

* Symbolic computing allows a customer to pursue applications that are difficult or impossible to approach using conventional computing techniques. These complex applications require a sophisticated development environment. Customers purchase systems because of speed, but equally important is a high-performance LISP compiler and a superior programming environment.

* Symbolic computing can fill the need for automation of nonclerical tasks. Just as word-processing has had a major impact on productivity for clerical personnel, symbolic computing can have a profound effect

on the productivity of professionals doing such tasks as integrated-circuit design and computer systems maintenance.

The majority of users evaluating AI technology want to develop expert systems. They want tools that are easy to use and lower cost systems with competitive performance. They need access to AI technology through training by vendors or consultants. Systems need to be easy to use with superior user interfaces. Standard networking facilities are needed so they can coexist with multiple vendors. Customers want integration with existing languages such as COBOL, FORTRAN and Pascal to allow use of existing code to extend the usability and simplify the maintenance of their conventional programs.

Expert Systems

Expert systems is the fastest growing segment of the AI market today. An expert system is a program that can reason, make recommendations and explain its reasoning, based on extensive knowledge of a given field such as geology or medicine. It can be thought of as a form of "intellectual cloning". An example of an expert system that is well-known is General Electric's CATS-1. This expert system diagnoses problems in diesel locomotive engines. The knowledge and rules for the system were culled from a 40-year vetern at GE who was an expert in locomotive trouble-shooting. In the commercial market, the term "expert system" is used to encompass most knowledge-based systems. All knowledge-based systems today, whether or not they are truly expert, address a narrow problem domain. It is important to remember that today's expert system can only capture expertise that humans already have. We are still far from the goal of building systems that can reason as people do and learn from experience.

Anatomy of an Expert System

The major components of an expert system are the knowledge base, the inference engine, and the user interface. Other components may be optional such as external sources of information residing in external databases and external sensors. The knowledge base consists of facts and rules about a specific domain. The inference engine communicates with the knowledge base to determine what rules may apply to the current situation and it also keeps track of the status of the problem. The inference engine may also have optional access to external sources of information. In some cases when the data in the knowledge base is not definite, a "certainty factor" may be assigned. When certainty factors are used, the inference engine includes procedures for calculating and reporting the level of certainty for answers to problems. Some expert systems may support an interactive dialog with the user who may ask why a certain conclusion was reached or why a particular question is being asked. Some user interfaces for expert systems may include a natural language processor and a graphics editor.

Hewlett-Packard has an internal tool that is a flexible expert systems construction environment that provides advanced reasoning and representation tools for development of applications. It has served as the basis for the expert systems development activity at Hewlett-Packard.

Application Areas for Expert Systems

Generic categories of applications suitable for expert systems development include interpreting and monitoring sensory input, diagnosing equipment, prescribing and implementing remedies, planning and design and prediction systems. Possible application areas cover a multitude of industries such as finance, law, medicine, military, education, manufacturing and service, agriculture and mineral exploration. Many of the actual expert systems that have been built and are well-known were built at Stanford for medical applications.

While many application areas may be candidates for expert systems development, this should only be chosen over traditional programming techniques when the domain knowledge is not firm. When the knowledge and problem is firm, fixed and formalized, algorithmic programs are more appropriate than heuristic ones and should be used. If the knowledge is subjective, ill-defined and partly judgmental, expert systems and hueristic programming are appropriate.

DIAGNOSTIC AND MAINTENANCE: In today's environment, a list of expert systems would most surely include a multitude of diagnostic systems for equipment. Hewlett-Packard has expert sytems in use today in their response centers to diagnose and trouble-shoot data comm lines and disc drives. Other examples include DEC's SPEAR system for diagnosing problems in their tape drives and PRIME's DOC expert system for assisting with troubleshooting on their 750 minicomputer. IBM has DART for disk fault diagnosis and AT&T Bell Laboratories has ACE to manage telephone cable maintenance.

U.S. MILITARY: The U. S. military has targeted a considerable amount of funding for expert sytems projects and has declared AI-based applications critical to upcoming generations of military hardware and software. Examples of expert systems that have potential include a pilot's associate, battlefield advisor and roving vehicles for use in hazardous conditions.

FINANCE: This application area holds great promise for the use of expert systems. The management of money is a real, measurable goal and those companies that do it best are going to have a competitive advantage. Areas proprosed for use of expert systems are cash management, portfolio management, underwriting, loan management and corporate finance. Financial expert systems would be able to handle huge amounts of rapidly changing data and apply the rules of finance to assist in decision making. For example, when to go public, covering of foreign exchange exposure and so forth. Cognitive Systems, Inc., TAD expert system assists in tax preparation.

MEDICAL: The medical field has also been a natural application for AI because much of a doctor's skill involves applying rules to evidence presented by a patient. HP is marketing an ECG electrocardiograph expert system that aids physicians in the diagnosis of heart disease. The product supports diagnosis justification, modifiable rules, stand-alone use and integration with other systems. MYCIN and ONCOSYN and PUFF are expert systems from Stanford University. MYCIN diagnoses meningitis and other blood-borne bacterial infections and recommends theraphy. ONCOSYN recommends drug treatment for cancer patients. PUFF is used for pulmonary function test interpretation.

MANUFACTURING: This is fertile territory for expert systems to be the brain for robots and to assist managers and engineers in decision-making with massive amounts of data. Hewlett-Packard has a Manufacturing Research Group in our applied research laboratories at Corporate headquarters that is specifically addressing advanced technologies for the manufacturing arena. We are very interested in robotics and machine vision.

SCIENCE & ELECTRONICS: Activity in many of the major electronics firms such as Hewlett-Packard, IBM, Xerox, Texas Instruments, DEC and others bear out predictions that increased importance is being placed on AI technology for in-house use. HP has developed an IC photolithography advisor that diagnosis fabrication defects and recommends correction action. The system allows for single or multiple faults, and gives conclusions and justification and can display video images to help identify faults. HP also has an IC design tool in the form of a procedural langage embedded in LISP for designing integrated circuits from library parts. VLSI is a ripe area for expert systems since we are rapidly approaching the density of one million transistors on a single chip. To design this manually is like designing a city block by block - the complexity goes up exponentially and the opportunities for error are great. Using AI techniques allows the design engineer to take a much more modular approach to design - much like writing a program with the added flexibility for change and prototyping providing definite productivity gains. Other examples of expert systems in use by electronics firms include DEC with their R1 (alias for XCON) expert system to help configure computers and their XSITE expert system to recommend computer site preparation. Xerox has KBVLSI which aids in VLSI design, Daisy Systems markets GATEMASTER and LOGICIAN for performing gate array layouts.

OTHER: Of course there are many other areas that have potential for use of expert systems. In HP, we have internal research applications underway for a number of internal tools, including natural language understanding, expert systems for the office and expert systems to provide more powerful and sophisticated programming environments.

Future Trends

The AI market milestones from now through the 1990's seem relatively clear. The industry and university collaboration that has already lead to development of several expert systems and the emergence of hardware tools and development environments will continue. More sophisticated software and hardware tools will be introduced in the near term and hardware prices will continue to edge down. At the moment, no AI tool supplier can claim to be the leader, but as further refinement takes place in the market and tools, companies will start to turn from "research- driven" to "market-driven" to get their AI products into the marketplace. By 1987 and 1988 it is felt that "off the shelf" expert systems will start emerging for the consumer. AI will become a normal part of the software market. By 1990 AI capabilities will be incorporated into most products.

However, while AI research continues at a brisk pace, full commercialization faces several potential hurdles, among which are (1) limited industry expertise in building knowledge-based systems, (2) the need for development tools and standards that are easy to use and economically feasible and (3) the need for characterizing the still ill-defined market and addressing critical market issues such as niche definition, pricing, channels of distribution and customer training and support. The period of market definition is likely to continue for the next two or three years. Software metrics also need to be gathered to give a more solid indication of productivity gains, since this technology is likely to involve a lengthy learning curve.

Summary

The world of the mainstream computer supplier is changing and the arbiter of that change is symbolic computing. Symbolic computing is the area of computer technology where the aim is to optimize the processing of symbols, representing objects or concepts rather than numerical data or text. In the past, changes in the 8 and 16-bit microprocessors had profound effects in the market, creating new industries and new competitors. Now the emergence of symbolic computing will have even more impact; not only will new markets be created, but existing markets will be severely impacted over the next few years.

In 1985 some major players in AI -- including IBM, Xerox, Hewlett-Packard, and Texas Instruments -- introduced new AI-related products and reaffirmed their committment to AI research and development. The products included AI programming "shells" to build expert systems, programming languages and tools and high-performance, lower-cost workstations that support AI products. It is anticipated that in 1986, these products will become even more widely available, marking a milestone in the evolution of AI from the research and development laboratories to the commercial marketplace. As the market becomes more clearly focused, all major computer companies, mainframe, minicomputer and microcomputer, will enter into direct competition and will vie to offer systems of comparable functionality. This will elevate the competition to a new level.

For AI suppliers, success in the AI market will not necessarily be determined by previous success or failure in the 16 or 32-bit world. More likley, success will depend on the manufacturer's ability to produce on schedule these order of magnitude more complex systems; the ability to adequately supplement the systems with support for AI programming languages (LISP and PROLOG); high performance mass-storage devices and high-resolution graphics displays; the implementation of a large virtual address space and large amounts of physical memory; and the garnering of third-party software support for languages, development tools and applications.

For the major computer manufacturers such as HP, IBM and DEC, it has become a question of not whether they should develop an AI-based product line, but rather which system processor they should choose and when. As with any technological upheaval, the emergence of symbolic computing on high-performance workstations and PC's give new players the opportunity to get the jump on the existing competitors. However, difficult decisions must be made on the basis of technical, marketing and support issues. Choosing the right market niche and third party suppliers will be tantamount to choosing the right horse. Also, choosing a 32-bit processor family is not enough, as many system manufacturers will choose the same processor family. The issues of system design, price/perforamnce ratios, marketing strategies and software support will make the critical difference between success and failure.

For the potential customers of Artificial Intelligence, it is well to heed the siren's song -- with a clear eye and an alert mind. End users owe it to themselves to become educated about this technology and informed of the activity that is taking place. Many vendors are becoming very agressive in marketing tools and an escalation of this activity will surely take place over the next couple of years. It is important to examine your business for potential applications, keeping in mind that commercially viable products on the market today are far from fulfilling the promise of AI. Researchers are still trying to understand and capture the nature of human intelligence -- we do not have machines today that can reason as humans do and learn from experience.

Biography

Sharon Bishop is a product manager in Hewlett-Packard's Information Technology Group in Cupertino, California, with responsiblity for several products, among which are AI lanaguges and programming environments. During her eight years with Hewlett-Packard, she has held senior programming positions, managed a group of application software engineers and, most recently, moved into product marketing for computer languages.

BUSINESS BASIC PHASE II,
THE NEXT STEP TO A FRIENDLY COMMERCIAL PROGRAMMING LANGUAGE

Dr.Ulrich E. Fauser
WEIGANG MCS
Stöberlstr. 68
D-8000 München 21, W-GERMANY

Summary

Whereas the already powerful phase I released with T-Delta 2 and presented at the Washington conference was primarily aimed towards former BASIC/3000 users, the new phase II to be released mid-86 adds even more advanced statements so that a real flexible and rich language for commercial programming will be available.

This and the nevertheless comparable performance makes HPBB a real competitor to the other prevailing commercial languages COBOL and PASCAL.

This paper will give an overview of the new features from an OEM's view who has been also a beta-testsite from the beginning of the HPBB project.

Phase II will include features like:

softkey integration for userfriendly programflow control

cursor and screen I/O

a full fledged report writer facility in a 4th generation style

database search and sort with multiple threads

other database extensions

a new forms handling package working in character mode allowing fieldwise program interaction

All this and the existing conversion tools make HPBB not only a language with a high programmer productivity but also the perfect migration tool for BASIC/260 users as it is approx. 80 - 90% compatible to the 260 language after which of course HPBB was modeled.

Another feature making HPBB attractive is HP's commitment in making it available not only with the high end spectrum series but also including the low end 260 in its range of commercial computers through the JOIN/3000 program.

In this way an OEM will have a reasonable portable language throughout the range of the HP computer family.

Introduction

This paper will be presented in two different parts. The first will deal with the purely technical matters trying to give you an overview of the newly added statements and power to the phase II of HPBB like this language is officially named.

The second will bring you some of the experiences and problems that we as a major 250/260 OEM have met first as a beta testsite and then in converting our different manufacturing and commercial application packages to the 3000 environment, a still continuing effort.

Some History

The HP3000 Business Basic project was originally started in 1981 but it took a lot of pressure from some HP260 OEMs like us to really make it going finally in 83.

HPBB had several goals, first as the 250/260 OEMs and users slowly grew beyond the capabilities of their machine the demand for a growth path into a new hardware family was obvious but there was no suitable replacement with a similar friendliness and productivity within HPs range of commercial computers. Second BASIC/3000 users were not really pleased with their language so something had to be done to make it attractive again. And third many people were looking for an easy and friendly commercial language providing not only simple access to most of the necessary subsystems like IMAGE, VPLUS etc. but also increasing programmer's productivity by doing a lot of the housekeeping routines usally left to him, therefore leaving more time for better solutions.

Although most of HPBBs features were inherited from BASIC/260 one of the languages with the highest rating in user satisfaction most of BASIC/3000s features are incorporated too.

It was decided to release HPBB in two phases, phase I was primarily aimed towards former BASIC/3000 users or OEMs who wanted to start rewriting their applications with the still restricted language subset. This phase was released last year with T-Delta 2. Phase II which is scheduled to be released in mid 86 now provides a real upgrade path for former 260 users, especially after phase II was enhanced by a product made in HP Böblingen, called JOINFORM, giving the same screen handling facilities like on the 260. Thus the compatibility to BASIC/260 by not having to use VPLUS was drastically improved.

A short review of phase I

For those of you who haven't been in Washington and haven't read the proceedings here is a quick summary of the features of phase I.

Interpreter and compiler available

Full debugging and trace capability

Builtin editor with immediate syntax check and many edit functions and program development environment

Extensive ON LINE HELP

Always actice calculator and immediate command mode

All standard control structures
  GOTO, GOSUB, CALL, ON GOTO, ON GOSUB, function calls
  FOR-NEXT, IF-THEN-ELSE, WHILE-ENDWHILE, REPEAT-UNTIL,
  SELECT-CASE, LOOP-EXIT IF-ENDLOOP

Real subprograms and functions with parameters and local variables

Named and unnamed COMMON

Various OPTIONs to control interpreter and compiler

Variable and label names up to 63 characters

Seven datatypes INTEGER, REAL, DECIMAL (each SHORT and FULL precision)
  and Strings

Dynamic checks for ON ERROR, ON DBERROR, ON HALT, ON END

Statements for accessing all IMAGE intrinsics with easy handling

Rich output formatting features

Access to MPE commands (:SYSTEM) and control of son processes
  (:SYSTEMRUN)

Interface to routines written in SPL and PASCAL

More than necessary functions for stringhandling, numeric,
  mathematical, matrix, conversion, bithandling, logical, files

All possibilities to access the file system
  also special BDATA type file including type information with
  data
  KSAM only via intrinsics

and many more features of a modern language.

Now, what is new in phase II ?

Softkeys and typing aid keys

HPBB supports the 8 softkeys provided with most HP terminals in two ways: as typing aid keys and programatic keys. Typing aid keys are used for often needed charactersequences especially by programmers. GET KEY and SAVE key are used to save and store these. Definition and changes are made via standard terminal features.

More important are programatic keys or more often just called softkeys. These are used to allow the user control the flow of control for his program. In a program a softkey is defined using the ON KEY statement with a number (1 to 8), a priority, a keylabel and a softkeyaction which can be thought of as an interrupt routine. Only on the 3000 softkey actions are no longer real interrupts like on the 260 where an action could be invoked at the end of every statement or within INPUT statements and they always returned to the point of invocation, except with GOTO.

On the 3000 there are no real softkey interrupts, they are handled like special character sequences and only work when the program is in an input state. When the user presses a softkey during an input the statement is terminated and the corresponding softkey action is executed. Unlike on the 260 the flow of control here goes to the next statement after the input (not with GOTO) and it is left to the programmer what to do. To do this HPBB provides a builtin function RESPONSE which tells you how an input was terminated (softkey, HALT, timeout, data etc). This is not as it used to be on the 260 but it works and the additional handling around the input statements is automatically inserted by the conversion program.

The following statements are available: (examples)

ON KEY 7 CALL Selection;LABEL="SELECT DATA"
action can be CALL, GOSUB or GOTO

OFF KEY or OFF KEY 6 deactivates all or specified keys

ENABLE, DISABLE allow or disallow keyinterrupts to occur

PRESS KEY n makes HPBB think a softkey was pressed by user

CURKEY function returns the number of the last key pressed

Waiting for softkey, no data input allowed

```
  LOOP
    ACCEPT
  END LOOP
```

Waiting for data or softkey, exit also when HALT is pressed

```
  LOOP
    INPUT Var
    EXIT IF RESPONSE>0
  END LOOP
```

Cursor and screen I/O

The CURSOR statement allows you to place the cursor anywhere on the screen and define any of the available video enhancements

E.g. CURSOR (12,5) positions the cursor in row 12, column 5

CURSOR (15,1),("BI",-3) positions the cursor to 15,1 and displays a blinking, inverse field 3 spaces long

The builtin functions CPOS and RPOS return the current column or row position of the cursor.

The ENTER and LENTER statements work just like INPUT and LINPUT except that the input is taken from the screen. LINPUT reads a whole line including commas whereas for ENTER commas mean data item separators.

HPFORMS (VPLUS) interface

For interfacing the most common VPLUS intrinsics HPBB provides statements releaving the programmer from all the usual hassles of providing the right parameters or defining his common area. For special problems however VPLUS intrinsics still have to be called directly.

Statements:

OPEN FORM with HOME/OVERLAY/APPEND/FREEZE options

CLEAR FORM with REMAIN option

READ FORM <variablelist> with TIMEOUT and SKIP option
   including imbedded FOR loops

CLEAR FORM with DEFAULT option

WRITE FORM <expr.list> with CURSOR option
   including imbedded FOR loops

JOINFORM Forms/260 interface

To provide a higher compatibility for 260 users and also as an alternative to VPLUS HP Böblingen (BGD) decided to launch an additional project called JOINFORM for HP260 FORMS conversion and emulation in HP Business Basic. This team has implemented a set of library subroutines performing the same functions as FORMS/260. Calling these routines is integrated into HPBB i.e. several statements (like on the 260) check if a 260 form is active and they then behave differently.

All 260 forms can be converted into JOINFORM formfiles which are different from VPLUS files (filecode BFORM). There will also be a new friendly FORMS editor for defining, modifying and printing these forms.

JOINFORM does not work in blockmode, but in formatmode.

The forms handling statements are the same as for VPLUS, they decide from the filecode which subsystem to call.

OPEN FORM

CLOSE FORM and CLOSE FORM REMAIN

CLEAR FORM

JOINFORM allows forms to have an inputfieldorder and an outputfieldorder independent from each other for the two fieldtypes provided. Fields can be arranged in any order on the screen.There is no more extra tabfieldorder like on the 260. This is now equal to the inputfieldorder like it was done mostly on the

260 anyhow. Internally still three pointers are maintained, an input- output- and cursorfieldpointer.

The CURSOR statement was enhanced to position the cursor to these fields by adding the builtinfunctions IFLD, OFLD and CFLD

Thus CURSOR IFLD(5) sets input and cursorpointer to field #5

CURSOR OFLD(4) sets the outputpointer

CURSOR CFLD(3) sets the cursorfieldpointer

An additional builtin TFLD returns the fieldnumber of the last input field where return was pressed, this is the same as TFNUM on the 260.

All I/O interaction between program and form is done with normal I/O statements which internally behave differently when a form is active by calling the appropriate subroutines.

DISP and PRINT output to the actual outputfield thereby increasing the internal pointer.

INPUT and ACCEPT <variablelist> read from the current cursorfield for the benefit of the user it is only read what is really on the screen, so insert,delete and cursorkeys can be used, as the inputbuffer is simply discarded

ENTER <variablelist> reads starting from the current inputfield from the screen

Unlike VPLUS additional "normal" I/O can take place outside the form boundaries using LDISP and LINPUT/LENTER. The user doesn't have to distinguish between enter and return, all input is ended with return. This may cause some confusion if he uses his application together with VPLUS oriented utilities or other programs.

Database extensions

Using a socalled PACKFMT or an IN DATA SET statement allows DBGET, DBPUT and DBUPDATE to implicitly do packing and unpacking of the IMAGE buffer into/from program variables without extra PACK/UNPACK statements or program buffers.

Example:

```
Label: PACKFMT Partno$,Customer,Quantity
or
Label: IN DATASET "ORDERS" USE Partno$,Customer,quantity
```

DBGET Dbase$ USING Label;DATASET="ORDERS" will then immediately read databasevalues into program variables

The PREDICATE statement allows simple specifications of predicate for DBLOCK modes 5 and 6 without having to deal with wordcounts, entrylength and all that internal stuff.

```
PREDICATE P$ FROM "ORDERS" WITH Customer=1254;Quantity>100
```

Database SEARCH and SORT

Besides the report writer this is certainly the most powerful construct in HPBB. It allows programmers to select and sort IMAGE databases in an almost relational way, not only on single datasets but over multiple sets. HPBB supports the concept of workfile containing only pointers to selected/sorted records that is the database itself is not changed at all. This is available nowhere else on the 3000.

The DBASE IS Dbase$ statement only defines which database is actually to be searched and sorted

WORKFILE IS #1 defines the previously opened file #1 to be a special type for handling IMAGE pointers. Files with the WORKFILE option are treated specially and have some restrictions.

We have already presented the IN DATASET statement which is essential here too.

The THREAD IS statement defines which datasets are involved in a SEARCH or SORT. Up to 10 sets can be specified, no two consecutive sets of the same kind (master,detail) can be connected. If a master is linked to a detail with more than one path the PATH option allows to select which one to use. There is an additional contruct called a synthetical link when there is no real path from a detail to a master in the thread. (LINK Variable option)

The SEARCH statement searches all involved datasets according to the thread and evaluates its conditional expression. There are certain programming techniques keeping the number of records in the first dataset of the thread small because this is the only one to be read sequentially thus greatly influencing the performance. The result is stored as tuples of pointers in the workfile, one pointer for each dataset.

The SORT statement allows sorting of the pointers according to the related data in ascending or descending order with up to ten sortkeys. If a SEARCH preceded the SORT only the already selected records are sorted and vice versa.

Example (see also appendix)

```
Dset1: IN DATASET "ORDERS" USE Partno$,Customer
Dset2: IN DATASET "PARTMASTER" USE Partno$
Dset3: IN DATASET "PARTS" USE Partno$,SKIP 46,Storageloc$

ASSIGN "FILEA" to #1
WORKFILE IS #1

Thread1: THREAD IS Dset1,Dset2,Dset3

SEARCH USING Thread1;Customer=1254 AND Storageloc$="STORE_B"
```

This selects all records for customer 1254 whose parts are stored in STORE_B.

```
SORT USING Thread1;Storageloc$,Partno$
```

Now sort them according to storagelocation and partnumber

Now a LOOP would read the workfile, access the database with DBGET in mode 4 and process the records in whatever way.

Syntax and semantic is somewhat different from what the 260 had, there it was much more flexible, but due to be compilable some restrictions had to be made.

The REPORT WRITER

This is probably the most powerful tool available to the programmer allowing him to easily define and produce reports. The definition of a report is done in a 4th generation style by telling the system how it should look like instead of programming every little single step. The report writer system takes care of all routine work like pageformat and numbering, headers, trailers, totals and averages. It also lets you define when logical breaks are to occur and what happens then.

The actual production of the report is controlled by simple output statements, the DETAIL LINE construct.

Therefore the report writer system has 3 distinct parts

The report definition part

Statements controlling the activation of the report

Functions giving various informations

The report writer definition

The REPORT HEADER defines

page format	PAGE LENGTH
margins	LEFT MARGIN
which GRAND TOTALS are to be taken	GRAND TOTALS
under which conditions a break occurs	BREAK n IF <cond>
	BREAK n WHEN var CHANGES

with up to 9 break levels
and also the appearance of a one time report header

The REPORT TRAILER defines a one time report ending trailer

The PAGE HEADER/TRAILER defines what is to be done when a new page starts or a pagebreak occurs

The HEADER/TRAILER level statements define what has to be done when a break for a certain level occurs, which local TOTALS are kept, under which conditions a line is printed at all, thereby protecting sensitive information.

A REPORT EXIT section defines what is to be done in case of abnormal termination via a STOP REPORT command.

For an example see the appendix

REPORT WRITER executable statements

A report definition alone does nothing, it has to be activated before it can be used by a BEGIN REPORT Rptlabel statement No printing is done yet only internal initialization.

All actual printing is done triggered by the DETAIL LINE statement. It checks whether the report has started at all, which break occurs, printing appropriate headers and trailers, keeping local and grand totals and so on, all with one statement.

For rare cases when this is not enough special flow control can be activated be triggering some events like TRIGGER PAGE BREAK or TRIGGER BREAK level activating the corresponding sections in the report description.

END REPORT is the normal termination of a report, all remaining events are triggered, the report trailer is printed followed by a final page trailer. The report definition is deactivated.

STOP REPORT indicates an abnormal termination. No more normal processing is done, only the report exit section is activated if present usually giving some hints to the cause of the termination.

Report writer functions

They return various information about internal conditions and can be used to control the report even finer if necessary and to access numeric results.

AVG (level,i) average for the i-th expression of the TOTAL statement in level (0 being the GRAND TOTALS)

LASTBREAK level of last break condition

NUMBREAK (level) how often has a break for this level occured

NUMDETAIL (level) how many lines have been printed in level

OLDCV and OLDCV$ return the value of the last controlvariable for the BREAK WHEN statement, so even if you have read already a record which triggered the break the old value is still available

NUMLINE how many lines already on current page

PAGENUM current page number

RWINFO some more seldom used information

TOTAL (level,i) giving the i-th total for level

Missing features

For those of you familiar with BASIC/260 here is a list of features not available in HPBB. Programs using these statements have to be rewritten.

AVAIL
BUFFER#
CATLINE
CATFILE
CHECKREAD ON/OFF
DET without parameter
DIRECT
DIRECT NOUPDATE
DOOR LOCK/UNLOCK
DUPTEST
EDIT
EDIT KEY
HOLE
INDIRECT
LINK
LIST KEY
LOAD / STORE BIN
ON / OFF DELAY
PRINT LABEL
READ LABEL
REQUEST
RELEASE
RES
SD / SI
SET DATE TO
SET TIME TO
SYSID$

All TASK Statements

All PERFORM Statements

All TIO Statements

All MEDIA Statements

<u>In IMAGE</u>

IN DATA SET DIM ALL
IN DATA SET USE ALL   replace through IN DATA SET USE <list>
IN DATA SET IN COM
IN DATA SET FREE
DBCLOSE MODE 4
DBINFO Mode 4xx
DBCREATE
DBERASE
DBMAINT
DBPASS
DBPURGE
DBRESTORE
DBSTORE
READ / WRITE DBPASSWORD
XCOPY

There are no QUERY controlnumbers and no corresponding DBINFO

Performance hints

Like in phase I there is also an interpreter and compiler in phase II but unlike phase I all the new statements of phase II are compilable without exception.

The purpose of the interpreter is mainly for programmers use only or at most for one shot programs as its speed is too slow for everyday production programs. So its advantage is primarily for program development, testing and symbolic debugging in its fullest sense, because in the interpreter all source information is still available. You can not only look at or modify variables but also add, delete or modify the source program.

For enduser applications the compiled code is the only reasonable way to go, maybe with some programs needing noncompilable statements still interpreted. Remember you can call compiled programs from the interpreter but not viceversa.

Some speed ratios HP260 vs HP3000/37

HPBB interpreter 2-5 times slower than BASIC/260

HPBB compiler 5-10 times faster than interpreter

-> on the /37 a compiled program is on the average 2 times faster than the interpreted program on the 260

That shows clearly that the /37 is no upgrade path for users who have outgrown their HP260

Comparison to other languages

So far only measurements with phase I have been made as phase II features are not available in most other languages and they indicate a slightly slower performance compared to PASCAL/COBOL, we talk about 10 to 20% for CPU bound programs. For I/O or IMAGE bound programs there is of course not much difference since most of their time is spent in the system intrinsics anyhow.

With the coming phase II extensions performance should favor HPBB because of the really powerful statements especially if you take into account a faster development cycle and better and more flexible programs.

Please note that all tests are preliminary as especially phase II is not yet performance tuned.

For ways of improving performance even more by switching off some off the features like dynamic error checking etc in the compiler see the appropriate sections in the HPBB manual.

Experiences with the conversion process

Those of you familiar with BASIC/260 can probably tell by now that converting existing 260 application packages still requires some manual work but also that there are no real big obstacles. Just how much effort do prospective OEMs still have to put in before their programs are running in HPBB and in an HP3000 environment.

Let me give you some of the results we have got in converting our first packages.

Before any conversion can be done some very timeconsuming preparations have to be made first, I mean preparing all your files and databases for transfer and the filetransfer itself. There are two ways two do it, a slow and cheap one using a RS232 connection or a faster, expensive one using INPs. We had only the slow connection available at 9600 Baud (HP recommends only 4800, but we didn't have any problems) and with the preparations on the 260 side and the filetransfer you can easily spend some 3-4 days or more, mostly depending on the size of your database which has to be unloaded, transfered and loaded set by set. Conversion of databases is straightforward as the 260 uses almost the same IMAGE subsystem.

The good thing is most preparation and transfer procedures can be automated using PERFORM on the 260 or batchfiles for the transfer, so this can easily run overnight. An example is shown in the appendix, without further comment.

The cheap transfer tool is TRNSFR a 260 utility provided with HPBB, the expensive one is DSN/DS and INP hardware on both sides.

The conversion itself is done by a supplied utility BBCT250 which takes care of all the syntax changes and some but not all of the semantic changes. After this is done the converted program is loaded into HPBB with the GET command and now its your time to do the remaining manual adaptation.

The worst problem we encountered was a program too big for HPBB to accept in one piece. As the 260 has a single 64 KB address space for one user it is possible to write programs up to 60 KB as a single subroutine or main program. Unfortunately this is not acceptable by HPBB, it can only accept so called subunits (main, subroutine or function) up to approx. 20-25 KB or about 400-500 lines of code but of course many of these subunits i.e. the whole program can get much larger but there is that damned restriction on the subunit size.

Well no problem, we said, we just segment the big ones into some smaller subroutines and a control program. Easier said than done, I have to admit because some of them just turned out to be ugly, unstructured monster programs which were much simpler to rewrite than to segment and convert. But we were lucky, these programs were less than 10% of all, nevertheless this subunitsize restriction can be a serious bottleneck for a successful conversion effort.

The small enough programs (90% or more) were relatively easy to convert the conversion program BBCT250 doing most of the work. However some minor manual changes still had to be made. (The following list is not complete)

Create a unique subprogram name, on the 260 all our application subprograms were called Pgm because they were all selectively loaded just before the CALL by the menu.

Change all references to the database statusarray to doubleword references, the 260 only had 16 bit pointers and capacities

Make file and formnames acceptable to the 3000, the 260 allowed upper and lowercase and also special characters

Change DBGET, DBPUT and DBUPDATE and insert the USING clause, the 260 did the variable assignment dynamically, here it is static

When using direct wordpointers for accessing files, check them because due to other storage requirements they may have changed

The biggest manual changes however requiring some knowledge about the programflow are adapting the IN DATASET, THREAD, SEARCH and SORT statements as their logic is somewhat different.

And some minor stuff, some of them could be done using the HPBB editor functions and by redirecting HPBB's inputfile to a commandfile.
All in all it took us about one hour per program for the conversion. I think an acceptable result as it offers a whole new dataprocessing environment keeping OEMs software investments and almost the same functionality in the applications. Thus a user having to move to a bigger machine due to size limitations of the 260 can immediately continue to work in a familiar way without a timeconsuming learning process.

Schedule

for the current phases

Phase I released in mid 85

Phase II and Joinform beta test in February 86
release mid 86

Preview into the future

Further development within HPBB (no HP commitment yet)

After the release of phase II in mid 86 there might be ongoing efforts to enhance further the useability, power and friendliness of HPBB.

Several topics have been discussed including:

File sort statements, thus getting rid of intrinsic calls

data structures, at least simple ones

printing into and reading from strings, like PL/1 GET/PUT STRING

support of fully qualified filenames for distributed systems

full screen editor with immediate syntax check within HPBB

and some more

All these points are still subject to discussion and no commitment has been made by HP so far.

Porting HPBB to SPECTRUM

This project started in Nov 85

According to Dave Elliot (HPBB manager phase I) about 10% of the existing code has to be changed to adapt HPBB to the new hardware but for performance reasons approximately 20% will be modified. HPs plan is to release HPBB on SPECTRUM at the same time as on the 3000, mid 86.

Concerning database access it is planned to provide a softwarelayer between HPBBs DB statements and the new database so that a programmer can continue using his well known IMAGE DB statements without changes.

In doing this he will of course not be able to exploit the full range of the features of the new database, therefore additional DB statements have to be integrated too.

Right now no direct interface is in discussion between HPBB and the SQL like DB language, this can be done at first using e.g. a PASCAL subprogram.

This is probably of not such a big concern to HPBB users because already todays THREAD, SEARCh and SORT commands provide relational like access to IMAGE.

Increasing compatibility downwards to the 260

As the 260 despite its limitations is still going strong especially in Europe some tentative projects are discussed (JOIN/260) enhancing the BASIC/260 language with constructs from HPBB and maybe even make some changes to already existing statements with the goal of being more compatible with the grownup members of the family.

With all these activities going on you can clearly see that Business Basic has a strong support within HP. HPBB is planned to be <u>the</u> commercial Basic for all future systems, from the 260 across the 3000 up to SPECTRUM.

Acknowledgements

I am very grateful to HP Böblingen and Cupertino for enabling me to stay with the research team directly in Cupertino for over 3 months. Also to my company for letting me go for such a long time to ensure the usefullness of the product for 260 OEMs.

My thanks also go to the HPBB team for the good cooperation despite my being sometimes a very uncomfortable demanding customer but I think it turned out to benefit both sides, HP and HPBB user.

Authors biography

Dr.Ulrich Fauser
has been with WEIGANG MCS since 1981.

He is R&D manager and system software specialist therefore responsible for the selection of future hard- and software for use within his company.

He also deals with the europewide system software and technical support for all used computersystems, right now exclusively HP products.

He studied computer science at the university of Stuttgart with emphasis on compiler construction and microprogramming. While he was teaching computer science there afterwards his interest shifted to distributed relational database systems of which a prototype was implemented by his research team. This has been also the subject of his doctoral thesis in 1980.

Before he started with his present company he spent a year at the University of California in Santa Cruz researching within Prof. F.Deremers Translator Writing Systems project.

His primary interest today are operating systems, databases and programming languages.

Privately his favorite pastimes are sailing, skiing and scuba diving.

References

HPBB Reference Manual , 32115-90001

HPBB Quick Reference Guide , 32115-90002

HPBB Programmers Guide , 32115-90003

BASIC/3000 to HPBB Conversion Guide , 32115-90004

BASIC/250 to HPBB Conversion Guide , 32115-90005

Phase II External Reference Specification

JOINFORM External Reference Specification

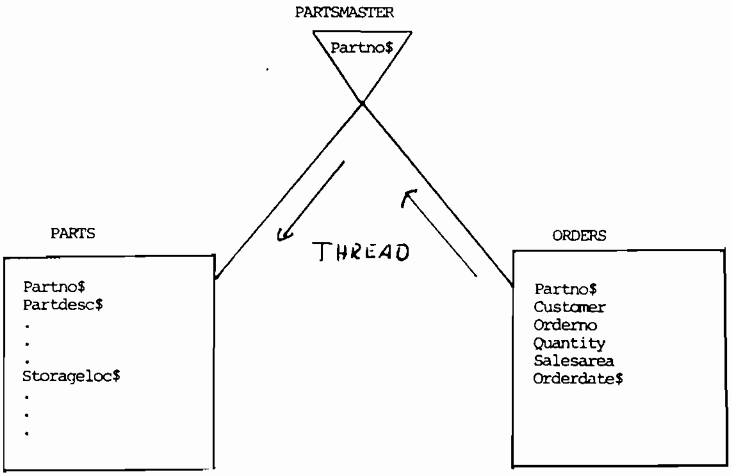
no manuals for phase II are available yet

```
1     :! Perform "PERF01:M"    ! COMMAND file
2     :! Files, beginning with FBH (FIBU) and of type
3     :! PROG, DATA or FORM , are read per CATLINE from
4     :! tape and copied to :M, PROG files are saved as
5     :! ASCII-Files
6     :! In a parallel process an EDITOR file is created,
7     :! later to be used as a batchfile for the TRNSFR program
8     :! The program WART1 only waits for the parallel task to
8.1   :! be in INPUT state
9     :! ***********************************************
10    REQUEST#7
11    SEND CONTROL HALT#7
12    SEND COMMAND#7,"RUN "&CHR$(34)&"EDITOR:M"&CHR$(34)
13    RUN"WART1:M"
14    SEND INPUT#7,"S LENGTH=160"
15    RUN"WART1:M"
16    SEND INPUT#7,"A"
17    RUN"WART1:M"
18    :DIM PARM(8)
19    :SET PARM(1) TO 0
20    :LOOP
21    :SET PARM(1) TO PARM(1)+1
22    CATLINE PARM(1) ON ":K" ,A$
23    :EXIT IF A$[1,3]="EOD"
24    :IF A$[1,2]="FB"  THEN
25    :IF A$[14;4]="PROG" OR A$[14;4]="DATA" OR A$[14;4]="FORM" THEN
26    :SET PARM(5) TO NUM(A$[4;1])
27    :SET PARM(6) TO NUM(A$[5;1])
28    :SET PARM(7) TO NUM(A$[6;1])
29    :IF A$[14;4]="PROG" THEN
30    LOAD A$[1,6]&":K"
31    B$="FBP"&CHR$(PARM(5))&CHR$(PARM(6))&CHR$(PARM(7))&":M"
32    SAVE B$
33    SEND INPUT#7,"DATA_FILE_TRANSFER "&B$&" TO FBH"&B$[4,6]
34    RUN"WART1:M"
35    :ELSE
36    :IF A$[14;4]="DATA" THEN
37    B$="FBD"&CHR$(PARM(5))&CHR$(PARM(6))&CHR$(PARM(7))&":M"
38    :END IF
39    :IF A$[14;4]="FORM" THEN
40    B$="FBF"&CHR$(PARM(5))&CHR$(PARM(6))&CHR$(PARM(7))&":M"
41    :END IF
42    SEND INPUT#7,"ARCHIVE_TRANSFER "&B$&" TO FBH"&B$[4,6]
43    C$="FBH"&B$[4,6]&":K"
44    COPY C$ TO B$
45    RUN"WART1:M"
46    :END IF
47    :END IF
48    :END IF
49    :END LOOP
50    SEND INPUT#7,"//"
51    RUN"WART1:M"
52    SEND INPUT#7,"K"&CHR$(34)&"TRATES:M"&CHR$(34)&",UNN"
53    RUN"WART1:M"
54    SEND INPUT#7,"E"
55    :END IF
56    :END
```

```
:! SAMPLE OUTPUT OF PERFORM COMMAND FILE

DATA_FILE_TRANSFER FBP044:M TO FBH044
DATA_FILE_TRANSFER FBP045:M TO FBH045
DATA_FILE_TRANSFER FBP046:M TO FBH046
DATA_FILE_TRANSFER FBP047:M TO FBH047
DATA_FILE_TRANSFER FBP048:M TO FBH048
DATA_FILE_TRANSFER FBP049:M TO FBH049
DATA_FILE_TRANSFER FBP051:M TO FBH051
DATA_FILE_TRANSFER FBP052:M TO FBH052
DATA_FILE_TRANSFER FBP054:M TO FBH054
DATA_FILE_TRANSFER FBP065:M TO FBH065
DATA_FILE_TRANSFER FBP057:M TO FBH057
DATA_FILE_TRANSFER FBP055:M TO FBH055
DATA_FILE_TRANSFER FBP053:M TO FBH053
DATA_FILE_TRANSFER FBP056:M TO FBH056
DATA_FILE_TRANSFER FBP058:M TO FBH058
DATA_FILE_TRANSFER FBP059:M TO FBH059
ARCHIVE_TRANSFER FBF301:M TO FBH301
ARCHIVE_TRANSFER FBF302:M TO FBH302
ARCHIVE_TRANSFER FBF303:M TO FBH303
ARCHIVE_TRANSFER FBF304:M TO FBH304
ARCHIVE_TRANSFER FBF305:M TO FBH305
ARCHIVE_TRANSFER FBF306:M TO FBH306
ARCHIVE_TRANSFER FBF307:M TO FBH307
ARCHIVE_TRANSFER FBF308:M TO FBH308
ARCHIVE_TRANSFER FBF309:M TO FBH309
ARCHIVE_TRANSFER FBF310:M TO FBH310
ARCHIVE_TRANSFER FBF311:M TO FBH311
ARCHIVE_TRANSFER FBF312:M TO FBH312
ARCHIVE_TRANSFER FBF313:M TO FBH313
ARCHIVE_TRANSFER FBF314:M TO FBH314
ARCHIVE_TRANSFER FBF315:M TO FBH315
ARCHIVE_TRANSFER FBF316:M TO FBH316
ARCHIVE_TRANSFER FBF317:M TO FBH317
ARCHIVE_TRANSFER FBF318:M TO FBH318
ARCHIVE_TRANSFER FBF319:M TO FBH319
ARCHIVE_TRANSFER FBF320:M TO FBH320
ARCHIVE_TRANSFER FBF321:M TO FBH321
ARCHIVE_TRANSFER FBF322:M TO FBH322
```

```
! simple HPBB example program demontrating database extensions
! and report writer
! not every detail shown
! (line numbers have been ommitted)
! YES I know it can be done in a different and easier way

SUB Orderreport
  COM Dbas$,Limit  ! lets assume database opened excusive in MENU

! Variable definitions only partially

  DIM Partno$[16],Storageloc$[6]
  INTEGER Customer,Orderno
  DECIMAL Quantity


Dset1: IN DATASET "ORDERS" USE Partno$,Customer,Orderno,Quantity,
        Salesarea,Orderdate$
Dset2: IN DATASET "PARTSMASTER" USE Partno$
Dset3: IN DATASET "PARTS" USE Partno$,Partdesc$,SKIP 46,Storageloc$

  Prt=FNSelectprinter      ! ask user which printer to use
  SYSTEM "FILE LP;DEV="+VAL$(Prt)
  DBASE IS Dbas$
  ASSIGN #1 TO "WORK"
Input: !
  OFF KEY
  CALL Clearscreen
  CALL Ask_user("Report for which salesarea",Area$)
  CALL Ask_user("for which month",Month$)
  WORKFILE IS #1
  POSITION #1;RESET
Thread1: THREAD IS Dset1,Dset2,Dset3
  SEARCH USING Thread1;Salesarea=VAL(Area$) AND VAL(Orderdate$[3;2])
      =VAL(Month$)
  IF NUMREC(#1)=0 THEN
    CALL Msg("No orders for this salesarea and month")
    ON KEY 8 GOTO Final;LABEL="EXIT"
    ON KEY 2 GOTO Input;LABEL="NEW INPUT"
    LOOP
      ACCEPT
    END LOOP
  END IF
  IF NUMREC(#1)>1 THEN SORT USING Thread1;Customer,Partno$
  ON HALT GOTO Exit
  SEND OUTPUT TO "*LP"
  BEGIN REPORT Rpt1
  FOR I=1 TO NUMREC(#1)
    READ #1;Orderptr,Dummy,Partptr
    DBGET Dbas$ USING Dset1;DATASET="ORDERS",MODE=4,KEY=Orderptr
    DBGET Dbas$ USING Dset3;DATASET="PARTS",MODE=4,KEY=Partptr
    DETAIL LINE 1 USING Dt1;Orderno,Quantity
  NEXT I
  END REPORT
Exit: STOP REPORT
Final: SEND OUTPUT TO DISPLAY
  SYSTEM "RESET LP"
SUBEXIT           ! return to menu
```

```
! Report layout for monthly order printout

Rpt1: REPORT HEADER WITH 3 LINES
    GRAND TOTALS ON Quantity*Price
    PAGE LENGTH 72,2,2
    LEFT MARGIN 10
    BREAK 1 WHEN Customer CHANGES
    BREAK 2 WHEN Partno$ CHANGES
    PRINT USING Hd1;Title$,Area$
  PAGE HEADER WITH 2 LINES
    PRINT USING Ph1;DATE$(1),TIME$
  PAGE TRAILER WITH 0 LINES         ! can also be omitted
  HEADER 1 WITH 2 LINES
    TOTALS ON Quantity*Price
    GOSUB Readcustomername
    PRINT USING Hd1;"Orders for "+Custname$
    HEADER 2 WITH 2 LINES
      TOTALS ON Quantity,Quantity*Price
      PRINT USING Hd2;"Orders for "+Partno$,Partdesc$,Storageloc$
    TRAILER 2 WITH 2 LINES
      PRINT USING Tr2;"Quantity for this part ",TOTAL(2,1)
      PRINT USING Tr2_1;"Ordervalue ",TOTAL(2,2)
  TRAILER 1 WITH 2 LINES
    PRINT USING Tr1;"Ordervalue for this customer ",TOTAL(1,1)
  REPORT TRAILER WITH 3 LINES
    PRINT USING Rtr1;"Total ordervalue for salesarea",TOTAL(0,1)
    IF TOTAL(0,1)<Limit THEN PRINT "Orderlimit not reached"
  REPORT EXIT WITH 3 LINES
    PRINT USING Re1;"TERMINATION DUE TO USERREQUEST"
  END REPORT DESCRIPTION

! Output format specifications would be here

SUBEND

! Other subroutines like Ask_user, FNSelectprinter etc would follow
```



Jorge Guerrero
Hewlett-Packard Cupertino
U.S.A.

NOTE: See page 627.

HP BUSINESS BASIC/3000: AN UPGRADE PATH FOR HP260 APPLICATIONS

Mark L. Hoeft
Hewlett-Packard-Computer Languages Lab
Cupertino, California, USA

Summary

One of the goals of HP Business BASIC/3000 is to provide an upgrade path from the HP260 to the HP3000. The next release will be targeted specifically for this market. Because the HP260 and the HP3000 are very different machines this conversion will not be 100% automatic. This paper will discuss how to convert HP260 applications to HP Business BASIC/3000, dealing with the mechanics of the automatic conversion and how to make the manual conversion as easy as possible. How to have your applications make the most out of HP Business BASIC/3000 will also be discussed.

Introduction

HP Business BASIC/3000 Phase I was introduced last year and Phase II will be introduced later this year. It is a complete program development system providing editing, debugging and program execution in one environment. The main reason that HP has brought out HP Business BASIC/3000 was to provide upgrade paths from HP260 to HP3000 and to provide a superior BASIC for BASIC/3000 users. HP Business BASIC/3000 was started in August 1981, after several years of investigation and specification. It has taken over 60 man-years to complete, and contains over 310,000 lines of PASCAL source (180,000 lines of real code). HP Business BASIC/3000 will be the standard for BASIC on all future HP commercial systems.

HP Business BASIC/3000 is intended to be used in developing business applications and utilities. Its easy subsystems access makes it convenient to write large applications and its rich language features and interpretive environment make it easy to write quick utilities. HP Business BASIC/3000's features allow the programmer to spend time on WHAT should be done and not on HOW to do it.

There is a conversion package to be used in moving HP260 applications to the HP3000. The conversion package will allow the transfer and conversion of BASIC/260 source programs, data files, databases and forms files. The run-time behavior of each HP Business BASIC/3000 statement has been carefully designed to duplicate, as closely as possible, the behavior of the corresponding BASIC/260 statement.

The specific details for the conversion from BASIC/260 are described in the BASIC/260 to HP Business BASIC/3000 conversion guide. It should be read before attempting a real conversion. This manual describes the mechanics of doing the automatic conversion. However, it does not describe many of the little tricks which can be used to make the conversion easy, nor does it describe what to do about the incompatibilities that do exist. This paper will try to do both of these. It will begin with a quick overview of the conversion process, and then will go into more detail supplying the user with hints to make the process easier and pitfalls to avoid. The sections dealing with the manual conversion, and enhancement and optimization go into detail about how to get your application up and running, and running well. Also, hints will be given to those programmers with advanced skills who will be converting large applications.

Overview of Conversion

Before you start you should make sure that this application will be successful on the HP3000. Was it written many years ago and is now out of date or are there already other applications on the machine which do just what yours does. Perhaps only some of your applications need to be converted. Perhaps some of the applications can be converted to Phase I instead of waiting for Phase II to convert all of the applications at once. Also, keep in mind what you are going to be doing with new applications.

In general, there are 5 steps in converting an application. These are site preparations, transferring the files, automatic conversion, manual conversion, and enhancement and optimization. Site preparations deals with making sure everything is ready to transfer the files, and to do the conversion. Transferring the files from the HP260 to the HP3000 can be done in several ways: using a DS line or a terminal line between the two systems. The automatic conversion does most of the needed conversion. The manual conversion is the remaining work necessary to get the application up and running. The enhancement and optimization portion of the conversion involves making the application run well, and letting it take advantage of the features of HP Business BASIC/3000 and the HP3000. The first three steps are fairly easy. It is the step of manual conversion in which most of the work will be done.

Before the conversion process is explained, some expectations should be set. First, some parts of HP Business BASIC/3000 are very compatible with BASIC/260 (i.e. exactly the same). Other parts are different or do not exist at all. Fortunately, most of the language is compatible, but how much work will be needed depends on the mix of features that the application uses. Some applications will be very easy to convert; others will be very difficult. This paper and the conversion guide will help you figure out how difficult the conversion is. Second, the HP3000 is a different machine. Even though much of BASIC looks the same, things are different and hopefully better. Third, the highest level of compatibility is in the interpreter because many applications use non-compilable features such as GET SUB and COMMAND. The compiler is many times faster than the compiler for CPU intensive programs. The proper use of the interpreter is for development, and then compile the application for production use. This will result in greater user satisfaction. Fourth, there are a fair amount of details to be dealt with in the conversion. Programs won't work automatically. However, the more time taken to make the programs better, the more satisfying the end product. Because of the differences in the BASIC, a program which is converted "just so it runs" will not run well. Take the time and do a good job.

Many applications can be converted into the Phase I version of BASIC, however, some may need to wait till Phase II for some features such as forms, keys, reports and some advanced data base features. Even if this choice is made, it is to the user's advantage to get onto HP Business BASIC/3000 during Phase I to learn this version of BASIC and to learn the HP3000 in general. There will be an updated version of the conversion program when Phase II is released.

Site Preparations

The first step in site preparations is to understand what you are going to be dealing with. This means you should know your application, the HP260, the HP3000, HP Business BASIC and any subsystems which will be used. A good way to start this is to read the manuals on these items. You should have up-to-date manuals and listings of everything that you will be using. This means HP manuals (HP260 and HP3000), your manuals, program listings, data bases schemas (from DBMODS), forms layouts (from PFORM) and any internal documentation of your application.

Also, you need to have solid equipment and software. Make sure that you are using a fully supported system. Both the HP260 and the HP3000 should have the latest software releases. A link must be established between the two machines. This can either be a DS (Distruted Systems) line or a regular terminal line. The DS line is recommended. Both machines should be completely backed up.

It will help if you set up an account on the HP3000 to be used for the conversion. It should have groups in for files just transferred, files from the conversion program, reports from the conversion program, files which need hand conversion, files which are fully converted, documentation files and miscellaneous files. The names of these groups could be HP260, CONVERT, REPORT, HALFCVRT, FULLCVRT, DOC and MISC. Some users may find if they only have a small number of files to be converted that they can use one group with letters appended to the end of the file names to distinguish what version the file is. Since the maximum number of characters in an HP260 file name is 6, and the maximum number of characters in an HP3000 file name is 8, this should be no problem. Remember that only letters and numbers can be used in HP3000 file names, that they must start with a letter, and that the case of the letters has no meaning.

PROGRAMS	:	unprotected, unsecured, unrunonly, DATA (SAVE) files
DATA FILES	:	unprotected DATA files created with the CREATE not the FCREATE command
FORMS	:	unprotected FORM files
DATA-BASE SCHEMAS	:	unprotected DATA (DBMODS) files
DATA-BASE DATA SETS	:	unloaded, unprotected BKUP (DBUNLD) files
KEYS FILES	:	cannot be converted

If you have a large number of files to prepare, you may wish to use the PERFORM statement and write some small programs to build scripts. For instance, you could use the CATLINE statement to get a list of all of the PROG files on a disk and use this to build a script which would do a SCRATCH P, LOAD "PROG" and SAVE "DATA" for each program file. This can also be useful in any of the other tedious tasks which must be done for a lot of files.

If you have the opportunity, you may wish to change your application on the HP260 to not use features which are not available in HP Business BASIC. This is true if you are planning to enhance it and will be changing it anyway. Sometimes the choice of features is arbitrary and now that you know that it makes a difference you will choose features which are easily converted. Also, whenever you write new code you should use features which are easily converted. See the section on manual conversion and Appendix C in the BASIC/260 to HP Business BASIC Conversion Guide.

Transferring Files

The next step in converting an application is to transfer all of the files used in the application from the HP260 to the HP3000. There are two suggested means of transferring the files: the preferred method DSM/DS/260, and a program based on LK3000 called TRNSFR. Put all files in the HP260 group on the HP3000.

On the HP260, files of characters are just regular DATA files, but on the HP3000 these files are called ASCII files and are different from BINARY files. You must know what kind of file you wish to use on the HP3000 when you do the file transfer, and there is a different style of transfer for the two file types. Here is a table of the correct styles:

PROGRAMs	:	DATA FILE TRANSFER of the saved program
DATA FILES	:	ARCHIVE FILE TRANSFER
FORMS	:	ARCHIVE FILE TRANSFER
DATA-BASE SCHEMAS	:	DATA FILE TRANSFER
DATA-BASE DATA SETS	:	ARCHIVE FILE TRANSFER of the UNLOADED file
Documentation files	:	DATA FILE TRANSFER
Miscellaneous files	:	DATA FILE TRANSFER

For information on DSM/DS/260 see the DSN/DS/260 manual. For information on TRNSFR/260 see appendix A of the Conversion Guide. As to which of the two means of transfer to choose, there are some good points and some bad points to each.

1) DS/260 uses a distributed systems line between the two systems. This line takes care of any errors that occur between the systems. TRNSFR/260 gets confusing if there is any noise on the line.

2) DS/260 runs at 9600 baud. The fastest supported speed of TRNSFR is 4800 baud and all bytes are transmitted in hex and, as a result, take two characters each on the transmission line. Therefore, the fastest speed of TRNSFR is really 2400 baud, which is 4 times slower than DS/260.

3) DS/260 makes ARCHIVE transfers directly from the file. In the current release of TRNSFR, a separate file is made which contains hex. The extra writes and reads to and from this file have a serious negative effect on the performance of TRNSFR. This file is 2 to 3 times the size of the original file which may create disk space problems.

4) TRNSFR/260 uses regular terminal lines and therefore doesn't require the extra board in each system that DS/260 requires resulting in a cost savings. DS/260 needs to be installed by an SE. For TRNSFR to be used, file BBCTHEX.PUB.SYS needs to be present on the system.

5) TRNSFR/260 is slightly easier to use for transferring several files one after another. The PERFORM DROM must be used to do this for DS/260 and because of the need to push softkeys to do the transfer, this is a bit tricky. However, this is described in Appendix D in the HP260 to HP Business BASIC Conversion Guide. TRNSFR some decisions about the number of records and the size of them for ASCII files on the HP3000. The decision about the record size makes it a little easier but sometimes it is very inefficient. The record size TRNSFR uses is 160 bytes in a record. This is correct for program files with long lines, but if all of the lines in the program are shorter or you are transferring documentation, your file probably takes up twice as much space as it needs to.

Regardless of the method of transfer, here are some hints. If you have a large number of files try to write a script for use with PERFORM or BATCH to let the transfer run unattended. This lets the programmer spend his or her time doing something productive instead of watch a terminal. If the transfer is to run overnight, be aware that you will not be able to create new files when the HP3000 is being backed up.

Automatic Conversion

The core of the automatic conversion is the file BBCT250.PUB.SYS. However, there may be a few extra steps necessary for the conversion depending on the file type used. To use this program RUN BBCT250.PUB.SYS and type CONVERT <in file name>,<out file name>,<report file name . The file type of the conversion can be used instead of the word CONVERT to avoid confusion. It is suggested to make a report of all of the files converted. Here is a list of how the conversion is done. This list has been paraphrased. For a detailed description, see the Conversion Guide.

PROGRAMs:

```
:RUN BBCT250.PUB.SYS
=> PROG MYPROG.HP260,MYPROG.CONVERT,MYPROG.REPORT
=> EXIT
:BBASIC
>COPY ALL OUTPUT TO MYPROG.MISC
>GET MYPROG.CONVERT
>SAVE MYPROG.HP3000
: EXIT
```

DATA FILES:

```
:RUN BBCT250.PUB.SYS
=> DATA MYDATA.HP260,MYDATA.FULLCVRT
=> EXIT
```

FORMS:

There are two kinds of forms in HP Business BASIC. One is based on VPLUS and one was written from scratch to provide a higher level of compatibility with the HP260. Here is a description of the conversion to a VPLUS form. The conversion and use of the other forms file type is described in the Conversion Guide.

```
      :RUN BBCT250.PUB.SYS
      => FORM MYFORM.HP260,MYFORM.FULLCVRT,MYFILE.REPORT
      => EXIT
```

DATA-BASE SCHEMAS:

```
      :RUN BBCT250.PUB.SYS
      => SCHEMA MYSCHEMA.HP260,MYFILE.FULLCVRT
      => EXIT
```

Create the data base in the correct account with DBSCHMEA.PUB.SYS

DATA-BASE DATA SETS:

```
      :RUN BBCT250.PUB.SYS
      => DATASET MYDSET1.HP260, name of database
      => DATASET MYDSET2.HP260, name of database
      => EXIT
```

DOCUMENTATION AND MISCELLANEOUS FILES:

Move these to the group DOC and MISC. This may be a simple file copy or you may wish to change the record length.

After all of this is done you should make an offline listing of the names of your files using the LISTF,2 MPE command, print of all the files and make sure a backup is done soon (perhaps that night). The files to be printed include programs, reports, schema and documentation.

There are some options in doing the conversion. These are OPTION INPUTLOOPS and OPTION REAL. OPTION INPUTLOOPS places loop around INPUT statements in converted programs. It is described in chapter 3 of the Conversion Guide. OPTION REAL converts all decimal values to real values during the conversion. There is no direct hardware support for DECIMAL data types and so they are slower than REALS, however, they are more accurate.

Manual Conversion

Although it may be tedious, everything that has been discussed so far has been rather straight forward and easy to do. The remainder of the work to be done may be easy or may be difficult, depending on your application. This is the place where technical expertise is most needed. Manual Conversion is that part where you take a program and change it so that it syntaxes, and then runs correctly. Normally this is the bare minimum that must be done, but there may be a lot of room for improvement in the application. The resulting programs should be placed in the group FULLCVRT.

Most of the incompatibilities between BASIC/260 and HP Business BASIC are described in Appendix C of the Conversion Guide. However, there are a few to watch out for:

1) Programs can be much larger than they were in BASIC/260, however, the individual subunits (main, subprograms and multi-line functions) must be smaller. For a well modulized program this will not be a problem, but you have to break up a large program into multiple subunits, and this can be very time consuming.

2) There are certain features which do not exist at all, in which case those portions of the application which use these will need to be rewritten.

3) you may find that the problems encountered in manual conversion are with features of your application which can be removed

because these deal with minor features.

4) You may find that the work necessary to deal with the problems encountered in manual conversion is greater than the rewriting portions of the application. If this is the case, rewrite those portions.

Optimization and Enhancements

Now you have an application that runs but there may still be some work to do. There may be some features to be added to the application. The application may not take advantage of some of the features of the language or the system. The cost of optimization and enhancement should not be counted in the cost of converting your application but they can add greatly to the success of your application.

There was a paper presented at the INTEREX'85 conference in Washington, D.C., USA by the author in September, 1985 called Developing Cost Effective Applications and Utilities using HP Business BASIC/3000. It gives many suggestions on performance tuning. These are a few of those suggestions:

1) Get your program to compile. Normally, this will be just removing COMMAND, GET SUBS and DEL SUBS from the program. However, there may be more to it if your subunits are too big. Compiling your program will make the CPU intensive portions run many times faster. If you cannot compile the entire application you may be able to compile portions of your application.

2) There are several compiler options to improve the performance of your application. These are the COPTION NO RANGE CHECKING and COPTION NO ERROR CHECKING.

3) Some part of the application could be rewritten to take advantage of features that did not exist in BASIC/260.

Conclusion

This paper is intended to give the user a better understanding of the how to convert BASIC/260 applications to HP Business BASIC/3000 applications. Hopefully, the hints and discussion of the process will help in the conversion. Please note that this is only one of the sources of information on this topic and needs to be used in connection with the BASIC/260 to HP Business BASIC/3000 Conversion Guide. With the proper preparation and understanding of the conversion process you should be able to provide a successful application on the HP3000.

References

For more information on the conversion process or on any of the features in HP Business BASIC/3000, please refer to the HP Business BASIC/3000 Manual series. This is a set of five manuals which constitute the user documentation of HP Business BASIC/3000. There is also an on-line HELP facility in the interpreter which provides information on syntax and functionality of all of the HP Business BASIC/3000 keywords and statements.

BASIC/260 to HP Business BASIC/3000 Conversion Guide, Part No. 32115-90010

This guide describes how to convert BASIC/260 applications into HP Business BASIC/3000 applications. It also describes the incompatibilities and the conversions which take place.

HP Business BASIC/3000 Programmer's Guide, Part No. 32115-90007

This guide is for those who want to learn how to program in HP Business BASIC/3000.

HP Business BASIC/3000 Reference Manual, Part No. 32115-90006

This manual describes all of the features of HP Business BASIC/3000. It is for those who wish to look up how an HP Business BASIC/3000 feature works.

HP Business BASIC/3000 Quick Reference Guide, Part No. 32115-90008

This is a "Quick Reference" version of the reference manual.

BASIC/3000 to HP Business BASIC/3000 Conversion Guide, Part No. 32115-90009

This guide describes how to convert BASIC/3000 applications into HP Business BASIC/3000 applications. It also describes the incompatibilities between the two BASICs.

The following manuals may also be helpful:

BASIC/260 Programming Manual

BASIC/260 Utilities Manual

HP/260 Console Operators Guide

DSN/DS/260 User's Manual

FORMS/260 Reference Manual

IMAGE/260 Programming Manual

TIO/260 Programming Manual

REPORT WRITER/260 Programming Manual

MPE Commands Reference Manual

IMAGE/3000 Programming Manual

HP3000 Intrinsics Manual

VPLUS/3000 Reference Manual

SORT-MERGE/3000 Reference Manual

Developing Cost Effective Applications and Utilities Using HP Business BASIC/3000 - Mark L. Hoeft, 1985 INTEREX, Washington D.C.

Biography

Mark Hoeft
has worked at Hewlett-Packard for 6 years and has been on the HP Business BASIC/3000 project for 5 years. Before that, he worked on the HP260 Lab. He wrote the HP260 to HP Business BASIC/3000 conversion package, and worked on the User Interface portions of HP Business BASIC (KEYS, FORMS, etc.) He also presented a paper at the 1985 INTEREX meeting in Washington, D.C. on the topic of HP Business BASIC.

Richard Irwin
Hewlett-Packard Cupertino
U.S.A.

NOTE: Due to the nature of the subject is not possible to include the full paper in the Conference Proceedings at this time.

DEVICE INDEPENDENT GRAPHICS SOFTWARE FOR THE HP3000

Peter Neuhaus
Hewlett Packard Company
Cupertino, California, USA

Summary

Occasionally, the requirements for displaying data graphically go beyond the capabilities of existing software packages. Consequently, it becomes necessary to begin the unpleasant task of developing custom code. Since this typically involves a significant investment of resources, it is important that the software be written so as to maximize its longevity and maintainability. To achieve these goals, it is necessary to understand the basic principles of graphic device independence and the concept of the Virtual Device Interface.

This paper will discuss these topics along with a discussion of how available graphics software tools can be used to assist in the development of device independent graphics software for the HP3000.

Background

In the early 1970s, the computer graphics industry realized that it needed to standardize some of the methods used in developing graphics software. The resulting conventions made it possible to create graphics in one environment (computer) and transport them to another with a minimum of recoding.

To date, only a few standards have been established, but others are under investigation. The Graphics Kernal System (GKS) has been adopted by the International Standards Organization and is being used extensively throughout Europe while the Siggragh CORE system, proposed in 1979, has not gained much acceptance. The debate continues, but GKS seems to be pulling ahead.

Regardless of whether or not one chooses to follow a strict standard, considerable improvements can be made in the writing of graphics software by following a few simple guidelines.

Frequently, companies plan to use only the specific graphics output devices that they already own, for example a HP7550 plotter or perhaps a non-HP graphics terminal. To support these devices, the specific commands required by the devices would be scattered throughout the application program (see figure 1). The result would be very efficient but would necessitate excessive modifications if new or additional output devices were acquired at a later date.

The First Step

Device independence is nothing new to the professional programmer. Common functions such as cursor control are often modularized into separate subroutines (device drivers) that can be easily modified or replaced to accommodate new output devices that require different commands for their proper operation. When it was necessary to drive more than one output device, a duplicate set of subroutines is written for each device (see figure 2).

In addition, if more than one device might be used simultaneously, it is necessary for subroutines with identical functions to have different names, such as LINE1, for drawing a line on device 1, or LINE2 for device 2. At this level, device independence was still not achieved since the LINE1 and LINE2 calls must be embedded in the application program.

Step Two

By inserting another level between the application program and the device drivers, the interface between the application program and the outside world is standardized. If this new level, perhaps a commercially available GKS package or CGL/3000 from the contributed library, contains a function that allows the application program to select which output device should be used, it is possible to remove the references to LINE1 and LINE2 and substitute a call to the new LINE function in the GKS/CGL package (see figure 3).

At this point, true device independence has been achieved since new devices can be supported without modifying the application program as long as someone writes a device driver for the new device. However, creating these new drivers can consume enormous amounts of programming effort because each device is unique in that it requires specific nonstandard "escape sequences" to perform a given task.

VDI - The Last Step

The graphics industry is attempting to standardize the hardware instructions required by graphic output devices through a concept called the Virtual Device Interface (now often called the Computer Graphics Interface). A VDI driver accepts all the command that a generic device might receive but only implements or emulates those that its device can perform.

If all graphic devices understood the same commands, the need for device drivers would be eliminated (see figure 4). Essentially, the device drivers would be implemented within the device's firmware. However, until the VDI concept becomes commonplace, it is necessary to employ the basic concepts of device independence when writing graphics applications. Several alternatives are possible.

Ways to be Independent

The most straightforward solution would be to obtain a graphics software library either from the computer manufacturer or from an independent third party. Such packages include a number of device drivers for the most popular graphic devices. The disadvantage to this solution becomes evident if it is necessary to change host computers at a later date.

Even switching between computer lines offered by the same manufacturer can cause significant problems. Therefore, when shopping for this type of software product, it is important to investigate the possibility of moving the product between systems. Packages written in standard languages such as Fortran or Pascal help simplify portability. But even standard languages often do not port well.

The ability to move to another CPU may sound like something that would not be done too often, but as desktop computers become as powerful as typical multi-user systems, many applications will be moved to smaller workstations. It's much like the user who feels he needs only 50 megabytes of disc storage, orders 100MB even though he "knows" it will never be needed, then runs out of disc space six months later. Applications and technologies change continuously. Investing the extra resources to implement a flexible solution often pays high dividends at a future date.

Sharing Graphics Data

Frequently, graphic databases created on one system need to be processed on another. To address this need, a standard format for exchanging databases, called the Initial Graphics Exchange Specification (IGES), has been established and is currently supported by a number of graphics packages. A similar newer standard, the Virtual Device Metafile (VDM), performs much the same functions.

Within HP, a standard called the Graphics Peripheral Interface Standard (GPIS) has been developed and is currently used by some HP150 graphics programs. Since it is similar to VDM, it can be easily modified to conform to the final version of VDM.

By simply using the IGES or VDM device driver, an application can store the resulting image or object description onto a transportable media such as magnetic tape, which can then be read by another IGES/VDM compatible system. Applications written in a device independent manner are able to utilize this useful feature.

Summary

The trade-offs involved in the decision to standardize the development of computer graphics software deals mainly with short term versus long term benefits. Projects that seem to be "one shot" programs may not appear to necessitate the features of device independence. But often, the programs are modified and used again, possibly for another "one shot" application. In general, establishing standards or guidelines in a programming environment leads to increases in productivity. The slight performance degradation created by the overhead of a graphics subroutine library can be offset by the ever decreasing costs of computer hardware.

Once standards have been implemented, applications can be developed faster since it becomes unnecessary to reinvent the wheel for each new project. In addition, program maintenance is simplified since each programmer understands the basic strategies used by his fellow graphics programmers. Overall, the need to be device independent will become increasingly important as the number and capabilities of systems and graphic devices expand.

Biography

Peter Neuhaus is a computer graphics specialist in the Information Systems and Networks Group, Marketing Communications Graphics Department. He is involved in exploring the use of computers by graphics artists to increase their productivity. During a leave from Hewlett-Packard, he taught computer graphics to engineers and artists at California State University, Long Beach.

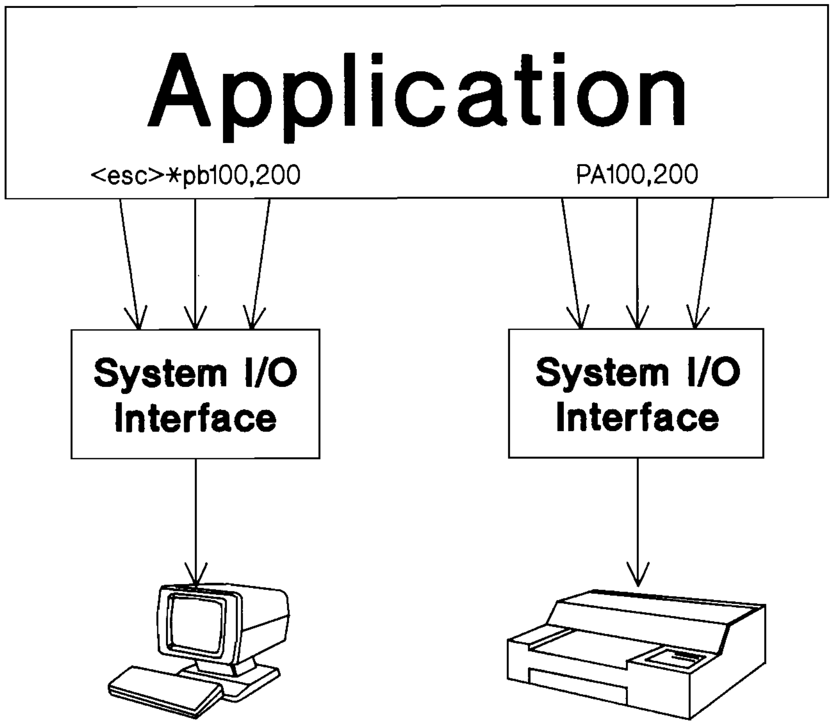


Figure 1 - Device dependent commands scattered within application program

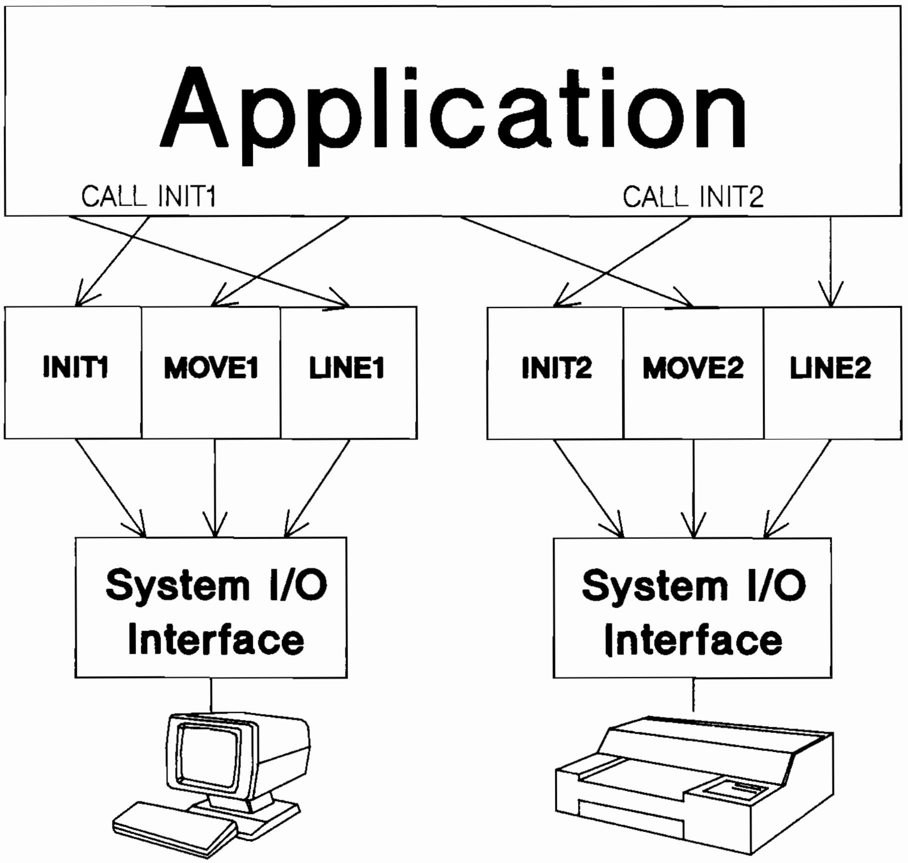


Figure 2 – One set of subroutines for each device

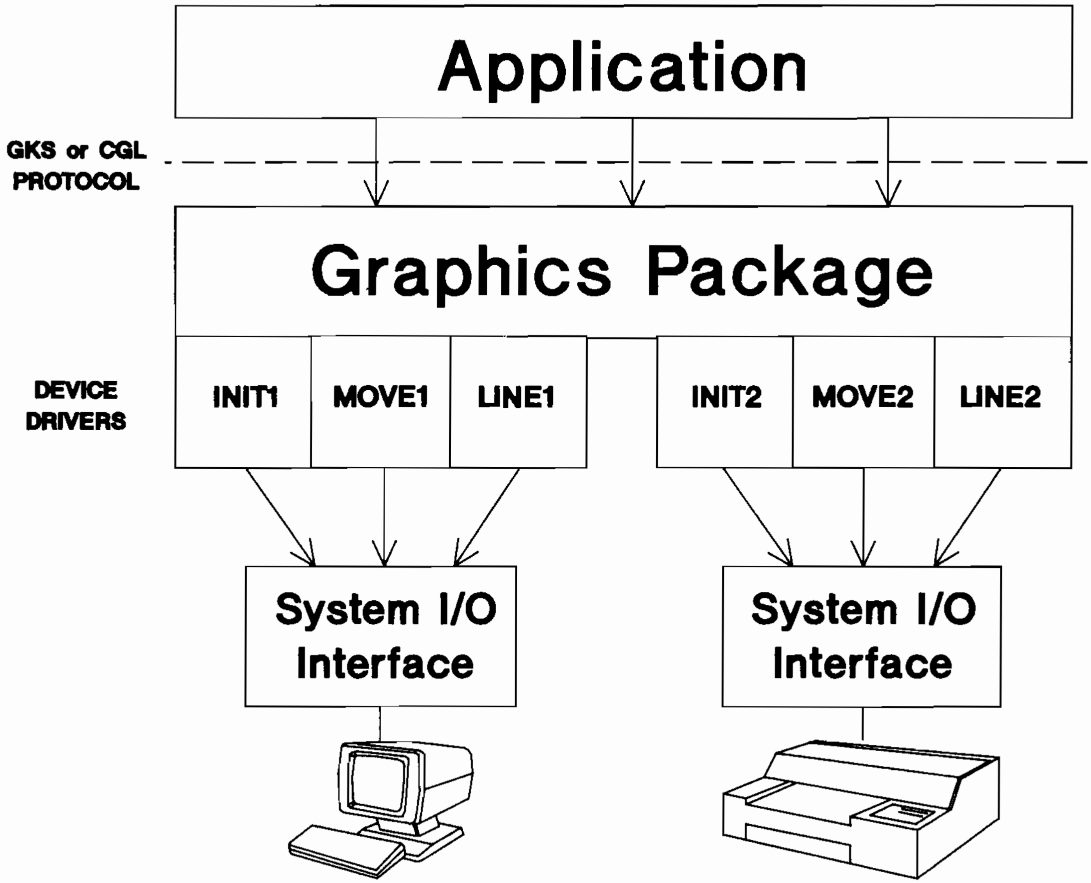


Figure 3 - Graphics package with its device drivers

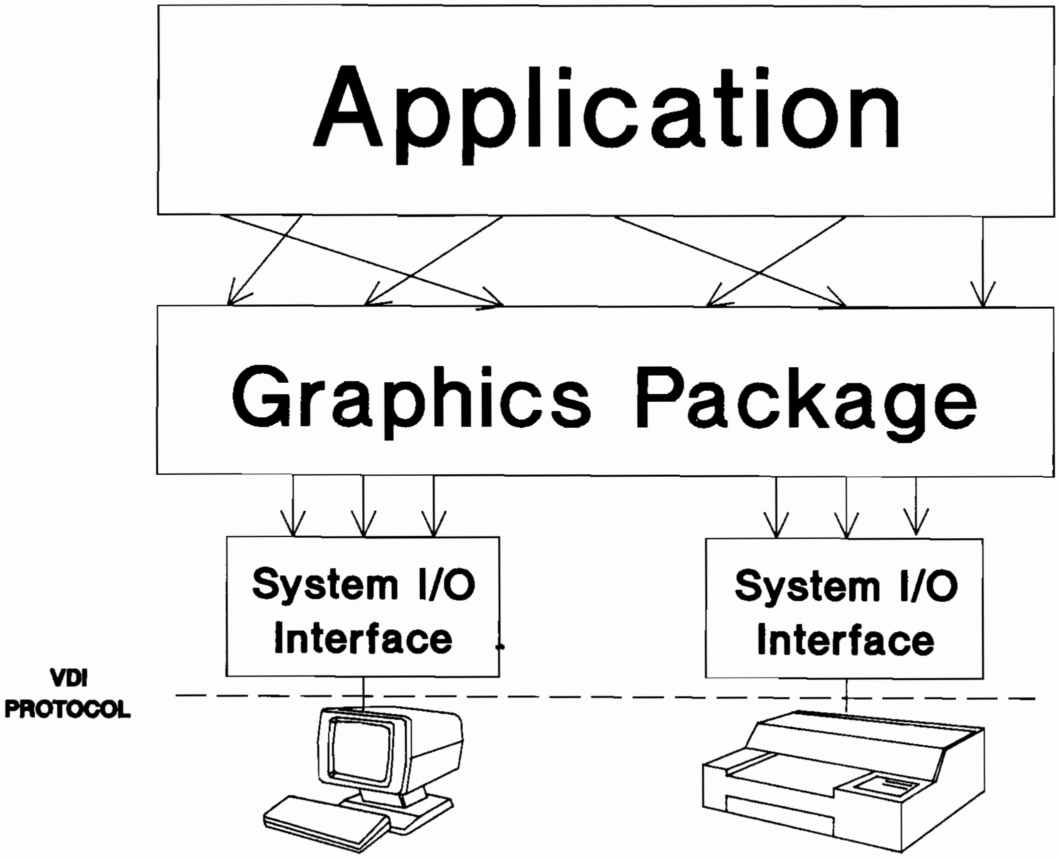


Figure 4 – Devices with internal VDI drivers

Raymond Ouellette
Infocenter
Canada

"Prototyping and Systems Development Using 4GL".

NOTE: Because of reasons out of the hand of the Host Committee, this paper will not be published in the Conference Proceedings.

C-sick?

or

How To Convert From SPL To C Without Making Waves

by

Stan Sieler

Stan Sieler has had seven years of experience on the HP3000, including over four years with Hewlett Packard in the operating system laboratory for the HP3000. He has been professionally involved with programming since 1972, and is currently a member of the Adager R & D team. He holds a BA degree in Computer Science from the University of California, San Diego.

This paper was made possible by Adager, S.A.

----

0. Introduction

For many years, SPL has been the language of choice for implementing efficient, high level programs on the HP3000.

Before I offend TOO many readers, let me mention that I am primarily referring to programming languages used for major programs that must be fast and maintainable. I realize that every HP3000 language has been used in such projects at various times, and for many people using something other than SPL is the preferred choice (for a variety of reasons, e.g.: not having any programmers who know SPL, or needing COBOL's packed decimal arithmetic).

During the early years of the HP3000, only two reliable languages existed: SPL and FORTRAN. After COBOL II was released, the number grew to three, but SPL remained preeminent. Of the three, SPL was clearly the most tightly coupled to the HP3000. That coupling, plus its ALGOL-like structure, made it the most powerful of the available languages. (That the power could be abused is true, but not relevant here.)

Eventually, other languages started appearing for the HP3000: RPG, BASIC, SPLII, APL, Pascal, FORTRAN 77, Business BASIC, and Fourth Generation Languages. But all of these had various drawbacks, and never managed to displace SPL.

However, the HP3000 is aging, compiler technology is improving, and computer architectures continue to change. As a result of this, many programmers on the HP3000 are looking for a language that will work well today, and will be available on tomorrow's computers as well. SPL clearly works very well today, but its death knell has sounded. HP has made it clear that a native mode SPL will not exist for Spectrum, the successor to the HP3000. What language, then, can we use instead of SPL to achieve the twofold goals of efficiency and portability?

Pascal immediately comes to mind. Unfortunately, Pascal was designed as a teaching language, and lacks a number of features that would qualify it as a replacement for SPL. Within HP, it is not Pascal that is being used to write their next operating system but MODCAL, a MODified version of pasCAL which overcomes most (but NOT all) of the weaknesses of Pascal. But...this language is not available to the HP3000 programmer today.

FORTRAN/77 also comes to mind, but this language has three strikes against it. First, many programmers will look at the name FORTRAN and dismiss it...this isn't fair, but it happens. Second, it is a new compiler on the HP3000 (released in late 1985) and has some reliability problems at present. Third, it isn't clear that FORTRAN/77 compilers will be available on most other machines. This third point, portability, is important for people who might want to move their programs to non-HP computers in the future.

What other languages are available on the HP3000? Until recently, the answer was: none. In 1986, two companys have announced C compilers for the HP3000.

What is C? Can it aspire to be the replacement for SPL?

To quote from the C bible (*'The C Programming Language'* by Kernighan and Ritchie), "C is a general-purpose programming language which features economy of expression, modern control flow and data structures, and a rich set of operators". While I have found the phrase about "economy" to really mean: "we hated to type much, so there is a lot of cryptic stuff in C", I do generally agree with the authors. C provides the programmer with a lot of power (and it is even easier to abuse than SPL!).

C compilers exist for most computers, ranging from the tiny 6502 microprocessor (the Apple II), to the mighty Amdahl 580. Now, C is on the HP3000.

The ease with which the C compiler, and the UNIX operating system, has been ported from the PDP-11 to many different computers says a lot for the portability of C.

The major strike against C on the HP3000 is the compiler reliability question. C is a new language on the HP3000, so we can expect some problems with the compilers initially. However, I think that the compilers will mature quickly. Unlike the FORTRAN/77 compiler, which was probably new code, I expect that both of the C compilers were ported from different machines, which would result in a compiler with less new code to debug. Additionally, I wouldn't be surprised if there is a larger number of programmers who want to use C rather than FORTRAN/77, so the C compilers will be "tested" more quickly.

For this paper I happened to use the C from CCSC. This should NOT be taken as an endorsement for CCSC's C over Tymlab's C, since I currently do not have enough information to make such a decision.

1. Introduction to C

This section is designed to introduce the reader to C, so I had better mention something about coding style in C. As a relative beginner in C, my C coding style still resembles my SPL coding style. I have noticed in the past that when I learn new languages, I often try to impose the style of an older language on the new one. This rarely works to anyone's satisfaction. It is important when using a language to try to code somewhat like the rest of the users of the language, so you can read their code, and vice versa. This shouldn't be taken to extremes...remember that the refrain "but everybody does it" does NOT necessarily make something right. One of the most important aspects of coding is to pick a style and stick with it.

Programs in C consist of one or more functions, including one special function called *main.* C refers to "function" in place of SPL's *procedure*, but the word "function" is actually never used in the language.

An example program that calculates the modulus-11 checksum for a bank account number is shown on the next page, with every line numbered as an aid to referring to them in the text. (The algorithm was extracted from the back of the V/3000 manual.)

```
#include stdio.h.ccsc
#include stdefs.h.ccsc

short
   weights     [6] = { 2, 3, 4, 5, 6, 7 };

/***********************************************************/
short calccheck11 (ptr, len);
   char *ptr;
   short len;
  {
   short
      rslt,
      winx;

   printf ("MOD11 (%s) --> ", ptr);

   winx = 0;
   ptr += len - 1;        /*point at units digit*/

   rslt = 0;
   while (--len >= 0)
      {
      rslt += (*ptr - '0') * weights[winx];
      winx = (winx + 1) % 6;
      ptr--;
      }

   if ( (rslt = 11 - (rslt % 11)) == 11)
      rslt = 0;
   if (rslt == 10)
      return (-1);
   else
      return (rslt);
   }
/***********************************************************/
main ()
  {
   char  num [10];
   short rslt;

   while (TRUE) {
      printf ("Enter 8 digit number: ");
      scanf ("%s", num);
      if (num[0] == '/')
         terminate ();

      rslt = calccheck11 (num, 8);
      if (rslt < 0)
         printf ("Unable to determine mod11 for %s\n", num);
      else
         printf ("Rslt = %d\n", rslt);
      }
   }
```

The outer block of a C program is the function *main*. This one starts at line 40 and declares two local variables: an array of characters called *num* (index from 0 to 9), and a 16 bit integer called *rslt*. Line 41 contains a *{*, which is C's version of SPL's *begin* (remember...the authors of C hated to type!).

Line 45 is the start of a *while* loop, but notice two differences from SPL here: the boolean expression is parenthesized, and there is no *do* after the boolean expression, just a *{*.

Notice the uppercase *TRUE* on line 45. Most C compilers differentiate between uppercase and lowercase, unlike SPL. This is unfortunate, as it allows variables like *i* and *I*, which are NOT the same variable.

The traditional C usage is: most things in lowercase, identifiers that are macros (defines) are in uppercase, and when variable names are composed of more than one word, make the first character of each word uppercase (e.g.: *CardCounter*). *TRUE* in line 45 was uppercase because it was a define (found in the file stdefs.h). This is the area where I will depart from the existing C conventions. In SPL I use lowercase everywhere, with procedure names being verbs and variable names being nouns. I intend to apply the same rationale to C. Thus, after this paper is finished, I will edit stdefs.h to say:

```
  #define true 1
```

instead of:

```
  #define TRUE 1
```

(Note: *TRUE* is defined as 1 in stdefs.h...this could cause some problems when interfacing with other HP3000 languages, like SPL, where *true* = -1.)

C has NO I/O defined within it, just like SPL. And, just like SPL (or ALGOL), this was a serious mistake. Luckily, the users of C have developed a relatively standard set of functions to provide I/O capabilities. Most C compilers come with these functions. For example, line 46 will probably compile and run on every C compiler. The standard *printf* function is a lot like the Pascal *write* statement. It provides a variety of formatting capabilities. Line 55 shows two of them: the *%d* in the quoted string is a signal to *printf* that it should grab the parameter after the string (*rslt*), format it as an integer, stick it into the string "Rslt...", and print the result. Since *printf* doesn't do carriage control for you (just as *write* doesn't), the *\n* is a request to *printf* to do a CR/LF at the end of the line. Thus, any *printf* whose formatting string ends with a *\n* is similar to Pascal's *writeln*.

On line 48 note two differences between SPL's *if* and C's *if*. First, there is no *then*. Second, the boolean expression is parenthesized. The lack of a *then* doesn't bother me because I have always maintained that *then* is "syntactic sugar" (something that really didn't have to exist), which is why I always code my SPL *if* statements as:

```
     if .... then
        ...
     else
        ...
```

rather than as:

```
     if ...
     then        (or then ...)
        ...
     else         (or else ...)
        ...
```

If you feel you REALLY miss the *then*, and just MUST have it, why, then, C has the answer for you! Just include the following line in your C programs:

```
#define then
```

This is equivalent to SPL's:

```
define then = #;
```

Line 48 shows one of the quirks of C, caused by people who hate to type. SPL, Pascal, and ALGOL use := as the assignment operator, and = as a comparison operator. C uses = as an assignment operator (like FORTRAN and BASIC), and == as the comparison operator. THIS CAN CAUSE A LOT OF MISTAKES FOR SPL PROGRAMMERS MOVING TO C!

Consider the following code fragment:

```
if (a = b)
   terminate ();
```

What this code does is: set *a* to the value of *b*; then, if *a* is non-0 (after the assignment), call *terminate*. The SPL programmer who glanced at the code would have said it meant: if the value of a equals the value of b, then terminate. Mistakes of this type will happen more frequently if the SPL programmer succumbs to the temptation to use the null define *then* (shown above), because then that statement in C would have looked like:

```
if (a = b) then
     terminate ();
```

Or, in other words, it would have looked so similar to SPL that no mental flags would have been raised saying "look at me closer, I am different"!

Comments in C are different, and similar to Pascal's *(** and **)*. A comment is anything following */** and before the next **/*. Most C compilers do not allow nested comments, so the best practice is to not use them.

Line 8 shows the declaration of a function, *calccheck11*, which will return a 16 bit integer as its result. Notice that the word "function" is not used. Again, if you feel more comfortable, you could do the following:

```
#define function
```

and then say:

```
short function calccheck11 (ptr,len);
```

C often shows clues to the machine of its origin. It has two interesting operators, pre-increment and post-increment, that were one word instructions on the PDP-11. Line 27 shows an example of post-decrementing. Line 23 shows an example of pre-incrementing...it says: take the value stored in len, subtract one, store that number back into len, and use it in a comparison against 0. The basic pre/post operators are:

++foo increment *foo*, use new value in expression.
foo++ increment *foo*, use prior value in expression.
--foo decrement *foo*, use new value in expression.
foo-- decrement *foo*, use prior value in expression.

As we end our very brief introduction to C, let us look at line 20. The += shows that C originated before compilers were smart enough to optimize expressions like:

```
a[i+j] = a[i+j] + k;
```

Instead, C lets the programmer do the optimization:

```
a[i+j] += k;
```

which means the same thing (barring side effects while evaluating *i* and *j*). It's also easier to type.

2. SPL to C Translation

This section discusses techniques for translating SPL to C. It has two major components: the easy part and the hard part.

2.1 Easy Automatic Translation

Much of the conversion of an existing SPL program to C can be very easily automated by your favorite editor. In the following examples I use Robelle's QEDIT, but many other editors would work as well.

Note: the following lines are numbered to facilitate referring to various lines.

```
 1. set shift down 2
 2. pq down @
 3. c "<<"/*" @
 4. c ">>"*/" @
 5. c "$include"#include"@
 6. c "procedure"(S)""   @
 7. c "integer"(S) "short" @
 8. c "byte"(S)    "char" @
 9. c "real"(S)    "float" @
10. c "logical"(S) "unsigned short" @
11. c "double"(S)  "**TEMP**"@
12. c "long"(S)    "double" @
```

```
13. c "**TEMP**"long" @
14. c ":="~"@
15. c "<="<~"@
16. c ">=">~"@
17. c "="==" @
18. c "~"=" @
19. c "<>"!="@
20. c "then"(S) "" @
21. c "do"(S) "" "while"(s)
22. c "begin"(S) "{"@
23. c "end"(S) "}" @
```

Line 1 tells QEDIT that I will be downshifting SPL style text. Option 2 says: don't change anything within double quotes (").

Line 2 downshifts all text within the source file EXCEPT quoted strings.

Lines 3 and 4 convert SPL comments to C comments. You will have to manually search for SPL comments using the new "throw away the rest of the line" comment character *!* (introduced in MPE V/E SPL). With most editors, this might change a few occurrences of "<<" and ">>" that you did not intend to change (for example, if you had "<<" in a quoted string, then you probably did not want it changed to "/*").

Line 5 changes references to SPL $include files to the C format. Note: be sure to change the include files too!

Line 7 changes SPL 16-bit integers to C 16-bit integers. With CCSC's C, *int* and *short* both mean 16-bit integer, but I chose *short* because I would guess that *int* might mean 32-bit integers on Spectrum...it is an aspect of C that is NOT well defined.

Lines 8 through 12 change the common SPL variable types to their C equivalents. Note the fancy editing in lines 11, 12 and 13. C uses the terms *double* and *long* EXACTLY backwards from SPL. Thus, the simplistic approach of saying: change *double* to *long*, and then change *long* to *double* would merely change all *double* and all *long* to *double* with no more *long*s in the file!

NOTE: changes like those in lines 10 and 12 can cause problems if the resulting line is too large for your text file.

Lines 14 through 19 effect the following changes:

SPL	:=	<>	=
C	=	!=	==

The extra trick here is to avoid changing things like <= into <== by accident. Again, these change commands could accidentally change items within quoted strings that you did not want changed.

Lines 20 and 21 get rid of the unnecessary *then*s and *do*s. Note that line 21 is qualified so that it only affects *do*s that are on the same line as *while*s, thus not touching *do/until loops*.

2.2 Easy Manual Translation

SPL *defines* can easily be converted to C *#defines*. For example:

```
define  twox   = (x + x) #, fourx  = (4 * x) #;
```

would be changed to:

```
#define twox   (x + x)
#define fourx  (4 * x)
```

Similarly, *equates* in SPL are easy to handle as *#defines* in C. Another method of handling *equates* might be chosen for some usages. For example, consider an input parser that returns the type of token found, via equates, in a variable called *iclass*:

```
equate unknownv = 0,
       tokenv    = 1,
       numberv    = 2,
       stringv    = 3;
integer iclass;          <<always is: unknownv...stringv>>
```

In C this could be changed to use *enum*, which is similar to Pascal's enumerated types:

```
enum scanner {unknownv, tokenv, numberv, stringv};
enum scanner iclass;          NOTE: not all C compilers support enum.
```

2.3 Harder Aspects Of Translation

Some of the conversion of an existing SPL program to C can be difficult. Consider some of the following constructs of SPL:

- address equation at variable declaration;
- subroutines within procedures;
- *move* statement;
- *scan* statement;
- *assemble* statement;
- register usage (*push* and *pop*);
- entry points;
- if expressions.

Some of these constructs will be very difficult to move to C, particularly *assemble*.

Although C does offer a form of address equation with the *union* statement, you may not need to use it since address equation is not used often in high-level SPL programs.

Subroutines pose a large problem. C does not allow functions within functions, which would have been a nice solution. If a short subroutine has call-by-name parameters (or call-by-value ones which are not altered), then it might be converted into a macro. Consider the following SPL example subroutine:

```
integer subroutine min (x, y);
        value   x, y;
        integer x, y;
   begin
   if x < y then
      min := x
   else
      min := y;
   end <<min sub>>;
```

It can be converted into the following macro:

```
#define min(x,y) ( (x) < (y) ? (x) : (y))
```

Then, calls to *min* will work as before.

Sometimes, a subroutine can be converted into a separate function. Ideal candidates for this are subroutines that use none of the local variables of the surrounding procedure.

If a subroutine only uses a small number of a procedure's local variables, then you might consider making it a separate function and pass those variables in as additional parameters.

The SPL *move* statement can be translated into a function call:

```
move p1 := p2, (10);
```

translates into:

```
move (p1, p2, 10);
```

This, of course, assumes that the original SPL 2move was a byte-oriented, not word-oriented. Note that you may have to write a *move* function yourself in C. Alternatively, the following macro accomplishes the same SPL move:

```
#define move(p1, p2, len) {short ktr;  for (ktr=0; ktr < len; ktr++) \
                                        p1[ktr] = p2[ktr]; }
```

(The backslash (\) indicates that the line is continued on the next input line.)

Or, as another alternative, some C compilers provide the routine *memcopy* for moving bytes:

```
#define move memcopy
```

The SPL *scan* statement can be handled in a similar manner.

There is almost no hope for the SPL *assemble* statement. But, luckily, there is almost no reason for it to be in your SPL programs anyway.

Similarly, any SPL code that explicitly accesses any hardware registers (e.g.: *Q*, *DB*, *X*, *S*, *DL*) will have to be examined by hand.

Some C compilers for the HP3000 do not support entry points. While most programmers do not use entry points, and therefore will not miss them, some do. For example, many HP programs in PUB.SYS have documented entry points (e.g.: EDITOR) that drastically affect their behavior. I have a policy of having a HELP entry point in every program I write, so that a new user only needs to say:

```
run tapedir, help
```

to get safe help information (such programs ALWAYS terminate after delivering the help information).

The C *if* statement differs in an important way from the SPL *if* (besides in appearance). In SPL, the *then* part will be executed if the boolean expression is true...but "true" is defined as "bottom bit is a 1". If C, the *then* part will be executed if the expression is non-0. This difference can drastically affect programs. Consider the following SPL code that checks to see if a variable is an odd integer:

```
if logical(k) then
```

If this were translated to C without thinking as:

```
if (k)
```

then it means: "if k is not 0". Instead, a proper C translation would be:

```
if (k & 1)
```

which does a "bitwise and" with *k* and 1, returning 0 if the value was even and 1 if the value was odd.

3. Conclusion

C is a very powerful language, and exists on many different computers. Because of this, I feel that it is an excellent candidate for replacing SPL.

I am not abandoning SPL right now, but I am going to start coding SPL with the thought in the back of my mind that someday, soon, I will be moving to C. This knowledge will provide an incentive to avoid those features of SPL that ARE hard to convert to C (or to ANY other language). Additionally, I plan to work further in C on both the HP3000 and on various microcomputers in an effort to get "up to speed" in it as fast as I can.

Remember...keep an eye on C...it may be the language in your future!

Performance and COBOL

Bruce Toback
OPT, Inc.
2205 Fulton Road
La Verne, CA 91750 USA
(714)593-1681, Telex 532678

Performance and COBOL

The first standard COBOL was proposed in 1960, when computers were very different from today's mini- and microcomputers. They were single-user, multi-priest behemoths with architectures designed to fit the electronic components available at the time. Fast memory was prohibitively expensive; slow memory was merely expensive. The processor itself required a large corporation and a large staff to purchase and maintain it.

The original COBOL standard was designed with these facts in mind, and many of the programming practices and accumulated wit and wisdom of the COBOL programmer come from those times. Presented here is a selection of that wit and wisdom, and how it relates to COBOL/3000 (and indeed most modern computers).

1. Indexing is faster than subscripting.

Remember registers? (If you ever programmed in assembly language, you certainly do. If you've only programmed in a high-level language, you probably don't.) General purpose computers usually had a limited number of these high-speed memory locations, generally eight or sixteen. The compiler, when generating code for a COBOL source program used several of these for itself, but the remainder were available to the programmer for USAGE IS INDEX items. Indexing was much faster than subscripting because in order to access an array element, a subscript would have to be brought into a register, and possibly converted to a different data type. Indexing, by definition, meant using an item which was already in a register, so bypassing the conversion and data movement steps. Worse still, some computers had addresses that could be operated on only by special instructions (e.g., the Burroughs B200/300/500 series). This is the reason that indices can only be added to or subtracted from.

On the HP3000, though, there is only one index register, andit is shared by all arrays. Therefore, as long as the subscript you are using is USAGE COMP PIC S9(4), there is no difference between indexing and subscripting - either in the generated code or the speed of the resulting program. Shops which try to speed up programs by converting them to use indexing would be better off devoting the time to programmers' vacations: system performance would then at least be improved because of a smaller program development load! However, performance can be improved by changing your subscripts from COMP-3 to COMP: the compiler emits code necessary to do the required conversion, but this is relatively expensive in run time. (But see 5.)

2. COBOL sorts take longer than external sorts.

This is very application dependent, and again has strong roots in history and folklore. In the Dark Times, the COBOL SORT verb generated in-line code to call some routines the compiler folks wrote. The compiler folks generally had better things to do than write sorts, so the sorts were not necessarily very good. In addition, these sorts were not very adaptive to circumstances, and had a limited performance range. They might involve additional overlays that had to be read in from (heaven forbid!) cards. Besides, in a batch-oriented, job-step environment there was not much point to an internal sort.

On the HP3000, though, all sorting is done by the SORT/3000 subsystem, regardless of whether you use SORT.PUB.SYS or the COBOL SORT verb: the HP3000 COBOL compilers simply emit code to call SORT/3000 intrinsics on behalf of your program. The result of this is that sorts done with the same sets of keys and in the same kinds of environments will take the same amount of time, regardless of whether you invoke them through COBOL or through MPE.

The key here is in the same kinds of environments. SORT/3000 needs memory to work: the more, the better. If your COBOL program requires 20k words of memory for itself when you execute the SORT verb, SORT/3000 will get only about 9k. Experiments show that when sorting 80-byte records, SORT/3000 needs about 8k words to produce acceptable performance, and works best with at least 16k words. So, you'll have better performance with an external sort...

Sometimes. To determine whether to use an internal sort or an external one, you should look at your entire application. Many batch-oriented systems converted to run on the HP3000 have jobsteps that include a sort to a temporary file, a report on the temp file, a different sort to a temp file, a report on the new temp file, and so on. This means that each record in your master file is being handled three times: once as input to the sort, once as output to the temp file, and once again as input from the temp file. Replacing the external sort with a SORT verb with an OUTPUT PROCEDURE reduces this to only one: records are read once by SORT/3000 on behalf of your program, and then passed to your program via the output procedure. (Of course, SORT/3000 handles records several times during its execution, but this number is largely irreducible except by providing more memory for the sort.) The results of this kind of redesign can be dramatic: reductions of 2:1 or 3:1 in runtime are possible.

3. COMPUTE is faster/slower than the ADD/SUBTRACT/MULTIPLY/DIVIDE verbs.

Like SORT, this depends on how you use the various computational verbs available to you in COBOL. If the object is to perform a complex series of arithemetic operations, using COMPUTE will generally be faster (by a few microseconds). If you are simply adding values to an accumulator, ADD and COMPUTE will generate exactly the same code, and so there will be no performance difference.

4. COBOL is inefficient at arithmetic.

As the previous point shows, how "efficient" a language is at a particular task depends mostly on how you use it. (This generalization applies only to general-purpose languages.) By following a few simple rules, COBOL is as efficient as, say, FORTRAN at arithmetic. In particular, avoid either implicit or explicit type conversions: don't mix COMP-3 and COMP items, and try not to mix COMP PIC S9(1-4) with COMP PIC S9(5-9). And in addition, avoid arithmetic with USAGE DISPLAY items, since these always require type conversions. (The HP3000 has instructions to operate directly on COMP and COMP-3 items, but lacks arithmetic instructions to operate on DISPLAY types.) For an extended discussion of this, see Jim May's paper Programming for Performance , published in the Proceedings of the 1982 HP3000 IUG Conference, Edinburgh.

5. Searching: IMAGE and internal techniques.

earches, or table lookups, are very common in data processing operations. You probably do them without thinking in most cases, since every IMAGE master lookup (DBGET mode 7, or DBFIND) is really a search operation. If you routinely use IMAGE to do your table lookups for you, you might be surprised at how much time can be saved by using your own code to perform the same operation.

The examples used for this article are derived from a real-life applicaiton: printing a purchase order report. Purchase order records were contained in a detail data set containing four items: a part number, a vendor number, a purchase date, and a quantity. (Other items were left out for the purpose of this example.) Each part number's corresponding manual master record contains a description for it, and each vendor number's corresponding manual master record contains the vendor's name and address. To test the methods outlined here, I wrote a small COBOL-II program to generate a simple purchase order report, and then modified it to try different search methods.

In the first example, the item being looked up was the vendor's name. The program in Figure 1 is representative of most such programs: a serial read (or perhaps, a sort output procedure) gets each detail record, and then a lookup is performed on the associated manual master to get descriptive information. The lookup is entirely contained in the paragraph **GET-VENDOR,** so that various lookup techniques can be tried without making significant changes in the rest of the program. (If your programs are written like this, you should be able to simply lift code from the examples, place it in your system, and enjoy the kudos.)

All of the examples, and the performance information derived from them, comes from a 1/2-megabyte Series 30 running T-delta-1 (MPE-V/T) and a single 7925 disc drive. You should keep this in mind when you examine the performance data. If you are running a faster system (you couldn't *possibly* be running a slower one), you should divide the "CPU-seconds" figures by 2 if you are on a Series III or Series 37; by 5 on a Series 4x CPU; or by 12 (!) if you are on a Series 6x machine. Clock times will not scale by as much, since non-cached I/O rates for a single disc drive are almost identical on all CPU's. In addition, all timing was done with disc caching turned off. Caching will in most cases improve wall-time performance, but leave CPU time almost unchanged. (Disc caching resulted in a slight performance degradation on this small-memory Series 30.)

The first test was run using the "standard" program in Figure 1. To produce this 9,000-line report took just under thirteen minutes, and used 475 CPU seconds. (If you are scaling these numbers for your Series 68 CPU, you should come out with about 5:45 clock time, and 40 CPU seconds.) Is any improvement possible? Since by now you have looked at the graph in Figure 5, you know the answer is "yes." The program in Figure 2 replaces the DBGET with a SEARCH verb, which performs a serial search of a table in memory. Of course, some preparation is required for this technique, and this is done in the paragraph **INIT-VENDOR-SEARCH. INIT-VENDOR-SEARCH** does a simple serial read of VENDOR-MASTER, and saves the vendor number and vendor name in VENDOR-TABLE. The result of running Program 2 is almost a 2:1 reduction in run-time, and savings of about one-third in CPU time. Clearly the extra effort of creating a table in memory was well-spent.

COBOL, though, supports an even faster search verb: SEARCH ALL. Using SEARCH ALL requires that the table used for the search be sorted into ascending or descending order. This, of course, requires some additional preparation, shown in Figure 3. SORT-VENDOR-TABLE is a generalized Shell sort, and you should be able to use this in any of your programs by changing only the identifiers used for the table. For sorting tables that fit entirely in memory, it is *much* faster than the SORT verb. (The sort would execute faster still if it were programmed in SPL, but the difference for small tables is not worth the extra effort.)

The result of using SEARCH ALL is a further improvement, although not as dramatic a change as eliminating IMAGE from the picture. The ratio will improve noticeably, though, with larger tables, as you will see shortly.

There is a search method even faster than binary search, however, for most commercial data. The "80-20" rule is a generalization about most things in business: twenty percent of the customers generate eighty percent of the revenue (or orders, or complaints); twenty percent of your vendors are responsible for eighty percent of your shortages, and so on. (As practical examples, think about the number of transactions in your accounting system that use the "accounts payable" or "inventory" accounts.) Because your computer records will (should) reflect the parameters in your business, you will probably discover that eigty percent of the records in a detail are linked to only twenty percent of your master entries. (Of course, for pure "control" information such as invoice numbers, this is not true.) This will probably apply to parts in inventory, to purchase order line items, to sales order items, and other details requiring "descriptive" information from associated masters.

You can use this property of business data to come up with a new search rule for your memory array: Whenever an item is found in the table, it is moved up one entry. Repeated application of this rule rapidly and automatically organizes your table by frequency of use. The resulting program is shown in Figure 4; the very simple change is in paragraph **GET-VENDOR.** About 20 percent less clock time is required now, and about about 15 percent less CPU time as well. (The test data used for these timings was designed to have approximately an 80-20 distribution.)

To summarize the information gathered so far, look at Figure 5. (You've probably already done this, but look again.) Clearly, the "standard" method is the worst of the lot. Given the small effort involved in adding *any* of these internal searches, they are probably an easy way to gain performance. The small amount of "overhead" time in each bar is the time required to do any pre-processing before the report starts. In program 1, the time is spent only in opening the data base and output files. In programs 2 and 4, there is additional time required to read the master data set, and in program 3, some additional time is required to sort the array. The overhead time is very low for all three "fast" methods.

These techniques will work on small master data sets (a few hundred entries or so), but what about masters too large to fit into the stack? (Unless you are sorting using a separate process, you should leave at least 12K words for any required sort.) In the case of the vendor file, the maximum would be about 1100 entries. In the case of the part master, however, the maximum would be only about 650 entries. (In a practical program, there would be even fewer entries, since these small test programs make no allowance for V/3000 space, extra files open, or other uses of the stack. Moreover, there would probably be several manual masters on which lookups were to be performed.) A modification of the technique used in Program 4 can still be used as long as the master data set or sets are not extremely large.

In this case, rather than reading the entire master data set into memory at once, entries are read in "on demand." No initial "preload" of the array is performed. Instead, the table size is set to zero during initialization, and as each detail record is read, a table search is performed. If the table search fails, the program looks up the key in the appropriate master data set, and the newly read record is added to the growing array. (If the requested record *is* found in the array, its position is adjusted as in Program 4.) When the array becomes full, there are two choices. First, the entry at the bottom of the array can be thrown away to make room for the new entry. This method will efficiently handle cases in which detail records with a particular key are "clustered" - newly-read records will tend to migrate to the top of the array as their "cluster" is read.

6. A perspective on performance.

With the exception of searching and sorting, most of the performance questions analyzed here, and most performance debates in general are matters of microseconds on modern computers. You should take this into account before embarking on any large rewrite projects. For example, changing a subscript from COMP-3 to COMP may save 50 microseconds (millionths of a second) per array access. If you access the table in question ten million times over the course of a three-hour run, your rewrite will save 500 seconds, or about eight minutes out of 180 - a 4-1/2 percent improvement.

Conversely, if you are doing an external sort followed by a report, for a total run-time of three hours, changing the application to run with a single internal sort might save an hour or more, a savings which would justify a two-day redesign. (Assuming that the report is run more often than once a year.)

Finally, performance is much more a matter of choosing the right design in the first place, rather than "tweaking" a poor design to get more speed out of it. As the sorting example shows, shaving 20 percent off the run-time of a bad design will usually pay a lot less than finding a good design to start with. For an excellent discussion of this, see The Elements of Programming Style, Kernighan and Plauger (1974), Chapter 6.

SOFTWARE DEVELOPMENT IN TRANSITION:

USING TRANSACT / 3000

by Norm Wright

Summary - The paper presents a comparative study of programmer productivity on several recent development efforts on the HP3000. The development projects are good subjects for comparison, since the software effort was of a similar size and scope, and each used a dedicated staff with approximately equal capabilities. They also involved the same customer and demanded that the end product be comparable from the point of view of the user. Two of the projects involved Cobol, Image, and VPlus; the third used Rapid/3000, including Transact, almost exclusively. Statistics are presented which delineate the modest productivity savings which were achieved. Additional improvements which were achieved in user presentation are mentioned, along with a brief discussion of the expected impact of Transact on program maintenance. Additional discussion will focus on the the experiences of the development staff in adapting to Transact/3000. Changes in concepts, procedures, and strategies of Cobol programmers using the new dictionary-based development methodologies are discussed.

For the past three years, our organization has been involved in a number of medium to large-scale software development efforts, all on Hewlett Packard 3000 equipment. Among three recent projects, two involved the conventional HP3000 development triad: Cobol, Image, VPlus. A third major project launched boldly into the fourth generation, utilizing HP's most recent software development tools: Dictionary/3000 and Transact, with its built-in access bridge to VPlus and Image. The discussion which follows is based upon perceptions which evolved from this succession of projects.

For most users of HP3000 equipment, there are several initial deterrents to undertaking major software development effort in Rapid/3000. One deterrent may be the high performance overheads which are reportedly experienced in production software using Transact. Almost all users who have gotten beyond casual or experimental use of Transact sooner or later will experience this problem -- this is true at least with the present generation of HP equipment. Fortunately for those users, the well-known problem has an equally well-know solution. The solution comes in the form of a well-known Transact optimizing compiler which has become a standard remedy for users who are experiencing performance headaches.

A second major deterrent may be the high cost of the Rapid/3000 package. For many small organizations, especially those in competitive markets, the cost of development software itself may seem prohibitive. Only if the user organization is large and spends a considerable amount on in-house software development is the cost of Dictionary/3000 likely to be justifiable. If the differential can reasonably be expected to be absorbed by the reduced development or maintenance costs of new systems, the software becomes cost justifiable.

Unfortunately, the subject of developmental time and cost savings has seldom been reasonably addressed. Vendor claims for the economies which can be expected from the implementation of fourth generation methodologies tend to be moderately to wildly exaggerated. Claims ranging from 100 to 1000 per cent leave the prospective user of fourth generation languages feeling like the incredible shrinking man. Above all, such vendor claims seem rarely based upon realistic and reasonable assessments of practical user situations. The discussion which follows is an attempt to realistically measure the impact of Transact on a real life software development process.

Table 1 presents the historical experience of our organization in software development with three recent projects. The columns C1 and C2 both represent Cobol-Image-VPlus software development, while the column T1 represents the development effort which utilized Transact almost exclusively. The three projects offer a variety of built-in advantages for comparison. All were carried out with a similar staff makeup, comprising a range of programming skills and experience levels. Unique to the working environment, all three development efforts were carried on with "captive" staff -- the personnel were dedicated completely to the project at hand and had no other professional duties which were outside the area. All three projects were designed and implemented for the same client, hence all three attempted to hit a level of sophistication, user expectation, and documentation that was approximately the same.

The development time figures are expressed in man months, which have been adjusted for overtime and holidays to reflect a work week of approximately 40 hours. The figures shown represent the total development, beginning with the systems analysis and database design. The time overheads involved in loading the data dictionary are included for the Transact code, as well as the time for building VPlus forms in both Cobol and Transact. Program design, coding, and debugging times are included, as well as the time necessary to integrate and test the system prior to implementation. The man-month figures also include the time involved in producing reasonably complete user, program maintenance, and operations documentation for each system. They do not include the initial project definition and functional specification phases.

Measures of programmer productivity which have been applied in the past frequently relied upon the total number of lines of code produced. Accordingly, the table presents the total number of lines of code comprising each system, broken down for comparison of interactive versus batch programs. Seperate categories show the size of associated copy libraries, include modules, and other supporting code.

		System C1	System C2	System T1
1.	Interactive Programs	34	37	64
2.	# Lines Code, Interactive	26,720	37,714	60,529
3.	Average Prog Size Interactive	786 lines	1019 lines	946 lines
4.	Batch Programs	58	35	56
5	# Lines Code, Batch	40,047	28,022	29,046/48 *
6.	Average Prog Size Batch	690 lines	801 lines	605 lines
7.	Total Programs	92	72	120
8.	# Lines Misc Code: Copylib/Include/etc.	4,224	4,388	3,758
9.	Total Lines	70,991	70,124	93,333/112 *
10.	Average Prog Size	772 lines	974 lines	833 lines
11.	Total Man Months	74	52.5	60
12.	Man Months per Program Module	.80	.73	.50

* Eight batch COBOL reporting programs in system T1 averaged 904 lines each.

- TABLE 1 -

SOFTWARE DEVELOPMENT STATISTICS

FROM THREE SYSTEMS DEVELOPMENT PROJECTS

No adjustment is made for the Transact system to attempt to capture the size and coding involved in the data dictionary itself. The figures for the Transact system also do not include a significant (and growing) number of Report and Inform auxilliary programs.

In comparing Transact with Cobol, one would intuitively feel that a measure involving the total absolute number of lines of code would be inadequate. After all, one of the most well-publicized features of fourth generation languages is their syntactic brevity, and the ability to condense complex programming structures into a short sequence of statements. But this reputation for brevity is not borne out by the absolute number of lines of code produced in these three projects. Instead, Table 1 shows as much variability between the two Cobol systems in terms of the number of lines of code per program, as it does between the Cobol and Transact systems. Of course, had the statistics extended to the level of capturing the number of words or characters produced, then it is very possible that the fourth generation's good reputation could have been restored in this respect.

But even though the figures do not show a consistent differential between Transact and Cobol in the number of lines of code produced, it seems wise not to base any measure of productivity upon such a measure. A better measure of productivity would make some attempt to quantify the program complexity. In particular, it would seem relevant to try to measure the functionality of the programs produced -- the overall "work" or "complexity" of the resulting system. Ideally one would like to know how much time would be saved in writing exactly identical systems -- one using each development methodology.

Fortunately, in the case at hand, there is no need to attempt to develop a new programmer or productivity measurement. There is a much easier way, offered by the fact that all three systems were produced at a similar level of design sophistication. While the three systems are not identical -- in fact they are quite different applications -- they do represent perhaps the closest it is possible to come to this condition in a real world setting.

Since all three systems are similar from the standpoint of both design and user presentation, we can take the program itself as a functional unit of measurement. This is to say that, roughly speaking, given the three systems presented here, a single program from any system behaves in the same way and performs the same functional "work". Of course some programs will be more complex, accessing a large number of datasets and performing many functions. But overall in the three systems shown here the total "functionality" of the system is roughly proportional to the number of programs in the system. Such a measure would, of course, be inappropriate for systems which were designed to different standards of either functionality or modularity. But the three systems under consideration offer somewhat of an ideal case in this regard.

Using the program itself as a measure of functionality, the productivity measure which is listed last in the attached table can be derived, the figure "Man Months per Program". This measure can be used to give a close approximation of the overall time savings, experienced in these projects, of Transact over Cobol. As the table shows, programmers on the Cobol projects required between .7 and .8 man months to produce each functional program module. The time for a similar module using Transact was .5 man months. This would suggest that productivity improvements in the range of 28% to 38% were experienced.

The reader will note that the variability between the two Cobol projects (C1 and C2) is one third as high as between Cobol and Transact. In fact, the project C1 involved high startup costs in terms of a team of programmers getting oriented toward a new client and a new environment. The time savings which were experienced in C2 are more the result of a team pursuing established methods, and capitalizing on working tools (program shells, drivers, database routines, copy libraries, etc.) established during the first project. To a certain extent, some of these established methods were also useful in T1 -- the Transact project -- in that they did at least provide an existing standard which had only to be adapted to the new programming methodology. Although the startup overheads of learning and adapting to Transact were considerable, especially in the first few months of the project (T1), the project was also facilitated in many phases by being the third in an established series. Overall, it is at least convenient to assume that the overhead of learning Transact in T1 was counterbalanced by the beneficial effects of the pre-existing models in Cobol.

An additional factor is not satisfactorily reflected in the statistical presentation in the table. The use of Transact in T1 facilitated a number of significant improvements, particularly in the area of user friendliness and ease of use. As two examples, the increased use of sorted record presentations and the increased capabilities of browse-type record accesses in T1 could be cited. Both types of presentation are vastly easier in Transact than in Cobol. The programmatic ease of such capabilities in Transact meant that many functions in the T1 system utilized the features, where the nearest Cobol equivalent did not. Overall this did lead to a significant improvment in user presentation for the T1 system users over that provided by the two Cobol systems. Our installation does not appear to be unique in this experience. The increased sophistication of Transact code from the user point of view has been mentioned by a number of other users of Transact.

Another very interesting topic which we will not attempt to cover here is the matter of improvements in program maintenance to be expected from Rapid/3000. To some extent, program maintenance is a continuation of the same processes which were going on in development. Programs are being enhanced and extended, screens are being changed and rearranged, and the database must also occasionally be revised and restructured. This would suggest that as a minimum, we could expect the same economies as during the development process.

Our own preliminary experiences with maintenance of a Rapid/3000 system would suggest that this is true. In addition, it would be wrong not to add that the capabilities of Report and Inform have also proven to be tremendous advantages. The defacto and specialised reporting capabilities are especially useful in reducing the backlog of maintenance activities. This allows the maintenance staff to be in a better position to react to the more complex and far-reaching requirements of users. This is the reverse of the maintenance predicament at most installations, where the steady background of minor changes and request overwhelms the staff resources long before more complex changes can even be considered.

Well, the productivity gains which were experienced during the development phase of our project were modest, especially in view of the elaborate claims which have been made for some fourth generation languages. Perhaps it is important at this point not to forget that productivity gains in excess of 25% in almost any industry are bound to have important, far reaching impacts. Above all, it is probably wrong to expect the entire process of computer analysis and software design to vanish into nothing at this particular stage of its development. Whether it is the fourth generation or natural language interfaces, important problems still remain in the formulation of complex, real-world problems into computer algorithms and processes.

Norm Wright - has worked on a variety of software systems for HP3000 computers since 1975. He has spoken at previous Interex conferences in Baltimore, Orlando, and Copenhagen. He is currently associated with INFORMICA, an independent software firm, doing business in the Kingdom of Saudi Arabia.

Interex HP3000 Madrid Conference, March 1986
Theme: 'Migration to 2001'

. . . universal software in 2001 . . . some UNIphiles would say that the year 2001 will see one world, under UNIX™, indivisible, with liberty and c shells for all . . . here's a light-weight introduction to conversational UNIX™ so you can get food, directions and maybe lodgings for the night in the Migration to 2001 . . . .

UNIX* thru the Eyes of MPE

Sam Boles, Member Technical Staff

HEWLETT PACKARD

With UNIX™ evolving as the de facto "industry standard" operating system, Hewlett-Packard now includes this important dimension in its array of computer technology. The HP9000 computer family currently supports HP-UX, a powerful dialect of UNIX™, with more under development.

Here you can get a view of UNIX™ thru the eyes of MPE. The friendly vernacular of MPE -- second language to HP3000 users around the world -- becomes the familiar basis in terms of which those new to UNIX™ can acquaint themselves with the terse power of this operating system.

Starting with fundamentals that map one-to-one, you'll see some MPE UDC's used with HP summer students to accelerate their productivity by providing a transitional mechanism from their UNIX™ background. From there you'll move into the more complex facilities that the UNIX™ productivity engine gives to both programmers and end-users alike.

***. . . since 1969 . . .
a little UNIX™
lore, mystique
and
culture . . . .***

Archeological evidence of UNIX™ dates back to 1969. For the children among us, that's the year one of the leaders in the computer industry "***unbundled***" in recognition of the fact that software was no longer just the packing that came free with the hardware to keep it from rattling around in the carton. Anything that's survived in this business since 1969 has some ***lore, mystique and culture***. UNIX™ is no exception.

First of all, it's an operating system ***of the programmers, for the programmers, by the programmers***. That's why we ***byte hacks love it***. And that's why ***civilized people spend large sums of money for*** commercial ***shells to cover*** the lean terse power of UNIX™.

**UNIX™ is a trademark of AT&T Bell Laboratories.*

Some say it's ***unfriendly***. But you have to remember that one man's "friendly" may be another man's "verbose."

It's the ***classic controversy of efficiency vs friendliness***. Do you have an elaborate high-overhead ritual to establish friendliness or do you get right to the point? Do you ask for a confirmation that the user really wants to purge every file in his group, or do you assume that if he's smart enough to ask for it you'll do it for/to him?

You can look at it this way: a lot of development nodes save trees by running OUTFENCE high, going to hardcopy on only a small portion of printer output. That means to get hardcopy you need to enter something like

```
ALTSPOOLFILE #O936;PRI=10
```

Now that's ***mnemonic and intuitive***, probably. It's maybe what you'd call ***friendly***. But about the third time you do it, you decide it's worth the trouble of updating your UDC's with something like

```
ASF SPL=0,PRI=1,COPIES=1
OPTION LIST
ALTSPOOLFILE #O!SPL;PRI=!PRI;COPIES=!COPIES
```

that you invoke with something like

```
ASF 936 10
```

That's the UNIX™ style.

It's ***terse***.

Crisp.

All right, ***cryptic***.

Powerful.

A rich repertoire of commands and options to do the kinds of things programmers do to build and document software.

. . . maybe something of a misnomer: today, there's little UNI in UNIX™ beyond UNIfying . . . .

UNIX™ may be a ***misnomer***. Legend has it that when the brilliant Ken Thompson named his brilliant child, he did it in counterpoint to the multi-tasking multi-user MULTICS at MIT. His was a single-user system for a single-engineer work station. Today there is ***little UNI*** in UNIX™ beyond the fact that it may be the single most ***UNI***fying element across the wide variety of hardware architectures and configurations in the industry today. Beyond that great ***UNI***fying attribute and signal contribution, UNIX™ is ***MULTItasking, MULTI-user, MULTI-noded, MULTI-shelled, even MULTI-processed*** on the HP9000/500 with its tri-CPU design.

One aspect of the "non-UNI" of UNIX™ is its ***multiplicity of dialects***. This is probably good and bad at the same time. Like any worthwhile software system, UNIX™ is evolving. It's inevitable that such a magnificent theme have myriad variations. The Bell Labs UNIX™, perhaps the seat of orthodoxy, has a System III and a System V. There's the Berkeley 4.1. And HP-UX, XENIX, VENIX, QNX, UNI-plus, -star, -plex and, of course, the powerful NIX of the anti-UNIX™ clingons.

Now we can't hold back tomorrow. We don't even want to. Just ***don't be deluded*** into thinking that the UNIX™ "industry standard" is going to get you entirely out of the ***technological retread business*** we've been confronted with for generations (computer, that is), with all its learning curve entropy and proactive inhibitions.

As Churchill once said about Democracy: ***It's not perfect by any means; it's just the best we've been able to come up with***. The same assessment might apply to UNIX™.

... UNIX™ and MPE side by side, going thru a few ordinary everyday commands ....

First a few words about the examples you see here. We all know the frustration of the ***example that doesn't work***. One way to reduce that problem is to capture the example right at the screen ***in vivo***.

Now to do this for the MPE part is a fairly straightforward exercise if you have an old 2647 like mine. You work the example on the terminal on-line to the computer, then position the cursor at the start of the example. You go into local command mode to do a

```
COPY ALL FROM DISPLAY TO LEFT TAPE
```

When you've gotten the latest batch of examples on tape, you do a

```
MARK LEFT TAPE

REWIND LEFT TAPE
```

Then you get into TDP (Text & Document Processor), find the place where you want to splice in the example, do an

```
/A nnn.nn
```

Then when you get the line number prompt, touch the READ key to get the cartridge tape contents spliced into your text file.

Getting the UNIX™ examples is a little different.

If you're using an HP9000/520, you've got an integrated 5" floppy disc drive. As you do the examples you precede the example with an `echo` or a `cat >>` to the disc file where you're collecting your actual examples. For example, for the `ps` (process show) command:

```
echo $ ps -e >> seb
```

(The `>>` means to append or concatenate the string after `echo` to the target file `seb`.) Then you actually do the command but redirect the output to the same file:

```
ps -e >> seb
```

This gives you what would have been on the screen in your disc file. Then, if you haven't bothered to engineer any better datacom, you can

```
lifcp seb /dev/rfd:SEB
```

to get your examples onto a floppy in LIF (Logical Interchange Format), take it over to your HP9000/236, do a virtual terminal file transfer to the HP3000 where the main body of your text is for doing your laser typesetting via TDP, and join the examples into the appropriate spot.

In the examples you see the HP-UX form, then the MPE form with an HP-UX-like UDC. In the UDC there's an `option list` to show you how the UDC gets expanded and executed. Imagine a Carriage/Cursor Return at the end of each line unless specified otherwise. If there's a ***Control-D*** you'll see `[ctl-D]`.

So much for the logistics. Let's get started.

First, get on the system:

HP-UX:

```
login: boles

Welcome to Hewlett-Packard System 9000 HP-UX
```

MPE:

```
:hello boles.cad
HP3000 / MPE V  G.B0.00 (BASE G.B0.00).  MON, DEC 24, 1984,  4:24 PM
```

Accounting in HP-UX is done generally at the ***user*** level, as opposed to ***user.account*** in MPE. There are some other differences, too. For example, if you have a password and key it wrong, MPE asks you several times to try again; HP-UX doesn't tell you whether it's the user or the password or a backspace that's the problem, but asks for everything again.

HP-UX:

```
$ who am i
boles    console Dec 24 13:26
```

MPE:

```
:whoami
SHOWME
USER: #S85,BOLES.CAD,UNIX      (NOT IN BREAK)
MPE VERSION: HP32033G.B0.00.  (BASE G.B0.00).
CURRENT: MON, DEC 24, 1984,  4:26 PM
LOGON:   MON, DEC 24, 1984,  4:24 PM
CPU SECONDS: 5          CONNECT MINUTES: 2
$STDIN LDEV: 22          $STDLIST LDEV: 22
```

Notice the blanks are suppressed in the UDC to get the `showme`.

HP-UX:

```
$ date
Mon Dec 24 13:48:48 PST 1984
```

MPE:

```
:date
SHOWTIME
MON, DEC 24, 1984,  4:26 PM
```

Basically the same but a little more time granularity in HP-UX and a GMT (Greenwich Mean Time) basis.

HP-UX:

```
$ ps -e
   PID TTY  TIME COMMAND
 27335  co  0:00 ps
 27279  co  0:04 sh
    35   ?  0:01 getty
    34  a2  0:02 getty
    33  a1  0:01 getty
    32  a0  0:01 getty
     1   ?  0:01 init
```

MPE:

```
:pse
SHOWJOB

JOBNUM  STATE IPRI JIN  JLIST    INTRODUCED  JOB NAME

#S69    EXEC        20  20       FRI  8:54A  OPERATOR.SYS
#S85    EXEC        22  22       MON  4:24P  BOLES.CAD

2 JOBS:
    0 INTRO
    0 WAIT; INCL 0 DEFERRED
    2 EXEC; INCL 2 SESSIONS
    0 SUSP
JOBFENCE= 0; JLIMIT= 5; SLIMIT= 60
```

The `ps`, like `showjob`, tells you what's running in the system. Some differences are cosmetic: syntax, format, nomenclature; but CPU consumption, state and start time are all useful but not available in both systems with these comparable commands.

HP-UX:

```
$ ls
seb
sebb
```

MPE:

```
:ls
LISTF @

FILENAME

SEBUNX
```

Again, mostly cosmetic differences.

HP-UX:

```
$ ll
total 3
-rw-rw-rw-   1 boles     101          370 Dec 24 13:51 seb
-rw-rw-rw-   1 boles     101          100 Dec 24 13:48 sebb
-rw-rw-rw-   1 boles     101          365 Dec 24 13:51 sebc
```

MPE:

```
:ll
LISTF @,2
ACCOUNT=  CAD          GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                 SIZE  TYP        EOF       LIMIT R/B  SECTORS #X MX

SEBUNX   *        72B  FA          48          48   7       16  1  1
```

The HP-UX "long" file list gives security, date and owner. The ***-rw-rw-rw-*** means it's an ordinary data file (not a directory nor a device special file), the owner of the file has read and write access but not execute permission, as do the owner's group and the public in general. The listing also includes number of directory links, owner, group code, size in bytes, date and time of last modification.

This touches on a major difference: the file system. The UNIX™ directory structure and file concepts are a major transitional consideration, and beyond the scope of this paper. You get glimpses here in the links information and in the `mkdir` ***and*** `cd` ***examples below. But remember this is only the tip — there's a real iceberg there.***

HP-UX:

```
$ cat
This is to show cat with no parms.
This is to show cat with no parms.
This is line 2 of show cat.
This is line 2 of show cat.
[ctl-D]
```

MPE:

```
:cat
FCOPY FROM=;TO=
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983


This is to show cat with no parms.
This is to show cat with no parms.
This is line 2 of show cat.
This is line 2 of show cat.
 < CONTROL Y >

2 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

This is the `cat` (for con**cat**enate) form without parameters. It's basically input from $STDIN and output to $STDLIST -- the CRT in this case.

HP-UX:

```
$ cat > file1
This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
[ctl-D]
```

MPE:

```
:catt file1
FCOPY FROM=;TO=file1;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
 < CONTROL Y >

3 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

This is the concatenation of CRT input to a new or replaced file on disc. An easy way to build a file without getting into the editor -- but you give up the more powerful edits. Notice the >. That's UNIX™ redirection from the default CRT to the named file. Be careful: UNIX™ has high regard for your presence of mind. If it finds a file out there already by that name, it doesn't ask as MPE does whether you're sure you want to purge it (unless you've removed the write permission with a `chmod`) -- it just writes over the old file.

HP-UX:

```
$ cat file1
This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
```

MPE:

```
:catf file1
FCOPY FROM=file1;TO=
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983


This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
EOF FOUND IN FROMFILE AFTER RECORD 2

3 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

Here with an implicit < redirection of input, we concatenate from the named file to the CRT.

HP-UX:

```
$ cp file1 file2
$ ll file*
-rw-rw-rw-   1 boles    101          121 Dec 24 14:01 file1
-rw-rw-rw-   1 boles    101          121 Dec 25 20:44 file2
$ cat file2
This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
```

MPE:

```
:cp file1 file2
FCOPY FROM=file1;TO=file2;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

EOF FOUND IN FROMFILE AFTER RECORD 2

3 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:ll file@
LISTF file@,2
ACCOUNT=  CAD         GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FILE1              80B  FA           3       1023   1      128  1  8
FILE2              80B  FA           3       1023   1      128  1  8


:catf file2
FCOPY FROM=file2;TO=
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983


This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
EOF FOUND IN FROMFILE AFTER RECORD 2

3 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

This is a simple file copy with no changes as you can see from the `ll` and `cat` listings. Note the wild card * that gives you all files starting with "file".

HP-UX:

```
$ cat >> file2
This is some more text to illustrate the
concatenation facility of UNIX in this game
of "Follow the Leader" with MPE.
[ctl-D]
```

MPE:

```
:cattt file2
FILE file2,OLD;ACC=APPEND
FCOPY FROM=;TO=*file2
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

This is some more text to illustrate the
concatenation facility of UNIX in this game
of "Follow the Leader" with MPE.
 < CONTROL Y >

3 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:catf file2
FCOPY FROM=file2;TO=
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983


This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
This is some more text to illustrate the
concatenation facility of UNIX in this game
of "Follow the Leader" with MPE.
EOF FOUND IN FROMFILE AFTER RECORD 5

6 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

Here you see a concatenation with the append redirection instead of the replace. Control-D signals End of Data.

HP-UX:

```
$ cp file2 file3
$ cp file2 file3b
$ cp file2 file3c
$ ll file3*
-rw-rw-rw-    1 boles    101          247 Dec 25 20:55 file3
-rw-rw-rw-    1 boles    101          247 Dec 25 20:55 file3b
-rw-rw-rw-    1 boles    101          247 Dec 25 20:55 file3c
$ rm file3*
$ ll file3*
file3* not found
```

MPE:

```
:cp file2 file3
FCOPY FROM=file2;TO=file3;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

EOF FOUND IN FROMFILE AFTER RECORD 5

6 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:cp file2 file3b
FCOPY FROM=file2;TO=file3b;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

EOF FOUND IN FROMFILE AFTER RECORD 5

6 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:cp file2 file3c
FCOPY FROM=file2;TO=file3c;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

EOF FOUND IN FROMFILE AFTER RECORD 5

6 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:ll file3@
LISTF file3@,2
ACCOUNT=  CAD         GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FILE3              80B  FA           6       1023   1      128  1  8
FILE3B             80B  FA           6       1023   1      128  1  8
FILE3C             80B  FA           6       1023   1      128  1  8
:rm file3
PURGE file3
:rm file3b
PURGE file3b
:rm file3c
PURGE file3c
:ll file3@
LISTF file3@,2
NO FILES FOUND IN FILE-SET (CIWARN 431)
```

Note here the generic purge, representative of the UNIX™ respect for the programmer's presence of mind. (Some of us who only marginally deserve that respect do a lot more back-ups under UNIX™.) You can protect yourself on sensitive files by removing write permission and thereby getting UNIX™ to prompt for confirmation of purge.

HP-UX:

```
$ ll file1
-rw-rw-rw-   1 boles    101          121 Dec 24 14:01 file1
$ chmod 600 file1
-rw-------   1 boles    101          121 Dec 24 14:01 file1
```

MPE:

```
:ll file1
LISTF file1,2
ACCOUNT=  CAD         GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FILE1              80B  FA           3       1023   1       128  1  8


:chmod600 file1
SECURE file1
```

Here's an example of changing access permissions on a file. We don't have a simple parallel in MPE. This now disallows group and public users to access the file. Note that the UNIX™ granularity of control could be approximated by a combination of the `secure` you see here and the `altgroup xxx; access=` facilities in MPE.

HP-UX:

```
$ cp file2 file3
$ ll file*
-rw-------   1 boles    101          121 Dec 24 14:01 file1
-rw-rw-rw-   1 boles    101          247 Dec 25 20:53 file2
-rw-rw-rw-   1 boles    101          247 Dec 25 21:06 file3
$ mv file3 file4
$ ll file*
-rw-------   1 boles    101          121 Dec 24 14:01 file1
-rw-rw-rw-   1 boles    101          247 Dec 25 20:53 file2
-rw-rw-rw-   1 boles    101          247 Dec 25 21:06 file4
```

MPE:

```
:cp file2 file3
FCOPY FROM=file2;TO=file3;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

EOF FOUND IN FROMFILE AFTER RECORD 5

6 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

```
:ll file@
LISTF file@,2
ACCOUNT=  CAD         GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP         EOF       LIMIT R/B  SECTORS #X MX

FILE1              80B  FA            3        1023   1       128  1  8
FILE2              80B  FA            6        1023   1       128  1  8
FILE3              80B  FA            6        1023   1       128  1  8


:mv file3 file4
RENAME file3,file4
:ll file@
LISTF file@,2
ACCOUNT=  CAD         GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP         EOF       LIMIT R/B  SECTORS #X MX

FILE1              80B  FA            3        1023   1       128  1  8
FILE2              80B  FA            6        1023   1       128  1  8
FILE4              80B  FA            6        1023   1       128  1  8
```

Here you see the *move* or rename facility in action.

HP-UX:

```
$ pwd
/users/boles
$ mkdir dir2
$ ll
total 15
drwxrwxrwx   1 boles    101          0 Dec 25 21:26 dir2
-rw-------   1 boles    101        121 Dec 24 14:01 file1
-rw-rw-rw-   1 boles    101        247 Dec 25 20:53 file2
-rw-rw-rw-   1 boles    101        247 Dec 25 21:06 file4
-rw-rw-rw-   1 boles    101       2536 Dec 25 21:28 seb
-rw-rw-rw-   1 boles    101       1055 Dec 24 14:07 sebe
-rw-rw-rw-   1 boles    101       1055 Dec 25 20:43 sebf
-rw-rw-rw-   1 boles    101       1607 Dec 25 20:54 sebg
-rw-rw-rw-   1 boles    101       2059 Dec 25 21:04 sebh
$ cd dir2
$ pwd
/users/boles/dir2
$ cp ../file1 file1dir2
$ ll
total 1
-rw-------   1 boles    101        121 Dec 25 21:35 file1dir2
$ cd
$ pwd
/users/boles
```

MPE:

Here we don't have an MPE analog closer than hello with a new group specified. In the example, while in the home directory, you ***make*** a new ***dir***ectory with mkdir. The directory file (initial "d" in the ll listing) now appears in its parent directory. A cd (change ***d***irectory) to the subdirectory dir2 is confirmed with a pwd showing the path name up the directory chain. The cp uses a ../ to indicate the parent directory of the current working directory. A cd without an explicit directory gets us back to the home directory, which the pwd confirms. This is just a quick dip in the deep end of the pool. Don't worry about this for the brief glimpse of UNIX™ you get here. But do be aware that the UNIX™ file system is different from MPE.

```
$ who > file4
$ cat file4
boles    console Dec 24 13:26
$ date >> file4
$ cat file4
boles    console Dec 24 13:26
Tue Dec 25 21:42:10 PST 1984
$ ll file4
-rw-rw-rw-   1 boles    101          59 Dec 25 21:42 file4
$ wc file4
      2     11     59 file4
$ grep 'MPE' file1
the UNIX vernacular to an MPE environment.
$ ll file1 > temp1; wc file1 > temp2; cat temp1 temp2 | grep 'file'
-rw-------   1 boles    101         121 Dec 24 14:01 file1
      5     22    121 file1
```

MPE:

Don't try to map this one-for-one to MPE. You see some redirection and a new counter command, wc, to set up an illustration of ***piping*** (the | operand) and the string finder, grep. The wc counts lines, words and characters. The grep lists the lines that contain the ***g***ene***r***al ***exp***ression ("mess": ***m***n***e***monic***s*** are merely a ***s***tate of mind) search string argument. Here the cat has 2 input files that it ***pipes*** to grep which outputs the two lines containing the search string 'file'.

Before wrapping up, let's scratch the surface of the UNIX™ ***shell***. First some simple shell scripts. Suppose you want your UNIX™ to speak MPE. Here you see a file called listf that contains ls. At first it won't execute but the chmod fixes that.

```
$ cat > listf
ls
[ctl-D]
$ listf
listf: cannot execute
$ chmod 777 listf
$ listf
dir2    file2 . . . temp1    temp3
file1   file4 . . . temp2
```

Here you see a file called purge with a $1, the symbol for the first argument, which is the name of the file you want to purge. The chmod makes it executable.

```
$ cat > purge
rm $1
[ctl-D]
$ chmod 777 purge
$ purge temp4
```

Next you see a shell control loop that edits all the files starting with "file", using the commands in sebmod.

```
$ cat sebsh
for i in file*
  do ed $i < sebmod
  done
```

Here's what sebmod looks like. It gives the file name, lists the first record, inserts a new line, then lists the first three lines, then quits.

```
$ cat sebmod
f
1
i
Begin with Mar 1986 Interex HP3000 Conf . . .
.
1,3p
q!
```

Here's an execution:

```
$ sebsh
107
file1
This is to show the translation of
Begin with Mar 1986 Interex HP3000 Conf . . .
This is to show the translation of
the UNIX vernacular to an MPE environment.

233
file2
This is to show the translation of
Begin with Mar 1986 Interex HP3000 Conf . . .
This is to show the translation of
the UNIX vernacular to an MPE environment.

59
file4
boles    console Dec 24 13:26
Begin with Mar 1986 Interex HP3000 Conf . . .
boles    console Dec 24 13:26
Tue Dec 25 21:42:10 PST 1984
```

Epilogue . . .

There you have it: a glimpse thru the Looking Glass from MPE-land into the Land of UNIX™. You've seen our "MPENIX": some of the elementary functions in our quasi-UNIX™ UDC's that we built to help our summer students. From there you sampled some of the UNIX power that enables a computer user to reach new levels of productivity. You've seen some of the features that have enabled UNIX™ to establish a good track record as the common link that lets us move with reasonable gracefulness across a substantial portion of the computer world today.

About the Author ....

Sam Boles *is a Member Technical Staff in the Hewlett-Packard Information Software Operation in Cupertino, California. With HP since 1976, his computer experience started back in the AUTOCODER days of the 1401/1410, migrated thru the 360/370 era, and now focuses on next-generation operating system software. Sam received his MS at UCLA in Information Systems.*

sebiug48 2135 27jan86

TITLE FROM BUSINESS ACTIVITIES TO ONLINE APPLICATION DESIGN

Theo Gielens
Database Consultants Europe B.V.
Amsterdam, Netherlands

ABSTRACT

All Design begins with Analysis and, in this paper, we look at how the results of a top-down, structured analysis of activities, using functional decomposition, can be input into a structured online application design.

The applications, when developed, should support the natural flow of operational and administrative activities and, further, should ensure that the organisation's data is kept consistent, up-to-date and accurate.

The following topics are covered:

- Activity Analysis
- Online Application Systems Design.

1. INTRODUCTION

Figure 1 shows the System Development Cycle required in a Shared Data Interactive environment. The left side of the diagram is concerned with the analysis and design of the data ie. Data Analysis, Logical Database Design and Physical Database Design.
The right side is concerned with the analysis and design of the activities ie. Activity Analysis, Application System Design and Structured Program Design.
The centre of the diagram covers the steps required to ensure that the interactions between data resources and business activities are fully catered for during the analysis, design and eventual implementation.

This paper covers the following areas:

1. Activity Analysis
   A method used to understand and record the activities and the flow of data between them.

2. Application Systems Design
   A method to transform the decomposed activities from the activity analysis phase into structured and clearly defined online or batch programs.

It is evident that the participation of the user in both areas is of vital importance. This implies that the techniques used by the analysts and the designers should provide them with simple, diagrammatic tools that enable effective communication with the users.

The business area "COURSE ADMINISTRATION" has been chosen for illustration.

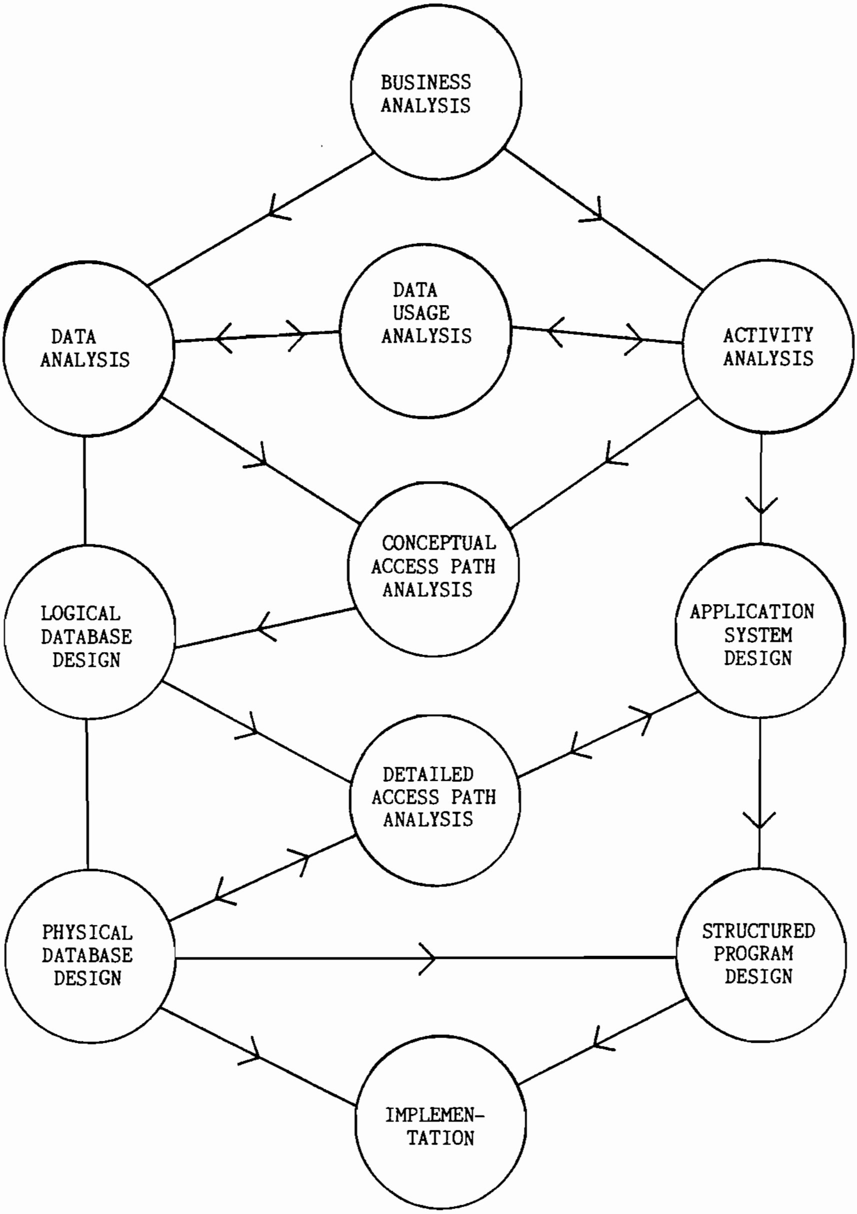


FIGURE 1 - The Systems Development Cycle in a Shared-Data, Interactive Environment.

2. ACTIVITY ANALYSIS

Activity analysis is a step-by-step, diagrammatic method used to analyse business activities by starting at a high overview of business areas and progressively decompose them, through lower levels of activity, into greater detail.

2.1. Deliverables

The output from activity analysis comprises Data Flow Diagrams and Activity Decomposition Diagrams supported by appropriate documentation. The DFD shows the activities, concerned with a business area, and the flow of data into and out of them.
The ADD provides an overview of the activities showing commonality, trigger events and the sequence in which the activities are performed (top-to-bottom and left-to-right).

2.2. Tasks for Activity Analysis

Activity Analysis consists of the following tasks:

2.2.1. Decompose the activities

The first objective is to clearly identify the major activities within the business area and the flow of data concerned with these activities (see Figure 2).
The next step is to break these activities down into smaller activities. The information is gathered using a number of sources as follows:

- interviews
- documentation of current systems
- documents currently in use
- etc.

2.2.2. Identify Elementary Activities

The decomposition of activities stops when a level of Elementary Activities has been reached (see Figure 3):

An elementary activity is one which cannot be decomposed further without either destroying the consistency of the data it uses and outputs or (in case of an enquiry type activity) ceases to provide all the output necessary to the objective of the activity.

Elementary activities become candidate transactions

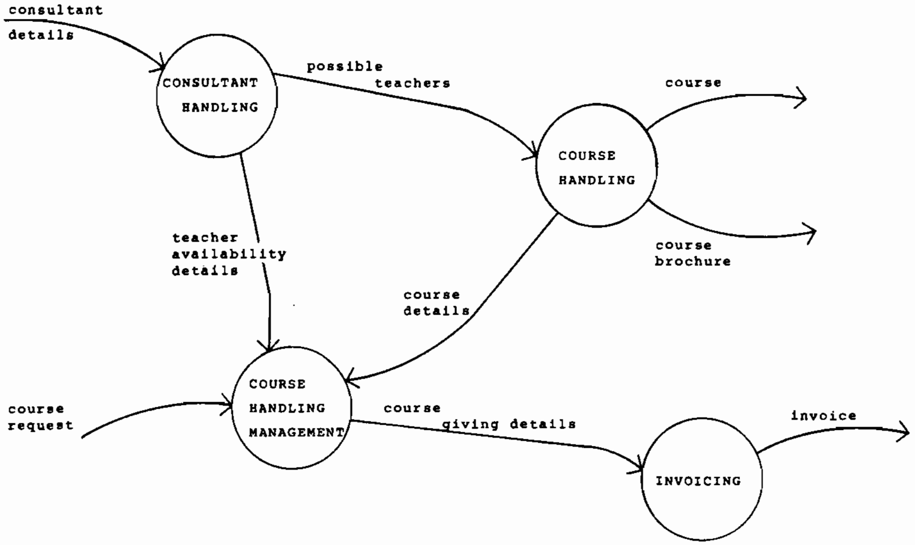
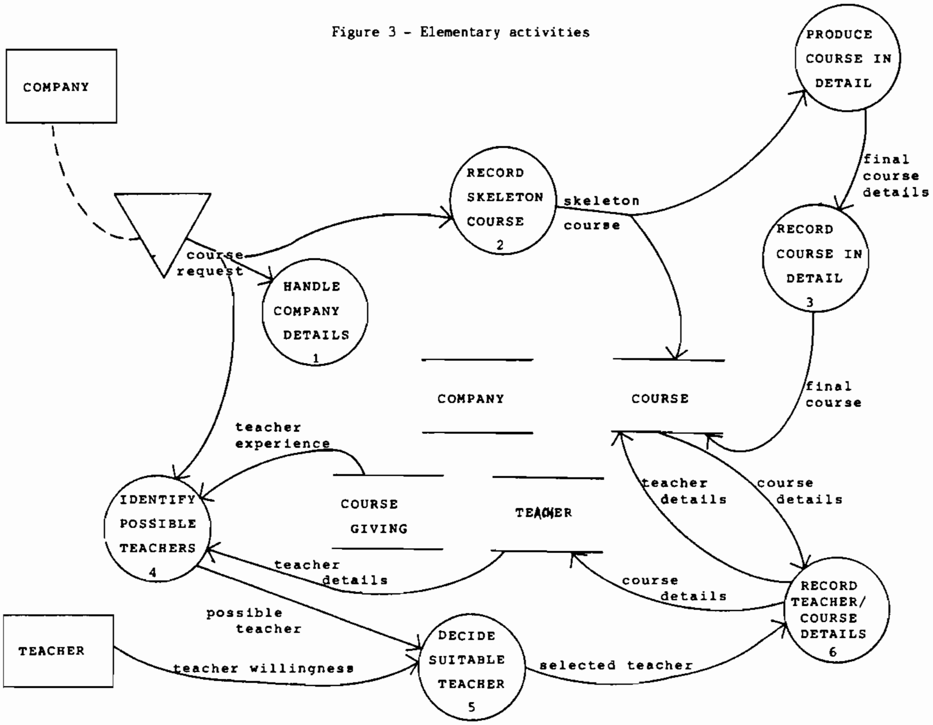


Figure 2 - Major activities in a course administration

Figure 3 - Elementary activities



2.2.3. Document low level DFD's

Since the DFD's are input to both the System Application Design and Conceptual Access Path Analysis phase it is essential to document them showing:

- description of the activity and the data flows.
- volumes
- where the activity is performed

2.2.4. Describe the elementary activity in detail

In order to avoid misconception and misunderstanding it is essential to describe the elementary activity in detail. A very useful technique to employ is structured text. This technique also provides a means of identifying commonality during System Application Design.

2.2.5. Create the ADD

The ADD is a hierarchical representation of the business activities showing how they decompose from the highest level activities down to the elementary activities. The sequence in which the activities are being performed (implied within the DFD) is also shown (see Figure 4).

2.3. Activity analysis and its interaction towards the data

The activities that need to be performed on the data will normally influence how the database is organised. From the elementary activities the entry points, navigation paths and access sequences are identified during a process called Conceptual Access Path Analysis (see Figure 1). The resulting output comprises:

- Activity Logic Diagrams
  diagrams showing what entities are accessed/created/deleted and the sequence that is required.

- Activity Usage Figures
  figures showing the frequency of logical accesses on entities.

3. ONLINE APPLICATION SYSTEM DESIGN

Application System Design is a method of further decomposing the elementary activities into processes that can be implemented as programs, modules or sub-routines, providing, therefore, a technique to design the structure of the online system.

3.1. Tasks for Online Application System Design

3.1.1. Identify common processes

In order to design an optimum systems network and to avoid repetitive design and development of those processes which may be common to more

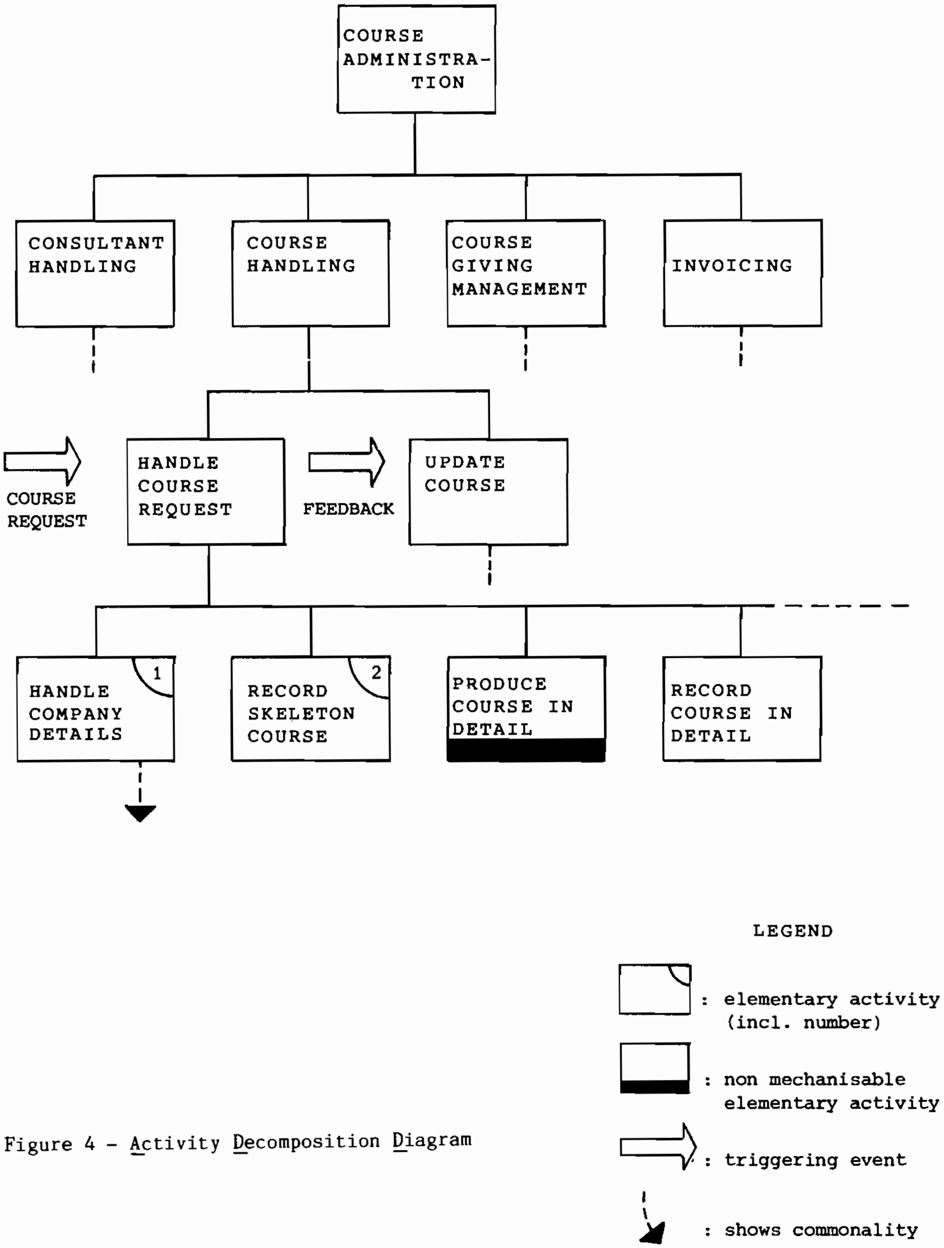


Figure 4 - Activity Decomposition Diagram

than one activity, it is necessary to identify commonality as the first step in Online Application System Design.
Commonality can exist at three levels, as follows:

- Elementary activities
- Components: action on an entity type and its attributes
- Primitives: action on an attribute type or its attribute values, or action on more than one attribute type in combination

Normally commonality of elementary activities would have been clearly identified, already, during the construction of the ADD. However, commonality of Components and Primitives can only be identified by a thorough examination of the structured text produced earlier.

COMPONENTS become candidate modules

PRIMITIVES become candidate modules or sub-routines

Both components and primitives are added to the ADD and, where they are common across more than one elementary activity, this is shown.

3.1.2. Systems Network Design

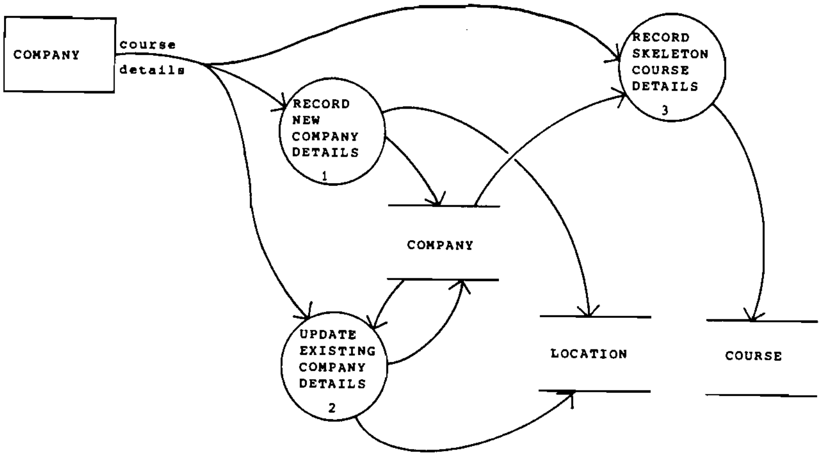
The systems network is designed using information contained in the DFD's and ADD and, in addition, the following considerations:

- standard navigation paths (define function keys)
- user preference on menus:
  . based on trigger events
  . based around entities
  . based around responsibility
  . etc.
- frequency : a navigation path for an activity which is being used frequently should be as short as possible.

The result is a Systems Network Diagram showing :

- hierarchy of transactions.
- flow of control (sequence, selection and iteration)
- listings (output)
- passed parameters
- how control is passed

To illustrate how a DFD can impact on the eventual SND an example is provided showing how a "course request" could be handled in two different ways. Figure 5a shows one example resulting in a SND as shown in Figure 6a.
Figure 5b shows a similar "course request" containing one extra activity, "CHECK IF COMPANY EXISTS", resulting in a significantly different SND, as shown in Figure 6b.



COMPANY
course
details
RECORD NEW COMPANY DETAILS
1
RECORD SKELETON COURSE DETAILS
3
COMPANY
UPDATE EXISTING COMPANY DETAILS
2
LOCATION
COURSE

Figure 5a

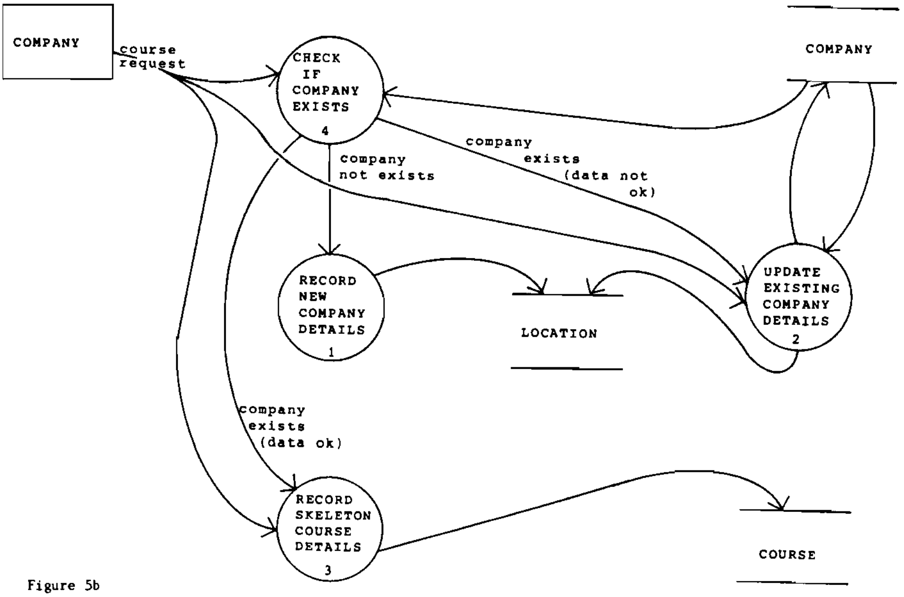


Figure 5b

Figure 6a

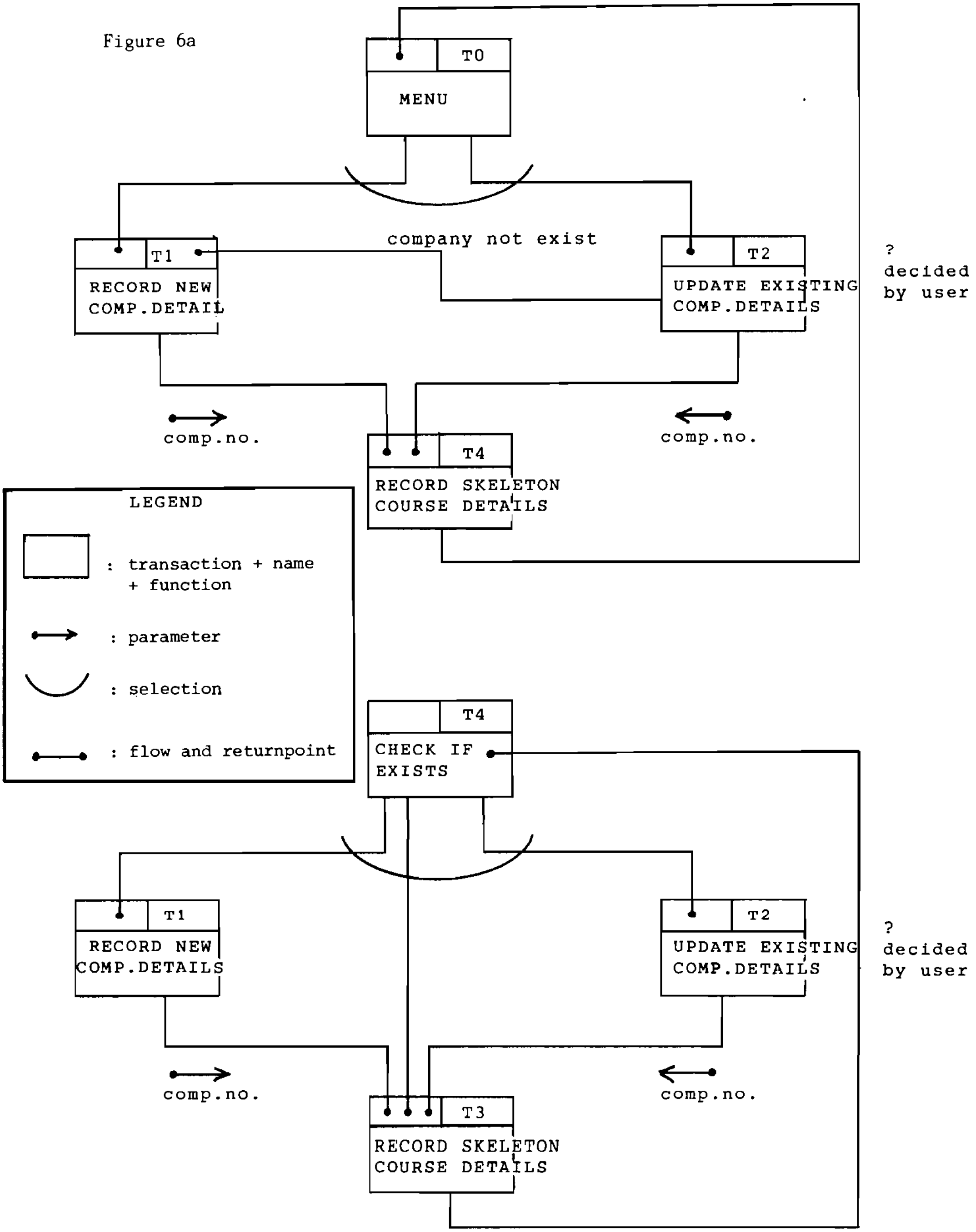


Figure 6b

Before the system network design is complete, it is necessary to check the following:

- Is every transaction available as required?
  Each user should be able to reach all activities he is entitled to use, subject to suitable authorisation.

- Is every common process fully integrated in the SND ?
  Where applicable, parameters should be added to the SND. Also all updating logic should be moved to "lower level" transactions in order to preserve the consistency.

3.1.3. Multiple implementation of Elementary Activities

It is possible that for some elementary activities, more than one transaction may need to be defined, for the following reasons:

- need for fallback systems

- different mechanisms for different users:
  - casual user version
  - experienced user version

- different installations (hardware/software)

For each transaction, however, the logic stays the same, although the mechanisms may be different. Whenever this occurs it affects the systems network:

- fall-back transactions and production transactions must <u>never</u> be put into the same network.
- transactions for different installations must <u>never</u> be put into the same network.
- transactions which provide <u>alternative</u> dialogues may or may not be put into the same network, depending on the users preference.

3.1.4. Design the dialogue(s)

The design is done together with the user. The design primarily consists of designing the user and computer actions for each transaction. The basic steps consist of:

1. determine the inputs and outputs
2. decide the technique of conversation
3. design the dialogue in broad outline
4. optimise and refine

3.1.4.1. Determine the input and output

From the structured text of each elementary activity, it can be determined what input (and when) is required to be able to perform the activity and what output is required to provide the user with what he requires.

3.1.4.2. Decide the technique of conversation

This depends highly on the users skills and preferences. Various possibilities exist, as follows:

- keyword (free format)
- question and answer
- form filling
- multiple choice (menu)
- etc.

3.1.4.3. Design the dialogue in broad outline

Again a diagram is used, which allows discussion with the user. The example (Figure 7) shows how new company details could be handled (refer to figure 6b activity T1). Note that the updates are performed in M06, with reference to the points discussed earlier regarding consistency for elementary activities.

3.1.4.4. Optimise and refine

Using prototyping tools the dialogue can be implemented. This could result in feedback from the user, as a result of which some steps may have to take place again. If no prototyping tools are provided, a desk check examining the dialogue should take place.

3.2. Interaction towards the data

Whereas, in the Conceptual Access Path Analysis (see Figure 1) the output is used to perform the Logical Database Design the outputs of Detailed Access Path Analysis provide the output that is used to either create a first pass physical database or optimise the physical database design.
The feedback from the database design could also enforce an alteration of some activities and possible the SND.

THEO GIELENS

Since the start of his data processing career in 1977, Theo Gielens has been particularly involved in the areas of Application Design and Activity Analysis. After joining DCE as a consultant, some two years ago, he extended his experience in the area of Data Analysis and he participated in the development of DCE's Application Design Course.

Project/System: Course Administration System	Sub-Project/Sub-System:	
Author: DLN	Date: 12/12/89	Page Nº:
Version: 5	Status: Agreed	
Transaction Name: Record New Company Details	Number: TO50	

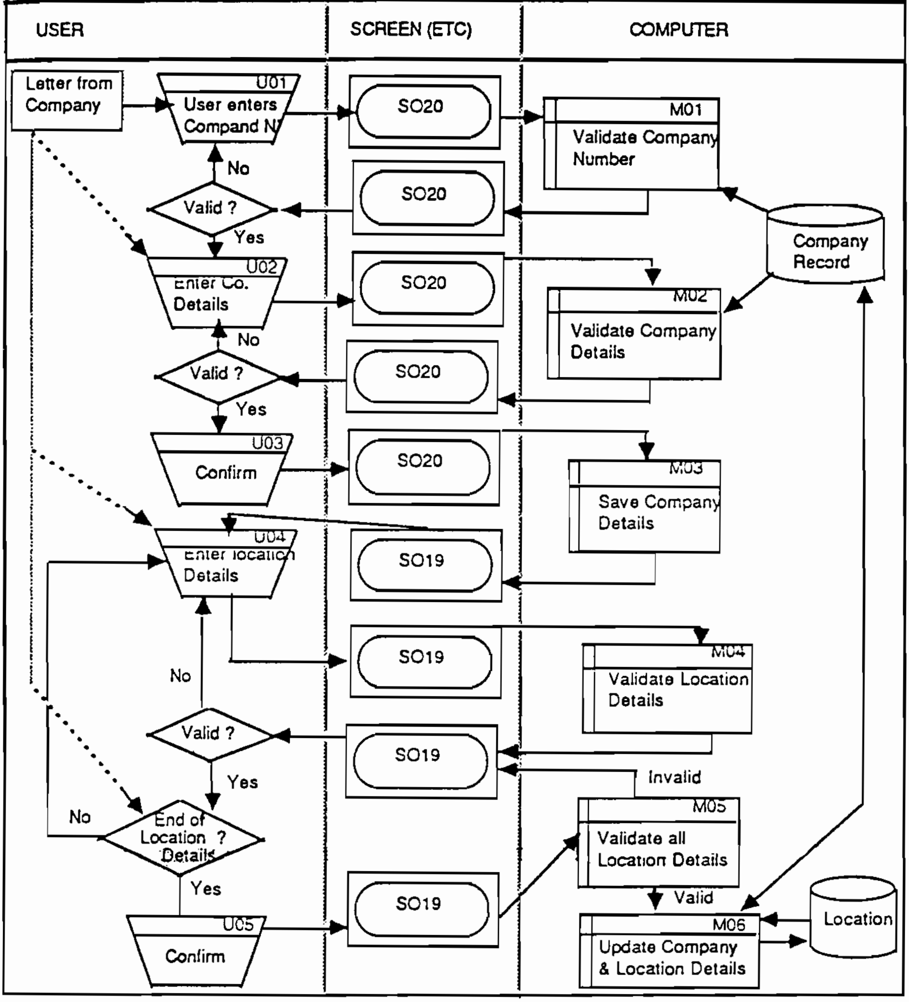


Figure 7

THE WALL METHOD - A NEW METHOD TO DEVELOP BETTER SOFTWARE

Matti Jamback
Kemira Oy, Helsinki, Finland

SUMMARY

The end-users are very seldom satisfied with the first version of a new application system. Something has to be added, something has to be changed and something is not necessary. This is very familiar to everyone who has worked in the software development projects.

There are a lot of reasons why software projects end like this, but one of the most common ones is the communication gap between end-users and software people. They have no common language.

In Finland a method called "The Wall Method", has been available in system design for over four years. The developer of the method is Kari Saaren-Seppala.

The Wall Method is a visual method that helps end-users and systems people work in teams and solve problems in a way that everyone understands.

The main characteristics of the wall method are:

- large, colourful, easily modifiable pictures on the wall
- the end-users, from different levels of the organization make the pictures in small, dynamically changing teams
- the working environment encourages creative spirit by visualizing the design phase in common terms.

DATA PROCESSING IS STILL YOUNG

Data processing has been done in large scale only about 25 years. Therefore, it is natural, that the system design methods are not as good and perfect as for instance in architecture. Yet, systems have been designed and some have succeeded and some have not.

WHY IS IT SO DIFFICULT TO SUCCEED?

The traditional way of designing application systems can be divided roughly in four parts:

1 Feasibility study, where the need and the benefits of the system are clarified.
2 System design, where the data content and the function of the system are designed.
3 Programming and testing.
4 Inauguration.

If you make big mistakes in part 1, e.g. you decide to make a system which is not necessary or the advantages of the system are not greater than the disadvantages, there is not very much matter what you do after, because the system will not satisfy the end-users.

If you make mistakes in system design, these mistakes can be corrected, but that costs often time and money which both are limited resources. To balance the budget and the schedule of the project compromises are made, but then the final target of the system is often lost.

The programming errors cannot be avoided but if these errors are caused by inaccuracies in the system design we are in trouble.

And finally, when the entire system should start, all the troubles begin. The end-users are not satisfied and a lot of corrections and changes have to be made until the system is satisfactory.

All the above mentioned is probably very familiar to those who have been designing application systems. There are many reasons why software projects don't succeed, some reasons are technical, some project management reasons and perhaps the most common ones are the definitional reasons. Usually it is the mixture of all these reasons.

LET THE END-USERS DEFINE THEIR OWN SYSTEMS

It is a fact that if you can't get the end-users to take part in system design the whole system is in great danger to fail.

Usually an inquiry is made for the end-users at the early phase of the system design. He is asked about the data content and the services he expects from the new system. After the proper answers have been received, the system professionals design the system and adp-professionals accomplish it.

The role of the end-user must not be just an object of the inquiries, but the area where they take part in the system design must be much wider. The end-user may quite well manage the design phase.

Four levels of participation for the end-user can be defined in the life-cycle of an application system.

Level 1:

- an inquiry is made about the present problems and about the proposal for the new system
- the facts that already are quite clear get clearer but the point of view of the end-users does not change.

Level 2:

- the plan for the new system is gone through actively and thoroughly in small dynamically changing teams
- the real material is used
- the end-users try to imagine the future form of their work
- ideas and experiments are made
- many rounds of improvements are gained
- when the work goes on, the point of view of the participants changes and gets wider and the end-users learn to know new features about their work.

Level 3:

- this level is mainly accomplishing and testing the nearly finished system
- the methods are about the same as on Level 2, but now the computer is also used
- the prototyping is very much recommended on this level
- the end-users learn to know the system which they have to use for the next few years and they can imagine the influences and changes the new system may cause to their work.

Level 4:

- this is the continuous use of the system which will last for several years.

Very often the end-users take part in the system design only on Level 1. Level 2, partly also Level 3 have been done by the system professionals. At this point the end-users have a rather limited chance to influence on the whole system.

The prototyping on Level 3 is very much recommended, because by using this method the end-users get almost a real impression about the system and a part of training is also taken care of.

Level 4 is very often forgotten. Yet it is very important to collect feedback about the system from the end-users, because they are the everyday users and they know best the advantages and disadvantages of the system.

SOME REQUIREMENTS FOR A GOOD DESIGN METHOD

It is vitally important to get the end-users to take part in the system design, but how to cross the communication gap between them and the systems people? If the end-users and the systems people cannot use exactly the same language about the problems they are solving, it is not possible to get satisfactory results.

There are many nice ways to define a system very precisely but for an end-user it is very often difficult to understand the formulas and other abstract symbols which are used in these methods. Still, most of these methods are not suitable for team work, which is essential in designing systems.

Let's take a short look at the methods which are used in other fields that require making design and various kinds of descriptions.

When an architect makes the plans of a house, he makes drawings about it from different angles. In these drawings he also puts trees, people, machines etc. in order to get the drawing as real as possible. A miniature model about a house is also often made and that model can even be taken apart to get an idea about the inner constructions of the house. In industry a prototype is made about a new product and this prototype is then tested in as truthful conditions as possible in order to find out all the weaknesses. These are then corrected before the product is put on the product line.

In these two examples the keywords are visuality and truthfulness. the visuality helps to compare the design to the real world and to find out the drawbacks of the plan. If the design is concrete enough, it is possible to test various alternatives of activities against that design and to get new ideas also to the real life. Still, a truthfull design method gives a good basis for communication, because the conceptions and the language are of reality, not some gibberish that only trained professionals understand. It is possible to start working immediately without wasting time to clarify various kinds of conceptions. And last but not least, the visuality activates imagination. The word imagination gives an idea of seeing imagines. The visual images, pictures, help the individual or the team to look for and to find out new and better solutions.

Despite of being visual and truthfull the design method must be exact, it has to keep the general view about the system clear, it must be suitable for team work, the conceptions must be clear throughout the entire system and the pictures must be easily modifiable. Still, a good design method should be easy to learn and easy to use.

THE WALL METHOD

The Wall Method has been developed in Finland in 1981 in three application system developmen projects. The developer is Kari Saaren-Seppala. In the years 1981-1984 over 100 seminars have been held about this method. In Finland it has been taken very widely in use, e.g. 9 of the 10 biggest industrial companies use The Wall Method. Many software houses have adopted this method as a part of their own design methods. Also a book has been published and an English version of that book will be published soon.

The Wall Method is not a licensed product but it is an open method that is available to anyone.

HOW TO USE THE WALL METHOD?

The design of a new system can be divided into six phases as follows:

1 Survey of the system area
2 Description of the existing system
3 Survey of the changes
4 Definition of the data
5 Definition of the adp-tasks
6 Testing.

This list of phases is not an exact order of working but the three first phases can all be done simultaneously.

The next chapters present each of these six phases shortly and also a list of techniques, that can be used in each phase, will be presented. The more accurate descriptions of the techniques will be presented a few pages later.

SURVEY OF THE SYSTEM AREA

The description consists of all the factors that the system possibly will handle (e.g. the divisions and the departments of the company, the authorities, etc.).

All the factors are first collected together in any order. When this is done, these factors are then grouped in two groups, those which will be included in the system (marked with "+") and those which will not (marked with "-"). This process is iterative and new groups are formed and some groups are included and some not. Finally we have the group of factors that will belong to the system area.

The techniques that can be used are

- "the glass house"
- horizontal projection
- data flowchart
- organization picture
- large-scale picture.

DESCRIPTION OF THE EXISTING SYSTEM

The purpose of this description is to give to the members of the project and the teams a realistic and similar image of the starting point. This helps the further communication, because the conceptions are clear and the language is same.

The company and its functions are described so that it is possible to see it as a whole and the situation of the new system in this wholeness.

The description includes for instance the functions, the jobs, the material-streams, the equipment and the accommodation of the company. The end-users and their current tasks and the current data-processing will be described, too.

The real material, reports, file-descriptions, card files etc. is included, also, in this description.

After the completion of this phase all the problems of the existing system must be found.

The techniques are the same as in the preceding phase.

SURVEY OF THE CHANGES

If there is no need for changes in the existing system, there also is no necessity to make a new system. A need for changes that is strong enough, is usually the activating reason to start a system project.

There are many ways to collect the needs for changes. Meetings can be arranged where participants tell their needs, end-users write lists about their need etc. To go through the description of the existing system is a good way of finding out changes.

The material is then analysed and the working teams go the results through and divide the changes in two groups, those which will belong to the system and others which do not. The changes that were chosen to the system, will be a part of the test material.

DEFINITION OF THE DATA

In this phase the data and the logical structure of the data will be described.

First a conceptual schema will be used. The sure conceptions of the existing system form a basis of the schema and this schema is then completed by the new conceptions, that the changes possibly have created. The result will be a conceptual model which then can be completed by the data-items of each conception.

The final result will be the logical conceptual schema and the lists of data-items completed with the details, such as the length and the mode of the data-item, volume-numbers of data-sets, is a data-item a primary or a secondary key etc. This description is the basis for the final data-base.

The techniques are:

- conception schema
- data-item list.

DEFINITION OF THE ADP-TASKS

The data-processing of all the work stations is divided into adp-tasks. A task may be updating a data set, making a report, entering an inquiry to the data-base etc.

Of each task a number of things must be described:

- how the data is accessed
- the checkings and defauls of data-items
- calculations
- display lay-outs
- report lay-outs
- security
- menus and display-paths
- etc.

The result of this phase is a final manual model of the future adp-system. The end-users are able to test the system with the real material.

It is very useful to use prototyping in the critical or otherwise important tasks in order to give a more realistic impression of the system to the end-user. Using a prototype is also very good training for the future users.

The techniques are:

- prototyping
- cross-references
- lists of the tasks in a work station.

TESTING

The end-users use their own real material as the test material. The list of changes is a part of that material.

The testing with real material and look-alike manual system will reveal weaknesses and defects of the system. Yet, it is easy to make corrections and improvements. The adp-professionals, who will do the final programming, should be present at all the testing sessions in order to get a precise and realistic image about the system.

The testing phase may last for weeks, but it is worth using enough time so that all the end-user groups are able to test the system definition and get assured that their needs will be fulfilled.

HOW TO ORGANIZE A SYSTEM DESIGN PROJECT?

In order to guarantee enough special experience to the project group it is recommended that at least the following skills can be found from the group:

- project management
- knowledge of the system design methods
- adp-knowledge
- knowledge about the system-area from the end-user's point-of-view
- knowledge about a working method, which is systematic and creative.

It is not necessary to get the representatives of all the end-user groups to the project group. Depending on the part of the system which is to be described a team of end-users is collected to do that description with the project group.

The first three phases of system design do not necessarily require the presence of adp-professionals but after that it is essential that also analysts and programmers are present.

ABOUT THE TECHNIQUES USED IN THE WALL METHOD

Since The Wall Method is based on large and colourful pictures on the wall, there can't be an exact technique or way to make these pictures. Yet, some techniques and materials have shown out to be useful and easy to use and they can be called as "standards" in The Wall Method.

Below short descriptions of some of these "standards" are presented.

"THE GLASS HOUSE"

This technique is based on the facade pictures of buildings. The walls of the buildings are transparent, made of glass. The figures and functions that are to be described, are situated this house on their own places.

With this picture it is easy to show functions of the company. The material and dataflows can be situated in their right places, the work stations, computers etc. are all visible and the connections to the outer world can also be described in this picture.

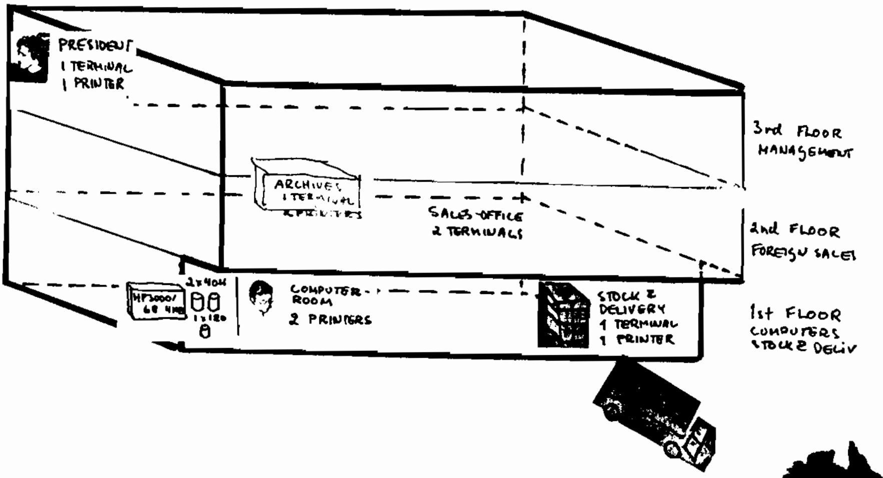


Figure 1. The Glass House.

HORIZONTAL PROJECTION

This picture is a horizontal projection about the building. It is used like "The Glass House".

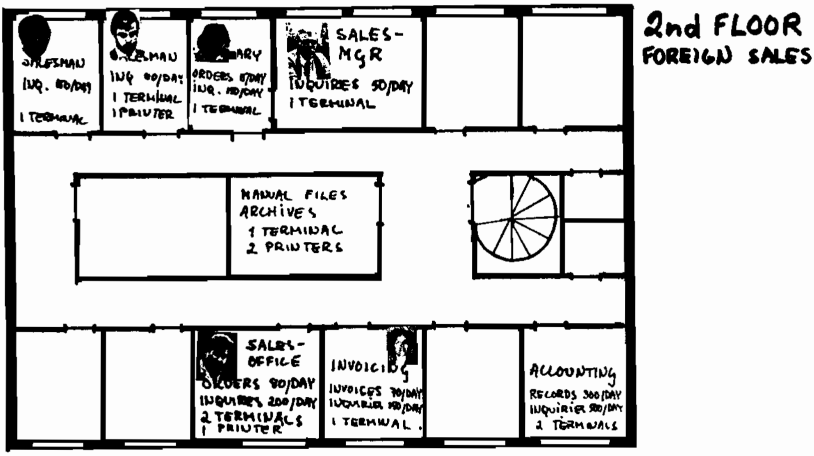


Figure 2. Horizontal projection.

DATA FLOWCHART

This picture is a reduced picture about "The Glass House" or the horizontal projection. Only the data or material flows are left.

It is also possible to make a flowchart that is combined of different types of pictures, "The Glass Houses", horizontal pictures etc. The pictures are connected to each other by arrows, which show the direction of the flow.

LARGE SCALE PICTURE

This picture is a combination of all the pictures which are already mentioned.

The large scale picture gives a general overview about the system when it is looked at from a far distance. When the observer steps closer, smaller and smaller details appear.

ORGANIZATION PICTURE

This picture describes those parts of the organization which will use the system.

It is useful to combine to this a cross-reference, which describes the responsibilities of the data administration.

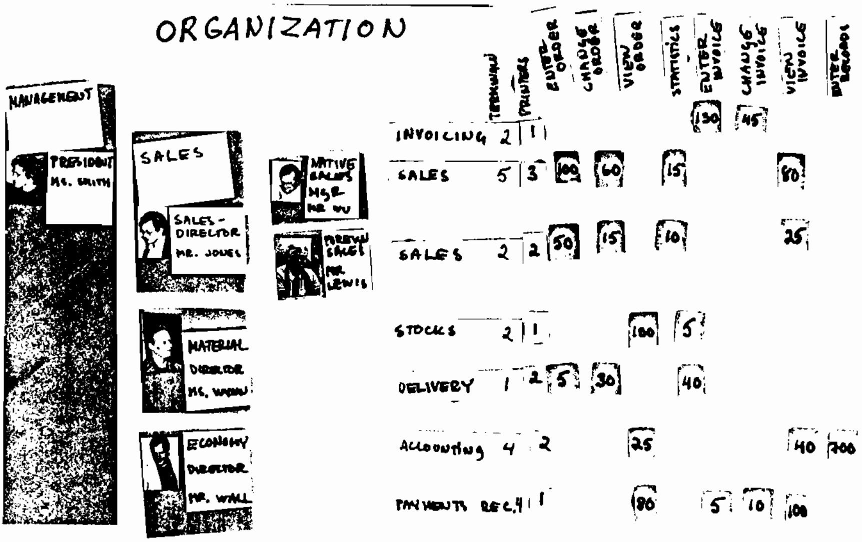


Figure 4. Organization picture.

CONCEPTUAL SCHEMA

This picture describes the logical structure of the data. When the lists of data-items are combined to this picture, there is a basis for creating the data-base.

The logical data groups of the system are at first grouped into data conceptions and then the relationships are defined between those conceptions. This is the logical structure of the data. The data-items of each data conception are then listed and the necessary information about each item is added.

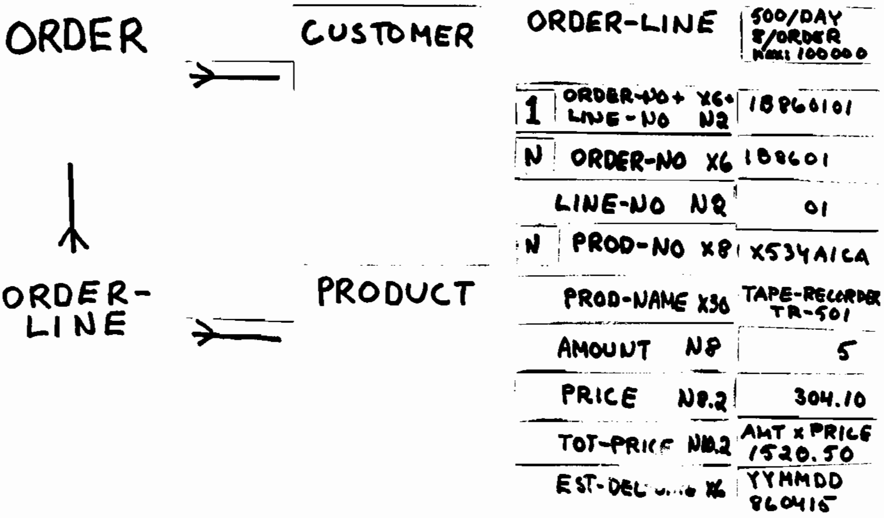


Figure 5. Conceptual schema.

MATERIALS

When using The Wall Method, the following materials can be used:

- a lot of empty wall space
- black felt tip pens (various gauges)
- pencils
- tape
- a kind of adhesive for fastening the cartons and papers
- large papers
  - white paper
  - white squared flip-paper, square 5 x 5 cm
- coloured cartons (about ten light colours)
- tools
  - scissors
  - photographic cutter for cutting cartons
  - ruler for cutting paper

HOW TO USE COLOURS?

The coloured cartons must be used throughout the system so that one colour always means the same thing.

The nex recommendation has shown out to be good:

- light blue: an end-user, a group of end-users, a department
- light green: a task of an end-user
- yellow: a data, a conception, a data-base, anything that accompanies to the data
- bright green: raw materials, products, all material
- gray: display lay-outs
- all red colours are alarm colours

HOW TO DOCUMENT?

The system description which is made by using The Wall Method, can't be moved from one place to another as easily as a description in the covers. So its is important that the room in which the work is to be done, is reserved for that use long enough. It is also recommendable that the room can be locked in order to prevent unauthorized persons from seeing the description.

The back-up copies can be made by photographing the wall from time to time. If a video-tape camera and recorder are available the walls can be taped and that tape can be used in training the end-users.

SUMMARY

The Wall Method got started, when it was realized that in system design the concrete pictures are better understood than the abstract figures and formulas. Such materials are for instance the real reports, forms, photographs, etc. When this material was used, the pictures became large and it was natural to take the empty walls in use.

The big size of the wall pictures requires an own room for the project. So it is natural, that the members of the project gather in one team to work together in that room. The work becomes effective and creative when the result is continuously visible to everyone. Since the pictures are easy to modify, all kinds of experiments are possible and new solutions are easily tested.

The training of the end-users actually takes place while making and testing the description.

THE ADVANTAGES AND DISADVANTAGES

Like any method The Wall Method is especially strong in some cases, neutral in others and it also has some weaknesses.

Advantages are:

+ efficient common language for the end-users and systems people, it is possible to talk about the problem itself, not about the technique
+ the method is easily learnt
+ the whole system can be seen as a whole
+ the state of readiness of the system can be seen all the time
+ the testing is easy
+ suitable for team work
+ cheap and easy to use because all the equipment and materials can be bought from the nearest stationer's
+ since the method is based on various degrees of accuracy, the system can be presented to the management from different levels of the organization.

Disadvantages are:

- part of the pictures must be transferred into ordinary paper documents
- security may be difficult to arrange
- the room must be reserved for the project for a long time.

Finally, if you are satisfied with your present system design method, use it by all means, but if you feel that The Wall Method might give you something new, try it first in a minor scale as a part of your present method. It may not be wise to abandon a method that you may have used for years but to take a new method in use as an alternative and having got some experience you can make the choice.

The preceding chapters have described The Wall Method only as a system design method but the method can be used in many other cases, too. All kinds of descriptions can be made by using The Wall Method and some pictures of the system description can be used also for other purpose than system design. Since the method gives a concrete view about the things, it can be used anywhere, where it is necessary to show or demonstrate various cases.

THANKS

I want to thank Kari Saaren-Seppalá for the valuable advise he gave to me while I was doing this presentation. I also used his book as literary sources.

BIOGRAPHY

Matti Jamback has been in adp-business for over 13 years as an analyst, project manager and system manager. His current position is in Kemira Oy, Fertilizer Division, Helsinki, Finland. He is System Manager, with responsibility for coordinating and designing systems for that division. He has been using The Wall Method starting from the first projects, where it was developed.

INFORMATION SYSTEMS PROTOTYPING

Orland Larson
Hewlett-Packard Company, Cupertino,California, USA

Summary

One of the most imaginative and successful techniques for clarifying user interfaces and generally improving the productivity and effectiveness of application development is a methodology called INFORMATION SYSTEMS PROTOTYPING.

With waiting time for new applications running into several years and those applications failing to meet the users needs, managers as well as users have been searching for more efficient and effective approaches to systems development.

Prototyping, as an application system design and development methodology, has evolved into a real option for both the MIS professional and the user.

This paper reports on the growing body of knowledge about prototyping. It begins by reviewing the changing role of data processing, the challenges facing the MIS organization, and the traditional approach to application development. It then defines prototyping followed by the step-by-step prototype development process. The advantages and disadvantages, as well as the cost and efficiency of prototyping, will be discussed followed by the essential resources neccessary to effectively prototype applications. In conclusion, to illustrate the benefits of prototyping, the speaker will present success stories of systems developed using the prototyping approach.

INTRODUCTION

The Changing Role of Data Processing

The data processing department has changed dramatically since the 1960s, when application development as well as production jobs were usually run in a batch environment with long turnaround times and out-of-date results.

The 1970s were a period of tremendous improvement for the data processing environment. One of the key developments of that period was the development and use of Data Base Management Systems (DBMS). This provided the basis for on-line, interactive applications. In addition, computers and operating systems provided programmers the capability of developing application programs on-line, while sitting at a terminal and interactively developing, compiling, and testing these applications. The end user was also provided with easy-to-use, on-line inquiry facilities to allow them to access and report on data residing in their data bases. This took some of the load off the programmers and allowed them to concentrate on more complex problems.

During the 1980s, the data base administrator and MIS manager will see increased importance and use of centralized data dictionaries or "centralized repositories of information about the corporate data resources." Simpler and more powerful report writers will be used by the end user and business professional. The programmer will see the trend towards the use of high-level, transaction processing languages, also known as fourth generation languages, to reduce the amount of code required to develop applications. Finally, the tools have been developed to effectively do application prototyping, which will provide benefits to the end user as well as the application programmer and analyst.

Throughout the 70s and 80s, information has become more accurate, reliable, and available, and the end user or business professional is becoming more actively involved in the application development process.

Challenges Facing MIS

One of the MIS manager's major problems is the shortage of EDP specialists. A recent Computerworld article predicted that by 1990 there will be 1/3 of a programmer available for each computer delivered in this country. Software costs are also increasing because people costs are going up and because of the shortage of skilled EDP specialists. The typical MIS manager is experiencing an average of two to five years of application backlog. This doesn't include the "invisible backlog," the needed applications which aren't even requested because of the current known backlog. In addition, another problem facing MIS management is the limited centralized control of information resources.

The programmer/analyst is frustrated by the changeability of users' application requirements (typically, the only thing constant in a user environment is change). A significant amount of programmers' time is spent changing and maintaining users' applications (as much as 60 to 80 percent of their time). Much of the code the programmer generates includes the same type of routines such as error checking, formatting reports, reading files, checking error conditions, data validation, etc. This can become very monotonous or counterproductive for the programmer.

The end user or business professional is frustrated by the limited access to information needed to effectively do his/her day-to-day job. This is especially true for those users who know their company has spent a great deal of money on computer resources and haven't experienced the benefits. The users' business environment is changing dynamically and they feel MIS should keep up with these changes. MIS, on the other hand, is having a difficult time keeping up with these requests for application maintenance because of the backlog of applications and the shortage of EDP specialists. Once the user has "signed off" on an application, he is expected to live with it for a while. He is frustrated when he requests what he thinks is a "simple change" and MIS takes weeks or months to make that change.

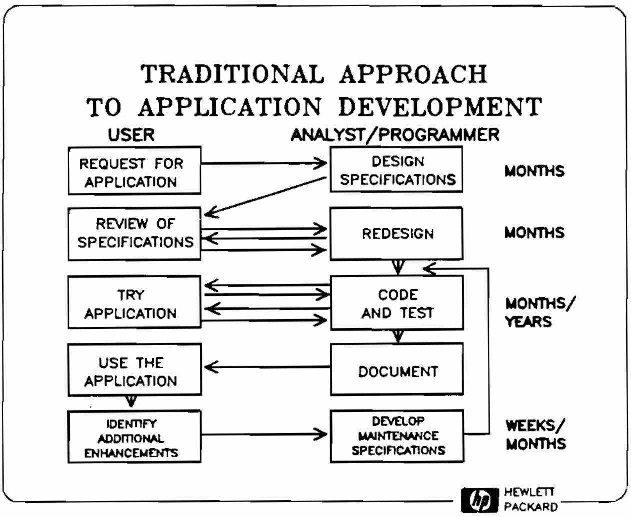
Traditional Approach to Application Development

There are some myths concerning traditional application development:

- Users know exactly what they want
- Users can effectively communicate their needs to MIS
- Users needs never change.

The traditional approach to application development has serious limitations when applied to on-line, interactive information systems that are in a state of constant change and growth. Communications among the user, analyst, programmer, and manager tend to be imprecise, a detailed analysis prolongs the process to the annoyance of the user, and specifications are either ambiguous or too voluminous to read. To compound this problem, the user is often requested to "freeze" his requirements, and subsequent attempts at change are resisted.

Let's review the traditional approach to application development.



- The user first requests an application and then an analyst or programmer is assigned to the application.

- The analyst or programmer takes the oftentimes sketchy user's specifications and designs more complete specifications.

- The user then reviews the analyst's interpretations of his specifications and probably makes additional changes.

- The analyst redesigns his specifications to adapt to these changes. (By this time, several days, weeks or months have gone by.)

- The user finally approves the specifications, and a team of analysts and programmers are assigned to develop, test and document the application.

- The user finally tries the application. Months or years may have gone by before the user gets his first look at the actual working application.

- The user, of course, will most likely want additional changes or enhancements made to the application. This is called adjusting the application to the "real world".

- Depending on the extent of these changes, additional maintenance specifications may have to be written and these program changes coded, tested and documented.

- The total application development process may take months or years, and the maintenance of these applications may go on forever.

In summary, the traditional approach to application development results in long development times, excessive time spent on maintenance, a multi-year backlog of applications, limited control and access to information, and applications that lack functionality and flexibility and are very difficult to change. The question is: "Can we afford to continue using this approach to application development?"

Prototype Defined

According to Webster's Dictionary, the term prototype has three possible meanings:

1) It is an original or model on which something is patterned: an archetype.
2) A thing that exhibits the essential features of a later type.
3) A standard or typical example.

J. David Naumann and A. Milton Jenkins in a paper on software prototyping (see reference 7) believe that all three descriptions apply to systems development. Systems are developed as patterns or archetypes and are modified or enhanced for later distribution to multiple users. "A thing that exhibits the essential features of a later type" is the most appropriate definition because such prototypes are a first attempt at a design which generally is then extended and enhanced.

Roles in the Prototyping Process

There are two roles to be filled in prototyping -- the user/designer and the systems/builder. These roles are very different from the traditional user and analyst/programmer roles under the traditional approach. The terms "user/designer" and "systems/builder" emphasize these differences and denote the functions of each participant under the prototyping methodology. Remember it is the user who is the designer of the application system and the systems professional who is the builder.

The user/designer initiates the process when he/she conceives of a problem or opportunity that may be solved or exploited by the use of an information system. The user/designer typically must be competent in his/her functional area (many times he/she is a manager) and usually has an overall perspective of the problem and can choose among alternative solutions. However, he/she requires assistance from the MIS organization.

The systems/builder is assigned by the MIS organization to work with the user/designer and is competent in the use of the available prototyping tools and knowledgeable about the organizations data resources.

Prototyping Process

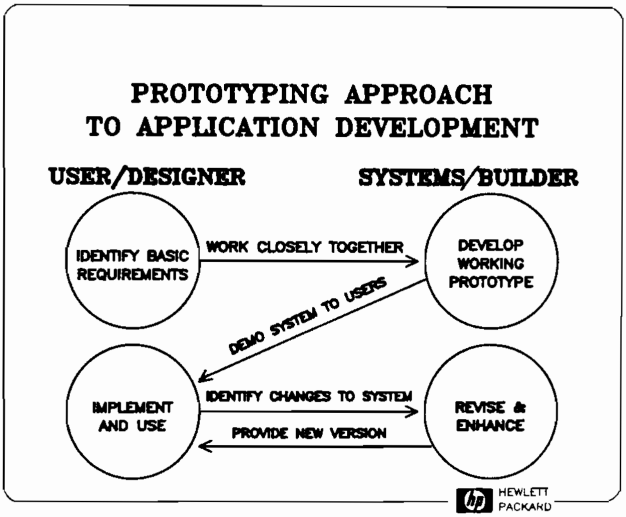
The process of application prototyping is a quick and relatively inexpensive process of developing and testing an application system. It involves the user/designer and the systems/builder working closely to develop the application. It is a live, working system; it is not just an idea on paper. It performs actual work; it does not just simulate that work. It can be used to test assumptions about users' requirements, system design, or perhaps even the logic of a program.

Prototyping is an iterative process. It begins with a simple prototype that performs only a few of the basic functions of a system. It is a trial and error process - build a version of the prototype, use it, evaluate it, then revise it or start over on a new version, and so on. Each version performs more of the desired functions and in an increasingly efficient manner. It may, in fact, become the actual production system. It is a technique that minimizes the dangers of a long formal analysis and increases the likelihood of a successful implementation.

Prototyping Methodology/Model

The prototyping methodology in general, is based on the following proposition: "People can tell you what they don't like about an existing application easier than they can tell you what they think they would like in a future application."

Prototyping an information system can be viewed as a four -step procedure.



PROTOTYPING APPROACH TO APPLICATION DEVELOPMENT

USER/DESIGNER

SYSTEMS/BUILDER

IDENTIFY BASIC REQUIREMENTS

WORK CLOSELY TOGETHER

DEVELOP WORKING PROTOTYPE

DEMO SYSTEM TO USERS

IMPLEMENT AND USE

IDENTIFY CHANGES TO SYSTEM

PROVIDE NEW VERSION

REVISE & ENHANCE

HEWLETT PACKARD

Step 1. User/designer identifies the basic information requirements:

- Write a brief, skeleton-like statement that captures the essential features of the information requirements.
- User/designer and systems/builder work closely together.
- Concentrate on users' most basic and essential requirements.
- Define data requirements, report formats, screens, and menus.
- Need not involve lengthy written specifications.
- For larger systems, a design team may need to spend a few weeks preparing a first-effort requirements document.

Step 2. Systems/builder develops the initial prototype:

- Systems/builder takes the notes developed in the user discussions and quickly builds the menus and dialogs.
- A data dictionary would be useful at this time.
- Design and/or define data base and load subset of data.
- Make use of defaults and standard report formats.
- Write required application modules using a fourth generation language.
- Prototype performs only the most important, identified functions.

Step 3. Users implement and use the prototype to refine requirements:

- Systems/builder demonstrates prototype to small group of users.
- Users gain hands-on experience with application.
- Users are encouraged to make notes of changes they would like made.
- Users discuss and prioritize desired changes.

Step 4. Systems/builder revises and enhances the prototype:

- Systems/builder modifies the prototype to correct undesirable or missing features.
- May require modification or redesign of data base, changes to existing programs and/or additional program modules.
- Deliver back to users quickly.

NOTE: Steps 3 and 4 are repeated until the system achieves the requirements of this small group of users. Then either introduce it to a larger group of users for additional requirements or if enough users are satisfied, demo it to management to gain approval for the production system.

When to Use Prototyping

1. To clarify user requirements:

   - Written specs are often incomplete, confusing, and take a static view of requirements.
   - It is difficult for an end user to visualize the eventual system, or to describe his/her current requirements.
   - It is easier to evaluate a prototype than written specifications.
   - Prototyping allows, even encourages, users to change their minds.
   - It shortens the development cycle and eliminates most design errors.
   - It results in less enhancement maintenance and can be used to test the effects of future changes and enhancements.

2. To verify the feasibility of design:

   - The performance of the application can be determined more easily.
   - The prototype can be used to verify results of a production system.
   - The prototype can be created on a minicomputer and then that software prototype may become the specifications for that application which may be developed on a larger mainframe computer.

3. To create a final system:

   - Part (or all) of the final version of the prototype may become the production version.
   - It is easier to make enhancements, and some parts may be recoded in another language to improve efficiency or functionality.

When Not to Use Prototyping

1. When an application requires a standard solution that already exists and is available at a reasonable cost from a software supplier.

2. When you don't have a good understanding of the tools available to prototype.

3. When the organization's data and software resources are not well organized and managed.

4. When MIS management is unwilling to develop a staff of professional systems/builders.

5. When the user/designer is unwilling to invest his/her time in the development of the application system.

Potential Problems

One of the initial problems typically encountered is the acceptance of the prototyping methodology by the systems people. This is due to the fact that people naturally tend to resist change. It may also encourage the glossing over of the systems analysis portion of a project. It is not always clear how a large complex system can be divided and then integrated. Initially, it could be difficult to plan the resources required to prototype (people, hardware and software). It may be difficult to keep the systems staff and users abreast of each version of the system. Programmers may tend to become bored after the nth iteration of the prototype. Testing may not be as thorough as desired. It might be difficult to keep documentation on the application up to date because it is so easy to change.

Even with these concerns, prototyping provides a very productive working relationship for the users and the builders. So it behooves all data processing management to learn to use this powerful tool creatively and to manage it effectively.

THE ADVANTAGES OF PROTOTYPING GREATLY OUTWEIGH THE PROBLEMS!

Advantages of Prototyping

One of the main advantages of application prototyping is that this methodology provides a capability to quickly respond to a wide variety of user requests. It provides a live, functioning system for user experimentation and accommodates changes in a dynamic user environment. One interesting aspect of this approach is that users are allowed and even encouraged to change their minds about an application's interfaces and reports, which is a very rare occurrence during the traditional approach. Maintenance is viewed right from the beginning as a continuation of the design process. Finally, prototyping provides an effective use of scarce systems/builders. One or a limited number of systems/builders will be required for each prototyping project; and while users are testing one prototype, the systems/builder can be working on another.

Cost and Efficiency

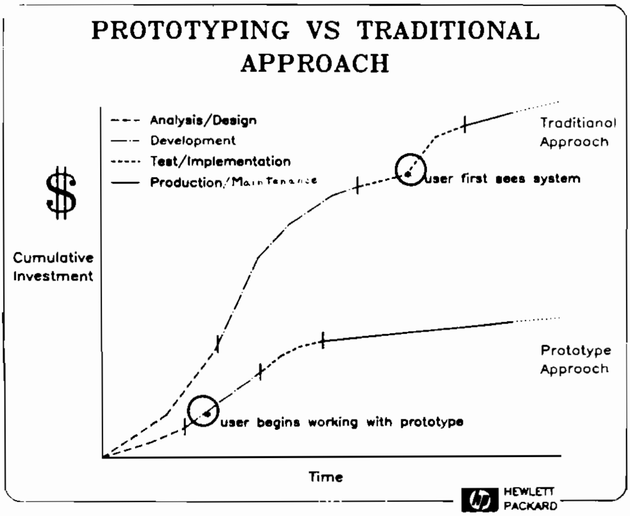
It has been found that there is an order of magnitude decrease in both development cost and time with the prototyping methodology.

It is often difficult to estimate the cost of prototyping an application system because the total costs of development, including maintenance, are usually lumped together. The cost of implementing the initial system is much lower than the traditional approach (typically less than 25%).

However, software prototyping could be expensive in the following ways:

- It requires the use of advanced hardware and software.
- It requires the time of high-level users and experienced systems staff.
- It requires training of the systems staff in the use of prototyping and the associated tools.
- Application run-time efficiency may be compromised.

The main thing to remember is that the main focus of prototyping is not so much efficiency but effectiveness.



PROTOTYPING VS TRADITIONAL APPROACH

Essential Resources

The following are the essential resources to effectively do application prototyping:

1. Interactive Systems

   - Hardware and Operating System - When doing application prototyping, both the builder and the system must respond rapidly to the user's needs. Batch systems do not permit interaction and revision at a human pace. Hardware and associated operating systems tailored to on-line interactive development are ideal for software prototyping.

2. Data Management Systems

   - A Data Base Management System provides the tools for defining, creating, retrieving, manipulating, and controlling the information resources. Prototyping without a DBMS is inconceivable!

3. Data Dictionary

   - A Data Dictionary provides standardization of data and file locations and definitions, a cross reference of application programs, and a built-in documentation capability. These are essential to managing the corporate resources and extremely useful when prototyping.

4. Generalized Input and Output Software

   - Easy to use data entry, data editing, and screen formatting software are extremely helpful in the application prototyping process to allow the programmer to sit down at a terminal with a user and interactively create the user's screens or menus.

   - Powerful, easy-to-use report writer and query languages provide a quick and effective way of retrieving and reporting on data in the system. A report writer that uses default formats from very brief specifications is most useful in the initial prototype.

   - A powerful graphics capability can be extremely useful for the display of data in a more meaningful graphical format.

5. Very High Level (Fourth Generation) Languages

   - Traditional application development languages such as COBOL may not be well suited for software prototyping because of the amount of code that has to be written before the user sees any results.

   - Very powerful fourth generation languages that interface directly to a data dictionary for their data definitions are ideal. One statement in this high level language could realistically replace 20-50 COBOL statements. This reduces the amount of code a programmer has to write and maintain and speeds up the development process.

6. Documentation Aids

   - Tools to aid in the maintenance of programs written in a 4GL.

   - Tools to aid in maintaining user documentation on-line.

7. Libraries of Reuseable Code

   - A library of reusable code to reduce the amount of redundant code a programmer has to write is an important prototyping resource.

   - This code could represent commonly used routines made available to programmers.

Hewlett-Packard's Tools for Prototyping

Hewlett-Packard is one of the few vendors that supplies the majority of the tools needed to effectively do software prototyping.

* Interactive Systems

  - HP 3000 Family of Computers
  - MPE Operating System

* Data Management Systems

  - IMAGE/3000
  - KSAM/3000
  - MPE files
  - HP Silhouette/3000
  - HP Access Central, HP Access

* Data Dictionary

  - Dictionary/3000

* Generalized Input/Output Software

  - VPLUS/3000
  - QUERY/3000
  - REPORT/3000
  - INFORM/3000
  - HPEASYCHART
  - DSG/3000

* Very High Level Languages

  - TRANSACT/3000

* Documentation Aids

  - EDITOR/3000
  - HPSLATE
  - HPWORD
  - TDP/3000

Additional Prototyping Tools Available from HP Third-Party Vendors

* Data Management Systems and Associated Utilities

- ADAGER	Adager
- CARESS, INTACT, SILHOUETTE/3000	Carolian Systems International, Inc
- DBACE	Snodgrass Consulting
- DBAUDIT, SUPRTOOL	Robelle Consulting Ltd.
- DB GENERAL	Bradmark Computer Systems
- DBMGR, IMSAM, OMNIDEX, CAPCHG	Dynamic Information Systems, Corp.
- DBTUNE (Europe Only)	HI-COMP
- HSC-COPYDB	Hawaiian Software Company
- IMAGINE	Technalysis Corporation
- MINISIS	Systemhouse Ltd.
- MIRAGE (HP 150)	Datasoft International
- PC/IMAGE (HP 150)	Advanced Data Services
- RELATE/3000	CRI, INC.
- SPEEDEX, SPEEDBASE (HP 150)	Infocentre

* Generalized Input/Output Software

- DATADEX/3000	Dynamic Information Systems, Corp.
- EASYREPORTER	Infocentre
- ENVY, HELPER	System Works, Inc.
- INDEX PLUS	Spectrum Solutions
- MONITOR, MISTRAL (HP 150)	Datasoft International
- PAL DATA REPORTER	Gentry
- PRESENTATION GRAPHICS	ARENS
- PRW/3000	Infotek Systems
- QUIZ, THE EXPERT, GRAPHICS	COGNOS
- RELATIONAL QUERY/3000	Upland Software
- SCREEN/3000	RMS Business Systems
- WHAT-IF	CIBAR, Inc.
- THE WRITE STUFF	PROTOS Software Company

* Fourth Generation Languages and Utilities

- ARTESSA/3000 (Europe only)	RAET Software Products
- CBAS/3000	Comprehensive Systems, Inc.
- FASTRAN (TRANSACT Compiler)	Performance Software Group
- FLEXIBLE	Sages American Group
- INSIGHT II	Computing Capabilities Corp.
- LL'SPIRIT	Singapore Computer Systems, PTE.LTD.
- PAL FAMILY	GENTRY
- POWERHOUSE (QUICK)	COGNOS
- PROGSPEC/3000 (COBOL Gen.)	Productive Systems
- PROTOS (COBOL Generator)	PROTOS Software Company
- Q-PLUS	Los Altos Software
- RELATE/3000 APPLICATION BUILDER	CRI
- SPEEDWARE, MICROSPEEDWARE	Infocentre
- THE SYNERGIST	Gateway Systems Corp.

* Documentation Aids

- DOCUMENTOR (Part of SPEEDWARE)	Infocentre
- LARC	LARC Computing
- QEDIT	Robelle Consulting Ltd.
- ROBOT/3000	Productive Software Systems, Inc.
- S/COMPARE	Aldon Computer Group
- SPEEDDOC, SPEEDEDIT	Bradford Business Systems, Inc.
- TESS/AIDE	Computer Consultants and Serv. Center

The preceding lists of HP third-party software are not 100% complete. The majority of the listed software was derived from ads placed in SuperGroup Association Magazine, Interact Magazine and The Chronicle. Please consult the Hewlett-Packard Business Systems Software Solutions catalog (Part # 30000-90251) for additional information.

Summary

Prototyping is truly a "state-of-the-art" way of developing applications.

- Software prototyping promotes an interactive dialogue between the users and the programmer, which results in a system being developed more quickly, and results in an interactive development approach which is friendlier for the end user.

- The prototype provides a live working system for the users to experiment with instead of looking at lengthy specifications.

- The users are provided with an early visualization of the system which allows them to immediately use it.

- The users are allowed and even encouraged to change their minds about user interfaces and reports.

- Maintenance is viewed right from the beginning as a continuous process and because the prototype is usually written in a very high-level language, changes are faster to locate and easier to make.

- Software prototyping results in:

  * Users who are much more satisfied and involved in the development process.
  * Systems that meet the user's requirements and are much more effective and useful.
  * Improved productivity for all those involved in software prototyping: the user/designers and the systems/builders.

Biography

Orland Larson
is currently Information Resource Management Specialist for Hewlett-Packard. As the data base and application development specialist for the Information Systems Tactical Marketing Center he develops and presents seminars worldwide on data base 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 data base 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.

Bibliography

Boar, Bernard H., Application Prototyping: A Requirements Definition For The 80's, John Wiley & Sons, New York, New York, 1984.

Canning, Richard G., "Developing Systems By Prototyping," EDP Analyzer (19:9) Canning Publications, Inc., September 1981.

Jenkins, A. Milton, "Prototyping: A Methodology For The Design and Development of Application Systems," Division of Research, School of Business, Indiana University Discussion Paper #227, April 1983, (41 pages).

Jenkins, A. Milton and Lauer, W. Thomas, "An Annotated Bibliography on Prototyping," Division of Research, School of Business, Indiana University Discussion Paper #228, April 1983, (25 pages).

Larson, Orland J.,"Software Prototyping - Today's Approach to Application Systems Design and Development," Proceedings 1984 International Meeting HP 3000 IUG, Anaheim, California, February 26 - March 2.

Martin, James, Application Development Without Programmers, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

Naumann, Justus D. and Jenkins, A. Milton, "Prototyping: The New Paradigm for Systems Development," MIS Quarterly, Vol. 6, No. 3, September 1982.

Naumann, Justus D., and Galletta, Dennis F., "Annotated Bibliography of Prototyping for Information Systems Development," Management Information Systems Research Center Working Paper (MISRC-WP-82-12), September 1982.

Podolsky, Joseph L., "Horace Builds a Cycle," Datamation, November 1977, pp.162-186.

Wetherbe, James C., "Systems Development: Heuristic or Prototyping," Computerworld, Vol. 16, No. 7, April 26, 1982.

A DATA DICTIONARY ARCHITECTURE FOR THE YEAR 2001

Leon Leong
Hewlett Packard
Information Networks Division
Cupertino, California USA

Summary

Data dictionaries will be a critical service in managing a company's information network. Data dictionaries are being applied in many areas: fourth generation languages, report writers, query tools, network directories and transparent network access products. Thus choosing a data dictionary with an appropiate feature set can be the key to a well managed information network. This paper gives an overview of several features that have been included in Hewlett Packard's new dictionary product, System Dictionary, how they function, and what the features can be used for.

Introduction

A data dictionary is a central repository of information about the data and resources that a company has on its computer systems and network. Dictionaries are used to store, manage and organize this information. The information about the data and resources that a company creates and uses is called "metadata"; metadata means data about data. Metadata can include information about the name, size, structure, use, ownership, location and other attributes of a company's data and resources.

Typical objects that are documented in a company's information environment include:

- data base schemas,
- file and record layouts,
- data element descriptions and their use,
- computer nodes and device configurations,
- network topology,
- users an their security information.

A dictionary does not contain the actual object; just a description of the object. For example, a company may have a field in a file called "PART-NUMBER". A data dictionary would contain information such as,

- size (10 bytes),
- data type (ascii characters),
- display length (10 characters),
- owner (manufacturing),

whereas the file would contain the actual part numbers, for example 3225490001.

Subsystems which have objects being described in a data dictionary may have their own method for describing its structure and usage. For example, a data base has a schema, a network has a directory. These other methods, however, are very specialized for the particular subsystem; for example, it is not possible to describe the network topology in a data base schema, or vice versa. A data dictionary provides users a common format for describing metadata, such that it can be shared, and not redefined for every subsystem.

At first impression, one would guess that the users of a data dictionary would be individuals such as data base administrators, system analysts and programmers. However, there are many different types of products which can also be called "users" of a data dictionary:

- Fourth Generation Languages

  A fourth generation language (4GL) can use a data dictionary to resolve data definitions at compile/load time. A 4GL programmer only needs to know the name of the files, data bases or data elements that they want to use; the 4GL then retrieves from the dictionary the data type and structure information when the program is compiled. Inconsistencies between programs are reduced because there is a single source of definitions - the data dictionary.

- Ad Hoc Query Tools and Report Writers

  An ad hoc query tool/report writer will use a data dictionary to display to a user possible data to report on. For example, the query tool will, based on a user id, determine from the data dictionary what that user is allowed to access. The query tool will then retrieve from the dictionary the files, data bases and data elements and display them to the user. The user will then choose what to report on. The query tool then uses the metadata in the dictionary to determine the optimal access path.

- Application Customization

  Many applications tend to be very specific to an organization's operation. However, if an application were to utilize the metadata in the dictionary, then the dictionary can become a customization tool for the application. For example, if all headers and screen field names were stored in a data dictionary then a subsidary of a company in a different country could change the application to their native language, just by changing the metadata in the dictionary.

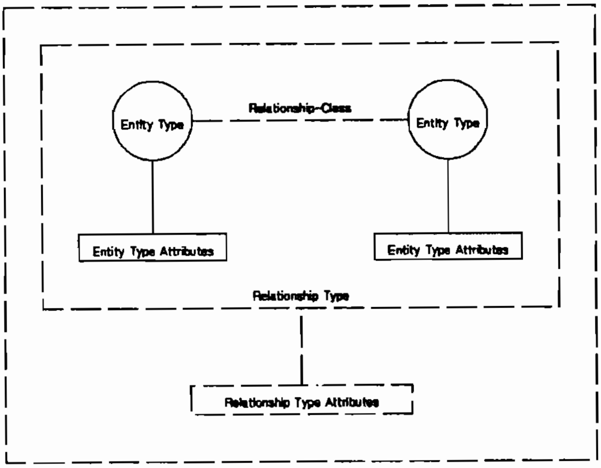
- Networks and Transparent Access

  A dictionary can be used to describe the location of nodes, and the topology of a company's computer network. In addition, a node's device configuration could also be included. This has potential uses for network configuration and management. By combining the network information with data structure information, applications can implement transparent network access for their end users.

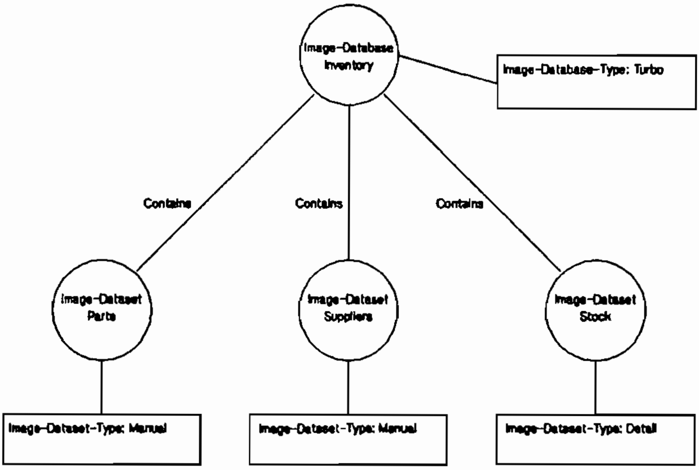
The uses of a data dictionary are many, and is only limited by the imagination of the people using it. But what of the features needed of a data dictionary, to make it usable for all these applications? Loading a dictionary can be an expensive operation; a dictionary should be evaluated for its feature set before being used as a central part of an information network. This paper describes some specific dictionary features which have been put into Hewlett Packard's new data dictionary product, System Dictionary and some the uses and benefits of those features.

Entity Relationship Model

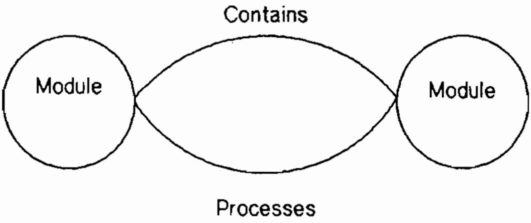
System Dictionary is based on the Entity-Relationship model. The Entity-Relationship model is composed of entities, relationships and attributes. Entities are objects in a company's information network. Every entity will belong to an entity type. An entity type is a template which describes the attributes associated with an entity. Relationships describe logical associations between entites. Attributes are pieces of information about either entities or relationships. Similarly, every relationship belongs to a relationship type; a relationship type is a template which describes the attributes associated with a relationship.



As an example, suppose a company has an Inventory Image database. The database is composed of three datasets: manual masters Parts and Suppliers, and a detail Stock. In the Entity-Relationship model, Image-Database and Image-Dataset will be entity types. Inventory is an entity of the entity type Image-Database. Parts, Suppliers and Stock are entites of the entity type Image-Dataset. The relationship type Image-Database Contains Image-Dataset, has three relationships in this example: Inventory contains Parts, Inventory contains Suppliers, and Inventory contains Stock. An attribute for the entity type Image-Dataset would be Image-Dataset-Type; for the entities Parts and Suppliers, the attribute would have a value of "manual", and for the entity Stock, the attribute value would be "detail".

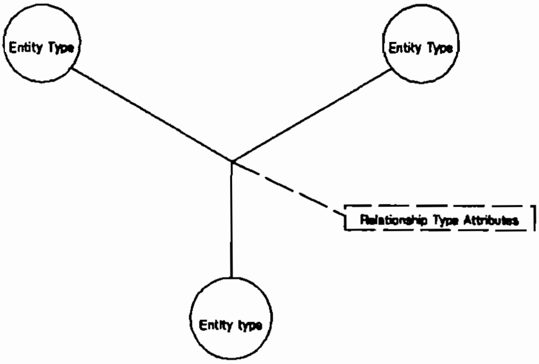


System Dictionary allows more than one relationship type to be established between entity types. For example, a code module may process another code module, or a code module may contain another code module. In order to allow for more than one relationship between entity types, System Dictionary includes, as part of the definition of a relationship type, an additional descriptor known as a relationship class. In the Image example, only one relationship type exists between the entity types Image-Database and Image-Dataset: Image-Database Contains Image-Dataset. Two relationship types exist between Modules: Module COntains Module, and Module Processes Module. The relationship classes are: Contains, and Processes.



Example of multiple relationship types
between entity types

Most relationship types will involve only two entity types. However there are cases in data modeling where, for integrity and consistency purposes, three or more entity types are needed in a single relationship type. System Dictionary allows a user to have up to six entity types in a relationship type.



A Three-Way Relationship Type

Extensibility

System Dictionary comes with a predefined set of entity types, relationship types and attributes; this predefined set is also known as the "core set". Thre core set of System Dictionary is extensive enough to cover the basic subsystems on the HP 3000. The core set includes structures which cover:

- MPE file system,
- KSAM files,
- Image/3000 data base management system,
- Network node locations,
- Code modules,
- MPE accounting structure,
- Cobol data structures.

The core set provides a stadard structure for most customers to work within. However, every customer's information model will be a superset of the core set; in addition, probably no two customer's information model will be exactly the same. Consequently, as customers adapt System Dictionary for greater use in their information network, they will want to extend the dictionary with their own entity types, relationship types and attributes to match their data model. In order to support this, System Dictionary allows a customer to create new entity types, relationship types and attributes. In addition, System Dictionary will allow relationship types to be established between existing core set entity types and customer entity types, and for customer defined attributes to be added to core set entity types and relationship types.

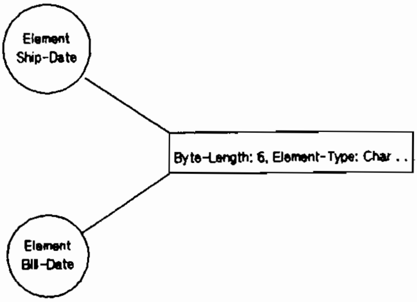
As another part of the extensibility feature, System Dictionary allows customers to specify default values for attributes, and also allows the specification of edit values for attributes. This capability provieds a form of integrity checking on attribute values entered. For example, the core set attribute Image-Dataset-Type will have the edit values "Manual", "Automatic" and "Detail" as its edit values. If any other value is entered for this attribute, an error will be returned. Customers can modify the edit values on core set attributes if they desire.

System Dictionary cannot anticipate every future use of the product today, let alone the uses by the year 2001. Extensibility allows the customer to grow their dictionary to meet their needs today and in the future, as they make more and more of their applications dictionary based.

Naming

Naming is an important part of the System Dictionary architecture. The most basic function of System Dictionary is as a name server. Users supply a name to the data dictionary system, and the dictionary system returns a set of attribute values for that name. Every entity in System Dictionary has a primary name; that is the name that it was created with.

System Dictionary supports "synonym" names for entities. Customers will often have situations where they have multiple names for an entity, and they would like to search the dictionary based on any one of the names. When a user gives a synonym name for an entity, the attribute values returned are the same as if the entity's primary name were used. For example, a customer may have a data element named "Ship-Date" in one application; in another application, the data element may be known as "Bill-Date". Since the name Bill-Date refers to the same data element, it should be created as a synonym for the data element Ship-Date. In System Dictionary, a user can create synonyms for any entity that they own; when the original entity is deleted, all the synonyms are also deleted.



System Dictionary also provides for "alias" names for entities. Alias names are different than synonyms, in that aliases are not used as keys into the dictionary. Aliases are attributes of an entity. Aliases are most useful when an entity's primary name is changed, due to naming conflicts. In the case of a naming conflict, an entity may be renamed, with the original name being assigned as an alias to the entity.

System Dictionary provides another naming feature: internal and external names. Customers will often have operations which span many countries. Each country will have its own language and terms that they use. A local data administrator may customize an application by modifying the names in the dictionary. However this can lead to problems when the originator of the application wants to make some modifications or develop utility programs which operate from the dictionary. System Dictionary provides two open modes to solve this problem: internal and external name mode.

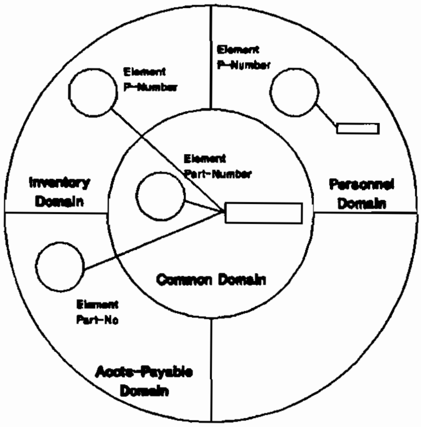
When a user creates a dictionary object, such as an entity, both an external and internal name can be supplied. If no internal name is supplied, then it defaults to the external name. External names can be modified to conform to the native language of the user; internal names can never be modified. Retrieval from the dictionary will be via the name open mode. Thus application developers which develop softwarebased on names in the dictionary can always use internal names, while the end users can use the external names.

Domains

Domains are separate name spaces in the data dictionary. Domains are useful for partitioning the dictionary by application system. Customers will often have many different application systems already in existence before purchasing a dictionary, or will have several application systems in development concurrently. Domains aid in migrating multiple applications to System Dictionary. These application systems will usually have conflicting names and definitions For example, an inventory application may have a data element "P-NUMBER", which is used to represent the object Part-Number. A human resources application may also have a data element "P-NUMBER", which in its context is used to represent the object "Personnel-Number". A third application, such as an Accounts Payable system may have a data element "Part-No"; Part-No refers to the same data element in the Inventory application known as "P-NUMBER".

These three application systems may already coexist when a dictionary is purchased to manage the overall information environment. Attempts to load the data definitions for the three application systems in the same domain lead to a variety of name conflicts. This would force the resolution of the name conflicts before the dictionary could be utilized, and it would also require changes on two the application systems. Domains are a useful feature to solve this problem. An INVENTORY domain could be created for the Inventory application, a HUMAN-RESOURCES domain for the human resources application, and an ACCOUNTS-PAYABLE domain for the accounting application. Since each domain is a separate name space, no conflicts would occur.

In order to promote the sharing of data definitions, however, a "common" domain needs to exist in the data dictionary. Users are allowed to link entities and relationships in the common domain. This linking allows the attribute definitions to be shared, yet it also allows each application system to keep its own name for entities. Local and common domain entities and relationships can be linked over time, and application system names adjusted over time as the application systems evolve. When an application system is moved to another node in the network, its definitions are separated out nicely by domain.

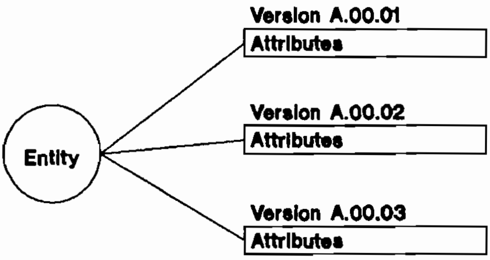


Linking definitions between domains

Version Control

Application systems are "living" systems. Application systems change, as a normal part of user feedback, growth, changing business environments and perhaps governmental regulation. A data dictionary should support the application system life cycle by having a version control feature. Version control as the name implies, allows entities and relationships to have multiple versions of values for attributes. For example, in version A, the element "Zip-Code" may have a length of five digits, while in version B of an application, the element may have a length of nine digits.

System Dictionary supports the concepts of version control. Every domain may have more than one version of entity and relationship definitions. Each version in a domain has a status: test, production, or archival. Within a domain, there may be as many test or archival versions as a user may want, but only one version may have its status set to production. Definitions in test versions may be modified, while a version which is set to a production or archival status becomes "read-only".



When a user wishes to access definitions in System Dictionary, the user must first open System Dictionary. The System Dictionary open operation requires the specification of a version in a domain, to retrieve the defintions from. The version can be specified by one of two methods: one method would be to specify the name of the version; the other method is to specify a version status. System Dictionary will look for the version which was last set to the specified status to retrieve definitions. A dictionary without version control requires all applications to upgrade to a given version at the same time. System Dictionary's version control feature allows each application to upgrade to the version at a time which best suits the user.

Dictionary Security

System Dictionary's security system is based on the ownership and capability. A user of the System Dictionary is known as a "scope". Every entity and relationship decides what level of access, also known as sensitivity, that other scopes may have. There are three sensitivity levels: public read, which allows any other scope read access; public modify, which allows any other scope read and modify access; and private, which does not allow any other scope access. In the case of private sensitivity, System Dictionary allows the owner of an entity or relationship to explicitly assign on a scope by scope basis, read or modify access.

Domains have two sensitivity levels: public modify and private. In the case of public modify, any scope has access to the domain; in the case of private sensitivity, only the owner scope may access the contents of the domain.

Every scope is assigned a set of capabilities. These include:

- Customization, which allows a scope the ability to create, modify and delete entity types, relationship types, attributes and relationship classes,

- Domain, which allows the creation and deletion of domains,

- Version Control, which allows the creation and deletion of versions,

- Security, which allows the ability to create and delete scopes,

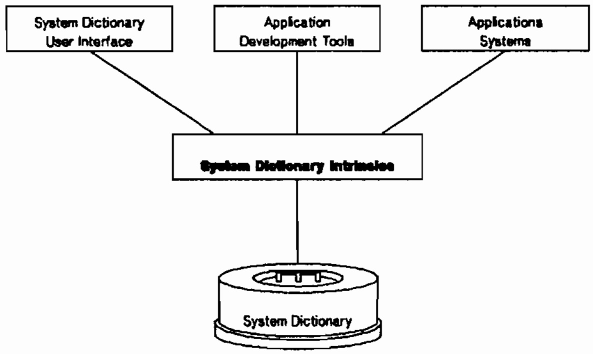
- Create, which allows the ability to create, modify and delete relationships and entities,

- Read, which allows the ability to read entity and relationships.

System Dictionary has one "superscope", known as the dictionary administrator (DA) scope. The dictionary administrator scope has full control over System Dictionary; this scope can perform any type of access on any domain, version, entity type, relationship type, attribute, relationship class, entity or relationship regardless of who the owner is.

Programmatic Access

An important feature of System Dictionary is that the product provides a standard, stable, and supported set of library routines, known as System Dictionary intrinsics. System Dictionary intrinsics provide independent software vendors and programmers the capability of accessing the dictionary contents without having to know how System Dictionary's internal structures are organized. By using the intrinsics, ISV's are protected from changes in the internal structures which may occur as data management technology improves, and System Dictionary takes advantage of those improvements.



Intrinsics provide a method for integrating applications with the dictionary and thus provide a more active environment, whereby changes in the dictionary can be reflected immediately in the applications. Active dictionary environments will be commonplace as customers evolve their information networks to the year 2001; intrinsics are a method for implementing it.

Summary

This paper has touched upon several of the features incorporated in Hewlett Packard's new dictionary product, System Dictionary, and what benefits the customer derives from the features: flexibility, migration, and growth to an active dictionary environment.

Biography

Leon Leong has been with Hewlett Packard for past seven years. He is the R & D project manager for the System Dictionary project.

MICROCOMPUTER WORKSTATIONS.
THE PATH TO FULL INTEGRATION.

Richard Linnett
Cognos Inc. Ottawa Ontario Canada.

Summary.

With the entry into the market by major computer manufacturers, microcomputers came out of the hobbyists den and became a legitimate tool in the office environment. A large number of microcomputers have been installed to handle small applications within an office - word processing and spreadsheet type applications. To a large extent these microcomputers are completely self contained, any data required from existing minicomputer systems being promoted into the system manually.

Many microcomputers are now being used not only for standalone wordprocessing, but also as replacements for terminals connected to central minicomputers or mainframes.

Exchange of data between minicomputer and microcomputer is becoming a major requirement - whether it is the extract of data from the corporate database for inclusion in a wordprocessor report, or the transfer of files between the two environments.

We are now seeing tools on the market that address the data transfer requirements. Within the next few years we should expect to see application development tools on the market that go far beyond file transfer to full integration.

This paper looks at the state of microcomputer integration, and paints a broad scenario of how we can expect the situation to change over the next few years as we move towards true integration.

What is MicroComputer Integration?

Much attention is being paid to the integration of software packages on the microcomputer, and we now have a number of integrated wordprocessor, spreadsheet and communications products available. These products, despite the criticisms, have made a step in the direction of integration on the microcomputer. However, the integration stops at the microcomputer, there is no real integration into the minicomputer world.

Integration of microcomputers into an office minicomputer environment entails far more than an easy to use front end, a full distributed environment is needed where:

> The minicomputer and Microcomputer work together in processing an application. A micro can typically support far better end user interaction than a minicomputer. A minicomputer can support larger databases and can control sharing of data across multiple users far better than a microcomputer. To be fully integrated, these capabilities must be used to the best ability.
>
> Load sharing between minicomputer and microcomputer has to be intelligent - load sharing should be self balancing depending on the load of the systems at the time of processing.
>
> There will always be a number of dumb terminals in use, or times when the central minicomputer system is unavailable. An integrated mini / micro environment must also be able to adapt to this.
>
> Data exchange between microcomputer and minicomputer will not be on a "batch" basis as with file transfer, but will be on a transaction basis.
>
> Data in the environment has to be available using terminology that the user understands. A data dictionary will have to be powerful enough to shield the user from the intricacies of Image database structure, allowing dynamic user views of data to be developed independantly of where or how data is stored.

Much of this could be achieved now with extensive coding at both mini and micro to develop an integrated application, but the magnitude of the work required precludes it. Future developments in application development software will make this intelligent use of all resources available to all users.

Where Are we coming from?

Microcomputers and minicomputers cannot really be considered as being integrated now. The uses of both environments is to all intents, completely divergent.

We have a very traditional environment in the minicomputer world:

> There is still has much faith in COBOL, structured programming techniques and complex databases.
>
> The minicomputer DP department suffers from the same problems as a large mainframe department - large maintenance loads and long backlogs of work requests are making the department appear very unresponsive to user demands.
>
> These backlogs are causing the DP department to move from COBOL to products that can aid productivity. But even with the increased turnover in development, the backlog is not being reduced because more and more requests are being added to the backlog.
>
> Many DP departments are experimenting with, or actively encouraging users to develop their own ad hoc requests using the same development tools as the DP are now using. However, with data in the corporate database normalised in a way that makes for efficient processing, it is not readily meaningful to the user.

The microcomputer environment is very much at odds with the structured minicomputer world.

> Small spreadsheets or Basic programs are developed as needed and thrown away just as quickly.
>
> A knowledgeable spreadsheet user can build very complex models that reflect an entire operation, and that model can be adjusted quickly to incorporate any operational changes that occur.

As long as these two worlds stay apart, the different methodologies can co-exist, the distain of the professional DP professional for the ignorant microcomputer user balanced by the distain of the microcomputer user for the DP department that never gets anything done.

Where are we now?

With few exceptions the level of integration between microcomputer and minicomputer is now limited to the downloading of data from the minicomputer for processing in a microcomputer application. Typically this involves:

A mini program - COBOL or possibly a report writer extracts data from the corporate database and build a file on the minicomputer.

Transformation of data into the optimal format for use by the microcomputer is not always attempted. Flat ASCII or DIF files are the common transfer medium.

Actual data transfer is managed using a terminal emulation package. The microcomputer, which may be capable of 750,000 instructions per second does no more than wait on a terminal port and transfer the occasional character to a disk file.

The spreadsheet user then has the data available for use.

There are many problems with this situation:

A program needs to be written on the minicomputer to extract the data. That requires knowledge of the host system and frequently scheduling of work by DP staff to write the programs.

Communications is a black box to most people. To configure a terminal emulation package to match the protocols used by the host system requires knowledge that most users do not have and do not care to have.

Transformation of the data to ASCII or DIF formats looses format information. When loaded into the destination spreadsheet, the user frequently sees misaligned columns and truncated values.

Very few sites look at the uploading of data back to the minicomputer as a viable option. Considerable problems exist with data integrity if significant portions of a database are offloaded from the host to distributed microcomputers.

Although these integrity problems preclude file transfer as a viable basis for distributed processing, there will be instances where the nature of the data allows for file transfer. As an example, a corporate budget file could be safely distributed across the individual cost centres assuming that a cost centre could only update that cost centres budget.

The first integrated packages

By limiting the process to data download only, many of the problems of data integrity can be avoided. For that reason, the first products that we can expect to see appear on the path to shared processing will appear in the guise of more efficient extract programs than available now.

As previously, mentioned, a microcomputer acting in terminal mode wastes a large amount of processing power. The more powerful microcomputers that have become available during the last year, and microcomputers based on announced but not yet available processors will have even more spare processing power during what is basically terminal mode.

By moving to multitasking (if not multiuser) operating systems, the option will be there to move a file transfer program into the background:

> This will allow the user to continue with micro based work during the time transfer takes.
>
> By hiding the file transfer inside such an environment, probably with an attractive windows interface, the microcomputer will become more effectively used than now. This is however very much a dead end.
>
> Nothing is done using this approach to offload the host, and the problems of currency control are not addressed.

The alternate, and preferable approach to using the spare microcomputer power is to offload the host system as much as possible.

> Definition of the extract request will be offloaded completely from the host system. By formatting and editing the request on the microcomputer, considerable savings in host resources can be made while improving the user interaction.
>
> There is no need for the host processor to use valuable cycles to reformat the data into a microcomputer format. That work can adequately be managed during the time that the microcomputer is waiting for input.

Together, the two approaches provide a major advance in what is available now.

Concurrent with the improved data extract packages we can look towards intelligent terminal emulation for microcomputers that do more than simple terminal emulation.

Many users are now using microcomputer terminal emulation products to provide an easy to use front end to the HP3000. The command language being used to control access to the host, providing autologon facilities and navigation through the available applications by a series of menus.

We should be looking for combined minicomputer / microcomputer programs that take this menu driven approach a step further. As microcomputer users begin to require more access to the minicomputer, some integration of minicomputer and microcomputer operating system commands is needed. This could take the form of:

> a menu driven option to MPE commands, much along the lines of PAM on the HP150.
>
> menu driven equivalents to utilities such as FCOPY.

Beyond such an enhanced terminal emulation feature, we should be looking for intelligence being distributed to the microcomputer during application processing. With such a front end, it will be possible to make significant reductions in host processing and improve the user interface dramatically. It is possible to offload the following:

> Screen formatting - boxes, windows etc require a considerable amount of communications traffic in addition to the host processor time. Major improvements in turnaround could be managed by this process.
>
> Data entry editing. If a field is known to be numeric and within a certain range, it would be reasonable to make such checks on the microcomputer before transmitting the data to the host. By keeping the processing local until database access is required, the microcomputer can be used to its best, providing a fast effective front end to the host database machine.

Even with all data remaining on the host system, a significant housekeeping process is implied, as meaningful savings require screen formats and editing rules to be downloaded to the microcomputer. Changes in the central system would need to be propogated to all microcomputers when they use the system.

It is likely that a fourth generation product will be enhanced to provide this level of integration in a fairly automated manner.

These product initiatives would seem to be fairly simple at a first glance, they can be implemented using existing technology. They do however, provide a major improvement in the utilisation of microcomputers in the minicompter environment and set the basis for further product developments.

Further advances require developments in the following areas:

A Data dictionary will need to be available to the environment that goes beyond the simple file and element descriptions that are in use now:

> The data dictionary is going to need to be far more active than now. All data access will need to be routed through the dictionary if dynamic data formats, as required by the microcomputer user are to be supported.
>
> The data dictionary will need to support distributed data, and the dictionary itself will need to be distributed and replicated across several systems.
>
> If central data storage is to remain in complex data base formats such as IMAGE, the dictionary is going to need to hold information necessary to navigate across the database.
>
> User views of data, based on multiple file access and selection criteria will need to be available.
>
> Database security and integrity are major requirements of the DP department. Strong controls will need to be available in the dictionary.

The same application development language will need to become available on both the minicomputer and the microcomputer.

> This may not necessarily entail mirror images of the language being available on both systems. There are many features required of a language by a DP guru that are not at all relevant to the microcomputer user. Within limits, the fine tuning commands that the guru needs only lead to confusion and rejection of the language by a no DP user.
>
> Despite the question about how exactly a microcomputer language duplicates the minicomputer language, there is a strong need for the ability to develop and run the same application on both mini and micro. Many small applications would benefit from being transfered to a microcomputer, with the ensuing release of minicomputer resources.

Application development is frequently a resource hog, and the availability of a microcomputer development environm◆t is a good way to reduce the mini load.

A question does remain about the power of microcomputers being sufficient to fully support a development language that was based on the size and capabilites of a virtual memory minicomputer.

Communications is an area that requires some change before the true integrated environment can develop:

Standards in data communications are needed if the integrated environment is to be available freely. This is happening now with the OSI layered protocol being supported in more and more products. The IBM LU 6.2 peer to peer protocol will also drive more standardisation into the communications world.

Packet switch networks need to become more widespread. The cost reductions for long distance connections and data integrity available through an error checking link are necessary.

Error checking and handling need to become more a part of the communications environment, allowing some reliance on data transmission. X25 networks and the newer error checking modems are beginning to provide this integrity.

The current mess of inter microcomputer communications needs to settle before much progress can be made. Few software developers can make significant progress in development of network software until some standardisation happens, and until the software developers begin to adapt to the networks real multiuser versions of microcomputer software will not be available.

The Fully Integrated Workstation.

Given the required enhancements in the data dictionary, the "universal language", strong communications and microcomputer power we can look for applications software that will allow the power of the microcomputer to be integrated with the power of the minicomputer. We will then have a truly integrated microcomputer workstation:

Applications will be developed using a common language. It may be that the microcomputer user has a series of screen painters and application generators while the DP expert has a concise syntax that allows for very exact specifications, but the base syntax will be the same.

Once developed, the application will be able to run on either microcomputer or minicomputer, depending on the data volumes and the scope of the application.

If an application is implemented on the microcomputer, it will be possible to port the application to the minicomputer as data volumes grow.

If an application is run on the minicomputer, the microcomputer terminal will offload as much of the terminal management as reasonable. The same application will be run side by side on a terminal and a micro, yet the microcomputer will provide much faster response times, and an easier interface than the pure terminal.

Truly distributed processing will be possible, with data available locally on the microcomputer, from a departmental minicomputer, or even through the local minicomputer to the central corporate mainframe database.

With standardisation of communications links, and program to program communications across a link, it will become possible for programs from different vendors to be integrated to a level only dreamed of today.

Microcomputer software such as a spreadsheet or a word processing package will be able to access data from the central host system easily and without need for host system knowledge. Just as it is possible to move data from the word processor part of an integrated microcomputer package into the spreadsheet component, it will be possible to extract from the host database into the word processor and spreadsheet.

The two components, microcomputer and minicomputer will be connected through a very strong data communications link, and that link will provide an intelligent highway between the two.

Biography.

Richard Linnett has a wide background in the technical aspects of Data Processing, gained over eighteen years of practical experience with a number of organisations.

Richard Linnett has been with Cognos Inc for four years. During this time he has been responsible for the implementation of a number of projects to integrate different hardware and software environments for clients with mainframe, minicomputer and microcomputer applications.

He is now a Project Manager in the HP Marketing and Development Division of Cognos.

FROM DATA ANALYSIS TO TurboIMAGE DESIGN

Jos Witteveen, Glenn Pereira
Database Consultants europe BV.
Amsterdam, Netherlands

ABSTRACT

Because information is one of the company's most valuable assets in any organisation and the availability of information one of the highest priorities, good Data Analysis and proper Database Design have become critical success factors in the system development environment today.
This paper will describe how Data Analysis, Conceptual Access Path Analysis and Logical TurboIMAGE Database Design should be carried out in order to meet the information requirements of an organisation.

INTRODUCTION

TurboIMAGE/3000 has been developed to replace IMAGE/3000 in order to provide increased performance and functionality.
The database architecture remains as a two-level network structure with data set relationships between owners and members. It enables the HP/3000 DP professional, working in an environment in which data is shared across many people, departments and applications, to develop databases which can offer a number of significant advantages, such as :

- Data integrity in an online environment
- Central control of data
- Scope for future development
- Recovery facilities
- Easy-to-use application development tools

These advantages, however, can only be realised if the database design is carried out correctly with due regard to the natural structure of the data and the way it has to be accessed and maintained. The risk of getting it wrong is high and the costs of putting it right afterwards can be enormous.

ENVIRONMENT

A DATABASE is simply defined as a 'COMMON POOL OF SHARED DATA'.
Recognition of the fact that the organisation exists in a shared-data environment is the pre-requisite for building a database. In a shared-data environment the following principles apply:

1. Data exists in its own right and has relationships with other data.
2. Data is not exclusively owned by any department or application area, it must be available to any authorised person or function within the organisation.
3. Data has to be organised within the computer in a manner that reflects its real-world existence.
4. The objective is to hold each piece of data only once in the database files with exceptions allowed only for good reason and subject to appropriate control.

5. Applications are built with due regard to the need for control and synchronisation of data.

Figure 1 shows the System Development Cycle required in a Shared-Data, Interactive environment. The right side of the diagram shows the steps which are commonly used in Computer System Development ie. Activity Analysis, Application System Design and Structured Program Design. The left side shows a similar cycle of events for analysing the data (Data Analysis), designing the data structure within the limitations of the DBMS structuring rules (Logical Database Design) and the final physical design of the database (Physical Database Design). The centre of the diagram demonstrates the steps required to analyse how the activities (and eventual computer processes) need to use the data.

All design begins with proper and thorough analysis. It is necessary to collect the information using structured analysis techniques and feed the results into a structured, systematic design procedure. Before database design we are concerned with:

- the structure of the data, Data Analysis
- the way this data is used, Access Path Analysis.

DATA ANALYSIS

Trying to understand the structure and the characteristics of data has become a science in itself. It is widely practised today using a number of different techniques such as Data Analysis, Information Engineering and Normalisation, the latter being, perhaps, the best known.

In essence, analysis of data consists of recognising all the entities and attributes that exist within the scope of your project and, subsequently, the relationships that exist between entities.

An ENTITY is defined as anything of interest and about which data can be kept. Typically, an entity can be an object (building), person (employee), place (country) or event (flight).

An ATTRIBUTE is defined as an element that helps to describe the occurrences of one or more entities. Typically, attributes can be codes, names, amounts, prices etc.

A RELATIONSHIP is defined as an association between entities.
Whereas many types of relationship can exist, the two most common are the 'one-to-many' relationship and the 'many-to-many' relationship. Each relationship is drawn as a straight line between two entities with some symbol (arrow, crows-foot or trident) to show the 'many' of a relationship.

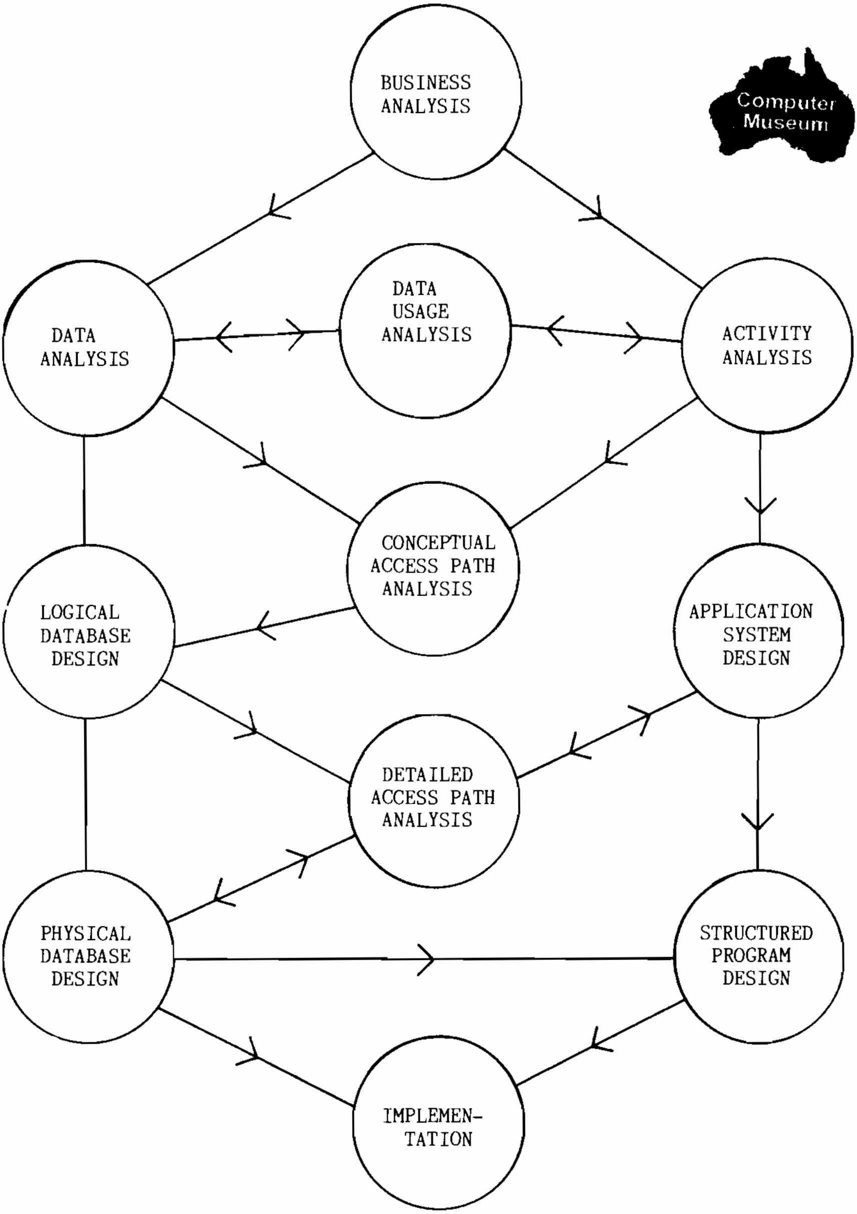


FIGURE 1 - The Systems Development Cycle in a Shared-Data, Interactive Environment.

If, for example, a customer can have many orders, but one order can be associated with only one customer, it may be represented as shown in Figure 2 below:

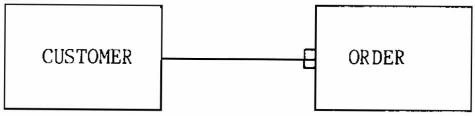


Figure 2 - The one-to-many relationship

If it was required to show a relationship between Supplier and Product where one supplier can supply many products and one product can be supplied by many suppliers, it may be represented as shown in Figure 3 below:

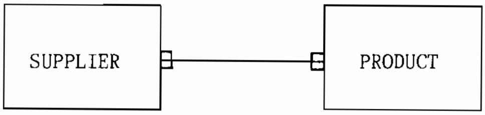


Figure 3 - The many-to-many relationship

A variation, which is commonly used, is the optional relationship. This shows that a relationship may hold in some cases, but not all. If, for example, a product may or may not be supplied by a supplier (ie. some products may be manufactured internally), the above straight line would be dotted at the product end, as shown in Figure 4 below:

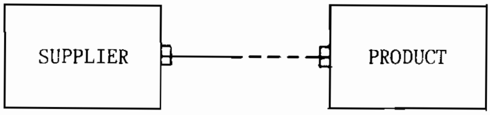


Figure 4 - Optional relationship

By identifying all the entities and relationships within the scope of your project, it is possible to build up a picture of the data in that area. This picture, which is normally called a Data Model, is a conceptual representation of the data - the real world - and it contains all natural associations.In addition to building the Data Model, all entities, attributes and relationships should be documented. This documentation, would contain precise descriptions, volumes, characteristics, special conditions etc. The attribute documentation should become, essentially, the Data Dictionary and should comply with any previous dictionaries set up within the organisation.

Figure 5 shows a Data Model for the Order Processing System.

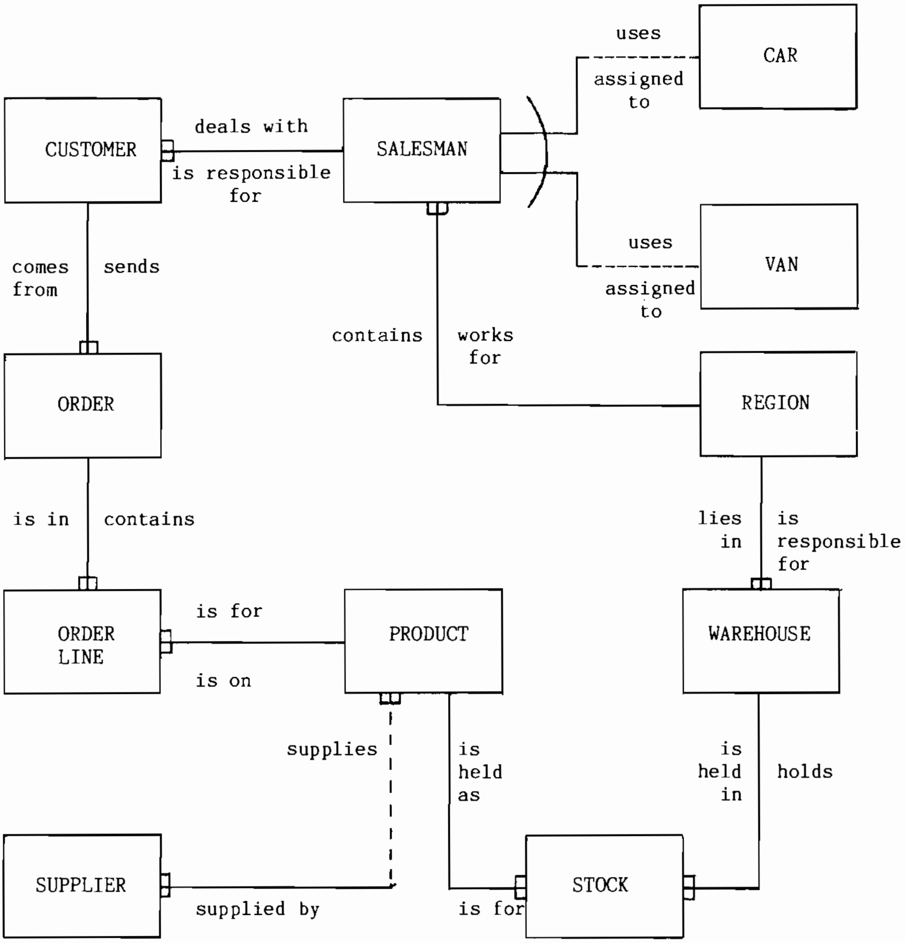


Figure 5 - Data Model.

ACCESS PATH ANALYSIS

Access Path Analysis is a technique used to identify how the data is accessed. It is carried out by drawing Access Profiles for all processes identified during the Activity (or functional) analysis.
Processes can be categorised as follows:

- RETRIEVAL Processes which, by definition, need to access data.

- MAINTENANCE Processes which need to insert, modify or delete data but, in doing so, would normally need also to access data (eg. validate that customer exists and credit status is OK before inserting an order).

The essential information required for logical database design is the access requirements ie. the ENTRY POINTS into the data structure and the NAVIGATION PATHS around the data structure. The overhead of updating the database is considered during Physical Design, when update performance will be evaluated.

An ENTRY POINT represents an initial entry into the data structure. It can consist of a search for one occurrence of an entity (normally using the unique identifier, eg. Customer Code) or a search for several occurrences (eg. all customers called Smith). In the latter case, the occurrences may be required in a particular sequence (eg. all customers called Smith in customer code sequence). Figure 6, 7 and 8 below, show how initial entry can be represented on an Access Profile. The single arrow indicates that one occurrence is accessed and the double arrow indicates that many occurrences are accessed. An 'S' next to the double arrow shows that the occurrences are accessed in sorted sequence:

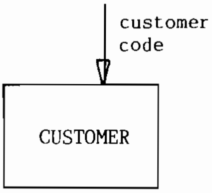


Figure 6:
Single Record
Access

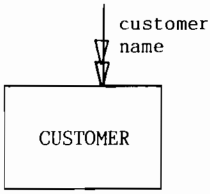


Figure 7:
Multiple Record
Access

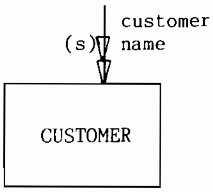


Figure 8:
Multiple Record
Access in Sorted
Sequence

A NAVIGATION PATH consists of one or more relationships that need to be used, by a process, to obtain all the data that it requires. Figure 9 shows the navigation path for a process that needs to enquire on orders, by customer, and display all the order lines in sequence.

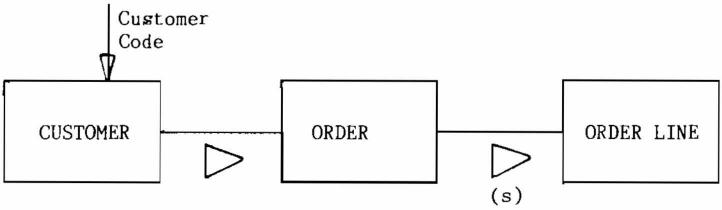


Figure 9 - Access Profile for Order Enquiry

In addition to drawing the access profiles, documentation forms should be completed for each process showing frequencies, volumes of data accessed, specific attribute usage within an entity, whether it is run in batch or on-line (if known at that stage), response time or run time constraints etc.

From this documentation it is necessary to compile a number of statistics and matrices which show the essential information required. Typically, these are as follows:

Usage Statistics - showing how frequently entities and relationships are used.

Usage Matrices - showing in which processes the entities and relationships are used and/or created.

Attribute Usage Matrix - per entity, showing where attributes are used and/or created.

Summary of Retrieval Access Requirements - showing all access paths required into and around the data structure.

LOGICAL DATABASE DESIGN

In order to separate the functional and structural design decisions from the physical design decisions, it is necessary to conduct an initial logical database design which produces a logical data structure that:

1. Is built in accordance with the TurboIMAGE structuring rules.
2. Contains all needed data.
3. Supports all access requirements.
4. Will be used as the input for physical database design.

Logical database design is carried out in two steps:

I - Refining the data model
II - Mapping to TurboIMAGE

REFINING THE DATA MODEL

The main objective of the refining process is to determine what is required of the data model, using the results of conceptual access path analysis.

During Data Analysis all relevant information is collected with regard to the entities, attributes and relationships defined within the scope of the analysis. The resultant data model should be reasonably close to the structure that needs to be mapped on to the database but would, normally, need to be refined in a number of ways as follows:

1. Eliminate many-to-many relationships from the data structure because TurboIMAGE does not support them. (In fact, very few Database Management Systems do).

   They are eliminated by the introduction of a new "in-between" entity (sometimes called a junction entity) and the creation of two one-to-many relationships in place of the many-to-many.

Figure 10 below, shows how the many-to-many relationships between Product and Supplier is expressed as two one-to-many relationships:

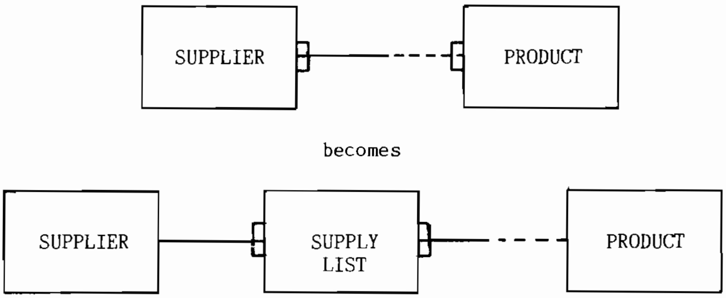


Figure 10 - Resolving Many-to-Many Relationships

Note:
The Access Paths for supplier - product need to be re-analysed.

2. Eliminate any entities, attributes and relationships which, although they belong to the conceptual Data Model, are not required for use by the computer application(s). This decision has to be made carefully, taking into consideration the following points:

   - all entities, attributes and relationships that are put into the database are likely to need maintenance to some degree.

   - due consideration must be given to future application development plans (say, next two years) and possible information requirements (new reports, ad-hoc enquiries etc.) that are not fully covered by the Terms of Reference for the application(s) currently being developed.

3. Add new relationships to the data structure if the access path analysis shows traversal between entities that are not directly related.

   For example, in the Data Model (Figure 5) there is no direct relationship between SALESMAN and ORDER (only an indirect relationship via CUSTOMER) but should there be a requirement to access ORDER from SALESMAN a new relationship could be added between SALESMAN and ORDER.

4. It may be possible to combine entities. Some judgement, based on experience, is required but two rules of thumb would be as follows:

   - where two entities are normally accessed together and their relationship, although one-to-many, is nearly one-to-one. This, typically, occurs where an order (or invoice) could have many lines or a customer could have many debtor numbers but, in practice, they almost always have one.

   - where two entities are very similar in terms of their attributes and they have the same relationships. For example, in our Data Model, Car and Van could be combined into one entity called Vehicle.

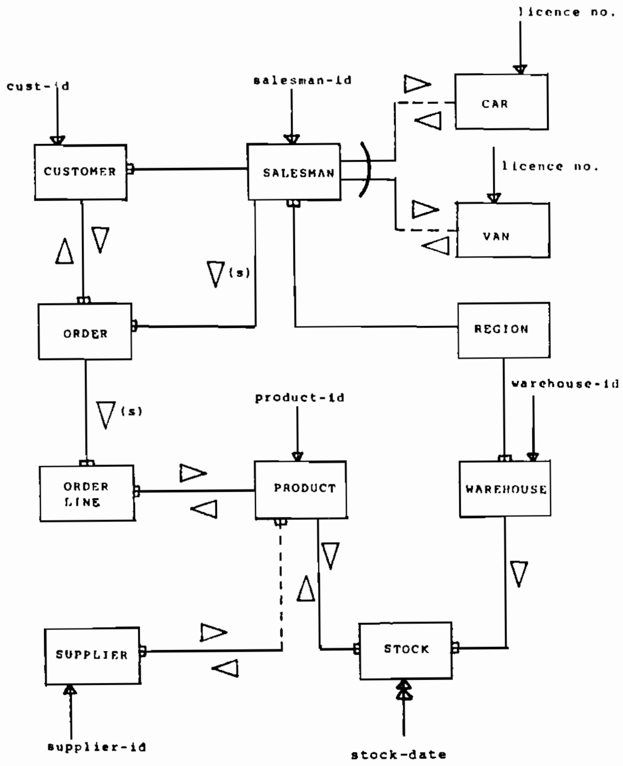


Figure 11 - Summary of Retrieval Access Requirements

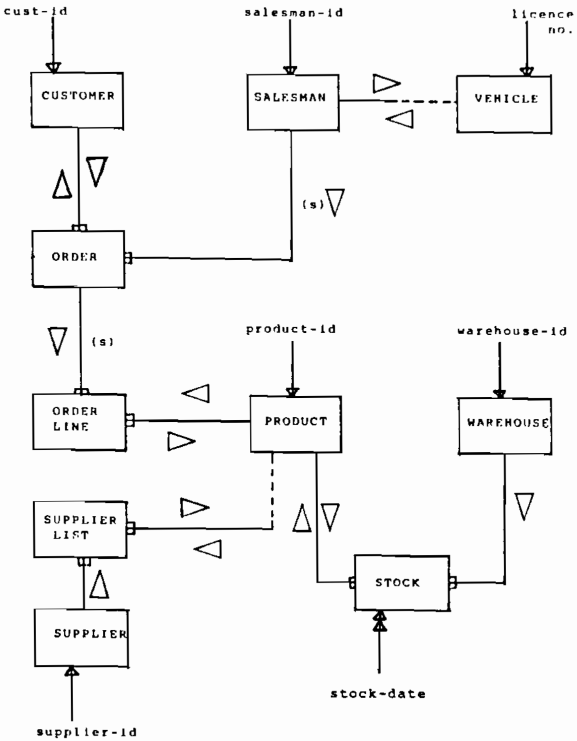


Figure 12 - Refined Data Model +
Summary of Access Requirements

In order to carry out the steps, outlined above, the information from the Data and Access Path Analysis is required.

Figure 11 shows a Summary of Retrieval Access Requirements for the Order Processing Data Model.

Figure 12 shows the Refined Data Model which has resulted from applying the above steps.

- The Supplier/Product relationship has been expanded into two one-to-many relationships.
- The Region entity has been eliminated.
- The relationships Region/Salesman, Customer/Salesman and Region/Warehouse have been eliminated.
- The relationship Salesman/Order has been added.
- The entities Car and Van have been combined to give a new entity called Vehicle.

MAPPING TO TurboIMAGE

The objectives of mapping are:

1. To translate the refined data model into a logical data structure that complies with the TurboIMAGE structuring rules.
2. To implement the access requirements into the logical data structure using the TurboIMAGE options.

Before the mapping is performed, it is essential to understand, very clearly, the structuring rules for TurboIMAGE.

1. STRUCTURING RULES

Within TurboIMAGE, two types of data sets can be supported, master and detail.
Relationships are supported between master and detail data sets.

A data set is a collection of records with the same record format. It is equivalent to what, in conventional terms, would be called a file with one type of record.

1.1. Master

A master data set must have a unique data field, the search field. TurboIMAGE supports calculated (hashed) access to a master data set using the value of the search field. A master data set may be linked to up to 16 detail data sets.
Two types of master data sets are supported, the manual master set and the automatic master set.

1.1.1. Manual

The manual master may contain more data fields in addition to the search field. A manual master set must be maintained by the application. If an entry needs to be added to an associated detail data set using a value of the search field that does not exist in the manual master, then the master record must first be created. Deletion of all detail entries will not cause the removal of the manual master entry.

1.1.2. Automatic

The automatic master set may only contain one data field, the search field. An automatic master is maintained by TurboIMAGE. When a data record is added to an associated detail data set using a value of the search field that does not exist in the automatic master, TurboIMAGE automatically creates a new entry in the automatic master for that value. Similarly when detail entries are deleted TurboIMAGE automatically removes corresponding entries from the automatic master. Typically, an automatic master set would be created to allow more efficient access to a detail data set for a particular search field eg. ORDER-DATE.

1.2. Detail

The detail data set can contain several data fields. A detail data set can be linked to a maximum of 16 master sets.

1.3. Relationships

A relationship or path is a means of connecting records, within the TurboIMAGE database, which relate to each other. A relationship can only exist between a master and a detail data set. One of the data fields within the record format of a master data set, the search field, must be included in the associated detail data set record format. The value of the search field must be unique within the master data set, but that value can be present in several record occurrences in the detail data set. TurboIMAGE will maintain a chain of pointers embedded within the data records for all entries which contain the same search field value.
This will enable access to all related data records of the detail data set for an occurrence of a data record of the master data set. The particular master and associated detail data records are referred to as a chain. The detail records in a chain may be sorted by a data field.

2. TurboIMAGE ACCESS METHODS

Although several access methods are supported by TurboIMAGE (TurboIMAGE ref. manual 32215-90050), the principal ones to consider, during the Logical Design stage, are as follows :

Access on master sets : I - serial, in the sequence in which they are physically stored.
II - calculated, using a search field value to find a unique record.

Access on detail sets : I - serial, in the sequence in which they are physically stored.
II - chained, all detail entries for one master record.
The sequence of the detail records in a chain may be controlled by defining a sort field for the chain.

3. TurboIMAGE SPECIFICATIONS

These specifications have been copied from the DATA MANAGEMENT specifications guide (32215-95002) :

- max. data item names per database : 1023
- max. data items per data entry (record) : 255
- max. data sets per database : 199
- max. detail data sets per master set : 16
- max. master sets per detail data set : 16
- max. search items per detail data set : 16
- max. entry size : 4094 bytes
- max. entries per data set : 2 billion (blk. factor 255)
- max. entries per chain : 2 billion
- max. characters per database name : 6
- max. characters per password : 8
- max. characters per data set name : 16
- max. characters per data item name : 16

The logical design stage is concerned with the facilities offered by the DBMS physical limitations (eg. CPU, disc-I/O etc.) will be taken into account during physical database design.
On-line access is the most important criterium when we have to consider alternative ways of implementing access possibilities.

4. MAPPING GUIDELINES

1 - Select all entities from the refined data model that only have 1:1 or 1:M (many) relationships with other entities and map them to a manual master data set.

2 - For each manual master select the search field.

3 - Map all remaining entities to detail data sets.

4 - For every detail set, implement the M:1 relationship that exists in the refined data model with the entities that were mapped to manual masters in step 1 and include the search field in the detail data record.

5 - Select all entities that were mapped to detail data sets in step 3 but have 1:1 or 1:M (many) relationships with other entities in the refined data model. Choose a suitable search field for these entities.

6 - Create automatic masters containing the selected search field for the detail data sets selected in step 5.

7 - Implement the 1:M relationships that exist in the refined data model between two entities that were mapped to detail data sets by relating them to the automatic masters created in step 6. Include the search field in the detail data record.

8 - Implement a relationship between manual master sets (1:1 relationship in refined data model) by creating a detail data set containing only the search items of the manual masters.

9 - Consider extra automatic masters for details on which direct entry is required or access via the existing master sets is clumsy.

10 - Remove all relationships (not the search fields) when the access profiles show that you only go from detail to master and never from master to detail.

11 - If more than one search item on a master set is required, consider mapping the master to a detail data set with automatic masters for the needed extra search fields. The original manual master will remain with only one field, the search field.

12 - Determine which master to detail paths should be sorted.

NOTES

If a detail data set has an optional relationship with an associated master, prepare a dummy (empty) master entry including a value for the search item because TurboIMAGE will not allow non-owned detail entries.

The logical data structure shown in Figure 13 was produced by applying the mapping guidelines on the refined data model and summary of access requirements in Figure 12. For each data set and relationship the appropriate guideline, that led to its determination, is indicated.

WHAT COMES NEXT ?

The logical data structure will subsequently be input to the physical database design process which is outside the scope of this paper. However, in summary, the physical design will be concerned with :

- performance optimisation
- considerations with regard to the environment (eg. CPU, disc-I/O, disc space)
- user design constraints (eg. response time, run time)

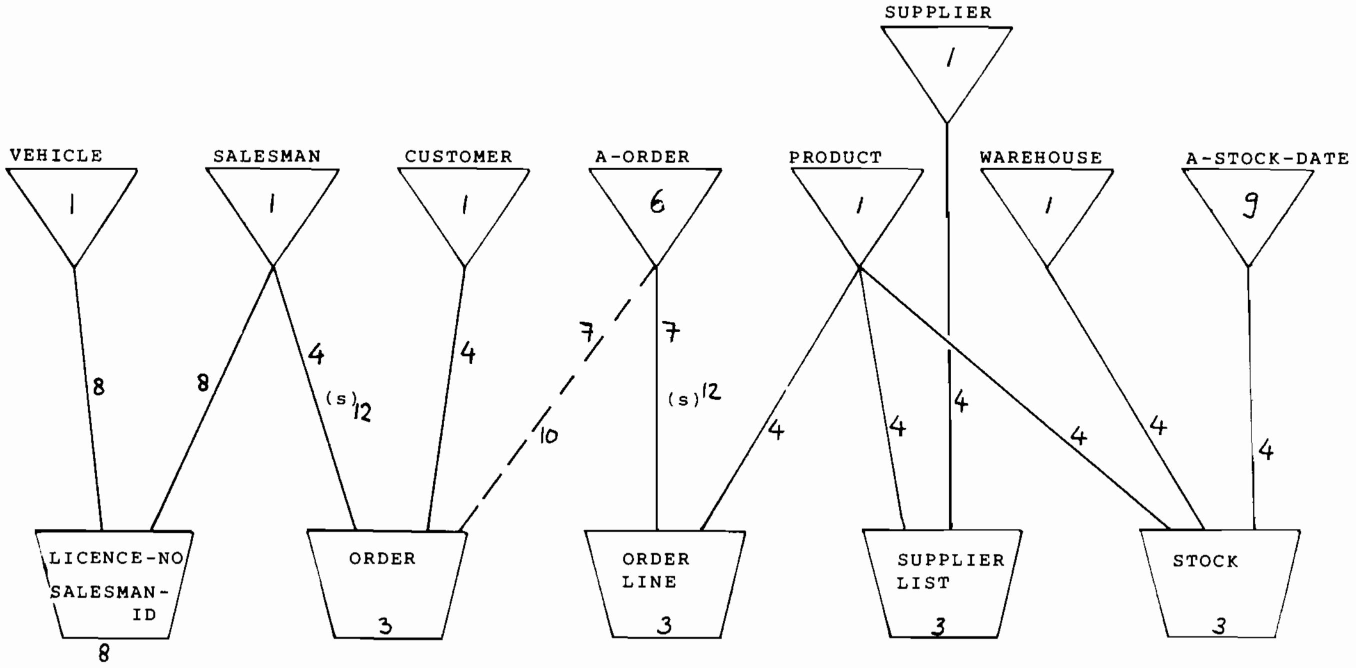


Figure 13 - Logical Data Structure for TurboIMAGE

JOS WITTEVEEN

Jos Witteveen has been working with database systems for the last six years, mainly on mini-computers. During the last three years, while working as a consultant for Database Consultants europe BV, he has been applying structured analysis and design techniques, particulary in the area of mapping conceptual data structures to a logical database design. His hardware/software experience includes Hewlett-Packard 3000 series and Digital VAX. With his previous employer he worked as a system manager providing technical consultancy and training to Hewlett-Packard installations.

GLENN PEREIRA

Glenn Pereira started working within EDP twelve years ago. During the last four years he has been applying structured analysis and design techniques in developing database systems.
His knowledge and experience includes: IMS/DL1, IDMS, IMAGE/3000, IBM/38 and VAX-11 DBMS.

BIBLIOGRAPHY

[Wierenga 1984] Wierenga, Hans, Physical Database Design, Amsterdam, Proceedings Decus, 1984.

[Pereira 1984] Pereira, Glenn, Logical Database Design, Amsterdam, Proceedings Decus, 1984.

[Green 1984] Green, Rego, White, Greer, Heidner, The IMAGE/3000 Handbook, Wordware, 1894 ISBN 0-914243-00-4

[Atre 1980] Atre, S., Database, Structured Techniques for Design, Performance and Management, Wiley, 1980

[Martin 1976] Martin, James, Principles of Database Management, Prentice-Hall Inc., ISBN 0-13-708917-1

DISASTER RECOVERY PLANNING - WHAT? WHY? and HOW?

Bryan D. Clapper
HEWLETT PACKARD Company, Santa Clara, CALIFORNIA, USA

INTRODUCTION

Information is a major asset for all Corporations. More and more companies are relying on computers to store and manage this vast asset. In some industries, maintaining a competitive edge over competition requires this information to be on-line and readily accessible. This in turn, requires maximum system availability.

Data Center disasters can render this information inaccessible. Disasters can result from natural threats such as a fire, flood, or earthquake, to intentional acts by disgruntled employees such as theft or arson. But disasters don't have to be 6:00 o'clock headliners, they can result from operational mishaps or power outages or faulty equipment.

A disaster can cripple a company's access to accurate up-to-date information necessary for sound business decisions. In addition, disasters can hinder a company's ability to conduct normal business activities such as making/receiving payments, meeting manufacturing schedules, booking orders, cutting purchase orders, and invoicing customers. Therefore, companies must be prepared to react quickly in the event of a disaster to minimize the impact to normal business operations.

Companies can achieve this protection by developing a **Disaster Recovery Plan (DRP)**. Such a plan would ensure the continuation of normal business activity by restoring necessary data processing functions in a timely fashion. This paper focuses on **WHAT** a DRP is, **WHY** it is necessary, and **HOW** to develop one.

WHAT IS A DRP

A DRP is a comprehensive document containing the actions required to restore data processing activities in the most timely and effective manner. It is intended to reduce the confusion created as a result of the disaster by clearly defining a course of action. Below is a brief description of the sections contained in a Disaster Recovery Plan.

* Scope and Objectives
* Immediate actions/Disaster notification
* Reovery team - Roles and Responsibilities
* Accessing the Damage
* Recovery Procedures
* Application Requirements
* Checklists
* DRP Maintenance Procedure
* DRP Test Plan

SCOPE and OBJECTIVES

Will this plan ensure a "total" or "limited" return to normal data processing activities? The answer to this question will provide the SCOPE of the DRP. It will be determined by a clear understanding of what applications are critical to the company's success and how long the company can survive before data processing activities must be restored.

The objective of every DRP is to ensure the restoration of data processing activity in a timely and effective manner. "Timely" and "effective" are loose terms, however, that may need to be qualified. For example, the objective may be to restore ALL data processing activity in 24 hours with no major problems.

IMMEDIATE ACTIONS/DISASTER NOTIFICATION

Confusion can overcome individuals not trained on what to do in the event of a disaster. This confusion could compromise the safety of company personnel and/or result in unnecessary property loss. Thefore, it is extremely important to define and train all individuals on the immediate actions to take during a disaster. This would include procedures to ensure above all else, personal safety. In addition, it would offer procedures for shutting down the computer(s), securing the data center, notifying the proper authorities and contacting the Recovery Team.

RECOVERY TEAM - Roles and Responsibilities

This section would identify all of the individuals on the Disaster Recovery Team. These individuals would be divided into groups and given very specific responsibilities to execute during the recovery process.

Below is a chart proposing such groups and their associated responsibilities. The number and/or names of the groups is unimportant. What is important, however, are the FUNCTIONs they are responsible for performing.

GROUP	RESPONSIBILITY
1. Facilities	* Assess damage to the computer facility * Estimate the time to repair/reconstruct the facility * Manage the replacement and/or repair of the facility
2. Consumables	* Assess the damage to all consumeables (paper, forms, tapes, etc.) * Ensure timely replacement of required consumeables * Determine consumable requirements (specified in the DRP) * Recovery salvageable consumables
3. Operations	* Ensure successful operations at backup recovery site * Retrieve necessities from off-site storage (backups, documentation, etc.)
4. Software	* Assess software and application requirements for recovery

	* Coordinate transfer of data and applications to the backup site * Install all software (OPSYS and applications) at the recovery site
5. Hardware	* Estimate damage sustained by hardware * Ensure the replacement and/or repair of hardware * Recovery Salvageable hardware * Coordinate transfer of useable and/or required hardware to backup site * Install (user installable) hardware at recovery site
6. Communications	* Assess damage to datacomm equipment * Ensure the repair/replacement of datacomm equipment * Recover salvageable equipment * Coordinate the transfer of useable and required equipment to backup site * Establish network at backup site
7. Logistics	* Coordinate the transfer of hardware, software, consumables, data, and personnel to the backup site * Provide transportation for all resources to and from backup site

ASSESSING THE DAMAGE

After a disaster has occurred, it is very important to assess the extent of damage sustained by the data center. This will be important in determining the appropriate recovery actions necessary to restore data processing activities. Based on the extent of damage, the recovery team will decide whether to continue processing on-site or relocate processing activities (all or part) to the backup site. Based on this decision, the appropriate recovery actions will be initiated.

RECOVERY PROCEDURES

This section will contain all procedures necessary to successfully restore data processing either on-site or at the backup site. In addition, it will identify what group(s) are responsible for the defined actions. Finally, this section will contain a procedure for returning from the backup facility. Below is a list of procedures to consider for this section.

* Notifying the backup site
* Identify applications to be recovered first
* Identify off-site storage retrieval requirements (consumables, documentation, data, software, etc.)
* Transporting resources to the backup site

- Configuration of the backup site
- Installing the operating system
- Installing the applications
- Scheduling applications/jobs
- Establishing datacomm links
- Instating security measures
- Restarting operations

APPLICATIONS REQUIREMENTS

This section defines and prioritizes those applications which are critical to the success of the company. Also, it defines all resources required by the applications including disc space, printers, paper, mag tape and datacomm equipment. Finally, it specifies the maximum amount of time the application can be "out of commission".

CHECKLISTS

Checklists are used as a reference guide by defining what resources are used and where they reside. For example, the consumables checklist would identify all consumables used in the data processing environment. In addition, it specifies the suppliers of these consumables, ordering lead time, availability and maybe even cost information. Listed below are the checklists maintained in this section.

- Consumables
- Off-Site Storage
- Hardware (CPU's, Memory, Terminals, Discs)
- Software (Operating system, utilities, applications)
- Datacomm Equipment (Modems, PBX's, Leased Lines)
- Documentation

DRP MAINTENANCE PROCEDURE

As companies grow, their data processing requirements and priorities change. These changes need to be reflected in the DRP as soon as possible. Therefore, a procedure for maintaining the DRP must be created and adhered to. DRP's that don't accurately reflect a company's business requirements are useless.

DRP TEST PLAN

DRP's, like software, must be tested for accuracy and effectiveness. Testing is designed to catch oversights which might prevent the Recovery from being successful. This section would contain such a test procedure.

WHY DEVELOP A DRP

Disaster Recovery Plans are required to recover from a disaster minimizing the impact to continued business. Justification for developing a DRP is found in understanding the costs associated with computer downtime. Some costs are quite easy to quantify while others are not.

For example, productivity losses in terms of salary for idle employees can be easily calculated. Assume, for instance, a company has an average of 60 on-line computer users per day and each user is active on the system an average of 5 hours/day. If the average salary for these employees is $10 dollars per hour and the computer were down for 1 day, salary costs would amount to $3000. If the system were down for one week that amount jumps to $15K and for a month $60K. Include in this model the cost for additional manual labor (manually tracking items through manufacturing, manually cutting invoices, manually writing checks for accounts payable etc.). Then add in the cost of interest expense for late payments to suppliers and the loss of valuable discounts for early payment. In addition, filter in the impact for delays of invoicing customers for products and/or services delivered. These delays in realizing income could greatly impact a company's cash flow position.

Intangible costs also need to be considered. Computer down time will directly affect the manufacturing and delivery of products. Delays in this area may destroy valuable relationships with customers and directly impact current and FUTURE business.

When the full impact of computer downtime is realized and costs are applied to the damages, the numbers can be staggering. It is then that the value of a disaster recovery plan is realized.

HOW TO DEVELOP A DRP

The development of a DRP requires the commitment of upper level management. This ensures that the appropriate (required) resources are available to develop, maintain, test and implement the DRP. The following steps are required in developing a DRP and are described below.

1. Assemble a Disaster Recovery Team
2. Define Company's business requirements
3. Define the DRP's scope and objectives
4. Develop the recovery plan.
    a. Identify critical applications
    b. Establish priorities for critical apps
    c. Define the resource requirements to run the applications
    d. Define the recovery alternatives (backup site)
    e. Develop damage assessment procedures
    f. Develop the recovery procedures for each alternative
5. Develop DRP maintenance plan
6. Develop DRP test plan

ASSEMBLING A DRP TEAM

The team should consist of representatives from Management, MIS, Facilities, and the various user communities. Including these groups in the development process adds viability to the plan. Management can relate business priorities and requirements which could directly affect the content of the plan. Users, for example, could bring information about their priorities, needs and concerns. In addition, these participants may be exposed to areas where their departments are vulnerable to disaster allowing corrective action to be taken beforehand.

DEFINING the COMPANY'S BUSINESS REQUIREMENTS

This step is necessary to determine what applications are most critical to the success of the company. Critical applications are those that directly impact cash flow. For instance, if an invoicing system is disabled, customer billing will be late and consequently payments for products and/or services will be late. This could adversely affect a company's cash flow position.

It is important to define for each application what costs (productivity, loss of business etc.) are incurred if down for 1 day, 1 week, or 1 month. This analysis will determine how long a company can survive without computer services.

DEFINING the SCOPE and OBJECTIVES of the DRP

Based on the company's business requirements, the SCOPE and OBJECTIVES of the DRP will need to be sufficient to ensure the company can operate (hopefully at a profit) in the event of a disaster. If it is determined, for instance, that the company must be fully operational (i.e. recover all applications) within 48 hours of a disaster, then the SCOPE of the DRP is to recover all applications and the objectives would be defined to meet the 48 hour requirements.

Once defined, the development team must review the scope and objectives with management and get approval to continue the development of the DRP.

DEVELOPING the RECOVERY PLAN

A tremendous amount of data needs to be gathered and analyzed in this step. For example, all critical applications need to be defined and prioritized. This includes disc space requirements, printing requirements, consumable requirements and datacomm requirements. Once the requirements are defined, a backup site to meet these requirements must be located. Some alternates to consider include mutual aid agreements, hot sites, cold sites and shells. Procedures must be developed to cover all pre and post recovery actions. And finally, a DRP maintenance and testing procedure needs to be developed.

MAINTAINING the PLAN

Again, the DRP is not a static document. This plan must be kept updated to reflect the changing business requirements and changes to the data processing environment. The minute new equipment is added to the data center, the DRP becomes outdated. Ample consideration needs to be given to HOW and WHO will maintain the DRP.

One way to ensure the accuracy of the plan is to make members of the Recovery team responsible for a particular portion of the plan. Meetings could be held quarterly to incorporate/discuss any changes to the plan.

TESTING the PLAN

Finally, the plan must be tested to ensure completeness and accuracy. Minimally, the plan should be tested on a yearly basis. Theoretically, however, once a change has been made the plan should be re-tested. This quite probably would prove to be too costly and time consuming. Therefore, criteria will need to be set which will determine when the plan should be tested.

BIOGRAPHY

Bryan Clapper has been with HEWLETT-PACKARD for 6 years. His first 4 years with HP was spent as a Software Development Engineer designing and developing Sales and Field Service applications. The last 2 years have been spent as a Commercial Systems Engineer supporting HP customers in the Santa Clara, California sales office.

VARIATIONS ON A TUNE - ANOTHER LOOK AT THE NEVER-ENDING STRUGGLE TOWARDS OPTIMAL PERFORMANCE

Steven M. Cooper
Allegro Consultants, Inc.
Redwood City, CA, USA

Research sponsored by
ADAGER, Antigua GUATEMALA

Introduction

Back in the Dark Ages (seven or eight years ago), we had no choice but to grope in the dark in terms of tuning our computers and databases for maximum performance; monitoring tools did not exist and the general guidelines that have become common knowledge had not yet been discovered and published. Relief began to come shortly thereafter, though, as articles on performance and tuning, many now considered classics, were written, providing us with a set of guidelines to follow.

Once the articles were written and the information disseminated, nobody seemed to give it much thought, considering the topic adequately covered. However, a lot has happened in the past three or four years in the HP3000 world and perhaps we ought to take another look at some of these issues. Comparing a Series III with one half megabyte of memory to a largish Series 68 running MPE V, the newer machine has a CPU that is five times faster, has up to 16 times more memory, and is capable of five or six times the effective disc I/Os per second (assuming caching). Most systems will also support four times the number of users. Given that we now have a much larger and differently shaped beast to deal with, some of the assumptions of those original articles must have changed, and therefore, so must the guidelines.

Unfortunately, we must all suffer through a period of semi-darkness again as we experiment with caching and its parameters, and learn by trial and error what is good or bad for these new machines. In an effort to encourage the re-opening of these topics, this article presents some observations and results of experiments done to date. I do not have all of the answers, and can't even explain all of the results that I've gotten, but perhaps combined with your own experimentation, we can continue to evolve these guidelines to keep pace with the maturing world of the HP3000.

CACHECONTROL

Caching is a wonderful thing if you've got the extra memory and are I/O-bound. The designers of caching left us with two dials and a switch to play with, so it is our duty as system managers and system tuners to turn and flick and see what happens. The "dials" are the RANDOM and SEQUENTIAL options of the CACHECONTROL command. They control how much is actually read from disc into a cache domain when we issue a read from a cached disc. (This is a simplification. We will discuss some of the complications later.) The RANDOM value tells how many sectors should be read when we do an FREADDIR call. (In COBOL, this happens for files with ACCESS IS RANDOM clauses.) The SEQUENTIAL value corresponds to FREAD calls. (In COBOL, ACCESS IS SEQUENTIAL.) Both parameters can be set to values between 1 and 96 sectors, indicating cache domains between 256 bytes and 24,576 bytes. By default, the system initializes RANDOM to 16 and SEQUENTIAL to 96.

It is important to note that no matter what type of DBGET or DBanything we do, IMAGE always issues FREADDIRs and FWRITEDIRs. (This is also true for KSAM files.) So for the typical lMAGE-based system, the vast majority of I/Os will be affected by the RANDOM value and not at all by the SEQUENTIAL value.

The MPE folks at HP chose the default RANDOM value of 16 because the HP7933 and HP7935 disc drives have a buffer size of 16 sectors. But you will need an HP7933/5 on the same GIC as another in-use device before you will ever use this buffer. In actuality, this buffer doesn't seem to have much affect unless you have two or more HP7933/5s on a GIC, these discs have firmware revision levels of 5.1 or greater, you have a version of MPE that supports rotational position sensing (most releases after and including MPE V/E), and you RPS is enabled on all of those drives. If you meet all of these requirements, a RANDOM value of 16 is probably the right value for your system.

However, if you don't meet all of these requirements, chances are that your system will perform better with a higher value for RANDOM. The bigger the value, the more will be read each time, and, assuming there is locality to your reads, the less you will have to actually go to disc. Furthermore, if we leave the value at 16, the number of domains tends to get very large, often over 2500. This causes extra work for the cache manager, since every time someone does a read, the manager must first see if the record is already in one of the 2500 domains. This suggests that the RANDOM value be increased, perhaps to 32, 64, or even 96.

But life is never that simple. Another thing to consider is the setting of BLOCKONWRITE. Let's assume that BLOCKONWRITE, the "switch" next to the two "dials", is set to NO. In this mode, when we issue an FWRITE or an FWRITEDIR, control returns to our program when the cache domain is updated, not when the actual disc I/O completes. The cache manager does, however, request an I/O at that time to flush that cache domain to disc. If we attempt to write to that same cache domain again before the I/O to disc actually completes, we will then get blocked and control will not return to our program until the following occur: the first I/O completes, our process gets launched again, and the cache domain is updated with our write request. (It is for this reason that some people say that we want to minimize the WRITE HIT statistic. This is only partially true; we want to minimize write hits to domains that are being written out at that moment. Write hits to "clean" domains are a good thing. Unfortunately, the cache statistics do not separate the good hits from the bad.)

Anyway, this seems to suggest that in order to minimize write hits to "dirty" cache domains, the RANDOM value should be set low, so only a smaller chunk of the file is "dirty" at any time. This directly conflicts with the advice of the first suggestion. "What now?", you ask? Well, this time we are in luck. The SHOWCACHE command tells us how effective caching is being and the CACHECONTROL command lets us tweak these parameters while the system is up and running. What we can do is STOPCACHE and STARTCACHE to zero out all of the statistics, issue our CACHECONTROL commands to adjust RANDOM, SEQUENTIAL, and BLOCKONWRITE, wait an hour or so, and then issue a SHOWCACHE command to see how we did. Try to find values that maximize "percent of user I/Os eliminated" and after that, minimize the "data overhead".

Let's focus on the BLOCKONWRITE parameter a bit more. If it is OFF, the system allegedly will run faster since our application programs will not have to wait for the disc outputs to complete. But we pay a price: unless we have asked for the file system to use the Serial Write Queue for the file, we will not know in which order our outputs will be flushed to disc. If the system should fail before all of the "dirty" cache domains have been flushed, we would have a higher likelihood of database corruption than we would have had with caching off or BLOCKONWRITE ON, assuming that we are not using the Serial Write Queue. How do you ask the file system to use the Serial Write Queue? For normal files, issue a call to FSETMODE. But for databases, there is no direct way to do this. IMAGE will issue the FSETMODE if and only if you are using Intrinsic Level Recovery (ILR) or Transaction Logging. If you use neither and have BLOCKONWRITE OFF, be aware of this new vulnerability.

It might be interesting to note some results we obtained on a machine with five megabytes. We ran ADAGER's DETPACK program in a stand-alone environment over and over, varying these parameters each time. This program runs through several IMAGE datasets and updates just about every record it looks at. Although this is a very different environment to one with 80 on-line users, it is similar to many shops' nighttime runs, with single-threaded report and update programs running. We found that the higher the value for RANDOM, the shorter the run times. And, much to our surprise, we had shorter run times with BLOCKONWRITE set to YES! This was probably due to the frequency of writing to "dirty" cache domains as described above. Hence, our best times were obtained with RANDOM = 96 and BLOCKONWRITE = ON. These were about 10% faster than with RANDOM = 8 and BLOCKONWRITE = OFF.

Extents

When we build a normal file, we can specify how many separate areas of disc we would like the file to be broken up into. We can also ask for these areas to be initially allocated or allocated the first time that a record in that area is referenced. IMAGE still uses its original algorithm for determining the number of extents that datasets should use. The algorithm always asks for the largest number of extents, 32, for medium size datasets or larger, and all extents are always initially allocated. This was fine in the old days; the larger the number of extents, the smaller each extent will need to be, and the easier it will be to find areas of disc that are the proper size.

But since the advent of caching, the number of extents can have a more important impact. (Now come the complications promised above.) When we do a random read to disc, the caching manager will read into a cache domain a chunk of the file as large as the larger of the RANDOM value and the number of words we requested, but not past the end of the extent. Hence, if a file has 32 extents in it, there are 31 brick walls built into the file, around which caching must work. We can improve the caching efficiency of nonIMAGE files, therefore, by lowering the number of extents in a file, to as low as one if enough contiguous free disc space is available. Larger numbers of extents are only justified when the file will grow over time and we want to minimize wasted disc space, or when disc is fragmented so that we cannot obtain large enough extents. (As an extra bonus, on tape drives newer than the HP7970, the fewer the number of extents that a file has, the quicker that file will STORE.)

For IMAGE datasets there is not much we can do until the algorithm used by IMAGE to build datasets is changed. One option, though, is to use ADAGER to create the datasets. Whenever an ADAGER function needs to create a new dataset, it uses an algorithm that will attempt to lower the number of extents in the dataset, without producing extents that are too large.

BLOCKMAX and BUFFSPECS

These are two tuning parameters that IMAGE allows us to set that are often ignored. Here again, it was safe to ignore them in the past, since the defaults were good values for the smaller system. But if we leave things alone now, we may not be happy with the results. For example, if you have a report program that runs stand-alone for three hours every night, IMAGE will be polite and use only 8,000 bytes of memory by default, even though megabytes are sitting unused. Caching minimizes this effect, since it will use all the extra memory it can find, but we would still be better off instructing IMAGE to be more aggressive in its use of memory resources.

The BLOCKMAX parameter is specified in the Schema. This specifies the largest size that we will accept for the block size for this database. Remember, IMAGE will decide what the blocking factor for each dataset will be. Then, the largest block size among all of the datasets will become the size of all of the buffers whenever the database is accessed. The default value is 512 words. We can increase this number up to 2048 words.

Look closely at the blocking factor for each dataset at the bottom of the DBSCHEMA listing. Sometimes IMAGE has a choice of a blocking factor that saves a bit of disc space, but does not pack as many entries into each block as will fit. IMAGE will let us override this choice. We can specify our own blocking factor for a dataset by putting the desired number in parentheses between the capacity and the semicolon in the schema. E.g.

```
    Capacity: 40000 (20);
```

would request a capacity of 40,000 entries and 20 entries per block.

The BUFFSPECS for a database can be interrogated by using the SHOW command in DBUTIL. The number of buffers is dependent upon the number of times the database is opened. The default BUFFSPECS are:

```
   BUFFSPECS = 8(1/2),9(3/4),10(5/6) ... 17(19/120)
```

which means that with one or two users in the database, 8 buffers will be allocated, three or four users will get 9 buffers, and so on until 19 or more users open the database, at which time 17 buffers will get allocated. To go back to our example, that stand-alone report program would run with eight buffers, since when it runs, it runs alone.

If you have many databases on your system, then all of this may be all right. But if you have one or two central databases, you might be better off giving more memory to these database to work with. In other words, change the BLOCKMAX and the BUFFSPECS. If you have ADAGER, the best way to change the BLOCKMAX is with the REBLOCK function.

This program will help you select a proper value, giving you a little lesson on the complexities of blocking factors along the way, and then will reblock your database. Choose a large value for a central database. It is bad enough that all users and all datasets must share the same set of buffers. The least we can do is make these buffers as large and as plentiful as is reasonable.

Now that we've adjusted the size of each buffer, we need to decide how many buffers we want to allocate. A good rule seems to be that the number of buffers should not be dependent upon the number of users. The number should be set to nine plus the largest number of paths going to any one detail dataset. If a detail dataset has five paths from master datasets, for example, set the BUFFSPECS as follows:

```
   BUFFSPECS = 14(1/120)
```

To do this, use the SET command of DBUTIL when no one has the database opened.

Cabling

Larger machines with two or three IMBs (InterModule Bus) and four or more GICs (General Interface Channel) can be cabled in many different ways. How things are cabled can have a greater impact on performance than just about any other factor. Do not assume that HP cabled you up optimally! Your discs should be spread over all of the IMBs and GICs that they can. If the system is accessing two files, performance will be best if the files reside on discs hooked to two different IMBs, next best if they are on two different GICs on the same IMB, next if they are on two different master drives on the same GIC, next if they are on a slave drive and its corresponding master drive, and worst if they are on the same disc.

Review your own IMB/GIC/disc drive configuration and fix things if they are not optimal. Then, with a tool such as FILERPT in the Contributed Library, identify the ten or so most used files. (We have contributed an updated version of FILERPT to the Contributed Library that will support both MPE IV and MPE V log formats.) Spread these files out among the discs according to the rules of the previous paragraph. We have found that disc file placement can have a major impact on performance, as long as we are concerned with the placement of the correct files.

Data Distribution

As databases get larger and larger, the way in which the entries are loaded in the database becomes ever more important in terms of performance. But since these large databases take so long to reload, many sites cannot afford the time it takes to reorganize their databases. This can have a dramatic affect on performance though. For example, a detail dataset with a blocking factor of ten has had so many puts and deletes over time, that its free entry chain points all over the place. When ten line items are placed on the same chain into this detail, they end up in ten different blocks. If this detail were packed along this path, all ten entries could reside in the same block. Instead of ten I/Os to read the chain, it will then just take one! A major improvement.

Programs such as DBLOADNG in the Contributed Library or HowMessy from Robelle Consulting will tell how badly your datasets need repacking. Then if called for, we can either take the time to reload them or use ADAGER's DETPACK and MASTPACK.

Conclusion

Just when we think we understand the rules and get our systems tuned accordingly, something comes along that changes the rules. Ignoring the changes can waste time and money. Tuning is a never-ending process that usually pays for itself over and over again. Give some thought to these guidelines, take a look at your system, and experiment. And when you discover something, share it!

Biography

Steve Cooper is a member of the Adager Research and Development Laboratory. He has had nine years of experience on the HP3000. He holds a BA degree in Computer Science from the University of California, San Diego and an MBA degree from the University of California, Los Angeles.

Gregory Stephens
Hewlett-Packard Cupertino
U.S.A.

NOTE: See page 653.

MESSAGE CATALOGS AND NATIVE LANGUAGE SUPPORT

Glenn Cole
Consultant
Fairfax, Virginia USA

Summary

One of the least-used features of the HP3000 is the Message Catalog facility and the more-recent Native Language Support. The message catalog is intended to be used in applications where there is a non-trivial number of (error) messages to the user. The classic application of this is MPE.

The other intended area of use is where the same application program is used by different people who understand different languages. In this case, only the messages need to be changed – modifying and recompiling the application is not necessary.

The message facility is easy to use, has minimal overhead, and may save substantial data stack area. This paper highlights a case history where use of this facility was virtually required, and then details the relatively painless way in which it was implemented. Afterwards, a comparison is given between the original Message Catalog facility of MPE and the similar feature within Native Language Support.

Introduction

The Message Catalog facility provides a means of programmatically accessing messages contained within a specially-formatted file by using standard MPE file system intrinsics. Parameter substitution is allowed, and messages may be routed directly to the list device, retrieved for use by the application program, or both. This facility is provided by Hewlett-Packard as part of the Fundamental Operating System (FOS), and thus is available under all versions of MPE on all HP3000 computer systems at no extra cost.

The Case: (Part 1 - The Problem)

At the time (late 1983), I was employed by a major Moving & Storage company as a Programmer/Analyst, working on a team developing an on-line Dispatch System. It was designed so that when a truck driver called his location in to a dispatcher, the dispatcher could see where the truck was going, update its current location, and pass along any messages to the driver. The Dispatch System also kept a history of each shipment that could be brought up at any time, along with any comments recorded by another dispatcher during handling of the shipment.

The Dispatch System used VIEW and IMAGE extensively, and was written in COBOL II running on a Series 64 under the Q-MIT release of MPE IV. It was a menu driven system with dynamic subprograms. Most of these subprograms used VIEW screens as well. Some of these subprograms could call other subprograms. The "longest path" was about 4 levels deep, not counting the main menu. Most of the subprograms in this path had many messages to retain — the average was about 40 messages of 80 characters each. These were kept in WORKING-STORAGE along with the other data requirements of the subprogram. Each time a subprogram was called, its WORKING-STORAGE SECTION — that is, the data area of the program — was added to the data stack. The end result was that when this "longest path" was used, it resulted in a STACK OVERFLOW and the program aborted.

This was not good news. The target date for implementation was approaching rapidly. Not only did we need a solution, but we needed one that could be implemented fairly quickly, that would not compromise the design set forth by the analysts, and that would work.

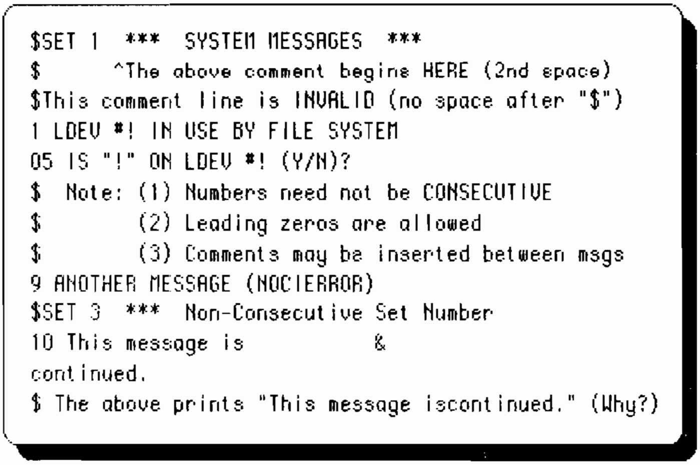
The Case: (Part 2 - A Solution)

The stage was set. I knew of the existence of message catalogs by virtue of reading the System Intrinsics Manual. However, I had never used this facility, nor had I ever seen an application that did. None of the other programmers knew anything about message catalogs either.

I happen to like the documentation provided for the HP3000. Sometimes it is difficult to find the desired information. Sometimes the information, once found, is neither entirely accurate nor complete. (Fortunately, on the occasion when it is incorrect, it usually says "This cannot be done" when in fact it can, instead of the other way around.)

In this case, the documentation seemed quite clear, and it appeared that a solution was at hand. First, I set up a small test program to verify that I understood both the concepts and the mechanics of Message Catalogs.

The concepts are rather simple and straightforward. A message catalog (Figure 1) is created initially as a standard MPE flat file, and consists of from 1 to 62 "sets," each containing up to 32767 messages. Sets are indicated by "$SET n" beginning in column 1, where "n" is the set number (1 - 62). The sets should be in ascending sequence by set number, but the numbers need



```
$SET 1  ***  SYSTEM MESSAGES  ***
$      ^The above comment begins HERE (2nd space)
$This comment line is INVALID (no space after "$")
1 LDEV *! IN USE BY FILE SYSTEM
05 IS "!" ON LDEV *! (Y/N)?
$  Note: (1) Numbers need not be CONSECUTIVE
$        (2) Leading zeros are allowed
$        (3) Comments may be inserted between msgs
9 ANOTHER MESSAGE (NOCIERROR)
$SET 3  ***  Non-Consecutive Set Number
10 This message is          &
continued.
$ The above prints "This message iscontinued." (Why?)
```

Figure 1 - Sample Message Catalog

not be consecutive. (For example, a message catalog with only 3 sets — numbered 2, 7, and 15 — is perfectly valid.) The remainder of the line following the space after the set number may be used as a free-form comment (e.g., "$SET 1 *** System Messages"). Other comments may be included by entering a "dollar-sign space" ($) beginning in column 1, with the remainder of the line (columns 3 thru 72) used for the comment.

Each message in the file is uniquely identified by its set number (described above) and by its message number (1 - 32767). Thus, message number 100 in set 1 may be completely unrelated to message number 100 in set 2. Messages are entered under the appropriate $SET heading with the message number beginning in column 1, followed by a single space, followed by the text of the message. Note that the message numbers within a set must be in ascending sequence, though they need not be consecutive. (Please review Figure 1.)

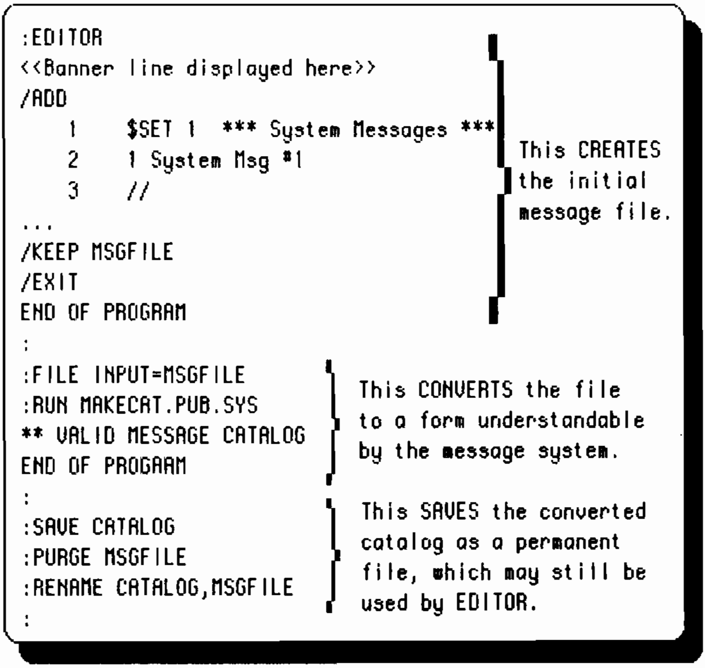
There are three (3) special-purpose characters that may be included in the message text. These include the continuation characters "ampersand" (&) and "percent" (%), and the parameter substitution character "exclamation mark" (!). When a continuation character is entered as the last non-blank character on a line, it indicates that the next line is to be included as part of the current message. It is important to note that all blanks immediately preceding the continuation character are IGNORED. Thus, there is some latitude as to where the continuation character may be placed. More importantly, this means that the following line probably should begin with a space.

Now, recall that the application program may elect to have the message routed directly to the user's terminal. This is where the two continuation characters differ. If the "percent" symbol (%) is used, then a carriage return and line feed are printed before continuing with the next line. The "ampersand" (&), however, indicates that the continuation line should be printed WITHOUT first printing a carriage return and line feed. In this case, the continuation line will be printed adjacent to the current line. These symbols correspond roughly to the SPACE (%40) and %320 carriage control characters.

Each message may contain up to five (5) exclamation marks (!) for parameter substitution.

Each passed parameter is inserted in the message where the corresponding exclamation mark occurs, with the first parameter replacing the first exclamation mark, the second parameter replacing the second exclamation mark, and so on, until all parameters are included. (Please review Figure 1 again. See — a picture really IS worth a thousand words!)

The mechanics of using a message catalog are almost as simple as understanding the concepts given above. Before an application can use a message catalog, it (the catalog) must first be prepared for use (Figure 2). Remember, the messages are entered with EDITOR (for example) and kept as one would keep FORTRAN or SPL source code, that is, as a numbered file with 80-byte fixed-length records.



```
:EDITOR
<<Banner line displayed here>>
/ADD
    1     $SET 1  *** System Messages ***
    2     1 System Msg #1
    3     //
...
/KEEP MSGFILE
/EXIT
END OF PROGRAM
:
:FILE INPUT=MSGFILE
:RUN MAKECAT.PUB.SYS
** VALID MESSAGE CATALOG
END OF PROGRAM
:
:SAVE CATALOG
:PURGE MSGFILE
:RENAME CATALOG,MSGFILE
:
```

This CREATES the initial message file.

This CONVERTS the file to a form understandable by the message system.

This SAVES the converted catalog as a permanent file, which may still be used by EDITOR.

Figure 2 - Preparing the Message Catalog for Use

The next step in preparing this catalog for use involves running the system utility program MAKECAT to install a "directory" on the file as a single user label. This utility reads the data from formal file designator INPUT and builds a temporary file called CATALOG. Any existing temporary file named CATALOG is renamed to CATnn, where "nn" is the first available number. (Note that the first file is renamed to CAT1, not CAT01, and so on.) A short (perhaps cryptic) error message is displayed for each line believed to be in error. If this happens, then the CATALOG file is not built.

Return to the EDITOR, text the file, fix the error, and try again.

Once validated, the message "** VALID MESSAGE CATALOG" is displayed and the CATALOG file is built. At this point, you may :SAVE CATALOG and then :RENAME CATALOG,myfile. (Of course, if a file named CATALOG already exists, you may have to use the :RENAME CATALOG,myfile,TEMP command and then :SAVE myfile.) Once this has been accomplished, the message catalog is ready for use. (Please review Figure 2.) Note that you can still modify the message catalog by returning to the EDITOR, texting the file, making the desired changes, keeping the file either under the same name or under some other name, and running MAKECAT again.

In order for the application program to access the message catalog, the file must first be opened. This is done with the standard FOPEN intrinsic. The important things here are the FOPTIONS and AOPTIONS parameters. FOPTIONS (file options) should include Old, Permanent, and ASCII (%5). AOPTIONS (access options) should include Multi-Record and Nobuf (%420).

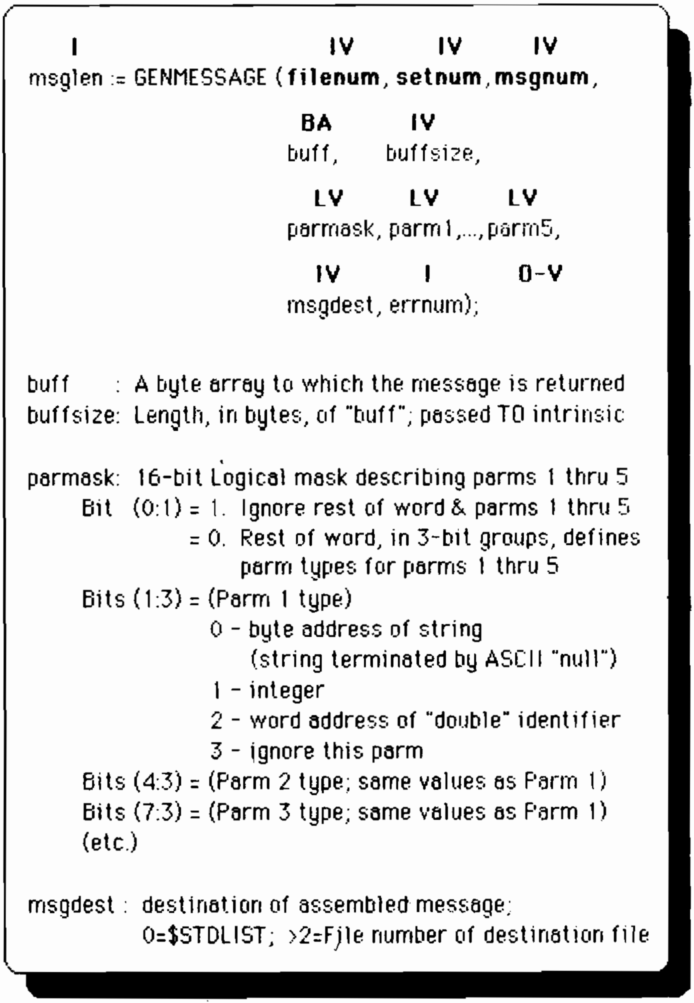
The intrinsic GENMESSAGE is the key to the whole operation. A (brief) description of this intrinsic is given in Figure 3. Recall that the target application that I was concerned with used VIEW screens. This meant that the message could not be sent directly to the terminal; the application had to use a buffer and then show the message using the appropriate VIEW calls.

The small test program that I used to verify I understood all of this worked fine. The questions now were: How can this be integrated into the Dispatch System? Should each subprogram call GENMESSAGE explicitly? Will the savings in data stack area be sufficient to allow the program to run without the STACK OVERFLOW?

If the subprograms called GENMESSAGE explicitly, then they would have to know the file number (returned by FOPEN) of the message catalog. This means that the routine performing the FOPEN would have to pass the file number to the other subprograms, which meant modifying the parameter list both in the calling routine and in the called routine. (The other alternative – each subprogram FOPENing the message catalog when beginning, and FCLOSEing it when leaving – could not be considered seriously because of the tremendous overhead the FOPENs and FCLOSEs would cause.) This seemed to have too much chance of programmer error.

There was still another alternative, one that appealed to me from the start, and which seemed to involve the least program modification. Recall that when a DYNAMIC subprogram is called, it gets an "original" copy of its WORKING-STORAGE section. That is, the variables in WORKING-STORAGE are re-initialized to the values defined when the subprogram was compiled. However, for a subprogram that is NOT declared as DYNAMIC, the variables in WORKING-STORAGE are NOT re-initialized. For example, let us imagine a non-DYNAMIC subprogram with a variable named COUNT described initially with a value of zero (0). Let us also imagine that during the course of this subprogram, COUNT is incremented by one (1). Now, for the first execution of this subprogram, COUNT will be zero (0) upon entry, and one (1) upon exit. However, the second time through, since COUNT is not re-initialized to zero, COUNT will be one (1, the previous value) upon entry and two (2) upon exit.

My idea, then, was to have a single subprogram that would do both the FOPEN <u>and</u> the GENMESSAGE calls. The subprogram would define the file number initially with a value of zero (0). This value would be checked and the subprogram execute the FOPEN only when the file number was zero, i.e., on the very first call. A successful FOPEN would change the file number to a non-zero value, so subsequent calls would not execute the FOPEN. (Note, however, that this approach meant that the message catalog would be closed implicitly by the file system when the application terminated, instead of being closed explicitly by the application.)



```
   I                        IV       IV      IV
msglen := GENMESSAGE (filenum, setnum,msgnum,

                              BA       IV
                             buff,    buffsize,

                               LV       LV       LV
                             parmask, parm1,...,parm5,

                               IV        I        O-V
                             msgdest, errnum);


buff     : A byte array to which the message is returned
buffsize: Length, in bytes, of "buff"; passed TO intrinsic

parmask:  16-bit Logical mask describing parms 1 thru 5
     Bit  (0:1) = 1.  Ignore rest of word & parms 1 thru 5
                = 0.  Rest of word, in 3-bit groups, defines
                      parm types for parms 1 thru 5
     Bits (1:3) = (Parm 1 type)
                  0 - byte address of string
                      (string terminated by ASCII "null")
                  1 - integer
                  2 - word address of "double" identifier
                  3 - ignore this parm
     Bits (4:3) = (Parm 2 type; same values as Parm 1)
     Bits (7:3) = (Parm 3 type; same values as Parm 1)
     (etc.)

msgdest :  destination of assembled message;
           0=$STDLIST;  >2=File number of destination file
```

Figure 3 - The GENMESSAGE Intrinsic

The other subprograms did not need the full flexibility available with GENMESSAGE, so adding a few limitations to keep things simple would not be critical. For example, no routine in the Dispatch System needed to substitute more than three (3) parameters in a message, and these values were never more than eight (8) bytes long. Thus, it seemed feasible to code the details of the message retrieval into a single subprogram. As an added benefit, this meant that the technique could be modified at some point in the future without having to modify the other subprograms. The remainder of this paper refers to this user-written message retrieval subprogram as GETMSG. (Appendix A shows the FORTRAN source code for GETMSG. The actual implementation was written in COBOL, but FORTRAN is more compact. Either language will suffice.)

Replacing the hard-coded messages in the subprograms with calls to GETMSG was not very difficult. Fortunately, it was not very time-consuming either. Since the immediate worry was the "longest path" within the Dispatch System, these subprograms were modified first. After all, if this "solution" did not work, then it was not a solution at all, and the effort expended thus far was merely educational at best.

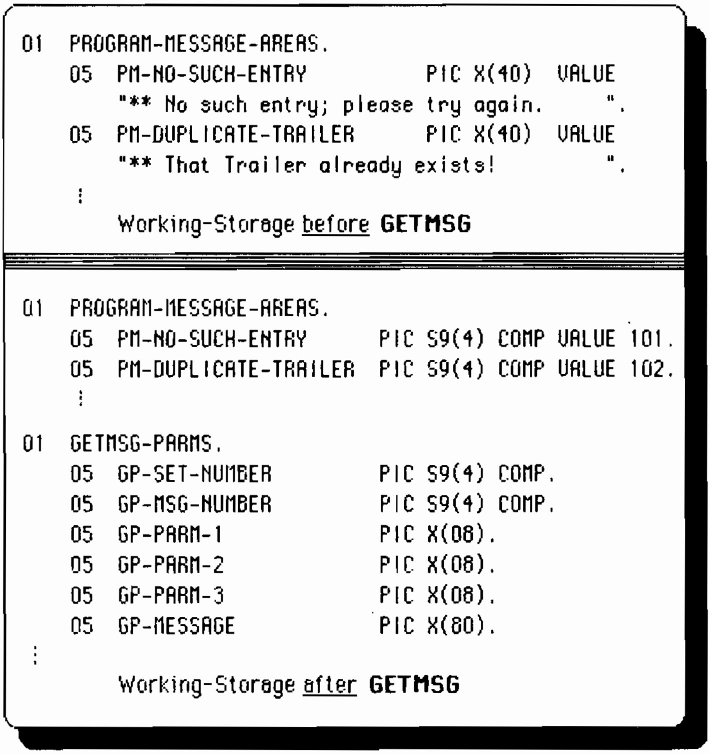
The subprograms had to know what Set Number and Message Numbers to use, since these were the main input parameters for GETMSG. These were assigned somewhat arbitrarily, based on the number of messages used by a given subprogram. (I think all four subprograms were assigned Set Number 1, and the Message Numbers assigned were 1-100, 101-200, etc. This gave plenty of room for future growth.)

The subprograms had been coded with the messages grouped together in WORKING-STORAGE. When each subprogram was text into the editor, this range of lines was kept in a separate file. (This auxiliary file was modified later so that it could be included as part of the message file. This meant that the actual messages did not have to be re-entered.) In the subprogram, the variable names assigned to the messages remained the same. However, each description was changed from PIC X(40) to PIC S9(4) COMP, and a VALUE equal to the Message Number was assigned. The Message Numbers were assigned in sequence, with the first message getting the lowest Message Number assigned to the subprogram. (Please see Figure 4 for a "before" and "after" look at this area.) Fields were also added to hold the Set Number, the generic Message Number, the retrieved Message, and three (3) 8-byte parameters for substitution (whether needed or not).

The PROCEDURE DIVISION was then searched for all moves to the VIEW window area. Each of these MOVE statements was replaced by two (2) other statements: A MOVE of the desired Message Number to a holding area, and a PERFORM of the utility routine (as yet unwritten) that would retrieve the desired message. (The Set Number did not have to be moved, since it remained constant throughout the subprogram.) Note that if parameter substitution was required for the desired message, the necessary value(s) would also have to be moved prior to the PERFORM.

The utility routine PERFORMed was quite simple, but I felt it was worthy of its own paragraph. It consisted only of a CALL to GETMSG, followed by a MOVE of the returned message to the VIEW window area. Note that if an error occurred in GETMSG (e.g., a missing Set Number/Message Number combination), this was indicated by the returned message.

The data area saved was tremendous, and the "longest path" worked, so this was indeed a solution. Perhaps just as significant is the fact that it took just two (2) days to research and implement it. There was very little new coding involved, and the technique was relatively simple and straight-forward.



```
01  PROGRAM-MESSAGE-AREAS.
    05  PM-NO-SUCH-ENTRY          PIC X(40)  VALUE
        "** No such entry; please try again.      ".
    05  PM-DUPLICATE-TRAILER      PIC X(40)  VALUE
        "** That Trailer already exists!          ".
    :
        Working-Storage before GETMSG
```

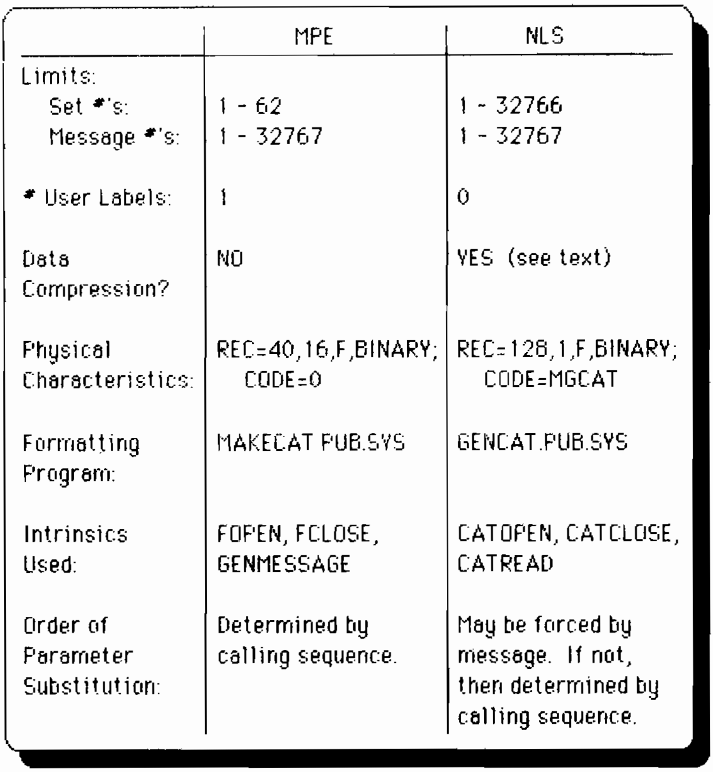
```
01  PROGRAM-MESSAGE-AREAS.
    05  PM-NO-SUCH-ENTRY      PIC S9(4) COMP VALUE 101.
    05  PM-DUPLICATE-TRAILER  PIC S9(4) COMP VALUE 102.
    :

01  GETMSG-PARMS.
    05  GP-SET-NUMBER         PIC S9(4) COMP.
    05  GP-MSG-NUMBER         PIC S9(4) COMP.
    05  GP-PARM-1             PIC X(08).
    05  GP-PARM-2             PIC X(08).
    05  GP-PARM-3             PIC X(08).
    05  GP-MESSAGE            PIC X(80).
:
        Working-Storage after GETMSG
```

Figure 4 - Working-Storage <u>before</u> and <u>after</u> **GETMSG**.

<u>Another Alternative: Native Language Support</u>

In the time since GETMSG was implemented, Hewlett-Packard has introduced Native Language Support (NLS). As with the standard MPE message catalog facility, this is included as part of the Fundamental Operating System; however, I believe you must be running under some version of MPE V. Native Language Support is an excellent idea, and it embodies far more than an improved version of message catalogs (now called the "application message facility" under NLS). However, only the application message facility is discussed below; most of NLS goes beyond the scope of this paper. (The manual is very readable; I strongly encourage you to read through it.)

A chart comparing MPE message catalogs with NLS is shown in Figure 5. Note that the two are very compatible. In fact, the GETMSG routine described earlier can be re-written to use NLS without modifying any other subprogram!



	MPE	NLS
Limits:		
Set #'s:	1 - 62	1 - 32766
Message #'s:	1 - 32767	1 - 32767
# User Labels:	1	0
Data Compression?	NO	YES (see text)
Physical Characteristics:	REC=40,16,F,BINARY; CODE=0	REC=128,1,F,BINARY; CODE=MGCAT
Formatting Program:	MAKECAT.PUB.SYS	GENCAT.PUB.SYS
Intrinsics Used:	FOPEN, FCLOSE, GENMESSAGE	CATOPEN, CATCLOSE, CATREAD
Order of Parameter Substitution:	Determined by calling sequence.	May be forced by message. If not, then determined by calling sequence.

Figure 5 - MPE Message Catalog vs. NLS

There are a few things worth noting that are not mentioned in the chart. First, the only data compression used by NLS is based on the blank space between the last non-blank character of each line and the logical end-of-record. (Remember, the Editor's line number is actually in the last eight (8) bytes of each record. The logical end-of-record mentioned above refers to the position just before this.) Thus, the worst place for the continuation character (in terms of data compression) is in the last position of the record. The amount of disc space consumed by this file compared to that of a standard MPE message catalog appears to be directly proportional to the average message length. Thus, a file whose average message takes up one-half of the record consumes about one-half of the disc space of the equivalent MPE message catalog. (As an example, the system message catalog takes up about 40% less space when formatted for NLS.)

Regarding physical disc I/O's: a brief study of the File Close records in the system log file shows that while the number of RECORDS processed remains the same whether or not NLS is used, the number of BLOCKS processed is reduced anywhere from three (3) to over ten (10) times. That is, NLS may require less than one-tenth of the disc I/O's of an equivalent MPE message catalog.

Note also that the blocks are smaller under NLS -- 128 x 1 = 128 words for NLS versus 40 x 16 = 640 words for MPE.

Conclusion

In conclusion, I would like to encourage you to look into message catalogs. They are easy to use, easy to customize, and in some cases, they may prove critical to the success of an application. I would also like to encourage you to look into Native Language Support. The reference manual is very readable. Among other things not mentioned in this paper, it explains how a single application can easily reference different message catalogs geared toward different languages. Hewlett-Packard has put considerable time, thought, and effort into this, and has made these tools available to us at no extra cost.

Biography

Glenn Cole has a B.S. in Mathematics from James Madison University, Harrisonburg, VA. He has been using the HP 3000 since 1978, and is now an HP 3000 consultant in northern Virginia (USA).

Appendix A. FORTRAN source for GETMSG.

```
      SUBROUTINE GETMSG (ISET, IMSG, CPARM1, CPARM2, CPARM3, CMSG)
C
      CHARACTER*08 CPARM1, CPARM2, CPARM3
      CHARACTER*80 CMSG
C
      INTEGER         IMSGLEN
      CHARACTER*28 CCATNAME
      CHARACTER*10 CPARML (3)
      LOGICAL         LPARM1, LPARM2, LPARM3
      EQUIVALENCE  (LPARM1, CPARML (1))
      EQUIVALENCE  (LPARM2, CPARML (2))
      EQUIVALENCE  (LPARM3, CPARML (3))
C
      DATA IFILE/0/, IMAXLEN /80/, CCATNAME /"CATFILE.PUB.PROD "/
C
      SYSTEM INTRINSIC FOPEN, GENMESSAGE
C
C     ------------------------------------------------
C     Open message catalog if not open already.
C
      IF (IFILE .NE. 0) GO TO 10
         IFILE = FOPEN (CCATNAME, %5L, %420L)
         IF (.CC.) 5, 9, 5
5           CMSG = "** Message catalog not open.  Call M.I.S."
            GO TO 99
9        CONTINUE
10    CONTINUE
C
C     ------------------------------------------------
C     Init parms (ASCII NUL ( %0) at end of each string).
C
      CPARML (1) = CPARM1
      CPARML (2) = CPARM2
      CPARML (3) = CPARM3
      DO 20 I = 1, 3
         CPARML (I) [9:2] = "  "
         CPARML (I) [INDEX(CPARML(I)," ") :1] = %0C
20       CONTINUE
C
C     ------------------------------------------------
C     Actual message retrieval; %0 indicates all parms are STRINGS
C
      IMSGLEN = GENMESSAGE (IFILE, ISET, IMSG, CMSG, IMAXLEN,
     +                        %0, LPARM1, LPARM2, LPARM3,,,,IERROR)
C
      IF (IERROR .EQ. 0) GO TO 99
C
```

```
      CMSG = "** Message retrieval failed.  Call M.I.S."
C
C     ----------------------------------------------------
C     Wrap up
C
99    RETURN
      EXIT
```

DATE HANDLING BEYOND THE YEAR "99"

by

Kevin Cooper

Hewlett-Packard Company
Pleasanton, California, U.S.A.

Summary

Most computer software that handles dates uses only two digits to represent the year. With the twenty-first century rapidly approaching, we must begin to think about how we will manage dates beyond the year "99". This paper focuses on three issues related to the handling and formatting of dates. What are the best ways to convert software that was written with the two-digit year limitation? What techniques should be used to design and code new programs that will avoid these pitfalls before the year 2000 arrives? And how can future productivity tools make all of this invisible to both software developers and users?

Introduction

Imagine these scenarios:

January, 2000 - You arrive at the office on Monday morning, January 3, ready to begin the new year. One of your users calls with a problem, and you promise to have it investigated immediately. When several more callers have similar problems within the next hour, you begin to wonder if someone has tampered with your computer system. Before noon you realize that you have a serious problem on your hands involving all your programs which handle dates.

June, 1995 - Your company has a business forecasting program which projects out five years. It worked just fine the last time you ran it, which was in December, 1994. But now in 1995 strange things seem to be happening with the program. Figures for the fifth year are showing up in the first column, and all the other years are shifted out one column. What could be causing this?

March, 1986 - I go to the bank to start a long-range savings program for my son's college education. I invest in a long-term note scheduled to mature in 2002, the year he will start college.

The computer printout from the financial institution thanks me for my business, tells me I will receive annual statements of interest earned, and concludes by promising to send me a reminder notice upon maturity - in March, 1902. Something makes me think I should have taken my money elsewhere.

The impending arrival of the year 2000 will force everyone associated with the data processing profession to take a good, hard look at the way computers handle dates. Most software that depends on date-related information today uses a six-digit date with a two-digit year, based on the assumption that the most significant digits of the year will always be "19". The dawning of the twenty-first century will change that sooner than many might think, and the programming staff will not be able to use the excuse that they were not given enough notice! Sort routines will place dates in the year "00" (2000) before dates in the year "99" (1999). And what about the potential confusion when the date 03/02/01 appears on a report? Is this March 2, 2001? February 3, 2001? February 1, 2003?

Before we look further at these problems, let's look back at a little bit of computer history.

Background

How did computers start using six-digit dates in the first place? The normal way people write dates is with six digits separated by slashes. Two people who communicate using the date 03/03/86 both understand this to mean March 3, 1986. (Note that this example avoids the confusion caused because Americans write dates with the month first while Europeans place the day first; we will discuss that later.) When the computer was first being used for data processing, data storage space was at a premium, so the early programmers had no reason to waste two extra bytes of storage space on every date just so they could cover the arrival of 2000. I think they were quite justified in making that choice.

That leaves us in the late twentieth century to solve the problems associated with the six-digit date, or more specifically, the two-digit year. As shown by the earlier examples, this will impact the computer world long before we actually reach the year 2000. It has already affected some organizations, such as financial institutions, who must handle notes and loans due to mature 15 or more years from now. The rest of us will become more aware of its consequences as the time draws nearer.

My first introduction to this whole problem came when I was a novice COBOL programmer working for Hewlett-Packard fresh out of college. I needed to program a six-digit date-editing routine to check whether February 29 was a valid date in a given year, which is really a test for leap years. I am not recommending the technique I chose, but I wrote the code to find leap years after 1977 like this:

```
IF YEAR = 80 OR 84 OR 88 OR 92 OR 96 ...
```

Then I stopped and pondered. What will happen when YEAR becomes "00"? I was appalled as I considered the impact that the turn of the century would have on the application I was writing. Sorts, edits, and countless lines of code which compared two dates would cease to function properly. This first encounter with the problems we will face when the year 2000 arrives made me realize that one of two things must happen before then:

1) I must try to find ways to solve this problem and get the word out so we don't face a computer crisis come January 1, 2000; or

2) I must get out of the data processing field altogether and seek a new career.

This paper represents my attempt at the first alternative. Hopefully, this means we can all avoid the second option.

By the way, one of the interesting sidelights about the year 2000 is determining whether or not it is a leap year. The general rule is that years divisible by four are leap years; based on this rule, 2000 should qualify. However, this rule has an exception. If a year is divisible by 100, then it is not a leap year. That seems to indicate that February, 2000, will only have 28 days, right? Well, no; there is another exception to the first exception. Years divisible by 400 are leap years. I had never heard of this 400-year rule until I saw it coded in someone else's date-editing routine, so I checked my almanac and found that the year 2000 truly is a leap year, with February 29 falling on a Tuesday. This means that leap year tests which ignore the exceptions and just check if the year is divisible by four will work until 2100 arrives. That is long beyond the expected life span of any computer program in existence today.

Conversion Methods

The first issue for discussion is the conversion of software that was designed with the limitation of a two-digit year. This problem may seem like it is a long way off, but we cannot avoid the issue for many more years. We have already seen that programs which project out to future years will have problems well before the twenty-first century arrives. Even if we assume that the majority of date handling programs will only have errors when we actually reach the year 2000, we are left with less than 14 years to completely eradicate the problem.

So what software should we be concerned about converting? To answer this question, we must try to figure out what software will still be around in the year 2000. One possible gauge of that is to look back 14 years and see what software from 1972 is still around today. Two obvious examples in the HP 3000 arena are the MPE operating system and the IMAGE date base management system. While most of their code has been rewritten over this time span, much of the original designs remain, such as the six-digit date used at system startup by MPE. There is no way to know whether these designs will still be around in the year 2000, but we can see from these examples that some of the software being developed today will in all likelihood survive the 14 years left until we reach the year 2000. Unless these new programs are designed and coded using some of the techniques to be discussed later, they will be the ones which require converting as the next century approaches.

Given the high probability that some software will need to be converted, when should all the necessary changes take place? Do we need to start converting our entire collection of existing software right away? No, I don't think so. Based on a typical life span of five to ten years, most of the computer programs in existence today should be obsolete by 2000 (although that is not the same thing as being completely out of use!). The time to evaluate this type of conversion will be a year or two before each software system begins to malfunction. For most software that means the late 1990's, which certainly gives us plenty of time to prepare!

Even so, I predict that the upcoming arrival of a new millennium will catch many of those in the data processing profession by surprise. Conversion will be a big issue because organizations will not realize the magnitude of the effort ahead of them. I would like to present three possible conversion methods to help you estimate the effort that will be required when the time comes.

1) Full Upgrade. This technique requires converting data and programs to use a four-digit year. This is the most direct approach because it is a complete conversion that involves no gimmicks. But it may require a massive effort to find every occurrence of a date field, change it, recompile every program which references it, and test it all thoroughly. The source code for every program must be available, and the current compilers of the day must be able to successfully recompile that code. The amount of time for testing should not be underestimated, even though this seems like such a straightforward approach. Because of the extra data space required, the possibilities for stack overflows and inefficient blocking factors (or whatever errors future computer architectures give us) are endless. Reports may not be able to accommodate two extra columns of data, so conversion routines will probably need to be written to print the

date with a two-digit year. People will have no trouble when they see "00" as the year after "99" - that only confuses computers!

2) Logic Upgrade. This technique leaves data as it is but implements some additional logic to handle dates properly. For example, it may be reasonable to assume that years above "50" belong to the 1900's while the rest belong to the 2000's. This method makes sense for programs which frequently use dates with two-digit years but do few comparisons (such as determining which year is greater). In these cases, the cost of a full upgrade as described above may be too high to justify. These new date comparisons could be implemented as common routines which could be called by many different programs. While this is less costly than a full upgrade, we must still be able to locate and recompile the source code. The effort to test should be less than that required for a full upgrade because fewer changes are being made, but again this should never be underestimated.

3) Data Upgrade. If source code does not exist or cannot be recompiled successfully, the only option besides a total rewrite is to attempt a data conversion of some kind. This can get tricky, but it just may work. If dates are stored in character format (rather than packed or integer), the ASCII collating sequence contains several special characters which have an internal value less than the value of the character "0" (zero). At the time when year "00" is about to be introduced to the system, all the two-digit years beginning with "9" can be updated to start with a special character such as blank or asterisk (*). This will force dates with year "00" to sort after dates with year " 9" or "*9". Users will see years of " 9" or "*9" on their screens and reports whenever they access the old data, but people can be educated in advance about the change. This will work best in instances when the older dates will be aged off the data base soon after the conversion. There is some risk here that the strange-looking dates will fail a logic edit or cause other problems, but this technique can be viewed as an acceptable work-around when the only other choice may be to scrap the programs entirely. A similar approach would use characters like "A0" to represent the year 2000, since they will sort after "99"; but such dates would be harder to work with and would remain in our data for a longer time. As with the other methods, any changes must be thoroughly tested in each environment before they can be considered successful.

New Design Techniques

As we have already stated, the best way to avoid the problems associated with conversion is to design and code new programs today in a way that eliminates the basic problem. It is not too early to begin using this strategy. Most of the software that will cross over into the twenty-first century has not yet been created. Unless we adopt some new design techniques, we will be building time bombs into all of this new software, set to blow up at some time around the turn of the century. If everyone in the computer world begins to adopt these techniques over the next few years, there will be no need to worry about the conversion methods already discussed.

The first technique is so obvious that it hardly seems to be worth mentioning, but it is the key to solving the whole problem. We must begin to use four-digit years! The cost is two bytes per date, a small price to pay for extra storage when compared to the cost of converting software. Sometimes these two bytes can fit into wasted space at the end of a block of records, requiring no additional storage. Even when that does not work, it will take a half million dates to use up an additional megabyte of disc space, with a thousand dates fitting into the same space as a 25-line editor text file. Most importantly, the amount of effort required to begin coding in this manner is very small.

The second technique involves adopting formats that will represent dates unambiguously. This brings us back to one of the more subtle problems associated with date handling even today, which comes up occasionally at an international conference such as this one. Computers aside, those of us from the United States are used to writing our dates in the format Month/Day/Year. In many other parts of the world, a different order is used: Day/Month/Year. This can lead to confusion about the meaning of a numeric date on correspondence. Introduce computers and the many organizations that use them across international boundaries, and we are certain to have confusion when a date such as 03/02/86 appears on a report. Is this March 2 or February 3? There are 132 days each year when this problem occurs (12 months times the first 12 days of every month, less one day per month when both month and day are identical, such as 03/03). Each of these dates has two possible interpretations, but at least there is no confusion about the year since there is no month or day numbered 86. In fact, at no time since the invention of the computer has it been possible to confuse the current year with any month or day.

Now let's introduce the twenty-first century, which actually begins with 2001. This is about the only area where a two-digit representation for the year 2000 will not give us grief, since year "00" (2000) does not conflict with any month or day. But the year 2001 will cause a whole new round of confusion if it is displayed with only two digits. Sometimes computers use not only the two formats discussed above (Month/Day/Year and Day/Month/Year) but also Year/Month/Day, the order in which dates are often stored internally for sorting purposes. So how do we interpret 03/02/01? As March 2, 2001 (Month/Day/Year)? February 3, 2001 (Day/Month/Year)? Or February 1, 2003 (Year/Month/Day)? The number of ways to confuse dates increases significantly during the first 12 years of the new century, since the values "01" through "12" can now represent the month, day or year. There are 1716 dates where this confusion can occur during these twelve years (12 months times the first 12 days of each month times 12 years, less one day per year when all three are identical, such as 01/01/01). Each of these dates now has up to three possible interpretations.

This problem should be taken care of now by choosing unambiguous formats for displaying dates. First of all, the year should be displayed using all four digits, such as "2001". This can be phased in gradually between now and the turn of the century, since the two-digit year does not confuse us yet. Then we must find a way to distinguish the month from the day. One method would be to use ordinal numbers, such as 1st, 2nd, 3rd, to represent the day of month. These representations often differ across languages, so this does not solve the international problem. Since people rarely use them in correspondence, they might be awkward to computer users, and they could still possibly be interpreted as months. Another approach is to use alphabetic representations for the months, such as Jan, Feb, Mar. These are more familiar to people, and they cannot be confused with the year or the day. The problem of language differences still exists, but these abbreviations could be translated into the local language for displaying. A common set of routines, similar in concept to HP 3000 Native Language Support, could be implemented to take care of this localization.

The resulting dates have a format like "MAR 02, 2001" or "02/MAR/2001", which are both much clearer than 03/02/01. The important point is not what format we choose, but that every user understands exactly which date we are representing.

The third technique that will greatly aid our system design is the use of a productivity tool called the data dictionary. While dictionaries do not represent a new concept, many software applications are still being developed without them. A data dictionary maintains information about all our data elements, including which programs and data sets refer to each element. If we must expand a six-digit date by two digits to accommodate the year, the dictionary simplifies the conversion process by locating all the places where changes must be made. I predict that its value as a productivity tool will continue to be demonstrated by increasing popularity over the next few years.

Future Productivity Tools

The data processing world needs a whole new way to look at date-related information. Currently, dates are usually treated like any other piece of data containing six numeric characters, but in reality they have several unique properties:

1) Dates are rarely printed in the same format as they are stored. Slashes are often added, requiring program logic or an edit mask for every date field.

2) Dates must be stored in Year/Month/Day order for ease of use in sorting and comparisons. Since this is not how most users prefer to view their data, editing logic of some kind must be coded to rearrange the order.

3) There can be ambiguity about what a date field represents, depending on the order in which it is presented. As we have already seen, this ambiguity will be even greater in the next century when two-digit years such as "01" can be confused with the numbers representing months or days.

I propose that all future programmer productivity tools incorporate a special data type for dates. This data type will be handled within a data dictionary just like integer or character data is today. Every other tool that accesses dates, such as screen handlers, programming languages, code generators, and report writers, will be based on the data dictionary. Since all these tools will understand the meaning of the type called "date", they will all handle dates in exactly the same way.

The key here is that the internal storage technique can be completely isolated from the way users see dates. There will only be one way dates are stored internally, and every tool which accesses dates will understand that storage method. The technique must allow for the turn of the century and beyond, implying the ability to handle four-digit years, and for easy comparison between dates, implying an .ascending order. One example of such a technique would use 32 bits as follows:

High-order	16 bits:	Year	0 to 9999
Next	8 bits:	Month	1 to 12
Low-order	8 bits:	Day	1 to 31

Another example would start counting at some set date in history and add one for every day. We could also use eight ASCII bytes. While techniques like these are not new, the problem up until now has been that programmers needed to know the internal representation. This will no longer be true with our proposed productivity tools! No one accessing a date field will need to understand the internal storage method; the outer layer of tools will manage that for us. Some languages today automatically convert numeric data to a readable format for displaying; dates need to be treated the same way.

For this to work, all access to these date fields must be handled by the tools associated with the data dictionary. One objection to this technique has been that generic tools (such as QUERY/3000) do not understand the special meaning associated with these fields. In the future, generic tools must know what type of data they are accessing, beyond just character or integer. To accomplish this, all tools will need to use data dictionaries for their description of data elements.

The end result of providing these tools is that programmers will not need to worry about handling dates at all. Once a data element has been defined with a type of "date", the computer will take over with a set of common routines that understand both the internal storage technique and the default display formats for dates. The software developer or end user only needs to refer to the data element, and the computer will handle:

Sorting and comparisons
Editing (for valid dates)
Range checking (such as dates between March 1, 1986 and March 31, 1986)
Displaying in the proper format

This represents another step toward making computers friendlier to everyone who uses them.

Conclusion

Although we still have plenty of time to implement all of this by the year 2000, the years will go by faster than we realize. We must start thinking along these lines now. Those who develop productivity tools should begin to plan how their software will handle the arrival of the twenty-first century. And those who create software using these tools should begin to ask about plans for the future. It is not yet too late by any means, but as the British poet Andrew Marvell once said,

> "At my back I always hear
> Time's winged chariot hurrying near."

The time will creep up on us sooner than we think. So let's be ready, because on Monday morning, January 3, 2000, no one in data processing will be able to say that they were not given enough notice to solve these problems!

Biography

Kevin Cooper is currently a Software Engineer (SE) with Hewlett-Packard Company, Neely Sales Region, supporting HP3000 installations in the San Francisco Bay Area. Prior to this he worked for eight years as a programmer/analyst and project leader in Hewlett-Packard's internal Information Systems organization. He holds a bachelor's degree in Computer Science from the University of California, Berkeley.

INTERACTIVE SYSTEMS: INTERFACING WITH THE HP3ØØØ

Antonia Stacy Jolles
SofTech, Inc. Oakland, California U.S.A.

Summary

The interactive system, interfacing large-scale and small-scale computers in the office environment, is in its infancy of development today, but is rapidly shaping the direction of future development. Research concerning interfacing and networking PC's and large systems is at the forefront of today's state-of-the-art designing. Our current methods of information management are inadequate for maintaining efficient and compatible interactive systems. A management scheme that considers the time line of technological advancement as well as the abilities and techniques of management practices must be developed now to insure the successful future of the interactive system environment.

The introduction of the personal computer into the office environment is changing our methods of information processing and information management. The relationship between personal computers and the larger systems they must interact with affects our strategies for information systems management, our utilization of hardware and software technology, as well as the directions of future development. The office of the future will be a haphazard collection of non-interactive systems unless we strategize for the implementation and management of successful interactive systems.

The interactive systems environment, resulting in offices with a distributed data base, presents many advantages that have influenced its popularity in the corporation. The use of personal computers presents a cost efficient method of providing additional intelligent terminals to tap into the large-scale system, either a mini or mainframe. The personal computer, less expensive than most intelligent terminals, provides an additional data input station to share the costs of utilizing the expensive peripherals already available as part of the mini or mainframe system. In most cases, for about one-quarter of the cost of an intelligent terminal, a personal computer can be used to interface with the office's large-scale system, either directly through a port, utilizing terminal emulation, or using a modem and telephone line.

The advantages of using a PC in the office environment are not limited to its role as an effective intelligent terminal communicating directly with the office mainframe. The personal computer provides the added advantage of acting as a stand-alone work station. Processing on a personal computer can provide the user with a less intimidating situation in which to develop computer skills. Applications programs written for the PC accomplish complicated tasks with a user-friendly approach. The user has a direct line communication with a personal computer, knowing that the computer is functioning for them alone.

The personal computer provides the flexibility of a mobile work station. Access to computing power may be sought from different locations, most often the office, but also from home and during business travel. The personal computer and the more portable lap computer provide a work station with enough mobility to carry on processing away from the office desk.

Numerous software packages have been written specifically for the personal computer. These are not packages that were first developed for large systems, and then scaled down for use on the PC. Rather, the software has been designed directly for the PC, taking advantage of the PC's dedicated processing, while at the same time freeing the large-scale system for the tasks best conducted on a large computer. Throughput time is increased on the large-scale system as more and more applications and tasks are run at the corporation's stand-alone work stations. Huge data base processing requiring the resources of a large system can be run without the added CPU burden of maintaining several other smaller tasks simultaneously. The large system is freed up to process exactly the kind of large-scale computing tasks it was designed to conduct.

The more sophisticated software development becomes, as it already has developed today, the simpler the system can appear to the end user. With the proliferation of the home computer and the recent introduction of the personal computer into the office environment, microcomputer software applications programs have been developed with the ability to conduct almost any complicated task previously reserved for the computing power of the larger systems. The inexperienced personal computer user is able to find user-friendly software to assist in the most sophisticated of tasks. These complex applications programs designed for the PC are increasingly requiring less computing knowledge on the part of the user, while at the same time performing more sophisticated functions.

The personal computer brings computing power to a wider audience. The easy-to-use applications available for the microcomputer have brought computer tools closer to the user so that the user has immediate access and control over their work. In addition, users are able to customize their tools. The PC as a stand-alone system allows the user to develop custom software to conduct any specialized tasks.

But the personal computer goes beyond functioning as a stand-alone work station. Interfacing to the larger systems has introduced recent computer users to the power of the mini and mainframe as well. The personal computer in the office environment, networked to the larger systems, has provided the missing link necessary to successfully introduce the inexperienced user to the large-scale computer. The relatively new user is now able to tap into the complex resources available on the mini or mainframe, but perhaps with the disability of not possessing the more sophisticated computing knowledge required in the mini or mainframe environment.

Further advantages are gained with the introduction of the personal computer into the larger context of the corporate computing environment. Off-site stations are able to perform computing tasks using a personal computer and then uploading the data via modem to the main site's mini or mainframe. The idea of the distributed data processing system becomes expanded when the stand-alone work station, away from the corporate large-scale computer system, can interface from absolutely anywhere a telephone line can be installed. Corporate branches can share the master data base of the main branch while still conducting their computing tasks at the local level.

Experience tells us that problems arise with the introduction of any technological information processing advancement into the work environment. Problems with information management occur from the end user's point of view as well as from the vantage point of system management.

From the end user's standpoint, data downloaded from the large-scale system and manipulated on the PC may be obsolete before the user has an opportunity to upload the processed information back to the main system. As the PC user is manipulating information at the stand-alone work station, others may be conducting further processing on that same data, with none of the parties aware of the other's actions. Not only does this waste the computing time of the users involved, but data rapidly becomes obsolete and incompatible. We have quickly lost all the advantages gained by the technology of a shared large-scale system. The possibility occurs of having the same or similar data maintained and updated in several different locations with no method to insure its logical coordination.

Hardware and software problems exist that effect the efficiency at both the end user and the management level. With the introduction of more and more personal computers into the office, problems of hardware and software compatibility arise. In the past, purchasing hardware or software within the confines of a single large-scale system necessitated the purchasing of compatible equipment. New acquisitions had to be compâtible with the existing large-scale system. But the purchasing of personal computers in the office environment can be done without regard for the compatibility with the large-scale system, until the time comes to network and interface. Then the problem becomes a crucial one. Software packages developed for use on PC's are sometimes incompatible with their counterparts on the larger system.

Files developed at the stand-alone work station simply may not be able to be uploaded to the larger system. But we are still left with slow processing time and multiple steps involved in using these software programs.

Emulation and conversion software is most often complicated and time consuming to use. Emulation software considerably reduces the response time on each process. In many cases, the emulation software itself requires a fairly knowledgeable user to conduct the several keystroke steps involved. Conversion software often requires several phases of conversion before the data is actually ready for transmission. Emulation and data conversion work, but within the confines of current software's capabilities.

Problems of hardware compatibility also occur. If a modem is not the networking tool, then the personal computer must be networked directly to the larger system. More and more personal computers bought for the office environment are being purchased with emulation boards already installed. Selecting the networking protocol must consider the compatibility of all the system's hardware and software involved. The definition of "the system" is expanded to include not only the mini or mainframe and its peripherals, but all the different designs of personal computers networked to the large-scale computer.

Many more problems are cropping up with the introduction of personal computers into the office environment. Management Information System (MIS) departments throughout companies are having to reassess their approaches to system management. The function of MIS, to manage the hardware and software of the corporation's mini or mainframe, and maintain an accurate data base, has become further complicated with the introduction of the personal computer. MIS functions have expanded from maintaining one large-scale system, to maintaining the large-scale system and the many stand-alone stations accessing the larger system. MIS is no longer managing a single centralized system, but is now trying to grab the tail of a decentralized computer network before the technology takes off beyond management's ability to regulate it.

Up until the introduction of the personal computer into the mainframe environment, MIS was able to control the flow of information in the large-scale system. The available technology provided adequate methods of regulating access to data through security measures activated by software. With the introduction of the PC networked to the larger system, MIS departments are losing control of the flow of information between the large-scale system and the PC user. A PC user networked to the company's mini or mainframe can download any information available at his security level from the large system down to the PC. Once this information resides on the PC, MIS has lost control over regulating the security of the data. Any other user can come along and access the data directly off of this same PC, or via a network, the information can move from PC to PC.

In addition, PC users can develop their own programs. Data originally downloaded from the large-scale system can be manipulated at the stand-alone station using custom designed programs without the supervision and control of the MIS department. This data can then be uploaded back to the mini or mainframe, thus corrupting the integrity of the data base of the large-scale system. The flow of information is out of the hands of a regulating MIS department, and cast into the hands of end users who lack the experience and the coordination to successfully regulate the flow of data.

The advent of hardware and software able to interactively network large-scale and small-scale systems has introduced the personal computer into the office environment at an exponential rate. Specific hardware and software problems had hardly been addressed before the personal computer began taking its stronghold in the corporate environment. The relatively inexperienced PC user has found himself having to learn a variety of applications packages specific to the personal computer quickly, and in most cases, on his own. In addition, networking the PC to the company's large-scale system, has forced this same PC user to learn emulation programs, networking software, communications packages, and conversion programs. Overnight, the office PC user has been forced to learn a vast amount of software packages with very little, if any, training assistance.

In addition, the PC user is finding that these software packages are more complex. Emulation programs generally take the user through a series of ten to twenty steps in order to begin passing data back and forth between the personal computer and the mini or mainframe. Users are finding, sometimes by hit and miss, that files created at their stand-alone work stations are incompatible with the larger system. In some instances, this means a complete duplication of work, or seeking out conversion programs that can handle the task.

Networking interactive small-scale and large-scale systems together is in its infancy of development. The software and compatible hardware necessary to insure successful interactive networks is slowly appearing on the market now. Many successful integrations of personal computers with the HP3000 are currently in use in many corporations. IMAGE/3000 data base extraction techniques are used to download data to a spreadsheet package on a PC for "what-if" analysis. Data can be extracted from personal computer data base applications and uploaded to the HP3000 for input to IMAGE/3000 based applications.

The Data Interchange Format (DIF) allows various unrelated applications to exchange data. Output from one application is formatted so that it may be processed and utilized by another applications program.

Command features of DSN/LINK can be used to set up a variety of time and keystroke saving routines. These routines can then be used for data transfer without user interaction. This eliminates the complex strings of commands sometimes necessary during data transfer applications between nodes in a network.

Development is still underway for future packages that will provide simple solutions to incompatibility and emulation problems. Future software for interactive systems must be able to link small-scale and large-scale systems regardless of the structure of the DBMS. Centralized batch data entry can be eliminated. We must be able to interchange data simply and efficiently between applications packages. Current networking and software emulation limitations must be overcome. But we are still looking to the future for technology to catch up with the complex of interactive networked systems already existing in many office environments.

Networking and interfacing technology has quickly advanced beyond the ability of our managerial skills to deal with issues of system compatibility, data base integrity and security. Future technology is not catching up soon enough with tools for regulating the security and compatibility of data on complexes of networked systems. Problems of data security, system management, and system compatibility must be solved, during the interim, while the technological tools catch up to our present day needs. MIS departments are at risk having to invent security measures for regulating their data bases and maintaining security on the growing network of systems, or risk losing their reign over the computer system.

From the viewpoint of the end user, the advantages of personal computers in the office environment far out weigh any disadvantages. The personal computer within the context of a networked system has taken hold within the corporation. The issue is not whether or not the personal computer can successfully integrate into the mini or mainframe environment. The issue is whether or not the mini or mainframe can adapt to the growing PC environment of the office.

MIS departments must catch up with the technology and begin strategies for information management within the context of networked systems, stand-alone work stations and distributed data bases. The first issue that must be addressed is providing training to the growing number of users tapping into the more easily accessible computing network. The introduction of personal computers into the office environment has brought many new and less experienced users on-line with the larger systems. Personal computer users are frantically trying to keep up with the latest in emulation, networking and communications software, as well as the myriad of applications packages available for the stand-alone work station.

In response to this growing need, information centers, support centers, or training centers are appearing in many corporations. In the broadest sense, the role of these centers in the office context is to insure the user's involvement in the corporations larger data processing environment. The proliferation of stand-alone work stations creates a situation in which the end user is able to isolate from the larger data processing environment. There is less communication between users within a corporation. The one time hope of computers eliminating the duplication of work between divisions within a corporations is lost as the information becomes isolated on the personal computer. The end user must still be involved with the larger system to insure communication between personal computers within a corporation and between PC's and the large-scale office computer.

The functions of an information center within a corporation can vary greatly. Essentially, the information center must provide a sharing of data processing skills. Foremost of these is acquainting the user with issues and methods of data security. With the growing number of personal computers in the office, acquainting the user with issues of data security becomes even more pressing. Users must be encouraged to maintain good habits of information management themselves. Issues that were once the realm of MIS departments are now spread to the user who becomes his own small-scale version of an MIS department managing the security of data on the personal computer.

The PC user is required to conduct his own daily or weekly backup of files. In the large-scale system environment, the MIS department conducts a disk to tape backup on a regular basis to insure the safety of files against gliches, or hardware failures. The PC user must be trained to conduct his own backups, either to a floppy disk, a tape backup or to the larger system itself. The latest version of the 2622 emulator allows the PC user to interface with the HP3ØØØ to upload an entire disk for backup. The PC data backup process is directly linked to the larger system. Information Centers should encourage good data management practices and provide the necessary training to PC users.

These Information or Support Centers often provide training for the office's personal computer users on a drop-in basis. The personal computer user, exposed to the vast marketplace of PC applications programs needs guidance on issues of purchasing, compatibility, as well as training. Users can work with the Information Center staff to identify the most efficient and compatible applications, with consideration given to cost factors.

The growing concern of company's starting up their own information centers is under who's authority should they reside. There are arguments urging that the information center is necessarily a part of MIS. There are just as convincing a set of arguments on the other side that insist the information center should be independent of MIS. The question should really begin with an understanding of the role of the personal computer in the office environment. Is it something that can come under MIS control at all. Our traditional methods of information management would insist that it does. But the personal computer assumes a different role entirely in the office than does the large-scale system.

MIS and the users need to define the role of the personal computer in the large-scale system environment. Areas of great security risk are best left on the mini or mainframe. There are areas of data processing that are best suited for application on the personal computer. Some accounting tasks are more efficiently performed on the PC, rather than the company mini or mainframe. Spreadsheet applications requiring large amounts of memory to load are best used on the PC where processing is dedicated and the mini or mainframe is unburdened. Spreadsheet packages are CPU intensive, slowing down the entire system in performing other tasks simultaneously. Spreadsheet analysis is best conducted on a PC using the dedicated processor, relinquishing the mini or mainframe processor to conduct other activities.

Custom programs geared to the specific needs of a user can be made available at the PC level for individuals, as long as these programs are catalogued and maintained by the Information Center. The cataloguing of custom as well as standard applications programs should be maintained by the Information Center, thus insuring the legal handling of software duplication as well as software maintenance.

The personal computer is an element in current data processing that defies our previous management definitions and abilities. In order for the Information Center to keep a handle on their job, the role of PC's in the office environment must be defined. The Information Center must keep the presence of different manufacturers of personal computers in the office to a minimum. By restricting the variety of models and the applications programs used, the Information Center can maximize the sharing of information between users on the PC network. This reduces the strain on the Information Center's resources.

With a strategic plan to guide the manager of the office computer network, fewer purchases of "state-of-the-art" software and hardware will occur. Implementation of a strategic interactive systems management plan would actively direct the purchase of only those technologically advanced tools that are in keeping with the direction of the office's future development. The office system network will begin to take on a cohesive and planned direction. It is up to management to redefine its role as information managers, and develop new methods of information management to include the role of the personal computer in the office.

Antonia Stacy Jolles has been with SofTech, Inc. for one year. She has been a Technical Writer involved in the computer industry for the past three years. Currently, she is working on an HP3ØØØ network designing and writing the documentation and graphics that accompany the custom software SofTech develops for the San Francisco Department of Energy. In the past, Ms. Jolles has worked extensively with personal computers and multi-user microcomputers documenting both engineering and end-user manuals. In addition, Ms. Jolles teaches Hewlett-Packard word processing skills and designs and teaches courses in Technical Writing and Graphics Production to university level students.

Winston Prather
Hewlett-Packard Cupertino
U.S.A.

NOTE: See page 647.

How to Compute Master Data Set Capacities

by

Kurt Sager

SWS SoftWare Systems AG
Schönauweg 8
CH-3007 Bern
Switzerland

Summary

Defining optimal capacities for IMAGE master data sets is not just a matter of getting a number from a prime number table. Many other factors influence the access performance of master data sets. A high percentage of synonyms not always means long DBPUT transaction times, 0.1% synonyms however may indicate severe performance degradation if other conditions are met. A method is presented which allows to compute master set capacities such that *perfect hashing* (no synonyms at all) is achieved in many practical cases.

References:

[1] IMAGE Data Base Management System Reference Manual HP-32215-90003

[2] The IMAGE/3000 Handbook / R.M. Green, F.A. Rego, F. White, D.J. Greer, D.L. Heidner. - Seattle : WORDWARE, 1984. - ISBN 0-914243-00-4

[3] IMAGE: An Empirical Study / B.D. Cathwell. - Proceedings HP3000 IUG 1984 Anaheim Conference. - p. 4-1 to 4-5

Introduction

In appendix C of HP's IMAGE Reference Manual [1], the following statement is written:

> *"A master data set capacity equal to a prime number or the product of two or three primes generally yields fewer synonyms than a master data set capacity of many prime factors."*

The appendix in the same manual shows a table of selected prime numbers from 101 to over 8 million.

Participants of IMAGE training classes are told to use *prime numbers* where possible to reduce the number of *synonyms* in master data sets.

ADAGER, the well-known data base utility package, refuses any number but a prime number as a master data set capacity to be changed. And other IMAGE utilities make similar propositions.

These statements and practices may raise the questions below:

> *Where is the mathematical proof which demonstrates the superiority of prime numbers for master data set capacities?*
>
> *What is the real impact of synonyms on system performance?*
>
> *If system throughput and response times suffer from synonyms, how should the capacity be changed to cure the problem?*
>
> *Are prime capacities really the solution?*

Findings from practical experience

Ten years of practical experiences with many IMAGE data bases prove that best performance is not guaranteed by just using prime numbers as master data set capacities.

Other factors have much more influence on the number of synonyms and possible performance degradation. It is generally known, that the load factor (the ratio of the actual number of entries over the capacity) of master sets should not exceed 70 to 80 percent. Although IMAGE allows master sets to be filled up to 100% severe performance degradation can be observed if entries are added to a master set already more than 80 to 90% full.

But in many cases adding entries to a nearly full master set shows no performance degradation. Sometimes however terminal response times become inacceptable due to synonym problems with master sets less than 50% loaded. Why?

It seems that the key item value distribution, or perhaps some specific value patterns, are much more important whether or not a significant amount of synonyms is generated, rather than choosing a prime or non-prime as capacity.

How Hashing Works

A very common task in today's data processing applications is to store an information entity on a secondary mass storage device (disc) for later retrieval. Each information block shall be identified by a unique key item value.

In computer science several methods are known to store an entity of information to a file and retrieve it at a later time, based on the value of the chosen key item.

On the HP3000 two techniques have been implemented and are available to programmers as part of the fundamental operating software package (FOS):

- the *B+ Tree technique*, used in KSAM files
- the *Hashing method*, used in IMAGE

The *B+ tree technique* implements an ordered sequence of the key item values as a tree in natural order (numerical or alphabetical order). The main advantage of this technique: processing in key order sequence is easy. The time to insert or to locate a given entity grows with file size.

With *hashing*, a defined algorithm computes a relative file address using the search item value as argument. Insert and retrieval times are independent of the file size. Ordered sequential processing is not possible, however.

IMAGE uses two different *hashing algorithms* to compute a file address, depending on the search item type.

Binary Type Search Items

If the the search item is of type I, J, K, or R (binary type), then a quite simple method is used:

$$a := (v - 1) \text{ modulo } c + 1$$

where

v is the search item value
c is the capacity of the data set
a is the computed data set address
and modulo means: "remainder of the division of (v - 1) by c"

The few examples below show how this formula works:

		computed
search item value	capacity	address
v	c	a
10	1000	10
100	1000	100
777	1000	777
1001	1000	1
1010	1000	10
1234567	1000	567
10	500	10
100	500	100
777	500	277
1001	500	1
1010	500	10
1234567	500	67

The mechanism of this technique can be visualized by mapping the domain of the key item values to a circle representing the address space of the master data set. Of course, the circle has a circumference of lenght c, the capacity.

ASCII Type Search Items

If the search item is of type X, U, Z, or P, then a much more complicated algorithm is used (see the *IMAGE Handbook* [2] for detail information and example calculation).

Synonyms

Starting from a key item value the hashing algorithm calculates an address between 1 and capacity c. The entry is then stored at this address if it is not yet occupied by an earlier added entry.

Only for a few special cases hashing algorithms perform so well that every (different) key item value maps to a different address, however. Thus sometimes a key item value generates the same address as an entry added earlier. This is called a *collision*. IMAGE then has to find the next free space to store the new entry. This new entry is called a *synonym* to the previous one, and is linked to it by forward and backward pointers. All different key items values hashing to the same address build-up a so-called *synonym chain*.

Hashing is said to perform well if few synonyms are found in a master data set relative to the number of entries. A good hashing algorithm spreads the entries as uniformly as possible into the available address space. As the master data set is filled up, the number of free locations diminishes, and therefor the probability of collisions will increase.

The programs *DBLOADNG* from the *Contributed Software Library*, and *HowMessy*, a bonus program for customers of *Robelle* products, show the percentage of synonyms and the average/maximum synonym chain length of every master set in a data base. *HowMessy* runs about 10 times faster as *DBLOADNG.* It is well documented in the *IMAGE Handbook* [2].

Performance considerations

Does a high percentage of synonyms mean bad performance for data base operations?

Not generally, performance depends on many other factors too!

The processing time to *read* data from a master set by a keyed DBGET is only marginally dependent on the synonym percentage (see the column *Inefficient pointers* in the *HowMessy* report).

The case of *adding* entries to a master set is more critical. If IMAGE can find a free entry in the near vicinity of the collision point, hopefully in the same disc block, then it's just a matter of pointer settings.

If there is no free space in the same disc block then the next block must be read from disc, and so on, until a block with free entries is found.

The maximum number of entries per disc block is equal to the blocking factor, the ratio of block size over entry length.

For data bases with many online accesses the block size should not be increased over the default 512-word size. The entry size of masters however should be kept as small as possible. Bad examples are manual master sets with many or big data items, good examples are manual or automatic master sets containing the key item only.

Therefor: the smaller the entry size the higher the probability of a free entry in the same disc block.

Many fully occupied blocks side by side, called *clusters*, are very dangerous if a calculated hashing address falls in the beginning of this zone. IMAGE then has to read many disc blocks to find a free entry for this DBPUT. During this sequential reading the whole computer system is locked for any other processes, including MPE!

In the column *Max.Blocks* the program *HowMessy* shows the maximum number of contiguous completely filled blocks for every master set. If this number is very low, say zero or one, then even a high percentage of synonyms does no harm, and DBPUT's will process as fast as ususal.

A large number for *Max.Blocks* indicates a potential performance problem, which actually already may occur if there are *any* synonyms present.

Every production data base should be analysed by *DBLOADNG* or *HowMessy* at regular intervals, once per week for example. The resulting output (one line per search item and per data set) should be checked for possible problem figures, as decribed in the *IMAGE Handbook* [2], chapter 23.

Analysing and correcting critical hashing situations

Our experiences show that especially search items of binary type need attention. The following remarks apply to this type of search items only: the I, J, K and R types.

It is quite frequent that binary serach item values are forming a (nearly) continuous sequence of integers, such as 101, 102, 103, ...

We can say they belong to a (almost) dense interval *[a,b]*, where *a* is the lower limit, and *b* the upper limit.

The corresponding master set capacity is therefor at least $c := b - a + 1$. Applying the simple hashing algorithm to this case we easily see, that every search item value maps to a *different* master set address. The result is *perfect hashing (no synonyms at all), no matter how full the data set is!*

If the search item values belong to *two or more intervals*, it becomes more complicated to see what happens. In fact this case is usually the *reason for very long response times* of DBPUT's if the intervals are very dense and mapping them to the address space causes overlapping of the intervals.

To overcome this quite frequent problem, the program *MASCAP* has been written and contributed to the *Madrid SWAP TAPE.*

For search items values belonging to several intervals *MASCAP* cpmputes master data set capacities such that *perfect hashing* will be achieved. The program aks for the lower and upper limits of every interval, the maximum allowed capacity, and then shows ranges of perfect hashing capacities.

The examples below show how *MASCAP* works (user input is written in *italics*):

```
:RUN MASCAP

MASCAP / Version 1.0 / (C) 1982 SWS SoftWare Systems AG, Bern

MASCAP computes Perfect Hashing Capacities for IMAGE Master
Sets with binary type search items (In,Kn,Jn).
This algorithm assumes that the search item values belong to
several relatively dense intervals.

The program will ask you to enter the number of intervals,
then for each interval the lower and upper limits.


Number of intervals: 2
Limits (min,max) of interval  1     ? 8400001,8412000
Limits (min,max) of interval  2     ? 8500001,8502000
Minimum capacity allowed:  14000
Minimum capacity: 14000
Maximum capacity: 25000
Perfect Hashing Capacities from 14572 (96.1 %) to 14666 (95.5 %)
Perfect Hashing Capacities from 17000 (82.3 %) to 17600 (79.5 %)
Perfect Hashing Capacities from 20400 (68.6 %) to 22000 (63.6 %)

Number of intervals: 3
Limits (min,max) of interval  1     ? 8400001,8413000
Limits (min,max) of interval  2     ? 8500001,8512000
Limits (min,max) of interval  3     ? 8600001,8602000
Minimum capacity allowed:  27000
Minimum capacity: 27000
Maximum capacity: 50000
Perfect Hashing Capacities from 28858 (93.6 %) to 29000 (93.1 %)
Perfect Hashing Capacities from 37334 (72.3 %) to 37400 (72.2 %)
Perfect Hashing Capacities from 40400 (66.8 %) to 43500 (62.1 %)

Number of intervals: 0

END OF PROGRAM
```

The figures above show that *perfect hashing* can be achieved using even numbers as master set capacities, for example 17000 from the first case above. When choosing a capacity outside the reported ranges catastrophical situations may occur.

In a real case observed at a customer site, a capacity of 15013 (a prime number of course!) and a load factor of about 70% one day produced a sudden jump in response time from 1 second previously to 45 seconds for every DBPUT to this master set. *Decreasing* the capacity to 14600 cured the problem!

An other author (see [3]) reports that the benefits of prime capacities are rather a myth than a miracle!

Program *MASCAP* needs the interval limits as input for each run. Sometimes however the number and the limits of each interval are not known in advance. To help in such situations the program *MASANAL* has been developped. It takes a specified master data set as input and then reports all relatively dense intervals found. The output can then be used by the program *MASCAP* to compute optimal capacities.

Conclusions

To obtain best master data set capacities in specific situations the following rules are proposed:

If the search item is of binary type and all values are contained in *one* relatively dense interval between *a* (smallest value) and *b* (largest value) then use a capacity of *c := (b - a + 1). The resulting hashing will be perfect (no synonyms) independent of the load factor.*

If the search item is of binary type and all values are contained in several relatively dense intervals then use the program *MASCAP* to compute *perfect hashing capacities.* If the interval limits and their number are not known then the program *MASANAL* should be used first to extract these figures from actual search item values stored in the master set.

In both cases the performance of existing data bases can be substantially improved without any changes in existing application programs.

If the numeric search item values are distributed more or less randomly then use a prime as master data set capacity and define the search item as binary type. In fact, the theory on pseudo random number generators explains that prime numbers are important in algorithms of "modulo type" to obtain good random distributions. Is this the origin of the prime number myth in IMAGE?

In all other cases define a search item as ASCII type and use *any* suitable number as master data set capacity, provided that it is not a power of 2 and that the load factor does not exceed 70 to 80%.

Interex HP3000 Madrid Conference, March 1986
Theme: 'Migration to 2001'

... networking heterogeneous environments ... a commonplace in 2001 ... with information captured and processed at one node communicated to other very dissimilar nodes for perhaps surprising transformations and end uses ... in the Migration to 2001 ....

What's a Nice Computer Like the HP3000 Doing in a Place Like CAD/CAM?

Sam Boles, Member Technical Staff

HEWLETT PACKARD

Here's a simplistic mechanism to communicate quickly and briefly the basic concepts of CAD/CAM (Computer Aided Design, Computer Aided Manufacturing), and the networking of the diverse technologies involved.

We start by building an IMAGE database on the HP3000 with name and other fields. We extract names from the database and network them over to the HP9000 where the ASCII string is sized, scaled, centered vertically and horizontally and transformed into tool path geometries for downloading to a 3-D CNC milling machine to produce name plates.

Then it's back from the HP9000 to the HP3000 for engineering specs documentation to wrap up this "CAD/CAM in microcosm," with a glimpse of a new methodology that can increase productivity by an order of magnitude in product development.

This unusual heterogeneous network result for an otherwise ordinary IMAGE database on the HP3000 can give you an informal introduction to the world of CAD/CAM.

... evolutionary perspective: it wasn't our size, strength or speed -- but our tools that put us on this end of the leash ....

Of all the animals whose ancestors crawled up out of the swamp that day, we're ***not the most imposing***.

Look at our ***size***. Lots of animals are bigger. Elephants. Whales. Giraffes. To name just a few.

Look at our ***strength***. Ever arm-wrestle a gorilla? Ever wonder why Budweiser uses Clydesdales to pull the wagon instead of a team of Arnold Schwarzenegger's?

And ***speed***. Do you think dolphins aren't allowed in the Olympics because they're not ***fast*** enough? How many gold medals do you think Spitz would've won that year if there'd been one even ***average*** dolphin in the swimming events? No, it wasn't our ***size, strength or speed***. It was our ***tools*** that made the difference. That's why the leash is around our ***hand*** and ***not around our neck***.

Sure, the ***opposing thumb*** was handy. And being able to balance, however precariously at times, on ***two feet*** was a help. But it was our tools that really made the difference.

Tools meant we could ***see things that weren't there.*** We could look at a tree and see a lever, a club, an axle, a torch, a spear, a raft, a shovel handle, even a baseball bat, for those who would take the world serious . . . .

. . . CAD/CAM as a transformation of data from the abstract to the concrete . . . a simple case . . . .

As the centuries passed our tools got more refined. One of these tools in the long heritage is CAD/CAM (Computer Aided Design, Computer Aided Manufacturing). Let's look at a simplistic example of CAD/CAM and see some of the basic principles at work. A "***CAD/CAM in microcosm.***"

Let's start with an IMAGE database on the HP3000. It can be a personnel-type database with people's names in it. Our data capture can be ordinary, too: thru a terminal where we touch the key with the letter "A" on it and thru some magic a nobit-nobit-nobit-bit-nobit-bit ends up in the memory of the 3000. A binary group in an NMOS memory array that can be interpreted thru the ASCII coding scheme as the letter "A."

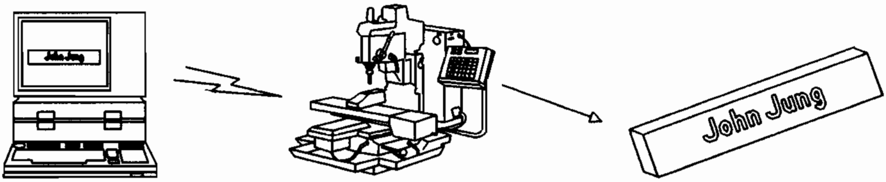
Now this is ordinary everyday commercial data processing on a computer. Nothing particularly marvelous about that. Well, marvelous, yes -- but we do it all the time everywhere so we've gotten used to the marvel.

Next let's extract some names from the database and communicate them via an RS232 ***network to another computer***, an HP9000. We can use QUERY to do the extraction and formatting and off-the-shelf drivers to send the name data serially over a twisted pair at 9600 baud to the 9000 running in terminal emulator mode. This is our CAD engine.

Now let's look at the next node in the network, that provides the CAM part of this "***CAD/CAM in microcosm.***"

. . . the CAM node in the network . . . a 3-D CNC machine tool with RS232 interface . . . .

The specific CNC (Computer Numeric Control) machine tool we'll focus on here is the ***Dyna 2400 milling machine.***



It's made by Dyna Electronics in Santa Clara, California, and represents some of the more advanced CAM technology available today.

The machine is designed to mill small, high-precision parts such as those typical of the electronics industry. The three step motors that provide the x-, y- and z-axis movement on the 2400 have a resolution of 1/10000 of an inch. The travel is about 6 inches in the x-axis, 5 inches in the y-axis and 4 inches in the z-axis. The maximum feed rate is about 30 inches a minute with synchronous and asynchronous control in each axis for full 3-dimensional capabilities, such as machining a hemisphere. The spindle can be throttled up to 10,000 rpm.

The on-board microprocessor is programmable in stand-alone mode or may be downloaded from a host via RS232 interface with line, subroutine and full-program granularity.

The instruction set includes the common control and vector move commands, but in addition is enriched with polar coordinate commands which simplify programming and provide powerful 3-D capabilities.

The machine's footprint is about 2 feet by 2 feet, with a height of about 2 feet and weight around 220 pounds. This gets it close to "***desktop***" class and enables comfortable positioning on a engineer's workbench 'or some interesting methodology implications as we'll see later.

. . . the "CAD" transformation of the ASCII symbol into 3-D CNC tool path geometries . . . .

Meanwhile, back at the HP9000, we have our ASCII symbol: someone's name or similar data. The data could just as easily be a ***vertex list for polygonal graphics or other 2- or 3-dimensional geometric descriptions with vector and/or polar coordinates.*** For our CAD/CAM example, we've chosen an ASCII symbol to give a simplistic CAD illustration of transforming an alpha character into a tool-path geometry. Here's a narrative of what we do:

```
    I.  Abstract:  This module is the CNC milling machine (Dyna 2400)
        stager and driver for name plates to illustrate CAD/CAM
        capability via a simplistic example.  The process includes the
        design implications of transforming text to tool path
        geometries with sizing, translation and scaling; the driving
        of the milling machine with vector and polar coordinates as
        well as other control commands is an example of relatively
        sophisticated computer-aided manufacturing.

        The module loads external tables of width, key and geometries
        for tool path generation, gets a name/string, scales/centers/
        gens tool path commands for the machine, downloads the path
        program via RS232 link, and graphically simulates the milling
        action.

   II.  Input
     1. External table of subroutine reference key and size
     2. Tool path geometries in Dyna format
     3. Name(s) from HP3000 IMAGE database via terminal emulation

  III.  Processing
  1. Initialize flags, counters
  2. Load path, subroutine, sub index and width tables
  3. Get name/string, scan, scale, center
  4. Gen tool path program for download to CNC
  5. Download tool path program from 9000 file to CNC
  6. Scan/parse, simulate tool path with graphics CRT

IV.  Output
  1. Graphic (CRT) replica of name plate, scaled and centered
  2. Temporary disc file ('TEMPPROG') of Dyna tool path code
  3. RS232 transmission of TEMPPROG to Dyna programmable controller
  4. Graphic (CRT) generation of tool path to simulate milling

 V.  Techniques/Controls/Considerations
  1. Subroutine/width table (default 'MMTBL')
      Format:  123456789
               A 67 2.45
               A=subject letter; 67=subroutine #; 2.45=width
  2. Tool path table (default 'MMLTR')
      Format: 123456789ABCDEFG
              851 GR a-180.000
              851=Dyna program line number; GR=Go Relative (to
               local zero/tool location; a=angle of polar
               coordinate;-180.000=angle size in degrees
   . . .

  3. Data structures and linkages
    Ordinal value of character in string is offset into
      |
      |-> subrtntbl (0..255 INTEGER)----------------->|
      |    program gen, offset set here is used to    |
      |    match subroutine # from pathtbl to build   |
      |    path index                                 |
      |                                               |
      |-> widthtbl (0..255 REAL)                      |
      |    letter width in base mm unit to size for   |
      |    scaling                                    |
      |                                               |
      |-> pathidx (0..255 INTEGER) <------------------|
            index to path table, points to start
      |<-- of geometries for a given letter
      |
      |-->pathtbl (1..16232 CHAR)
           tool path commands with variable number
           of 16-byte records comprising the sub-
           routine for the geometries of a given
           letter
```

For each letter in the alphabet, we design a scalable, relocatable tool path. For simplicity we use a Helvetica font style (that is, no serifs as in a Roman font), which blends aesthetically with a 1/8 or 1/16 inch ball end mill in the travel range available. To further enhance the aesthetics we use relative instead of absolute positioning at character termination to enable a proportional Helvetica font.

Here's an example, using the letter "B":

```
093 SUB        12
094 GO Z-   0.200
095 GR Y    5.000
096 GR X    1.100
097 ZERO     XY
098 ZERO AT
099    Y-   1.200
```

```
100 GR a-180.000
101 GRcX-   1.100
102 ZERO AT
103      Y-   2.500
104 GR a-180.000
105 GR X-   1.100
106 Z>C
107 GOfX    1.300
108 SUB RETURN
```

This shows you in the Dyna CNC vernacular, some of the mechanics of getting your CAD design transformed into something tangible like metal, plastic or wood.

Here's a loose translation:

The SUBroutine is #12 (93). GO absolute to -0.2 units in the z-axis (94). Note that the spindle is assumed "ON"; this is done in the initialization. The tool is assumed "clear" of the workpiece; see the end of this subroutine for the convention.

Go Relative (relative to the current tool location) +5 units in the y-direction (95). Note that the tool is assumed to be at the lower left corner of the "cell." This gives the left side of the "B." Go Relative 1.1 units in the x-direction (right) (96). This gives the top of the "B."

Create a local ZERO reference point at the XY coordinates where the tool is currently located (97). Create a local ZERO reference point AT a coordinate -1.2 units from the current reference point in the y-direction, but with the x-coordinate unchanged (98-99). This establishes the center point for the upper arc in the "B". The radius is from this point to the current location of the tool. Go Relative (relative to the last two points with the radius demarked by those two points) in a clockwise (-) direction for an angle of 180 degrees (100).

Go Relative and come back (c) 1.1 units to the left in the x-direction (101). Cut the bottom arc of the "B" (102-104). Go Relative 1.1 units to the left (-) in the x-direction (105). Move the tool clear of the work piece in the z-direction (106). GO fast (f) in the x-direction to positive absolute 1.3 units (107). This puts the tool at "absolute" x of 1.3 wrt the local ref x of 1.1 for a cell width of 2.4 units. This is the lower right corner of the cell, since the tool y-location was at the bottom of the lower arc when last heard from. RETURN to the next sequential instruction of the caller (108).

Notice the arbitrary "user unit" of measure. This is the base for establishing the correct relativities and proportions. As we'll see later this enables ***automated scaling*** and centering in the horizontal and vertical for varying size names and other symbols. This translates into simplified and automated set-up.

The convention of starting the the lower left of the character "cell" and ending at the lower right enables an aesthetically pleasing "proportional font" in which the letter "m" is wider than the letter "i."

One of the powerful features of the Dyna CNC instruction set is the use of ***polar coordinates***, as we see in the code for the letter "B" above. We're able to establish a reference point without tool movement from which to trace an arc with the 1/10000 inch resolution step motors in 3 dimensions. Alternatively we would have to do something like the segment fabrication we do in the CRT tool path simulation routine:

```
IF coodbuf1 = 'a' THEN BEGIN
  {to simulate the Go Relative polar (angle)
    coordinate, build the polygon around a point
    rotated clockwise or counterclockwise with
    5 degree resolution}
  anglegr:=xyz;
  anglesteps:=ABS(TRUNC(anglegr/theta));
  FOR q:=0 TO anglesteps DO BEGIN
    IF anglegr < 0 THEN BEGIN
      xpolar:=(xcur-xloc)*costheta+(ycur-yloc)*sintheta;
      xpolar:=xpolar+xloc;
      ypolar:=-(xcur-xloc)*sintheta+(ycur-yloc)*costheta;
      ypolar:=ypolar+yloc;
    END
```

Since the names and other symbols can be of varying lengths, and since we have a physical limitation of about 6 inches travel in the x-axis, we use the HP9000 for CAD scaling of the name.

We size the name by scanning character by character, calculating the width in basic "user units." Then we scale the horlzontal as the ratio of the x-axis travel limit to the calculated width. To maintain the original aspect ratio, we scale the y-axis with the same factor.

We are able in our original set-up calibration to position the spindle in the physical center of the work piece, so once we have the scaled x- and y-dimensions of the symbol, we can use half of the values to compute and execute the offset for the lower left corner. This way a simple physical calibration of the x-, y-, z-axis and spindle clearance can serve multiple runs on variable-length symbols.

It's beyond the scope of our "***CAD/CAM in microcosm***," but let's touch on some other issues you might want to address in more advanced problems.

You might want to input a material identifier that your CAD station could use to look up spindle rpm and tool feed-rate specs to download to the CNC device. You might want to programmatically do tool changes to handle more intricate cuts. You might want to control the injection of tool lubricant. You might want to coordinate the action of a loader/unloader robot. These things are within the capability of today's CAD/CAM technology.

. . . desktop computing . . .
. . . desktop drafting . . .
. . . now desktop machining for a major gain in productivity . . . .

In 1968 the HP9100 came to the engineer's workbench. A programmable calculator. With the HP9100, engineers no longer had to key in the algorithm each time they wanted to change a few variables. ***Desktop Computlng***.

This enabled engineers to get their sketch to the drafting department faster. But that was an ***interface***. A time-consuming interface. So personal CAD stations emerged to produce finished drawings faster. ***Desktop Drafting***.

This enabled engineers to get their finished drawing to the model shop faster. But that was an ***interface***. A time-consuming interface. So computers and CNC machines teamed up to give engineers CAD/CAM at their fingertips. ***Desktop Machining***. From "***art to part***" in minutes or hours instead of days or weeks.

. . . CAD/CAM engineering specs controlled with the data-words-graphics integration power of the HP3000 . . . .

For a professional grade documentation package that can leverage the investment in CAD/CAM, the scene can return to its starting point on the HP3000.

The 3000 has text and document preparation software such as TDP that can do ***typesetting*** at the quality level you see in this paper.

The ***graphics database*** on the HP9000 can be transformed and networked to the HP3000 where the vector form can be converted to raster for laser printer compatibility. With appropriate scaling and composition the documentation package can have a typeset "look" while retaining the ***flexibility*** of computer update.

Here's an example of graphics done with an HP9000 CAD system, ported to the HP3000 and integrated with text:

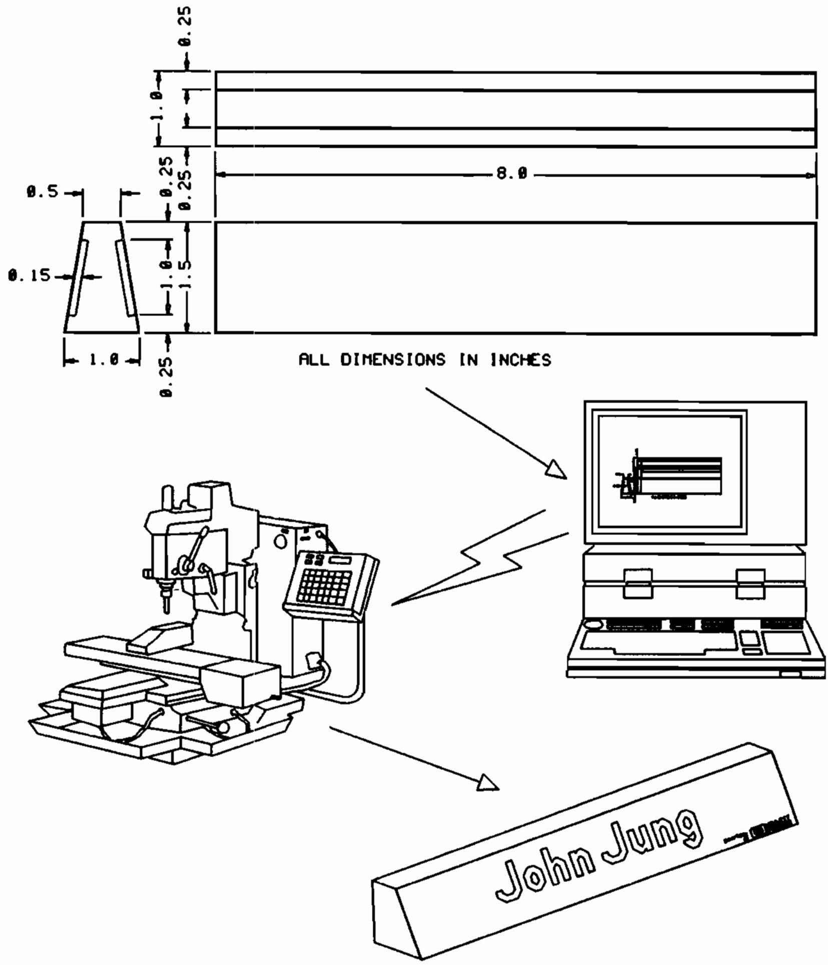


Figure 2. Our "***CAD/CAM in microcosm***" includes Engineering Data Control and documentation packaging thru a team effort of the HP3000 and HP9000 computers.

Epilogue . . . .

You're accustomed to the transformations and linkages for writing your data on disc, tape and paper. Here you've seen the transformations and linkages for writing your data on wood, metal and plastic. You've seen an ordinary ASCII string in an HP3000 IMAGE database networked to an HP9000 where it was transformed to tool path geometries executed by a CNC machine tool coupled with the HP9000. And you've seen the descriptive geometries of an HP9000 CAD system integrated with text on the HP3000.

This glimpse of CAD/CAM may help your understanding and appreciation of the other marvels we'll be seeing in our Migration to 2001.

About the Author . . . .

Sam Boles *is a Member Technical Staff in the Hewlett-Packard Information Software Operation in Cupertino, California. With HP since 1976, his computer experience started back in the AUTOCODER days of the 1401/1410, migrated thru the 360/370 era, and now focuses on next-generation operating system software. Sam received his MS at UCLA in Information Systems.*

sebiug27 2105 27jan86

Interex HP3000 Madrid Conference, March 1986
Theme: 'Migration to 2001'

... a picture's worth a thousand words ... and on a computer can take more processing and storage than 10,000 words ... in the Migration to 2001, if you find your graphics needs growing, here are some tips on quality and performance that may help you ....

A Blend of HP3000 and HP9000 for Computer Graphics

Sam Boles, Member Technical Staff

HEWLETT PACKARD

We all know that a picture's worth a thousand words.

And those of us who've drawn pictures with a computer know a picture can take more processing and storage than 10,000 words.

Then once you've gotten the basics under your belt, you get harder to please. You want resolution. You want performance. You want those end-points to meet, you want your curves smoother ... you might even want animation ... but, for sure, you don't want to wait.

This growing colony of computer artists with their growing appetite for the artistic can bring a multi-user commercial machine to its knees ... unless you resort to second-order Distributed Systems.

You've experienced first-order DS with your HP3000 talking to others like it, sharing programs and databases. There's a second-order distribution that networks into your HP3000 the high-performance high-resolution graphics capabilities of the HP9000 for data-words-graphics integration on your HP3000 with its laser printers and other powerful peripherals.

This description of graphics and technical publication techniques may give you some productivity ideas for your own installation.

... why graphics? ... for those who can't read ...

Once upon a time, there was a magnificent piece of ***americana*** called ***Life***. It came out once a week. And it cost only a dime. It was the week that was, in pictures. The glory of victory, the agony of defeat. The blood, sweat and tears that started wars and finished wars. The laughter, the sobs, the bad, beautiful, noble and ludicrous of the human condition -- photographed by some of the most courageous men and women in the history of journalism.

But not everyone viewed this piece of ***americana*** the same way. A young undergraduate (an English major) once remarked,

> *"Life is for those who can't read . . .*
>
> *Time, for those who can't think . . . ."*

Without commenting on its validity let's see if we can leverage this wisdom of a generation past, and come up with an answer to the question

Why Graphics?

Graphics is for those who can't read. No, it's not that they can't read because they can't read. They can't read because ***they don't have time to read***.

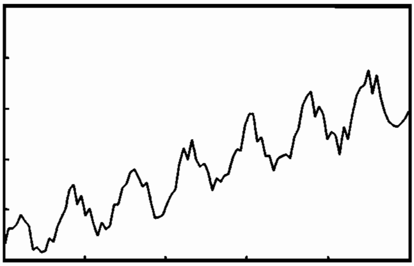
Let's look at an example out of one of the HP9000 reference manuals. Read these numbers:

```
0.1610 0.1625 0.1625 0.1628 0.1636
0.1631 0.1627 0.1608 0.1610 0.1606
0.1607 0.1617 0.1614 0.1626 0.1634
0.1640 0.1656 0.1660 0.1644 0.1651
0.1635 0.1641 0.1628 0.1619 0.1630
0.1624 0.1627 0.1644 0.1644 0.1657
0.1660 0.1670 0.1672 0.1666 0.1658
0.1662 0.1646 0.1633 0.1634 0.1636
0.1645 0.1652 0.1656 0.1677 0.1689
0.1680 0.1696 0.1680 0.1674 0.1677
0.1669 0.1655 0.1665 0.1662 0.1667
0.1668 0.1681 0.1688 0.1687 0.1707
0.1716 0.1716 0.1694 0.1698 0.1683
0.1683 0.1671 0.1681 0.1683 0.1684
0.1681 0.1698 0.1705 0.1723 0.1730
0.1734 0.1714 0.1722 0.1716 0.1696
0.1702 0.1699 0.1684 0.1706 0.1696
0.1715 0.1730 0.1737 0.1739 0.1751
0.1732 0.1747 0.1729 0.1717 0.1710
0.1707 0.1706 0.1709 0.1713 0.1720
```

Did you read them? No, of course you didn't. You ***don't have time*** to read . . . ***really*** read . . . a hundred numbers. And if you did have time you wouldn't waste it like that. (Notice how boring it gets after about the third digit?)

Besides why read a hundred numbers when today's technology can read them for you and maybe tell you something you might have missed -- because it's in ***between the lines***. Just browsing probably gives you the overall trend, but how about periodic motions and number of cycles?

Why Graphics?
The 100 Numbers You Didn't Read



Better, Easier, FASTER!

. . . and for those who can't think . . . again, because they don't have the time . . .

This maybe tells us something else, too: Graphics is ***for those who can't think***. No, it's not that they can't think because they can't think. They can't think because ***they don't have time to think***.

Sure, they could take a pencil and paper and calculate the deltas and get the pattern clusters in a few minutes. But that few minutes has an ***opportunity cost***. What they ***could've done***. Like get the corrective action launched. Or the next step of the design underway. Or whatever is the ***real work*** they could've been doing.

... to save time ... the Ultimate Unreplenishable ... at a rate of a thousand words per ...

It's the old story of "a picture's worth a thousand words." You get the message to your audience ***better*** ... ***faster***. ***You*** save time. ***They*** save time.

Let's look at this thing called ***time***. It's a unique commodity. Or maybe it's ***not a commodity*** -- ever try to buy some time? Anyway, it's ***unique***.

Remember back in the early 70's when we had to stand in line to get gas? Geologists for years had been telling us we were burning oil ***faster than it was being replenished***. Then we finally got the message: That meant we could run out. So the oil owners lowered production and raised the price. And suddenly there we were, waiting in line for stuff that not long before that a Gulf station in Los Angeles would sell for 18.9 cents a gallon during a gas war ... and clean your windshield while you bought it.

All of that for something ***relatively*** unreplenishable. But look at time. The ***Ultimate*** Unreplenishable.

If oil is ***slow*** at replenishing, it's nothing compared to Time. Time ***doesn't replenish at all***. You burn it and it's gone. Forever. And we burn it every second of every minute of every day. Not just when we drive to work. And we burn more of it faster all the time. Ask anyone who's been around awhile about how much faster you burn it as years go by. And how much ***more*** of it you burn.

The ***Ultimate Unreplenishable***.

Anything that improves performance in time utilization deserves attention. And graphics is one of those things.

So much for ***Why Graphics?*** Let's look at the evolution of the computer graphics artist.

... the evolution of the artist ... basics under your belt, you get harder to please ...

Remember a few years ago when ***DSG*** (Decision Support Graphics) brought charts right to our terminal on the HP3000? Bar charts, line charts, pie charts. We could eavesdrop a plotter on the line and get hard copy on the spot in minutes. Then there was ***HPDRAW*** to build the text and picture slides to go with the charts.

We could build our slide presentations quickly and conveniently. And update with the latest numbers in a matter of minutes.

The magic of ***Interface Reduction***.

We no longer had to queue up at the graphics department. And wait for the typesetter. We ***reduced*** these ***interfaces*** to zero. We saved time and money. A quantum leap in ***Productivity thru Interface Reduction***.

It was great.

Then as the elation wore off we noticed you could tell the computer slides from the typeset slides. It was the letters. Those stick letters. Like tapioca: ***good, but not exciting***.

We needed better letters. Nice smooth spline curves, with fill and boundary in different colors. And a wider range of fonts and sizes.

So DRAW II arrived with really world-class letters. A quantum leap in ***professional quality lettering***.

It was great.

Then as the elation wore off we noticed that next to the beautiful spline letters you really noticed when the end points in our drawings didn't quite meet, and some of the detail was a little ragged.

We needed ***better resolution***, and better and faster zoom, pan, grid snapping and . . .

On top of all that, something was happening in our work area.

Our colleagues were trying to figure out how we were able to get such good presentations ***so fast*** and ***still come in under budget***. They saw we were using DSG and DRAW, and they started to do the same thing.

CPU utilization began to grow. The last 3 days before quarterly review, ***response time*** got really ***slow***. We upgraded to a 68, and that helped. But not enough to keep up with the growing popularity of the tool.

. . . the issues of resolution, response time and load balancing . . .

As we experienced more and more positive results from our graphics, we wanted more and more quantity and quality.

More people were starting to use the tool. And the positive results from using the tool made them use it more.

Each iteration made the users more proficient with the tool, enabling them to do a better job of the next slide and increasing their appetite for perfection proportionately.

The ***need for fast-response high-resolution graphics tools evolved as the degradation of response time evolved***, aggravating the economic imbalance with a diminishing supply of CPU cycles, disc I/Os and main memory being confronted with an increasing demand for a higher service level by more users.

Compound the situation with a ***25 per cent annual growth rate*** and you get the scene that led us into second-order Distributed Systems for our computer graphics.

We already had our 3000's DS-linked so we could get at programs and databases on neighboring machines. And if we had a heavy-duty crunch that we needed to run we could off-load this to one of the light-load nodes.

But with the graphics overload we were looking for ***more than just CPU cycles***. We were looking for functionality. A richer command set. A more natural human interface. Higher resolution. Faster graphics performance in first-draws, redraws, transformation of primitives and cells.

We turned to our CAD-CAM family, the HP9000, and found what we were looking for.

Let's look at where the 9000 was coming from.

. . . the HP9000 genealogy . . . a "PC" before they invented the word . . .

If you trace the roots of today's HP9000 family, you go back to the days when we used words like "programmable calculator" and "desktop computer." If you look at the HP9825 (***circa*** mid-70's) you see the ***low-cost, small footprint, portable and individual work station*** that might've been called a "PC" if we'd used that kind of language in the medieval days of 10 years ago.

When we retired its jersey some time back, the HP9825 had been one of the top unit sellers in the history of Hewlett-Packard.

The tradition evolved into the HP9000/200 and /300 with 8-16 Megahertz processors with 3- and 4-plane color graphics, and the 3-CPU HP9000/500. The 300 and 500 support 8 planes of color (that's 256 colors from a palette of 16 million) with a graphics accelerator that pumps ***60,000 vectors a second over a 2 Megabyte bandwidth bus***.

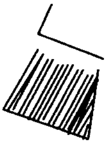
What all that bottom-lines to is ***high-performance high-resolution graphics*** with a wide range of price points.

Let's take a look at what even the low-cost range of this spread can do for your resolution, performance and load-balancing problems.

. . . a rich repertoire of commands, primitives and structurism . . .

Even in the low-cost HP9000/300 you find a feature set with a functional richness that puts you in a ***new graphics domain***.

With tools like ***EGS*** (Engineering Graphics System), you can ***zoom*** in on a particular detail and get the positioning you want, right down to the whiskers on the face (actually, a whisker's about 50 microns in diameter and the system has sub-micron resolution capabilities) . . .

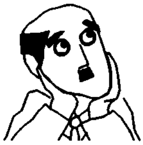


Using the ***cell instantiation***, component and level ***display selectivity*** and other functions so essential to CAD applications such as integrated circuit layout, you can build a basic cell one time . . .

. . . and ***scale*** and package it with a given instantiation in a given context . . .

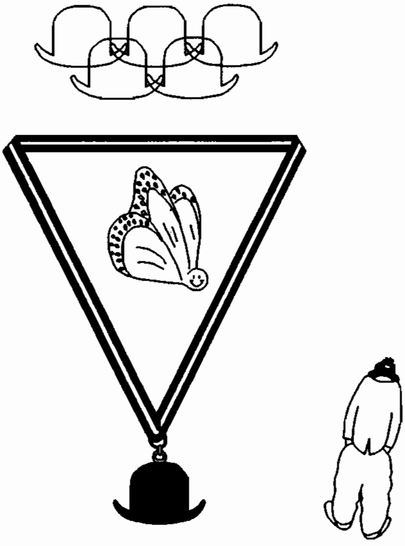
. . . include it in another instantiation with a different context . . .

. . . ***mirror*** the same cell for a different orientation in still another context . . .



. . . build up your ***hierarchical structure*** of cellular components with whatever ***scaling, translation, rotation, mirroring, zooming, panning*** are required, till you have the modules arranged in a multi-level composite that is your complete circuit . . . or whatever it is you're building . . .

. . . you can then take your integrated circuit layout or whatever it is you're building, "***plot to disc***" so you get an ASCII form of the HPGL commands that normally drive a plotter. You then use a ***terminal emulator*** (LAN's on its way, so your 9600 baud can move to the multi-megabit range) to get the vectors from the 9000 to the 3000. Here you've got the full power of TDP, lasers and other technology (such as EGS2FIG in the Contributed Library) to do your final packaging. Once established, your components can go into a library to provide a ***leverage base*** for future fan-out . . .



Epilogue . . .

The odyssey spans computer domains, operating systems, design disciplines. It gives you the rich functionality of the HP9000 CAD/CAM world, with it's high resolution, instantaneous response and natural human interface. It gives you the powerful data-words-graphics capabilities of the HP3000 and the laser printer. And it smoothes your processing load by spreading it out across the appropriate nodes to make your general user population a bit happier as their data processing engine is a little more responsive to their touch.

About the Author ....

Sam Boles *is a Member Technical Staff in the Hewlett-Packard Information Software Operation in Cupertino, California. With HP since 1976, his computer experience started back in the AUTOCODER days of the 1401/1410, migrated thru the 360/370 era, and now focuses on next-generation operating system software. Sam received his MS at UCLA in Information Systems.*

sebiug17 2035 27jan86

Title: In Search Of The Software Transistor

Authors: David Boskey and Tim Chase

Address: Corporate Computer Systems, Inc.
33 West Main Street
Holmdel, New Jersey 07733
U.S.A.

Telephone: (201)946-3800

Telex: 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

In this paper we will take a brief look at attempts to solve the problems which have been associated with software development. The reason we call it the search for "The Software Transistor" is that it was the development of the transistor which catapulted computer hardware into the advanced position which it holds today. In order for software to join those same lofty ranks, someone must develop the software equivalent of the transistor. We will, unfortunately, conclude that although some work is promising, there is still a long way left to go.

About Predicting

The theme of this conference is "Migration to 2001." With such a theme, it would appear appropriate to take a chance and make some predictions about what will be happening to software development at the turn of the century. This is a dangerous game, especially when the predictions are made about a year which will (hopefully) be reached by the authors. 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 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.

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 5 generations of 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, screen facilities which attempt 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 than programming languages in the classical sense, as we shall see.

By looking at this 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. This is amazing because 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 is to 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 <u>really</u> 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 no one 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 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 FORTRAN 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 ultimate customer is often lost by the result.

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 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. Regardless what the data sheets say, it is hard to write good programs which meet the customer's needs and anticipations. 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 2001 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.

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. Consider the language PROLOG. 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. However, 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.

In fact, we really have our doubts about the whole concept of nonprocedural programming. As Jean Sammet pointed out way back in 1969 in her book <u>Programming languages: History and Fundamentals</u>, 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 <u>how</u> 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 always 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 was 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. We predict that large expert systems will be plagued with the same HAL-like problems well beyond the year 2001.

And what of the Morlocks? They reap none of the benefits of the Eloi when it comes to programming ease. This means that even in 2001 someone will still have to bang the bits. Expert systems may become great at diagnosing blood 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.

Well, how about ADA?

If the expert system isn't the software transistor is there anything around 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 if we are to migrate to the year 2001.

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 features include the attempt to make a really portable language. Languages like C have been touted as being portable, but, in fact, most of the portability found is C is a result of the cleverness of the programmer. ADA's portability comes more from within. Portability will be extremely important in 2001. This is because systems built for the Eloi will be quite expensive 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 help out.

If language efforts like ADA are making important contributions to program portability, then they are also making changes in 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 2001.

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 finally starts selling 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.

ADA and other language systems, as opposed to compilers, will also aid in some of the organizational facing programmers. ADA compilers maintain application data bases which allow 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 peanut butter 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 hirsuit 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 address the design and support phases of software development. The cobbler's children often run bare foot. This is true for programmers. It seems as if they are often the last to benefit from computerization. Work must 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.

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....

Users of computers will have a field day by the year 2001. 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 around the corner. 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 that's the best we can hope for: a slow and steady sequence of steps toward Every Programmer's dream -- to put himself out of work.

NEVER CRY WOLF: THE CHALLENGE OF THE REMOTE TROUBLESHOOTER OR "THE CALL OF THE WILD"

Larry Abramovitz
Karen M. Devitt
Martin Marietta Data Systems, Greenbelt, Maryland USA

Summary

In "Never Cry Wolf: The Challenge of the Remote Troubleshooter", we describe the thrill of hardware troubleshooting in an international computer network. Following a detailed description of the network itself, we discuss our approach - past, present and future. How we help our users today will most certainly influence our future effectiveness, and the effectiveness of the network. Our solutions to problems - current and anticipated - can be extrapolated to the industry in general.

I. Preface

I was the newcomer to the company. I arrived with great expectations of doing "a little bit of everything": some troubleshooting, some equipment installation and configuration, and user training. Bright eyed and eager to learn, I was told that the network would eventually span the globe and consist of approximately 60 HP3000's, 3500 CRT's, 1500 character printers, loads of multiplexers and modems, plus miles of cable - all installed and maintained by Martin Marietta Data Systems (MMDS). The network was to be used for data capture, database maintenance and update, and, of course, remote data transmission. For now, there were 11 CPU's, with 2 more coming fast. The office was buzzing with activity. My first assignment was in the Maintenance Service Center (MSC). In the MSC, we serve as the intermediary between system operators or users and the vendors (the network uses HP and 2 additional vendors). We screen the trouble calls and try to resolve them before involving the vendors' field engineers. Around the clock, we attempt to catch and correct user errors, mysteriously changed configurations, disconnected keyboards, etc.

I could tell that this challenge would be different from any I had had before. How, with 2 telephones and a bookshelf of technical manuals (mercifully up to date), could we provide service to an international network of HP 3000 users? It seemed a formidable, if not impossible, task.

This paper presents our solutions and ongoing creativity in meeting the technical, analytical, and interpersonal challenges presented by remote troubleshooting; we describe our approach today and our future plans for this network which requires support well into the 21st Century.

II. Overview of the Network

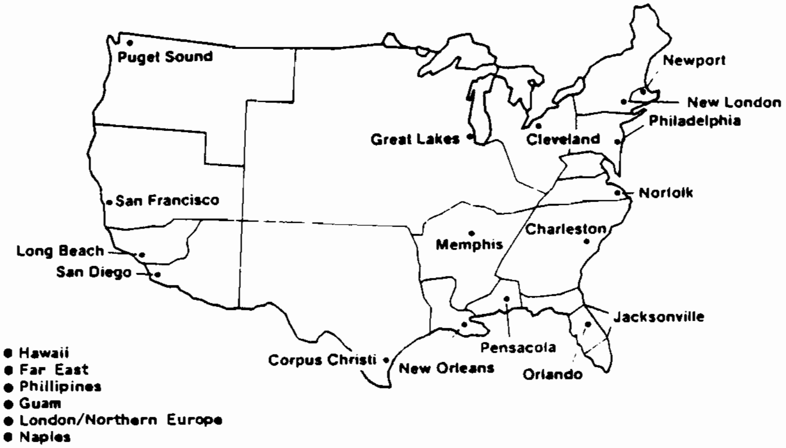
A. Introduction

MMDS supports a worldwide field reporting network for pay and personnel data. This network was conceived in 1976 when our client decided to modernize their employee pay and personnel system. Lengthy turnaround times and increasing errors in their manual system had become unacceptable. The modernization was to occur in 2 phases - functional consolidation and automated support.

In the first phase, approximately 3,500 separate pay, personnel and transportation offices were consolidated into 350 field sites. These field sites were organized into 25 geographic regions. The field site offices maintain employee records and provide all pay, personnel, transportation, and local information services to individual employees. The field sites also prepare transactions to notify headquarters of pay and personnel changes, audit transactions, and receive confirmation from headquarters. Processing Centers (PC's) are the physical locations of CPU's, where processing for a personnel region takes place. Most processing centers use more than 1 system (CPU and associated peripherals). All systems have the ability to communicate in a DS Distributed Systems Network environment.

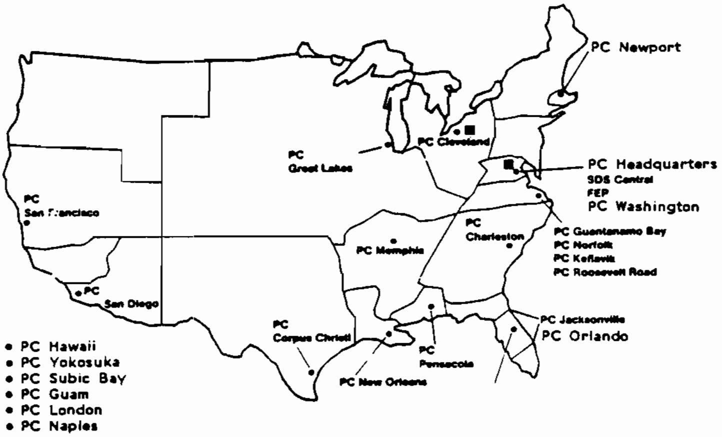
The map below shows the personnel regions in the network (regions outside the continental US are indicated on the lower left).

PERSONNEL REGIONS:



The Processing Centers that serve the regions are located in the following cities:

PROCESSING CENTERS

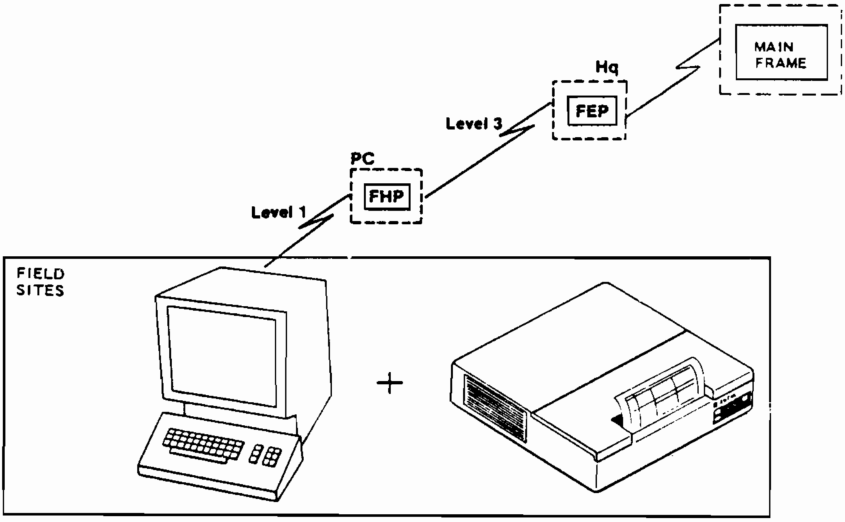


The second phase of modernization began in 1984. Applications software was developed to automate data collection and transmission procedures, hardware installations began, key personnel were trained, and the Maintenance Service Center started up.

B. The Network

What, Why and Where: This world-wide, online network integrates the automated field reporting and management information system to support the newly consolidated personnel offices. The network aims to improve pay, personnel, and transportation support services to the 400,000 employees. The benefits of setting up such an automated system to streamline these functions in a large organization cannot be overstated - accurate and timely collection of personnel and pay information, improved service to employees, automated input procedures, accountability, information support, and 2-way telecommunications. The network's 400 sites will ultimately reside in 50 states and 18 foreign countries.

The people at the field sites collect data on terminals connected to a host CPU (HP 3000) at the processing center; concurrently, supporting reports are printed out. The data then travels to one of two front end processors which exchange information (HP 3000/68's) and then on to the headquarters mainframes where the two master data bases reside.



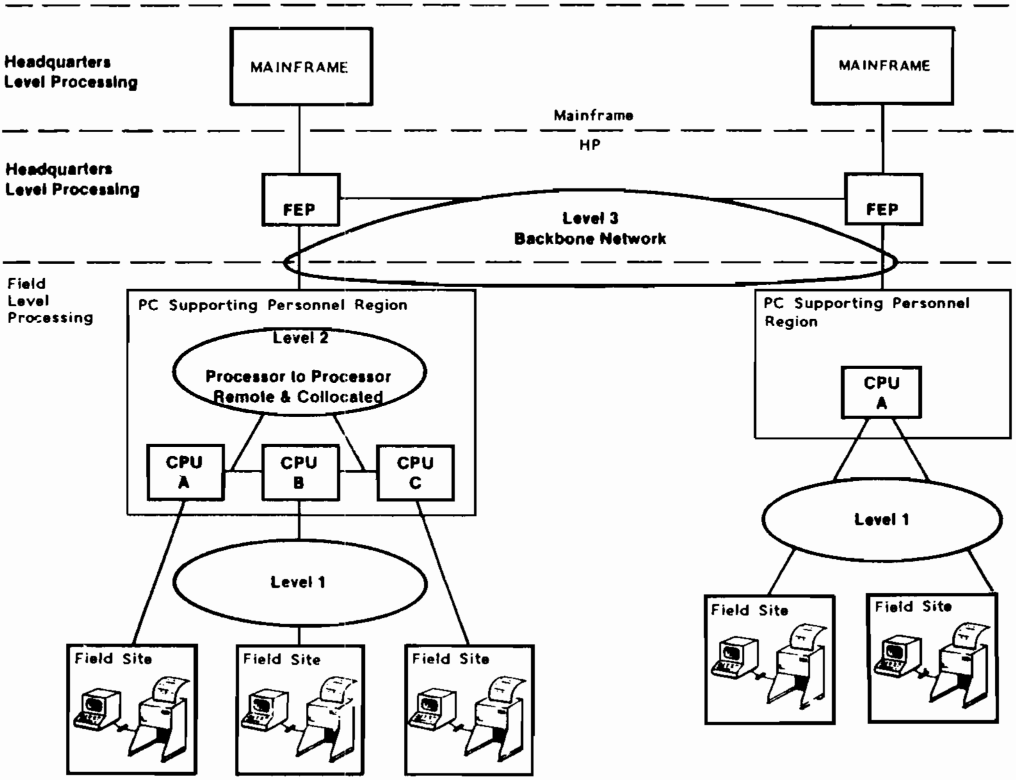
Each field site (with the exception of sub-sites) has a site manager, a system supervisor, and the appropriate complement of terminal end users. Each processing center "centralizes" hardware trouble calls for all its field sites. The significance of this will be revealed shortly.

Each processing center has system operators and a system manager in direct support of the equipment installed. Furthermore, each processing center has a technical control center where the local multiplexers, modems and telephone line interfaces are located. The integration of CPU power and network communication interfacing at a single site is of prime importance to MSC staff in diagnosing a problem at the processing center or remote field site. Thus, the major players in support of the network are 1) the system manager and technical control center staff at the processing center; 2) the site manager at each field site, and, of course, 3) the MSC staff. Each player in this triad is a central source of information during any troubleshooting episode.

HOW:

Hardware. To better understand the hardware configuration of this network, it might help to see the levels of communication in a diagram. Remote users communicate to processing centers. Processing centers communicate to front end processors which in turn communicate to headquarters' mainframes. We are concerned with the processing center and the field site level almost exclusively.

COMMUNICATION LEVELS



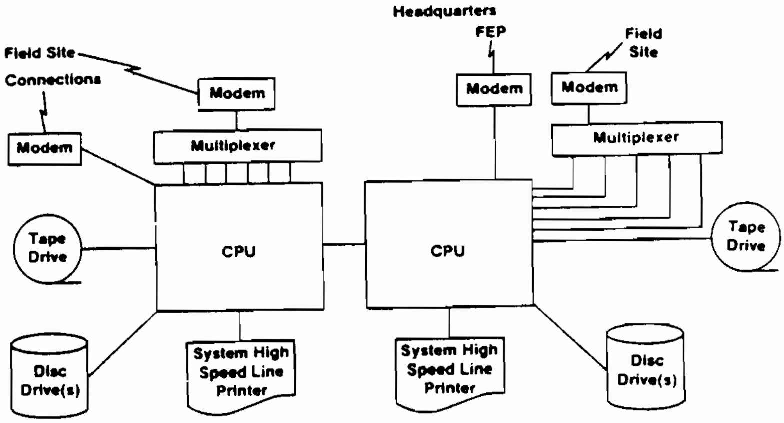
Processing centers are the operational data processing facilities in the network. Processing center equipment typically includes an HP 3000, tape and disc drives, high speed line printers, operator consoles with associated printers, and communications equipment. Processing centers often contain more than one CPU and associated peripherals.

Processing Center Equipment:

HP	3000/68	CPU (8 Meg)
	/48	(4 Meg)
	/42	(2 Meg)
HP	7933H	Disc (404 Meg; Fixed)
	7935H	(404 Meg; Removable)
HP	7976A	Mag Tape (1600/6250 bpi; auto-load)
	7978A	(1600/6250 bpi)
HP	2647F; HP150	System Console (/68)
	2392A	(/48, /42)
HP	2671A	Console Printer(/68)
HP	2608S; 2563A	Line Printer(400,300 lpm)
RM	OMNIMODE 96	Modem
RM	OMNIMODE 48	Modem
BE	Bell	Dialup/Autodial Modem
RM	OMNIMUX 320	Multiplexer
AR	Atlantic Research	Patch Panels
		Datascope

The processing environment for a typical processing center is shown below.

PROCESSING ENVIRONMENT FOR TYPICAL PC

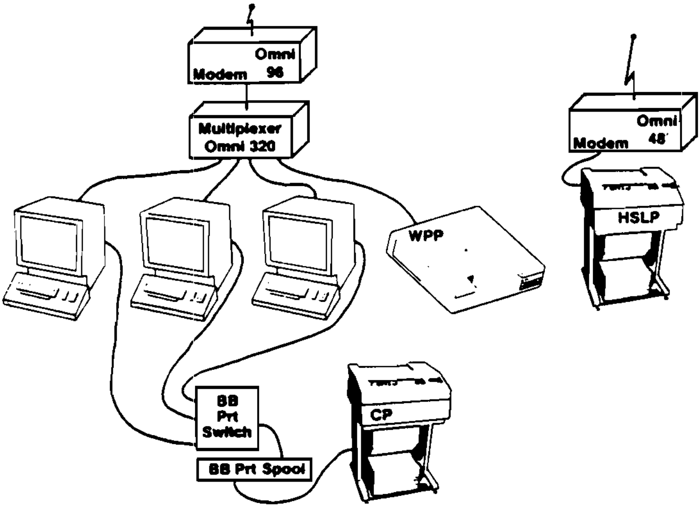


The processing environment for all field sites is essentially the same. There are significant differences, however, in the volume of work at various sites. Field sites have terminals, character printers, and high speed line printers and, of course, the requisite data communications equipment. Most field sites are supported by modems and multiplexers in a point-to-point configuration. Sub-sites are supported by MTS software and 4800 or 9600 baud modems in a multipoint/multidrop configuration.

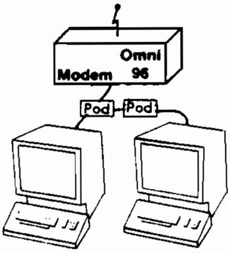
FIELD SITE EQUIPMENT:

HP	2624B	CRT
BB	PSI-4A	Print Switch
BB	PIA-60	Print Spooler
HP	2934A	CP (Companion Printer)
HP	2934A	WPP (Word Processing Printer)
HP	2563A	HSLP (Line Printer)
RM	OMNIMODE 96	Modem
RM	OMNIMODE 48	Modem
RM	OMNIMUX 320	Multiplexer
HP	2333A	Cluster Controller

TYPICAL FIELD SITE LAYOUT - POINT TO POINT:



TYPICAL FIELD SITE LAYOUT - MULTIPOINT



Databases. The network we are describing is essentially a data capture system for payroll and personnel data. After collection, the data is transmitted to the headquarters level where one of two front end processors (HP 3000/68) submits the data to the mainframe and also passes the data to the other front end processor, as you've already seen on the "Levels of Communication" diagram. The two data bases are geographically distant.

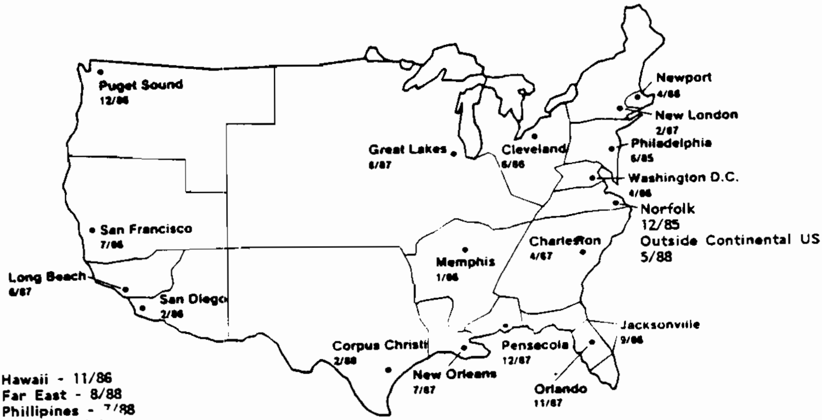
Communications Software. The systems in the network "talk" to each other via DS/DSN and a communications software system which uses a store/forward methodology to accommodate the different time zones. Data is transmitted from the processing centers to the front end processors using this customized networking software.

Systems Software. All HP 3000's in the network use HP FOS (MPE, FCOPY, EDIT/3000, Sort/Merge, IMAGE, QUERY and VPLUS). Other tools include TDP, MTS, DS/DSN, and special utilities designed to provide user logging and monitoring. Other than development personnel at headquarters, there are no programmers in the field. Compilers are not standard on the processing center CPU's.

When: The system was conceived in 1976. Equipment installation at processing centers and field sites began in July 1984. The Maintenance Service Center began operations in 1984 and will continue for 10 years. Prior to the first installation, key personnel attended training courses. Additional training is conducted at each field site following installation.

The current system implementation schedule is shown below.

IMPLEMENTATION SCHEDULE



Hawaii - 11/86
Far East - 8/88
Phillipines - 7/88
Guam - 10/88
London/Northern Europe - 12/88
Naples - 11/88

III. MMDS' Role in the Network

We perform 3 main functionc for the client's network: 1) hardware and systems software installation and activation (bringing a new system up); 2) user training; and 3) systems software/hardware maintenance. We coordinate and provide these services from our main office in Greenbelt, Maryland. Installation, while coordinated in Greenbelt, occurs (naturally) at the processing centers and field sites. We provide 3 phases of training: 1) for headquarters' staff personnel (given at MMDS' offices), 2) on-site training at each newly-installed processing center, and 3) follow-on training at MMDS in years 4-10 of the project.

The staff at MMDS rotates in and out of the varied tasks - a trainer one week, a cable layer the next, a troubleshooter the next. Installation typically is divided into 2 phases of 3 weeks duration each. Phase 1 is the DCE installation; phase 2 is the DTE. Training occurs 3 to 5 weeks after the installation and lasts for 2 weeks. Troubleshooting in the Maintenance Service Center continues 24 hours a day, 7 days a week.

Terminals, data communications equipment, CPU's and peripherals are installed by teams of MMDS employees. Each team consists of 3 people. One of the three is a designated team leader responsible for signing off on a site after verifying that all equipment is installed, working, and that all serial numbers have been recorded. After installation, training is conducted at that site for key personnel. We follow this routine for each installation. As of February 1986, we have installed 11 systems, representing 5 personnel regions with a total of 33 field sites.

During installation and activation of a site, maintenance is immediately available. We have noticed a number of distinct maintenance stages emerging in the process: 1) an install/burn-in time of approximately 1 month, followed by 2) startup/coming live when the users really begin processing, followed by 3) the ongoing, day-to-day use of the system, and 4) occasions when hardware and systems software are upgraded.

The focus of this paper is, of course, the maintenance function.

IV. THE MAINTENANCE FUNCTION

After rounds of negotiations, a centralized Maintenance Service Center (MSC) was selected as the approach. This MSC would be staffed by MMDS people who could remotely diagnose problems and call in vendors as needed. The Center provides 24-hour a day hot-line support to the network. Calls are placed by a designated person for each personnel region. Thus, the MSC serves as the central repository of all trouble calls reported by on-site users, and coordinated through the client's technical control center. This was preferred over having an alarming number of troubled users calling either MMDS or the vendors.

A. The Maintenance Service Center

Introduction. The Maintenance Service Center (MSC) was designed to provide maintenance in a centralized, controlled way. Lines of communication between the client and our staff are set up in the following manner: the user reports any problems encountered to the region's technical control center, located at that region's processing center. The person receiving the call contacts our troubleshooter on the MMDS hot line. We, in turn, call the "troubled" user and try to determine the exact nature of the problem. When necessary, we dispatch the vendor to the site. Upon successful resolution of the problem, all parties involved are notified as to the time and nature of the resolution.

Initially, for a period of 4 months, we provided service through the MSC 5 days a week, from 8 am to 5 pm; then, for 6 months, from 7 am to 11 pm. In November of 1985, we expanded service to 24-hours, 7 days a week and began an earnest search for more troubleshooters.

Staffing, Schedules, and Shifts. The expansion to 24 hours was dictated by an increase in equipment and contractual requirements. To accommodate the staffing needs of the project and the specific needs of the MSC, a schedule of 12-hour shifts was devised. This allows MSC staff a consistent schedule for 8-week periods while also permitting people to rotate in and out of various other project functions. One advantage to this kind of schedule is predictability: for 8 weeks at a time, we work the same shift. In addition, 3-4 days off every week (inherent in this approach) is compensation for the disruption to a "normal" work week.

The person on graveyard shift (8 pm to 8 am) has one disadvantage in that he/she works alone. Phone traffic is currently very low during this shift and boredom could become a factor. However, with the installation of a major site in San Diego and other planned installations, this is expected to change due to time zones and extended work hours.

Our scheduling (2 12-hour shifts per day) grew out of necessity more than anything else but has proven nonetheless to be quite workable. The people involved think so, and they seem to enjoy the usable chunks of time off every week.

One very important challenge we face is the problem of isolation. For 2 months at a time, 2 staff members work virtually alone - somewhat analogous to the bored shepherd boy on the hilltop who cried wolf. For some people, this is desirable; for others, it can be quite tedious. In any case, it requires self-motivation and determination.

How do we attract and keep good people? Routine rotation to other areas of the project provides a needed change of pace, and the added value of gaining professional experience. Our troubleshooters have installed, configured, and used all the equipment we maintain. Many of them have met and worked with the users at the field sites during installation, and some have instructed the "troubled" user in an on-site training phase. These factors contribute to a high level of technical expertise and staff morale. Our troubleshooters are kept aware of activities of the local HP users' group, data communications conferences, vendor tutorials, and a variety of professional activities. On a regular basis, staff members attend HP training courses in different subjects (System Manager, Programmer's Introduction, Query, Data Communications, etc.).

The people who staff our MSC represent a variety of disciplines - hardware specialists, software people, data communications people, and people with training experience. This mix of disciplines has proven very useful, and the resulting exchange of information and experience has broadened everyone's awareness.

How Effective Are We? To date, our success rate has been very good. A system of penalties is in effect (can you think of any better way to say that?) and our client rates us in a number of different categories including component down time, system down time, site down time, and excess response time. We are also motivated by some immediate requirements. For example, our vendors are given the same amount of time to have the engineer on site as we are (usually 4 hours; at some major sites, 2 hours). However, our clock begins to run after we receive the initial trouble call, while theirs begins after we have contacted the site, done our own troubleshooting, and then called them. Therefore, our own troubleshooting must be completed in a minimal amount of time. We are thus compelled to think and act quickly - and with good judgement.

Depending on our past experience with a particular caller and our own assessment of a situation, we will occasionally place a call to the vendor immediately, then contact the user. There are advantages and disadvantages to this. If we cry wolf too many times, eventually no one will listen. But taking a half-hour or so to diagnose a problem first could indeed save the vendor an unnecessary trip to the site. But then again, this could cause a "late" response time. This is the crux of our dilemma - and the core of the challenge we face. In spite of our good intentions and expertise, we sometimes place a wrong call to a vendor. More than once, we have mis-diagnosed a modem or multiplexer when the communications line was faulty. We have learned since then to harbor a healthy suspicion of phone lines "reputed" to be good.

Our effectiveness depends to a large extent on that delicate mix of technical, analytical, and interpersonal skills.

The_Resources. The Maintenance Service Center was developed, literally, from scratch. Starting with nothing has certain advantages. The usefulness of resources can be tested and determined empirically. In the course of our experience, we have established a very reliable set of tools for use during a troubleshooting episode.

The mainstay of our toolset is the Troubleshooting Guide, developed by the staff of the MSC. It provides a step-by-step diagnostic procedure for each piece of equipment, with cross-references to other connected hardware and manuals where appropriate. This guide leads the user through a series of questions and specific tests for verifying problems. The guide was set up as a complete "How To" manual. Due to the longevity of the project, we felt it was necessary to address the issue that new employees would be joining the MSC. Also, periods of time would elapse when an employee doesn't work the hot-line but is expected to be up to par following a prolonged absence. This guide assumes only a minimal familiarity with our equipment and configurations. It makes use of the Socratic method to determine the exact nature of the problem. In many cases, the dialogue between user and troubleshooter elucidates and even resolves the problem. Tests are done during this dialogue, such as hard and soft resets, data communication loopback tests, swapping out suspect equipment, etc. This method has resolved many trouble calls since the MSC began.

One important factor in keeping vendors in line with our objectives is to minimize false alarms. This ensures that when a vendor receives a call from MMDS, he can be reasonably assured that a failure exists and requires repair. Furthermore, we often can give them a clue as to the exact nature of the problem. They therefore arrive at the site better prepared. Never crying wolf further increases the probability that our needs are met when contention for a vendor's resources occurs. Naturally, a higher level of priority exists when we sound the alarm to our maintenance vendors than would exist if we habitually dispatched without due cause.

Equipment lists are another essential component of our library. It has proven absolutely necessary to keep accurate records of equipment by location and serial number. Since all of our vendors perform maintenance by serial number, these lists must be kept up to date at all times. Periods of warranty, renewed maintenance contract dates, terms of maintenance, points of contact, phone numbers - all this information must be accurate and accessible to our inveterate troubleshooters.

The maintenance function, as well as the installation function, makes use of the MMDS Installation Manual. This manual provides a complete snapshot of an installation. It contains a chapter on each piece of equipment, including procedures for installing, configuration parameters where applicable, cabling diagrams, user points of contact, site addresses and phone numbers.

Keeping accurate records to the extent necessary has proven to be a formidable administrative chore. To alleviate the tediousness, and to increase our overall effectiveness in troubleshooting, we ordered an in-house system. This system is comprised of an HP 3000/42, terminals (HP 2392A), a printer (HP 2934), and a modem (VA 212). We plan to use this system to tap into our users' systems when diagnosing certain problems. Our in-house system will also be used to provide the staff with hands-on experience. Eventually, we plan to do all our record keeping and forms processing on this system using IMAGE, VPLUS, TDP, etc.

Prior to and during an installation, data terminal and data communications equipment diagrams are created. For each installation we create a full set of diagrams by processing center and field site. Serial numbers, model numbers, and connectivity are illustrated.

In addition to the reference tools listed above, all of which are created and updated by MMDS, the MSC maintains a library of technical reference manuals provided by our vendors. Complete manuals for each piece of equipment are kept, as well as manuals of general interest (HP software products, AdvanceNet, Communications Handbook, etc.). Our troubleshooters are encouraged to delve into topics of interest on a particular piece of equipment or software product.

In order to provide around-the-clock coverage, we had a toll free number installed. Users can call any time of the day or night. If our troubleshooter has stepped out for a minute (calls of nature and the like), an answering machine records the caller's message. We also carry a beeper whenever we are to be out of the office for more than a few minutes.

Crucial to our dealing with specific trouble calls, a "Malfunction Report" was developed. This report provides a framework for the collection of pertinent information and the appropriate and timely response to problems. When a call comes in, a case number is assigned to it. The first 6 digits of the case number identify the site; the remaining digits refer to the year, month, and a sequential number that resets to 001 at the beginning of each month. This form becomes our official report for a specific call. At each crucial step in the process of resolution, detailed notes are made (e.g., called HP; HP arrived on site; HP cleaned end of tape sensor, etc.).

Frequently a trouble call remains "open" from one shift to the next. This occurs most often if a problem develops late in the day, and vendor maintenance coverage doesn't begin until 8 the next morning. To ensure that all open calls are followed up during the next shift, we devised a turnover log. The simple ritual of signing the turnover log guarantees that nothing falls through the cracks.

At monthly intervals, all calls are summarized and sorted by type of equipment and site. Interpreting the monthly summaries can be useful to track recurring problems or to spot emerging trends. Summary sheets also maintain crucial information as to down time and response time over the course of the month. This information has proven useful in certain geographic areas, for example, where vendors are underrepresented or where parts are in low supply.

Who Are We Helping? A profile of our users is helpful in understanding our approach to troubleshooting. Our client has a policy of hiring from within. As a result, many of our users, while proficient in the functional aspect of the network, are novices in the area of HP computer systems and often computers in general. They understand what the system is to be used for, but are learning how the pieces of the system work. We developed training courses to train our users in a variety of disciplines. Training encompasses a general orientation to the project, down to the level of detailed networking concepts and procedural software products (VPLUS, TDP, COBOL, etc.). In short, training was devised to be specific to the customers' uses of the product in this network. This kept the training courses of particular interest to the user and minimized questions of how this "fit" a user's specific needs. Better students make better end users who are more qualified to assist us when a problem arises. As you can see, doing a good job in one area can also help in another.

Our users are organized into the following categories: site managers, responsible for system operations; supervisors, responsible for supervision of operations; operators, responsible for data entry/retrieval and printing of hardcopy; and associate data base administrators, points of contact responsible for administrative support of field sites.

B. The Vendors

The vendors are, in effect, sub-contractors to MMDS. We maintain contracts with them to provide maintenance, and it is vital to our success to promote and maintain good relations with them. In the MSC, we not only deal regularly with users but also with vendors. We are the ones who send them to the far reaches of the globe; we are the ones who describe the problem at hand; we are the ones who are penalized if they don't meet the agreed-upon response time; we are the ones responsible for coordinating and following up calls; we are the ones who pass the on-site contact information on to the vendor. It is crucial to our effectiveness to maintain a good rapport -- to never cry wolf and only cry out when they fail to meet the customers' needs.

A specific example of our need for good rapport with the vendor occurred when a part flown in to repair a CPU arrived at the airport of a major U.S. city. The customer engineer drove to the airport late that night to pick it up, only to discover that the airport was closed. This could have been a very sticky situation; luckily for us it wasn't.

In addition to maintaining good professional rapport, we have found it necessary to keep our own accurate records of serial numbers, response times, maintenance terms, warranty periods, etc. The better our substantiating documentation is, the more weight our word carries with the local dispatcher. It has happened more than once that a piece of equipment - newly installed - required service before the vendor had entered the data in the warranty data base. With a response time of 2 hours, our facts must be correct. And in some cases, our facts are the only ones in town.

It has been to our advantage to get to know the dispatchers, supervisors, and customer engineers. In the same way that getting to know our users has proved helpful, knowing our vendors has smoothed the way to better service more than once.

C. How Are We Doing Now?

In the cold light of statistical evidence, we seem to be doing superbly well. Over the last 6 months, 250 trouble calls have made their way to the MSC. All of these have been successfully resolved either by MMDS on the phone or by vendor on-site support. System down-time has been kept to a minimum and many times exceeds the requirements of the customer. Vendor response to the customer has met customer expectations and is directly related to our ability to resolve a problem quickly. Even with new, large installations being performed every 2-3 months, we have maintained our success rate. As our troubleshooters gain experience, we are hiring and training new ones, and our MSC is functioning smoothly around the clock.

D. Where Are We Going?

As mentioned earlier, we are installing a system in-house. This will give us the resources to further automate our internal record-keeping processes. It will also provide our troubleshooters with continuous hands-on experience.

Beginning in 1987, we will commence overseas installations. Maintenance for overseas sites creates a new opportunity for us. In areas of the world where no HP customer engineers reside, we will train and relocate our own technicians.

Eventually, the MSC prime and graveyard shifts will be staffed by 2 troubleshooters. This will require further hiring and training of new people.

The physical environment of the MSC gained importance when the increase to 24-hour coverage took place. With people inhabiting one physical space around the clock, unique requirements surfaced. Security, kitchen facilities, and a "comfortable" environment are currently being addressed.

We previously touched on the unique scheduling requirements of running a 24-hour service. There is a human element we continually face in the process of shifting people from day to night, from the beginning to the end of the week, the loss of holidays, and the general disruption to personal lives. All of these things require a flexibility by staff and managers so that all needs are met - the needs of the people and the needs of the network.

The future will inevitably bring vendor hardware and software upgrades. This is a fact of the industry. Our resources (manuals, equipment lists, installation procedures, etc.) must be as easily upgraded as the hardware and software they represent. Areas of potential upgrade include Spectrum, new multiplexer models, new fixed disc drives, etc. And as technology moves hardware networking into the realms of the 21st Century, we expect to continue upgrading our basic procedures and techniques, so that we are well prepared to handle the new challenges that this technology will bring to the end-user and ultimately to us.

V. In Conclusion

Our experience troubleshooting this network has led us to make some brazen generalizations. We have distilled into a few pertinent commandments, if you will, what we think works best.

1. As in all things, a healthy dose of common sense works wonders.

2. Analytical ability is at least as important as technical expertise and ranks as high as attitude in performing well.

3. Use everything that happens as a learning experience. This attitude has proven invaluable. Learning can take many different forms.

4. Maintain an even temper. Grace under pressure cannot be overrated.

5. Don't be afraid to get other people involved. Ask questions. Brainstorm. Use HP's Response Center.

6. Accept the fact that you will make mistakes. Don't take things too personally. (See #3 above.)

7. Try to understand the big picture. Why does this network exist? Who uses it? What does it do?

8. Keep good notes. This is extremely important during a trouble call. Times, names, significant events - keeping records of these helps clear up questions later on if there is a problem or discrepancy. Also, there can be historical significance to specific trouble calls that may only surface much later, when memories fade.

9. Practice your Socratic method on the user. (What did you do? What do you think you did? What happened next? What does the screen _say_? Practice asking questions in a non-judgemental way, without insulting anyone's intelligence.

10. Remain flexible in your thinking. The ability to change direction can be very useful. As in software debugging, you have to know when to relinquish a certain avenue of investigation in favor of a new one. There is always an element of gambling involved.

11. Trust your instincts. Sometimes split second decisions must be made. The best private detectives trust their instincts. There is, however, no substitute for _clear_ thinking.)

12. Finally, Never Cry Wolf!

Larry Abramovitz is a Senior Technical Services Representative for Martin Marietta Data Systems. He has a B.S.in Decision Science and in Management from the Wharton School of the University of Pennsylvania. His experience in data processing includes instructing technical and managerial ADP users, troubleshooting, and installation.

Karen M. Devitt is a Principal Computer Systems Designer for Martin Marietta Data Systems. She has a B.A. in English Literature from the College of William and Mary. Her experience in data processing covers a wide range of activities: marketing support, technical writing, applications programming, system analysis and design, and most recently, remote troubleshooting.

DISTRIBUTED DATA ACQUISITION AND CONTROL FOR NUCLEAR PHYSICS EXPERIMENTS

Björn Dreher
Institut für Kernphysik, Universität Mainz, Mainz, F. R. Germany

Summary

A distributed system of one HP3000/68, three HP1000 systems, one HP9000-500 and one Perkin-Elmer PE3220 system is being used at the Institute for Nuclear Physics at the University of Mainz in West-Germany for the data acquisition during nuclear physics experiments at the 180 MeV Microtron. The microtron itself is controlled by two of the HP1000 systems. The backbone of the entire system is the distributed systems software, which has been developed in-house. It is using a packet switched message system to exchange data between processes on the various computers and can be used for local processes as well. The system is in operation since 1981. The HP3000/68 is the central node of the star topology system. Details about the data acquisition system with on-line analysis of the data on the HP3000/68 will be given as well as an general discussion of the control system of the microtron. Future developments of the communications software will include the usage of IEEE 802.3 LANs as hardware vehicle for the message transfer as well as the inclusion of other vendors MC68000 based VMEbus systems in the network.

Introduction

At the Institute for Nuclear Physics of the Johannes Gutenberg University at Mainz, West Germany, basic nuclear physics research is done in the field of medium energy physics with electromagnetic interaction using two different electron accelerators. The older linear accelerator produces a pulsed electron beam with a maximum energy of up to 400 MeV (million electron volts). The new racetrack microtron MAMI (MAinzer MIkrotron) yields in its current second stage a maximum energy of 180 MeV of c.w. electron current. The third stage, which is currently under construction, will deliver a continous electron beam of up to 800 MeV. Already for use during the experiments using the old linear accelerator a hierarchical distributed computer system was developed to be used for data acquisition and control [1]. In addition, the calculations needed during the preparation of the experiments and the time-consuming analysis of the measured data should be done with this system. Fig. 1 shows the current configuration. The network consists of three HP1000s, one HP9000 series 550, one Perkin-Elmer 3220, and one HP3000/68.

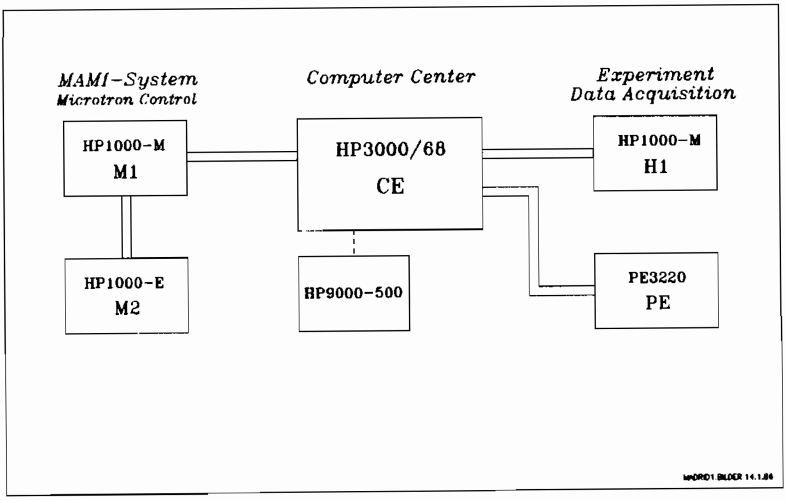


Fig. 1: Distributed Computer System

Overview

The HP3000/68 under MPE-V/E serves two major purposes. First, it is used as the main system in the institute's computer center for technical and scientific calculations. Second, it is the central node of the distributed computer system, which has essentially a star topology. Because the HP3000 is still a 16 bit computer system, the HP9000 system has been added to provide true 32 bit capabilities under the HP-UX operating system.

The next level in the hierarchy consists of somewhat smaller minicomputer systems with true real-time capabilities. They serve for data acquisition and for various process control purposes. The PE and H1 systems are used for fast data acquisition during nuclear physics experiments (more than 10 kbytes/sec). The loosely coupled M1 and M2 computers are being used for the control of the MAMI microtron. The HP1000 systems run under RTE-IVB, the PE3220 under Unix.

The HP3000/68 system

The central HP3000/68 has 72 ATP ports for terminals and other RS232 peripherals, e.g. letter quality and graphics printer and plotter. Two HP7925 and one HP7933 disk drives serve as on-line mass storage, four magnetic tape drives (two HP7970, two HP7978) are used two archive the experimental data. The HP3000 system is used during the preparation of the experiments which are being performed at the institute, for the analysis of the experimental data, either on-line during the experiment or off-line after the experiment has been finished, and for various technical and scientific calculations, which stretch from complicated solutions of problems from theoretical physics to the calculation of the spatial distribution of the magnetic field strength of the main magnets of the microtron and its graphical representation in the form of contour lines. Secondly, the resources of the central computer are being used by the front-end minicomputers via the network. This can be part of the central computer's processing power, e.g. during the data acquisition and on-line analysis of the measured data, or the transparent access of peripherals, e.g. line printer, magtapes, file system, etc., by the smaller computers.

The Interprocess Communications System

Backbone of the entire system is the communications software, which has been developed in-house. When the first implementation was begun in 1977, Hewlett-Packard could not deliver a system with all the capabilities required for our purposes. Today, only the new NS/3000 software has similar features as our own software. In the future, we plan to adopt at least the low level protocol layers (according to the ISO OSI standard) so that we can use standard hardware components as defined in the IEEE 802.3 standard.

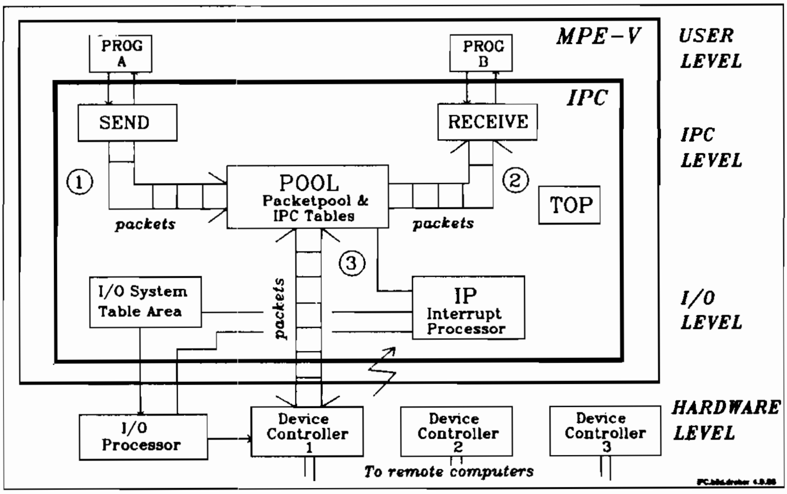


Fig. 2: Interprocess Communications System

Currently our Interprocess Communications System (IPC, not to be confused with HP's MPE IPC!) is implemented as a store-and-foreward packet switching network (Fig. 2). Information is being exchanged in the form of messages which are packed into one or more packets. They are sent from a sender to a receiver. It is relatively unimportant whether both reside on the same system or whether they are processes of different computers. If the data exceed the length of 2046 bytes, the message is split into two or more packets. Packets are the smallest entities that are transmitted between different computers. They consist of a header portion, which contains essentially address information in the form of symbolic names and the data area.

The main software interfaces for the programmer are two procedure calls: SEND and RECEIVE. Parameters are the symbolic names of the sender and of the receiver, a buffer with the data, and the amount of data to be transmitted. The communications software decides if this is a local communication (between processes at the same system), or if the packets have to be transmitted to another computer in the network via an appropriate I/O channel.

Since all packets are always buffered in a global data area before they are delivered to the receiver (packet pool), it was very easy to implement a store-and-foreward function. Therefore, if a packet comes in at the CE computer from e.g. the H1 system and the final destination is the M1 computer, it is simply put on the outgoing queue to the M1 computer and then eventually transmitted via the corresponding I/O channel.

The communications hardware is currently still a 16 bit parallel interface, which originates from the "Programmable Controller" product that was offered many years ago by HP for the HP3000-II. When we upgraded our Series III to the HP3000/64 we had to build a converter from the HP-IB (the internal I/O bus of the HP3000/64) to the old parallel interface. Therefore we had not to change anything at the side of the frontend computers. The maximum hardware data rate can be up to 1 MByte/sec. As an effective transfer rate we achieve 90 kBytes/sec if we transfer large blocks of 16 kBytes.

A general server program on the HP3000 and on the HP1000 systems allows the use of their file system and their peripherals by the other members of the network. As an example, the HP3000 serves very well as a remote spooling system for printer output of the smaller systems.

Data Acquisition and On-line Analysis in a Distributed Computer System

The systems H1 and PE are both being used for fast data acquisition during experiments. As the interface to the experimental set-up we use CAMAC, which is a very popular interface standard in nuclear and high-energy physics. It allows to transmit 16 to 24 bit words in parallel with a rate of up to 1 M words/sec.

During a typical experiment high energy electrons hit a piece of material that is to be investigated (target). A certain physical process happens at the point of interaction between the electron and the atomic nucleus. During that process the electron looses some amount of energy and changes its direction of flight (scattering). It may happen that the electron is absorbed by the nucleus and/or other elementary particles are produced and emitted out of the volume of interaction. All particles coming out of the target have to be detected in appropriate detector systems and their characteristic data, such as energy, angle of flight, and exact time of detection, are measured. With the help of Analog-to-Digital and Time-to-Digital converters (ADCs, TDCs) the data are transformed into digital computer readable data. These data are supposed to describe the physical process under investigation as completely as possible. They are read out via the CAMAC system and written in compressed form on disk or - for archival purposes - on magnetic tape. Like a transaction logging file they contain an exact history of the experiment being performed. Later, after the experiment has been finished, the experiment can be replayed on the computer simply by reading the magnetic tape.

This method of using a distributed computer system for data acquisition uses the small computer system with real-time capabilities for the time-dependent tasks that need fast real-time responses. The data is then sent on-line to the central computer, where it is archived on mass storage and where there is enough computing power to do a first on-line analysis of the data.

This is accomplished by the receiving process on the HP3000 by not only writing the data to magnetic tape or disk but by writing the data

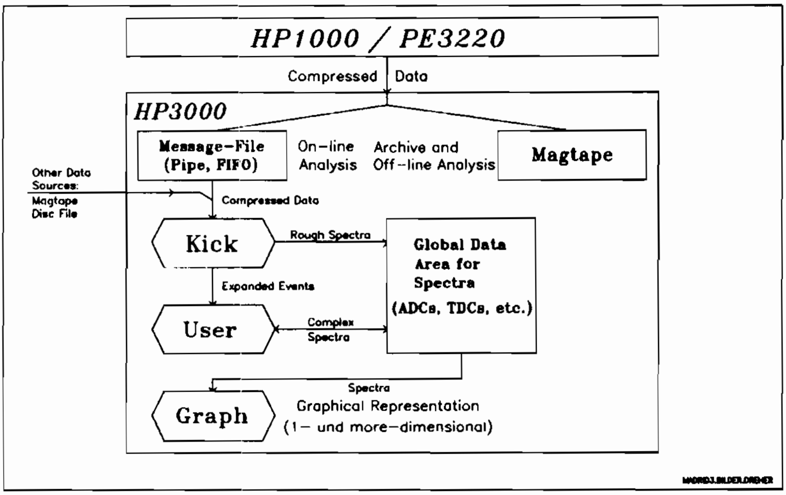


Fig. 3: On-line Data Analysis

to an MPE message file as well. At the other end of the message file the data analysis process reads the data for on-line evaluation (see fig. 3). This process acquires several large Extra Data Segments and sorts the data from the various sources (ADCs, TDCs) into one- or two-dimensional histograms. These spectra can then be displayed graphically by another process which has access to the same Extra Data Segments. Therefore already during the running experiment the experimentalist is able to analyze the incoming data. This is absolutely necessary for an efficient usage of the sparse and expensive beamtime. In fact, the same analysis program is used for the thorough off-line analysis as well. Therefore all the capabilities of the off-line evaluation are available online already.

The MAMI Control System

The M1 and M2 systems are used to control the MAMI accelerator. Figure 4 shows an overview of the MAMI control system. The two computers are coupled through the IPC message system and in addition by accessing a common file system on a shared HP7925 disk drive. In fact, our interprocess communications system as it exists today was originally designed for use in the MAMI control system and only later it was also implemented on the other systems. Particularly for the MAMI control system it turned out to be indispensable.

Here again the connections between the computers and the external process peripherals (operator desk and individual components of the

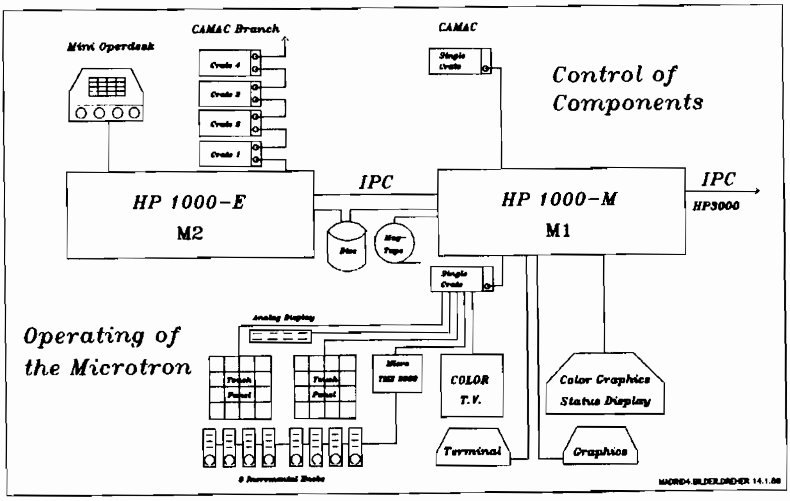


Fig. 4: The MAMI Control System

microtron) is through CAMAC. The communications tasks between the operator and the microtron run mostly on the M1 system while the M2 system is mainly used for the actual control and automatic optimization of the accelerator. The entire control software is, according to the necessary individual task, split into many small processes that are distributed over the two computers. The processes communicate through the message system.

Each component of the microtron that has to be controlled, or each class of similar components, has a special "service routine" assigned to it (Fig. 5). This service routine has the detailed knowledge of the hardware that is to be controlled (similar to an I/O driver in an operating system, but this is a regular user program). Characteristic data of the individual components, like hardware addresses, nominal values, limits, etc., are kept in a data base. They can be read or, if necessary, modified by the processes of the control system. In particular, the data base contains at any time a complete image of the actual status of the microtron (as far as the values are known to the computer). On a touch of a button this image can be stored in a file. Later, the data from that file can be loaded again into the actual data base. Thus on the basis of these values the microtron is brought into the same operating state as before.

The operating of the accelerator is done through three "touch panels", which function very similar to the HP150 touch terminal. Our touch panels have 16 fixed touch-sensitive areas which can be written through CAMAC (which is much faster than via RS232). By touching only few fields the operator is able to find in a tree-structured way the

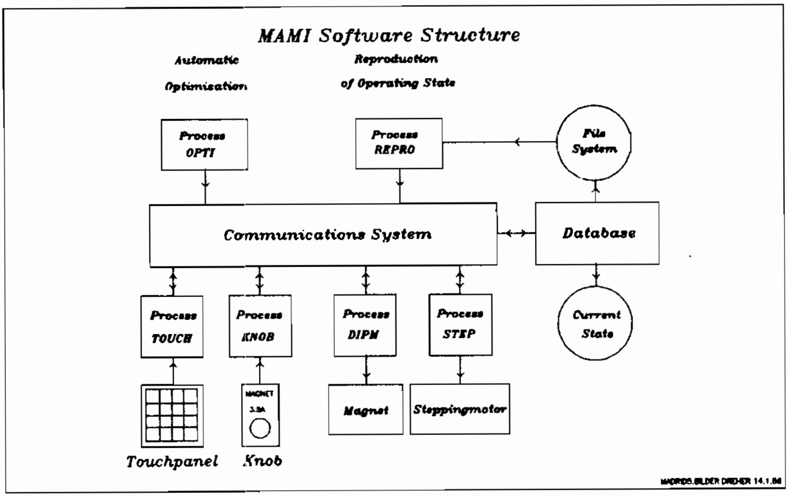


Fig. 5: MAMI Software Structure

final position, where he can for instance switch on or off an individual component of the microtron. To change analog values, e.g. the current of a magnet, manually, an incremental knob is logically connected (upon a touch) to that magnet. This knob has an alphanumeric display field where the computer displays the MAMI name of the magnet and the digital value of the actual current. A turn of the knob is converted to a digital signal which is sent to the corresponding service routine. If the requested value is legal, the service routine will adjust the magnet current accordingly. All this happens so fast that the operator has the feeling of using an analog potentiometer.

The global status of the entire microtron is displayed in graphical form on a high-resolution graphics display (HP2700). Its real zoom function allows the display of finest details.

Conclusion

The design and implementation of a distributed computer system for data acquisition, process control, and scientific calculations, was a great success for our institute. Many applications were only possible by features such as distributing different functions to the most adequate computers and then being able to communicate easily between the individual processes. Tranparent access from the front-end computers to the powerful peripherals of the central computer is a very economical solution for providing to all members of the network the access to those peripherals. Given the fact that the new DS products from HP, such as NS/3000, provide very similar functions as our system, many ideas of our design can probably very be easily adapted to all HP systems with the new software.

We ourselves will use in the future the hardware provided by these new products (IEEE 802.3 LAN) as a replacement for our old parallel link to connect the HP3000/68 or new upcoming products to other computer systems with a compatible LAN interface. Hopefully Hewlett-Packard will provide us with an appropriate software interface to the LAN at a level that is low enough to enable us to adapt our existing system to the new hardware. The LLA (Link Level Access) that is available in HP-UX for the HP9000 Series 500 is a good example for that.

Biography

Björn Dreher
is head of computing at the Institute for Nuclear Physics of the Johannes Gutenberg University at Mainz, F.R. Germany. He got his first computer experience with a Control Data 1700 minicomputer in the late 60s. The first Hewlett-Packard computer he worked with was an HP21MX in 1974, today known as the HP1000-M. In 1976 it was decided, that a distributed

computer system should be set-up at the institute using HP computers. Late 1976 a HP3000-II was installed together with two more HP1000 systems. Today the HP3000-II has grown to a HP3000/68 and the institute is anxiously waiting for the availability of the first member of the Spectrum computer family.

[1] Proceedings of the 1981 Berlin International Meeting of the HP3000 International Users Group, paper I.4

Siv Hermansson
PaBis ab
Belgium

"MPE Compared to HP-UX".

NOTE: Because of reasons out of the hand of the Host Committee, this paper will not be published in the Conference Proceedings.

HOSPITAL INFORMATION SYSTEMS: TODAY, TOMORROW AND AFTER.

George T. Horne
The Hospital for Sick Children, Toronto, Ontario, CANADA.

Summary:

Hospital Information Systems evolved in the wake of changes in the health care field. In-house development of an integrated system at the Hospital for Sick Children in Toronto resulted in a flexible network on three HP 3000/68 mainframes. Conversion of the system to a generalized bus structure with distributed intelligent workstations is under development. Future criteria and technical perspectives for the next decade are being investigated.

Out of the past...

As they have in almost all other areas of modern human endeavor computers have established themselves in health care, particularly in hospitals, and are showing every sign of being there to stay.

Contrary to the optimistic believes of the early sixties it has not been an easy walk. Indeed, few other fields are so full of pittfalls, beartraps and crocodile swamps as this particular one.

It has been said that those who don't learn from history are destined to re-live it. In order to investigate the - hopefully - successfull future of hospital systems, let us first look at the difficult past and the not allways glorious present.

Medicine is a very old profession. However the last 50-80 years brought more change to health care than all the previous centuries. As recently as in the first decade of this century there was, beside surgery, very little a physician could really do to cure a patient. Doctors were brilliant diagnosticians and could predict the course of an illness, there was a rather small number of drugs and techniques to alleviate pain and remove problems. Hospitals provided the necessary restful environment. There was little technology and only a few laboratory tests, most of which did not require complex equipment. The hospital was an extension of the family physician's field of action. It was controlled by doctors and control, being based mainly on the medical needs of patients, was relatively simple.

All this changed in the third and fourth decade: new tools, pharmaceuticals, particularly antibiotics, and the ever increasing use of complex technology made actual treatment and cure of illnesses an every-day practice. The costs and special skills needed to use these tools focused care more and more on hospitals, which in turn became complex entities not dissimilar to industrial plants.

Financial/social issues, e.g. availability of care, health insurance, etc. and the ever increasing costs also started to complicate the financial management of institutions, and their management became more and more the job professional managers - administrators.

As soon as usable computers appeared on the scene, enthusiasts believed that medicine was just the perfect place for their use.

So, in the early sixties we saw a group of physicians and engineers actually specifying, in a round table discussion, a suitable machine. The result was, as we know, the LINC - 8, which in turn became the PDP-8, and the rest is (non-medical) history.

Two areas of applications emerged: a) analysis of diagnostic measurements, typically ECG's, heroic patient monitoring, etc and b) processing of hospital/patient data, aimed at - hopefully - better management of information and resources.

Some major computer vendors (particulary one, associated somehow with a blue colouring) spent considerable sums in both areas, with relatively limited success - probably to their great surprise.

There was a number of reasons for this: analysts and programmers started out with their pre-conceived ideas about medical practice; diagnostic models proved to be just that, models that did not necessarily apply; the process by which a physician arrives at a diagnosis is more synthesis than anlysis; and human physiology stubbornly refuses to comply with computer logic.

On the management and information handling front it didn't look much better: most honest attempts to cost-justify computerization failed. Many companies are still at it and still find it difficult.

In spite of initial difficulties, hospital systems have gained wide acceptance in the last 10 years, have become imperative for modern health care and represent a substantial market.

It is interesting to note that to this day no computer hardware vendor has been extremely successfull in hospital information systems. Most of the existing, functional systems come from specialized software suppliers. I do not believe that we will see any major changes in this respect.

As we look at today's scene we have to start differentiating between the various major components of hospital systems. To simplify matters for the purposes of this paper we may safely state that most modern, at least medium-sized hospitals have at least some of their business functions (accounts, payroll, etc.) computerized. Basically these functions do not differ from similar applications in other industries, hence we shall not dwell on them.

At the other end of the scale we have the rapidly growing family of specialized medical instruments using computer technology as part of their basic function: computerized axial tomography (CAT), digital x-ray, ultrasonography and many more. They are now in the realm of companies marketing the instruments and DP professionals in hospitals are involved only marginally.

Let us concentrate on the middle field, known under the catch-all name of HIS - Hospital Information Systems.

We shall use the example of one hospital, the Hospital for Sick Children in Toronto, look at our present system, plans for tomorrow and dreams for the next decade.

The HSC System.

Our hospital is a 680-bed paediatric facility, aimed at all levels of care, with emphasis on tertiary care.

The first attempt for computerization was launched by the hospital in 1968 with the usual business systems and a rather grandiose plan for a "total" HIS. Part of this materialized in the form of an on-line Patient Admission/Discharge system as well as batch systems for laboratory test result, Medical Records and a number of other minor applications. The generally successfull, if expensive project, initially supported by Big Blue, ended in a major disaster in 1974 when, based on a consultants recommendation, an attempt was made to move to a remote data centre and to support a total of 10 teaching hospitals in Toronto. Result: nothing worked and the costs were astronomical.

In 1975 a small 4-member team proposed to start again, on an in-house basis. None of the commercially available packages were found satisfactory for our environment and we decided to develop our own customized system. There were severe, "post-disaster" financial constraints.

We started out with an HP 3000 CX, with 128 K and two 15 Mb disks. We decided to use SPL (for efficiency) and IMAGE only.

We wrote our own Screen Handler/Run Executive. It still stands up and so far we could see no advantage in using V-3000, or similar.

The lack of funds forced us into a design that is now the main virtue of the system. We could not start out, as most traditional HIS systems do, with one all-encompassing database. Each sub-system was to have its own database and had to be able to talk to the others. We wrote a communication file handler. To our great relief HP later invented IPC ; our purpose in life is not the development of system software, but rather of applications.

An A/D/T system, laboratory result reporting and a Medical Records systems were implemented within a year. Database space was obviously quite restricted and later grew as several 120 Mb drives came on stream. In 1978 we upgraded to a Series III.

It became very rapidly obvious that the machine would not be able to handle the volume of transactions and simultaneous on-line applications for more than a handfull of terminals with anything resembling reasonable response time. We upgraded to a 3000/64 and two years later added a second 64. This set-up enabled us to implement most of the basic applications that form a hospital system as it is perceived today. Disc space grew to 4000 Mb.

The Central Patient Index, which is, and will be the cornerstone of all patient systems holds demographic, and some additional information on almost a million patients. A Soundex program enables search on name and partial name if an ID number is not available.

All Registration functions (Emergency, Outpatients, Surgery,etc.) with their respective databases work through the CPR. Plastic ID plates and all necessary documents are produced on-line.

Laboratory sub-systems, with their databases, support laboratory test processing which includes on-line instrumentation and reorting of test results to the Nursing Stations, cumulative reports and laboratory statistics.

Introduction of terminals in the Nursing Stations represented a major challenge. We had to have a reasonably flexible communication network, that would serve us for some time into the future. We opted for the use of the hospital's telephone PBX and in addition to it we are using a second, voice-over-data network also on the standard telephone wires. This provides us with the capability for two data channels and one voice line at any location, as well as dial-up, modem sharing, etc., capabilities at all locations.

The machines were upgraded to mod.68's. A third, mod.48 machine was added for accounting & other functions. Down-time, planned (backup) or otherwise became crucial. Hospitals just refuse to conform with System Supervisor manuals.

It became necessary to mirror-write our databases to ensure functionality and data integrity - after learning some bitter lessons from hot-site situations and system crashes. We developed double-write capabilities, using IPC files and DS as well as our own logging system.

Later-on, when "Shadow/Silhouette" became available, we switched to this product, mainly for maintenance reasons.

As opposed to most computer vendors (and systems people) we were aware of the fact that nurses dislike typing - and most of the time have other things to do. We looked for ways to minimize typing. As we are using block mode, at least we didn't have to wory about the line-by line entry imposed by most other systems and 4GL's. Although we believe that a touch-screen would be an excellent tool, we found the resolution and handling of the HP 150 disappointing (and the cost prohibitive) and opted for a no-typing approach using inverse video light-bars operated by four function keys.

The use of separate databases for application systems in conjunction with Message Files has the great advantage that we can develop whole sub-systems, debug, test, and subsequently implement by just opening the "tube" while keeping the system operational for the users. This approach is also used for interfacing externaly purchased application packages, e.g. a Pharmacy system.

Fig.1 shows the evolution of various applications as it stood in 1985. The system is well received by the users - particularly since we did away with the need for signing on. By mid-'85 we approached an average 6-8 thousand transactions per hour on some 180 terminals and began to have serious response problems. A performance analysis showed that we are simply out of CPU cycles on our "production" machine. We had to spread the on-line load over two machines and lost valuable redundancy. An upgrade of our 48 to a 68 will give us another year of expansion - and time to think and plan.

Future development & plans.

The Hospital has launched a re-building program and by 1991 all patient Wards will be moved to the new wing. All rooms will be private. We have the rare opportunity to use the next five years for testing techniques and concepts that will take our hospital into the 21st century. Fortunately it seems that hospital organization and management is unlikely to change drastically. What we can expect is even more emphasis on management information needs, aimed at cost control and containment. However, in the areas of information handling, control and manipulation we will see drastic changes and growth as medicine is entering the computer age with a vengeance.

As we look at our present HIS, it does not substantially differ from a number of commercial packages - it is certainly more flexible, well integrated and has a higher level of data security, etc. but it still leaves us with a number of problems, unresolved questions and open philosophical as well as technological approaches. Some of these, in no particular order, are:

- How do we stem the river of paper ?
- Data entry vs. information retrieval; where, why, how?
- We do have plenty of data; how can we provide information?
- The storage problem; where do we put it all?
- The terminal problem; how to avoid typing.
- The signature problem; how to be safe and stay practical.
- The flexibility problem: ease of use/user control vis-a-vis data quality & integrity.

- Vendor independence, or Can you put your eggs in several baskets?
- Response: how to improve it and how fast is fast enough?
- Survival: as dependency on the system grows, how do we keep vital functions if a machine dies?

At first glance these questions do not differ from similar ones appearing in other fields, like industry and business. But in the hospital environment we usually find some circumstances making the answers less than obvious.

We do not pretend to have all the answers; we have some, some are very obviously motherhood issues and some are intended to present a challenge to the industry and a look into a still rather cloudy crystal ball. It has to be emphasized that in the medical field we should have preferably both feet planted on the ground; this author at least would not cherish the idea of making repairs to parts of his anatomy dependent on a system that has just gone "hot" and is being patched by the Response Centre.

First of all, there isn't the slightest hope for stemming the proverbial river of paper unless the HIS becomes a part of a comprehensive communication network in the hospital. The network has to include all aspects of communication: voice, data, graphics, image and video. It is also mandatory that future workstations in active areas of medical care be able to have access to all these media.

It should be possible for a physician to e.g. dictate diagnostic findings into a digitizing voice storage system accessible to other doctors via telephone; after a period of time important data from such findings should find their way into the relevant databases and eventually end up in the patient's medical record stored on optical disc for random recall.

Most existing hospital system have set as their main goal the entry of orders and retrieval of results,etc. at the Nursing Station. This seems to be a remnant of a programmer's understanding of hospital procedure. It should be stated that unless data is entered at source, we defeat the purpose of the system. The place for order entry and entry of patient and nursing data should be the bedside.

Conversely, opposite to general belief the bedside is usually not where medical and other decisions are made, but rather at the Nursing Station, the doctor's lounge or diagnostic areas such as X-Ray, etc.

These are the places where we need Information rather than just data. The users should be able to control extensively the form and format of such information, without having to rely on computer experts to do it for them.

Having stated the above, let us investigate how our system can evolve in the stated direction. Just throwing in more CPU power is clearly not enough. We have allready experienced system overload, mainly because we are forcing the machine to do both data management and terminal control.

We intend to make one machine into a "database engine" or archival machine, performing most of the database writing in the network and, under normal circumstances, virtually no terminal transactions. Several "terminal servers" would have copies of the databases. As access would be mainly for reading, we can take advantage of multithreading (Turbo-Image) and cacheing. Should a server machine fail, terminals can be temporarily switched to the archive machine, sacrificing speed but keeping operational; should the archival machine fail, existing data would still be available till alternative rescue is performed.

There are two pre-requisites for this: Software to perform the necessary file transfers between the machines and a fast enough communication system to make it work. Fortunately both are in existence in the form of Silhouette/Caress and HP-LAN respectively. We intend to test this arrangement in 1986/87, with a view to introduce a Spectrum machine into the network, probably in 1988. Our aim is to be able to support, by 1990, up to 500 terminals.

The other important step is to give users of information substantial independence, while improving visible response time. We see a viable solution in introducing intelligent workstations instead of terminals.

On the Nursing Station level this would be represented by a "Supermicro", e.g. HP 9000, Micro-VAX, Micro-EAGLE. The common,required characteristics are specified as: UNIX Operating System, a relational DB tool common to UNIX users rather than machines (e.g. MISTRESS) and the use of "C", Pascal, or a 4GL common to the 3000/Spectrum and the Workstation. Databases would be down-loaded to the workstations; this is transparent to the user. The users gain substantial independence and at the same time better response; the average number of patients per Nursing Station is only about 30 and hence the databases are small.

Also, the number of applications used is restricted, depending on the clinical speciality of the particular Ward.

With the increasing capabilities of Supermicros we may also expect to use them as local servers for terminals in patient rooms.

By adhering to some broad standards we may also achieve a measure of vendor independence on this semi-peripheral level.

Based on these philosophies we have stated our aims for the next five years as:

- Gradual conversion of the system into a bus-structure Information Network, encompassing voice and data communication, Workstations and graphics capabilities.
- Central archival database machine(s) and terminal servers. Workstations with read-only databases (relational) for user independence.
- Capability to support 500 - 700 terminal devices with adequate response and very high system reliability by 1990. Full redundancy of main databases. Major computer mainframe upgrade ("Spectrum") approx. 1987/88.
- Data entry at source, i.e. at patient level, for Orders, Nursing Notes, Q.A., bedside Test Instruments, etc., where appropriate.
- Information retrieval with maximum flexibility and under user control at Nursing Station/Ward level. Computerized Patient Chart. Potential access do digitally stored (Optical Disc) Medical Record. Similar capabilities at Clinics.
- Inclusion of additional Diagnostic and other services into the Network.
- Vendor-independent Operating System and language support at Workstation/Peripheral processor level.
- Common 4th Generation Languages for Micro- and Mainframe systems. Transition to "C" and Pascal as preferred languages on mainframes.
- Introduction of Optical Mass Storage, Voice Input/Control and other advanced technologies.

New Technologies:

It is the last point of our objectives that focuses our attention on what will be necessary and desireable in the future hospital environment. Let us now take our crystal ball and list some of the expected areas of interest, equipment, and, of course, associated problems that have to be resolved.

Mass Document Storage.

For every patient and every stay or encounter, the hospital amasses a very large amount of various data, most of it in the form of printed or written documents, but also graphic (ECG) material and x-rays. Warehouses are being filled with medical records, that have to be kept for many years. Records are being requested,sent out, searched and researched and frequently lost. There is now hope in the form of optical disc storage. One platter will typically store 1 Gigabyte, a "jukebox" device can handle 1-2 hundred platters. Documents can be stored via a digitizing scanning device and random retrieval in about 15 seconds seems to be feasible. A substantial effort would be needed to convert existing records. However, beside the obvious savings for storage space we can also see a very important qualitative change. We may stop shipping paper: for the first time it may be possible to retrieve parts of a patient record via remote terminals.

Problems to be resolved: Legal aspects, privacy & access security, combination of documents and computer generated data, and periodic housekeeping problems as arecord may be spread over more discs.

On a smaller scale, there is CD-ROM and WORM-type disc systems, that show great promise for e.g. Nursing Manuals, teaching systems, and infrequently updated lists like drug interactions etc.

Bedside input devices:

As we move data entry closer to its source, the patient, it is fairly obvious that the old-fashioned terminal just won't do. Nurses and doctors are not typists, we want to free them from clerical work,not add to it. The ideal device would be flat and could hang on the wall; it would have colour and graphic capabilities for chart retrieval; it it would be icon-driven and controlled by touch and/or voice. It has to be able to accept some form of signature. It should also be part of the voice communication system.

Problems: price, selective access security.

It is hardly surprising that major communication companies are involved in the development of such devices; they are also increasingly involved in Hospital Systems. We can probably expect more from them then from the notoriously introvert computer manufacturers, as their livelihood depends on communications. They will eventually make terminal protocols transparent and make our lives a lot easier.

We can expect that in the forseeable future 30-40% of routine laboratory tests will be performed on-line, at the bedside. This implies a plethora of quality control and interfacing problems. One more reason for protocol standards across the industry.

Expert Systems and AI:

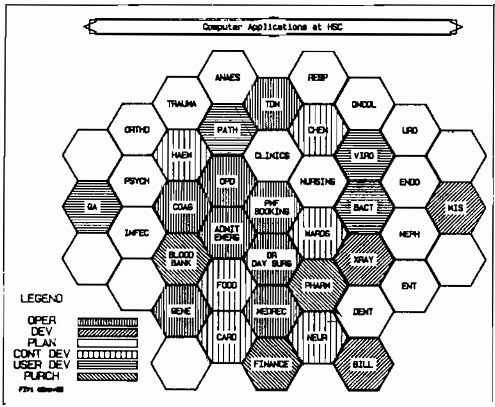
The medical profession is by its nature conservative and cautious. It will take some time until any, even well functioning diagnostic tools will earn acceptance. The sheer notion of AI is anathema to many doctors. There is however room, in the next decade, for a variety of uses of AI systems in combination with diagnostic instruments, quality assurance, quality control in laboratories and other bio-technical areas. A broad application field for expert systems is in training, education and clinical modelling. We are presently investigating some attempts in modelling the treatment of post-operative patients.

We do expect successful use of Expert Systems in the area of laboratory order processing and Pharmacy, as warning systems (potentially diagnosis dependent) for drug/test interactions.

Image processing:

Digital image processing has revolutionized the diagnostic field.

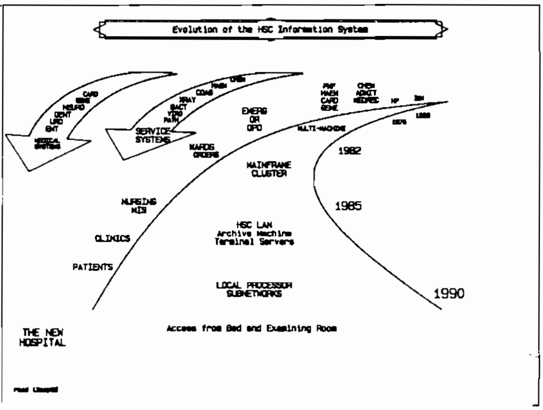
It is quite conceivable that we shall soon see the demise of x-ray films, although there are still numerous details to be worked out concerning the understanding of human vision, relative resolution, and also acceptance by physicians. From a storage point of view, we are faced with a major problem, waiting for appropriate technology. For example, the average daily output of x-rays at HSC, assuming a reasonable resolution, would require close to 6 Gigabytes of permanent storage.



Computer Applications at HSC

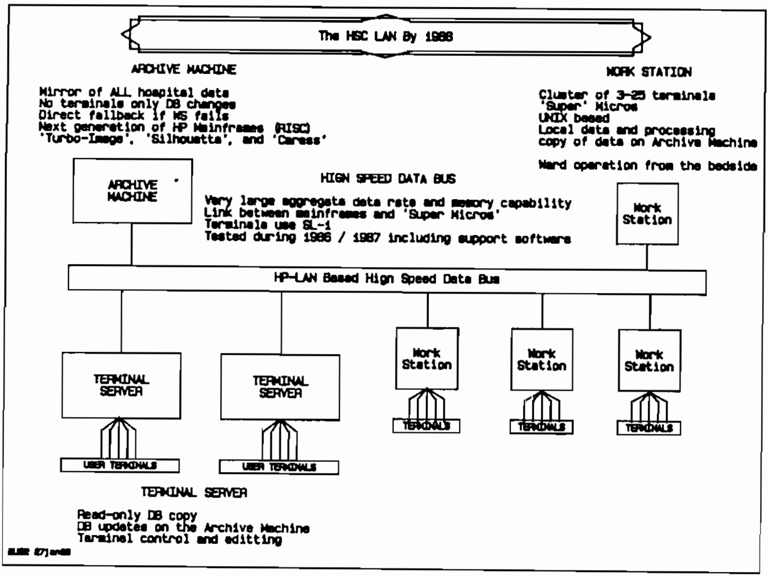
LEGEND

OPER	▥
DEV	▨
PLAN	□
CONT DEV	▥
USER DEV	▤
PURCH	▧



Evolution of the HSC Information System

Access from Bed and Examining Room



The HSC LAN By 1988

ARCHIVE MACHINE

Mirror of ALL hospital data
No terminals only DB changes
Direct fallback if WS fails
Next generation of HP Mainframes (RISC)
'Turbo-Image', 'Silhouette', and 'Caress'

WORK STATION

Cluster of 3-25 terminals
'Super' Micros
UNIX based
Local data and processing
copy of data on Archive Machine

Ward operation from the bedside

HIGH SPEED DATA BUS

Very large aggregate data rate and memory capability
Link between mainframes and 'Super Micros'
Terminals use SL-1
Tested during 1986 / 1987 including support software

TERMINAL SERVER

Read-only DB copy
DB updates on the Archive Machine
Terminal control and editting

Biography

George T. Horne
has been with the Hospital for Sick Children in Toronto for 16 years, for the last 11 as Director, Computer Systems. Prior to coming to HSC he worked in medical computing at the Royal College of Surgeons of England and in continental Europe. His background and education includes Engineering (electronics) and Medicine.

SERIES 58 PERFORMANCE

Jim Kramer
Hewlett-Packard Company, San Diego California, USA

Summary

The Series 58 is the new mid-range system in the HP 3000 line of computers. It has a more powerful processor than its predecessor, the Series 48, and its maximum main memory configuration is twice as large.

In this paper I will discuss the hardware characteristics which result in the improved performance of the Series 58, and will report the results of performance tests. For those system owners who might be considering an upgrade, I will present some guidelines to aid in determining whether an upgrade is likely to result in a significant performance improvement.

Series 58 System Description

The Series 58 introduces new processor, memory, and memory controller boards. Otherwise the system hardware is virtually identical to that of the Series 48, except that 1/4 megabyte memory boards will not work on the Series 58.

An upgrade to a Series 58 from a Series 44 or 48 replaces the CPU (two boards replace two or three), Control and Maintenance Processor board, and memory controller board. It also adds two megabytes of memory on a single board. A similar upgrade is available for a Series 39, 40 or 42, and the upgraded system is designated the 42 XP.

Supported memory sizes start at 2 megabytes and extend to 6 megabytes for the Series 42 XP and 8 megabytes for the Series 58. (The 2 megabyte minimums are possible if the system being upgraded has only 1/4 megabyte boards.) A new Series 58 will have 4 megabytes or more, and a "new" 42 XP (Series 42 XP upgrade purchased with a Series 42) will have at least 3 megabytes; note that these minimums are maximums for the 48 and 42 respectively.

The new hardware has been designed to run existing software; the processor identifies itself as a Series 48 processor in order to eliminate the need for changes. (A new machine instruction is available to distinguish the Series 58 processor from its predecessor, if that should be necessary. Since it will cause an illegal instruction trap on HP 3000's other than the Series 58, it cannot be used until MPE recognizes and emulates it). Supported software has been restricted to T-MIT and its successors to minimize system testing requirements.

The Series 58 and 42 XP, like the Series 42, 48 and 68, are always cached systems.

Performance Considerations

In the following discussion of performance considerations, comments about the Series 58 apply to the 42 XP as well (unless 7 or 8 megabytes of memory is being used).

The performance significance of the Series 58 compared to the 48 is its faster processor and the doubling of the maximum memory size. The processor provides substantially greater throughput than the Series 48 processor, as shown by the test results presented later in this paper.

For upgrade customers, the addition of caching to non-cached systems, and the addition of 2 megabytes of memory are also significant.

One thing to note about the new machines is that they are certain to match or outperform their predecessors. The processor is more powerful, their minimum memory configurations were maximums on the predecessors, and caching is available. The same claim could not be made about some prior upgrades: sometimes a Series 44 (if it only had one disc controller) performed worse than the Series III it replaced, and sometimes a 48 with caching turned on performed worse than the 44 it replaced.

The new systems address the three major performance bottlenecks: processor, memory, and disc I/O. The processor and memory bottlenecks are addressed directly with additional capacity; the disc I/O bottleneck is addressed with caching. Customers who already have caching are almost certainly CPU-limited, and will be helped by the additional processor power.

One can try to imagine a load on a 44 that would not run faster on a 58. It would be disc I/O limited, and the disc I/O would be so nonlocal that caching (and the additional processor and memory to support it) would not help. This situation is unlikely, however. Our experience is that caching does help when the necessary extra CPU and memory capacity are there, because MPE and most applications exhibit significant locality in their disc access patterns.

Please notice how much growth capability has been provided the Series 39, 40 and 42 in the last year. ATP's allow connection of as many as 60 point-to-point terminals. And the 42 XP upgrade provides a maximum memory size 50% greater than that of the Series 48, and greater processor power.

The Series 58 Processor

As mentioned earlier, the significant performance gains from the new systems are provided by the Series 58 processor and increased memory sizes permitted. Disc caching may also provide performance gains for customers upgrading from uncached systems.

The Series 58 processor contains a 32 Kbyte high-speed memory cache for buffering both instructions and data being read from memory. (Be careful not to confuse the memory cache with MPE's disc caching.) If a required word is in the cache, it is obtained in a single machine cycle. Six cycles are required by the Series 44 processor to retrieve a word from memory.

To take full advantage of the cache speed, hardware was added to speed up functions that in the Series 44 were performed while waiting for data to be read from memory. Microcode of the most frequently used machine instructions was rewritten to make use of the new hardware.

For performance, our main concern is that the processor speedup was achieved mainly through the provision of a memory cache, not through a processor clock speedup.

Initial performance testing concentrated on the new processor. Less emphasis was placed on the effects of additional memory and disc caching.

Performance Test Results

We will begin our summary of performance test results with some tests that do no I/O, and therefore isolate processor performance. It is important to note that these tests are timings of small programs that repetitively execute code within a loop. They therefore make excellent use of the memory cache, and show the series 58 processor to good advantage.

The Sieve of Eratosthenes is a prime number calculation routine that has been coded in many languages and run on many machines. It tests looping, array indexing, and integer arithmetic. The following table shows results on the 48 and 58 processors for four languages. The timings are in seconds required to execute ten loops, each of which calculates 1899 prime numbers.

	Series 48	Series 58	48/58
Fortran	2.71	1.56	1.74
SPL	2.75	1.59	1.73
Basic	2.20	2.00	1.60
COBOL-II	17.30	10.18	1.70

The column labeled 48/58 is the Series 48 timing divided by the Series 58 timing. This is the throughput ratio. The number 1.74 for Fortran indicates that the Series 58 can perform 74% more work of this type in a given time than the 48. (The "response time" improvement is 42% -- (2.71-1.56)/2.71. The response time improvement for a stand-alone task is always less than the throughput time improvement.)

The Whetstone benchmark tests floating point computations. It was done for both single and double precision.

	Series 48	Series 58	48/58
Single	9.31	7.02	1.33
Double	26.14	22.57	1.16

The throughput improvement is less than for the Sieve, because floating point instructions do extensive calculations on the data extracted from memory, so that the cache provides less help. The floating point and packed decimal instructions were not re-microcoded. Most applications do not use them as intensively as this benchmark.

An extensive set of tests was done testing various language constructs. The timings were obtained by executing a loop with and without the construct, and taking the difference. Below is a small sampling. The times are given in microseconds.

	Series 48	Series 58	48/58
SPL integer move	2.8	1.2	2.3
SPL real move	4.4	2.5	1.8
SPL long move	8.9	4.9	1.8
COBOL II S9(7) COMP-3 move	41.	34.	1.2
SPL move 80 bytes	62.	46.	1.4
SPL integer add	4.2	2.0	2.1
SPL real add	11.	8.0	1.4
COBOL II S9(7) COMP-3 add	160.	130.	1.2
SPL integer multiply	7.3	5.0	1.5
SPL real multiply	16.0	12.0	1.3
COBOL II S9(7) COMP-3 mult.	220.	190.0	1.2
SPL call/return, no parms	23.	16.	1.4
FWRITE 80 bytes, large BF	2300.	1600.	1.4

The pattern holds that speedups are greatest for the simpler operations which can make good use of the memory cache. The 2.3 speedup (130%) for the SPL integer move (LOAD, STORE) is the current series 58 speed record.

Now we have some tests which are much more representative of actual system usage. Descriptions of the tests come first, followed by a table of the results.

Compilation

1. COBOL Compile -- This test generated 32K lines of output. There were a few include files.

2. Pascal Compile -- A 3500 line program was compiled, no include files, and the output was sent to $NULL.

3. SPL2 Compile -- SPL2 is not available to customers, but some HP software is written in it. The program source was 5000+ lines, no include files, and the output was routed to $NULL.

TDP Operations

4. TDP Final 1 -- A 1515 line document was formatted to produce 2281 lines of output. There were 160 includes (each requiring a file open, read and close).

5. TDP Final 2 -- This test is the same as TDP Final 1 without the includes.

6. TDP Text and Keep -- A 10,000 line file was texted, a global change was made which modified and displayed 1136 lines, and the result was kept.

7. ED Text and Keep -- This is the same as the TDP test, but used an unsupported editor (similar to Unix's XED).

Data Base Access

8. ORBIT -- This is a production job from the MIS group of HP's Computer Systems Division. It reads, sorts and processes Image data. The sorting is done by Robelle's SUPRTOOL.

9. NEWJIT MRP -- This is HP application software that runs against an Image data base, and produces output files totalling more than 16K sectors.

All of the tests were run stand-alone on a 4 megabyte system with caching enabled.

	Series 48		Series 58		48/58	
	CPU	Wall	CPU	Wall		
Test	(sec)	(min)	(sec)	(min)	CPU	Wall
1 COBOL	1113	22	860	18	1.29	1.22
2 Pascal	169	4	122	3	1.39	N/A
3 SPL2	297	7	222	5	1.34	N/A
4 TDP Final 1	98	2	72	2	1.36	N/A
5 TDP Final 2	45	1	32	1	1.41	N/A
6 TDP Text/Keep	197	4	152	4	1.30	N/A
7 ED Text/Keep	120	3	84	2	1.43	N/A
8 ORBIT	2197	42	1801	33	1.22	1.27
9 MRP	10206	174	7214	123	1.41	1.41
Totals	14442	259	10559	191	1.37	1.36

The CPU throughput improvement of 1.37 is obtained from the CPU totals, and thus is a weighted average of the CPU throughput improvements for the 9 tests. A straight average yields 1.36.

A TEPE test was run of HPAccess, the new PC Central product which can extract data from the HP 3000 for use on personal computers.

The tests emulated five terminals performing such operations as project, join, select, sort, summarize and output. Two tests were run on both the Series 48 and Series 58. The tests differed only in the amount of data handled.

The following test results were taken from the emulator report. The response times are averages over all transactions for all terminals.

	Series 48		Series 58			
		Avg		Avg	48/58	
	Wall	Resp	Wall	Resp	Wall	Resp
	(min)	(sec)	(min)	(sec)	(min)	Reduct
Test 1	142	70.93	110	49.81	1.29	30%
Test 2	574	392.74	433	290.37	1.33	26%

Since the machines were almost never idle during the testing, especially for Test 2 (about 1% idle), the ratio of wall times can be taken as a good indicator of CPU throughput increases. The last column is the response time improvement.

The venerable EDP benchmark, which has been used to test the entire product line, was also run. Briefly described, the test comprises multiple terminals doing transaction processing with VPLUS and Image, and additional terminals doing program development. There are also batch COBOL compiles in the background.

For details of the test environment, consult the published performance guide, part # 5954-0401.

The following table shows throughput increases achieved by the Series 58, determined by comparing transaction execution rates at specified response times. The data provided by the tests gives transaction rates and response times for various numbers of terminals. To obtain transaction rates for the specific response times listed below, it was necessary to interpolate (linearly) between the provided points.

Response	Series 48	Series 58/4mb		Series 58/6mb	
Time	Trans	Trans		Trans	
(secs)	(/hour)	(/hour)	Increase	(/hour)	Increase
.5	4486	13428	2.99	13793	3.07
1.0	10879	20753	1.91	22100	2.03
1.5	16057	22147	1.38	24061	1.50
2.0	19253	23541	1.23	26022	1.35
2.5	20447	24935	1.22	26362	1.29

As the systems start to become saturated (2.5 second response), the Series 58 at 4 megabytes provides a 22% throughput increase. Additional memory is shown to help performance throughout, and near saturation the Series 58 with 6 megabytes provides a 29% throughput increase over the Series 48.

The next table shows response time reductions at specified transaction rates.

	Series 48	Series 58/4mb		Series 58/6mb	
Trans	Response	Response		Response	
(/hour)	(sec)	(sec)	Reduction	(sec)	Reduction
10000	.93	.38	58%	.37	60%
14000	1.24	.52	58%	.51	59%
18000	1.80	.76	58%	.66	63%
22000	4.37	1.45	67%	.84	81%

The response time reductions are substantial at all transaction rates. Again the extra memory is shown to be helpful as the system becomes busier.

Who Should Upgrade?

There are two good reasons to upgrade to a Series 58 (or a 42XP): to improve on-line response times, and to improve batch job turnaround times. The case of batch jobs is the easiest to consider: the Series 58 can be depended on to run them significantly faster. If the system being upgraded is uncached and the job is I/O intensive, caching should provide further help. However if the objective of the upgrade is to speed up just one or two critical jobs, it is definitely worthwhile to run them on someone else's Series 58 prior to upgrading to be sure that the hoped-for improvement will be achieved. I say this because a test completed very recently showed only a disappointing 10% speedup for a set of Image reporting jobs running concurrently.

The impact of a Series 58 upgrade on on-line response times is more difficult to predict. Improvements can range from dramatic to barely noticeable.

The dramatic improvements are seen when the poor response is caused by a shortage of a critical performance resource (CPU, disc I/O, memory). When one of these resources is near exhaustion, a small additional demand for it can make response times dramatically worse. Similarly even a small additional supply can make response dramatically better.

The chances for dramatic improvement are even better if the resource in short supply is memory, or if it is disc I/O on an uncached system. Here the benefits of the extra memory and caching add to the benefits of the faster processor.

The barely noticeable improvements, conversely, come when performance resources are adequate but the transactions are inherently long ones. The Series 58 will speed up such transactions, but reducing response time from 10 seconds to 7 or 8 seconds may not be all that was hoped for. The solution in such cases is, unfortunately, reworking the application or upgrading to an even more powerful machine than the Series 58.

How can you tell if poor response is due to a resource shortage or to long transactions? An easy way is to observe the response time for a single user running the problem application. If response for a single user is good, then poor response for many is due to a resource shortage. If response for a single user is poor, then the problem is long transactions. If you can say "Response used to be good, but has gradually gotten worse as we added more and more users," then an upgrade is almost certainly called for.

Another way to tell is by measurement. Tools such as OPT or Surveyor (in the TELESUP account) provide extensive information about system resource usage. However there are pitfalls to this approach that even experienced performance SE's have fallen into. For instance it is not enough to run OPT, observe that CPU usage is at 100%, and conclude that there is a resource shortage. This is because a significant amount of CPU used by batch activities (including such things as online compiles) really signifies the availability of CPU for short transactions, because of MPE's process priority structure. It is not enough to measure system resource usage; it is also necessary to determine which processes are using the resources.

Conclusion

The Series 58 and Series 42 XP provide help where it is needed in this era of disc caching: CPU and memory capacity. The processor performance increase, coupled with whatever help memory and caching provide, should give a significant boost to the current midrange machines.

Please join me in welcoming these new arrivals to the HP 3000 product line.

Biography

Jim Kramer has been an HP3000 Systems Engineer and Performance Specialist with Hewlett-Packard for eight years, and works in the San Diego, California, field sales office.

HP-2680A: THE MYSTICAL PRINTER

Richard Oxford
MCI Digital Information Services Corporation,
Washington D.C., USA

SUMMARY

This paper aims to more intimately acquaint the HP user with laser printers. This acquaintance is brought about through:

1. an explanation of technically what goes on in the box,

2. a description of the capabilities that are available to the user, and how to exploit them

3. and, an explanation of the data structures used in graphic data.

INTRODUCTION

Graphics are becoming an increasingly important part of computer information systems. Computer generated graphics range from simple bar charts, to multiple color presentation graphics, to very sophisticated solid modeling. With the advances in technology, and decreasing prices of hardware, graphic capable peripherals are becoming more affordable by the general public. They are no longer limited to large businesses with multimillion dollar DP budgets.

The decrease in price of laser printers, in particular, have made high speed printing and graphics more accessible. The need to generate graphics for the laser printer is increasing, but there seems to be only a small amount of effort being directed in this area. Here is a device capable of generating forms, logos, and signatures, as well as graphics, but yet it is quite frequently used only for high speed printing and forms generation. These printers have a lot of capability going unused, except for some packages developed by HP (TDP, HPDRAW, etc.). This "lack of use" can be attributed mostly to a "lack of knowledge" of how the laser printer actually works, and how to use it.

MCI DISC has been using many features of the 2680A laser printer for over 2 1/2 years. Many different logos, graphics, and signatures are printed during a typical job. Signatures and graphics are programmatically positioned on a page. All of this is done independent of an environment file! Most users are unaware of the ability to programmatically place graphic data, as well as text data, anywhere on the printable page. All functions performed by TDP are available to any programmer through intrinsic calls. One only needs to be familiar with the basic operation of the laser printer, as well as some additional data formats, to be able to take full advantage of the printer's capabilities.

Since we, at MCI DISC, are currently using 2680A laser printers in our applications, most information given will be in reference to this printer, but the same basic principles apply to other HP laser printers. Since the most popular print format is 8 1/2 x 11 inches, this will be used in the examples.

Basic Laser Operation

The old saying "It's all done with mirrors" really applies to the 2680A laser printer. Although it is not necessary to know how all of the mirrors function, it is beneficial to have a basic understanding of how the laser printer works. It will then be easier to understand data structures referring to raster scan lines and dot-bit images.

The generation of a printed image by the laser printer is a rather complex procedure. The description given here is a very simplified explanation of how the image is produced, slightly modified to be more easily understood. The generation of an image in the laser printer is accomplished through a process known as electrophotography. An image is produced on paper in three basic steps:

1. Generation of the image on the drum.
2. Transfer of the image to the paper.
3. Fixing of the image on the paper.

BASIC LASER PRINTER OPERATION

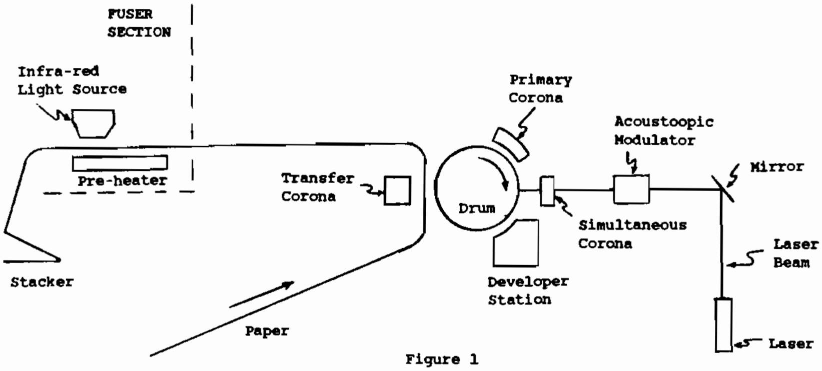


Figure 1

Figure 1 shows the basic layout of the 2680A laser printer.

Image Generation

The physical page on the 2680A is 8 1/2 x 11 inches and consists of a dot matrix of 2048 dot positions along the 11-inch axis and 1536 dot positions along the shorter axis, although the printable area is slightly smaller. The laser printer can handle a physical page size of 17 x 11 inches, but the 8 1/2 x 11 inch size is used for discussion. The laser beam scans from left to right, along the 11 inch axis, as the drum turns. Each page therefore, is composed of 1536 scan lines of 2048 dots each. A special device, called an acoustoopic modulator, receives data from the print control electronics and allows the laser beam to strike the drum when a dot is to be printed. The effect of the beam striking the drum, in conjunction with drum charging devices called coronas, results in a negative charge on the drum for every dot to be printed.

As the drum continues to turn, it passes by the developing station. By means of a "magnetic brush", toner is applied to the drum. The developer mixture used consists of iron filings called carrier, and tiny plastic balls referred to as toner. The toner is statically charged and attracted to the negative charge locations on the drum. The iron filings are used by the "magnetic brush" to help move the toner to the drum. When the drum has moved past the development station, it contains the final image to be printed.

Image transfer

Once the image is on the drum, it must be transferred to the paper. As the drum turns, the paper moves at the same speed as the drum, and passes between the drum and the transfer corona. The transfer corona imparts a high negative charge to the paper, which pulls the toner off the drum and onto the paper. The final image is now present on the paper.

Fixing the Image

As in most photographic processes, the final image must be "fixed" to prevent it from changing. At this point, before fixing, the image is nothing more than tiny plastic balls which can easily be smeared or brushed off the page. To fix the image, the paper passes through the final step in the fuser section of the printer. The fuser section consists of a pre-heater which heats the paper, and an infra-red light source that 'melts' the toner. The end result is that the toner is actually melted, or fused, into the paper.

SPOOLER RECORD LAYOUT

```
          MSB                                   LSB
          ---------------------------------------
word      | 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 |
          |-------------------------------------|
 0        | Logical Record Length in bytes +8   | ---
          |-------------------------------------|    \
 1        | Number of data bytes (including nulls)|   \
          |-------------------------------------|      \
 2        | Function code                       |
          |-------------------------------------|     one
 3        | Param1                              |   logical
          |-------------------------------------|   record
 4        | Param2                              |
          |-------------------------------------|
 5        | 1st data byte          2nd data byte |
          |-------------------------------------|      /
 6        | 3rd data byte          4th data byte |     /
          |-------------------------------------|    /
 n        | nth data byte              null *   | ---
          |-------------------------------------|
          |                                     |
          | Next logical record                 |
          |                                     |
          |-------------------------------------|
          |                                     |
          | nth  logical record                 |
          |                                     |
          |-------------------------------------|
          |  -1 (all ones) End of valid data    |
          |-------------------------------------|
          |                                     |
          |     non-valid data area             |
          |                                     |
          |-------------------------------------|
509       |                                     |
          |        This area reserved for       |
510       |              Spooler                |
          |                                     |
511       |                                     |
          ---------------------------------------
```

* all logical records must end on a word boundary. If they do not, a null is inserted, and included in the record length

Figure 2

COMMUNICATING WITH THE LASER

The next step in understanding how the laser printer works is to know how to get the image information from the HP-3000 to the laser printer. In order to maximize the throughput of the HPIB bus when communicating to the laser, data is sent to the laser in blocks of 512 words. The basic layout of these blocks is shown in figure 2. Each block can contain more than one logical record, with each logical record being a single function for the laser. Note that each logical record contains 2 byte counts, a function code, and 2 parameters, as well as any data being sent. The last 3 words in the block are used by the spooler. Although the block size for data transfer is 512 words, actual blocking is handled by the spooler and a buffer of any reasonable size can be written.

Most users have no problem sending files to the laser, or using TDP to communicate to the laser, but are limited to the constraints of those packages. Those who have used the LPS interpreter, or have written programs using 'P'intrinsics (IFS/3000 Programmatic intrinsics), have a better understanding of laser operation. If we look one step deeper, we find that the basic method used to communicate to the laser printer is via a single callable intrinsic, FDEVICECONTROL. This is a little known, even less understood intrinsic. This intrinsic gives us the ability to change character fonts, change logical pages, position text, download pictures or graphics, and much more. Table 1 gives a summary of the control codes used with the FDEVICECONTROL intrinsic and definition of each. (All of the control codes, with the exception of 139 and 144, have the parml and parm2 values shown in the MPE Intrinsics reference manual. The parameter definition for control codes 139 and 144 are given at the bottom of table 1.) Several of the control codes are used to set up or alter the current printing environment, load character sets and forms, or load Vertical Forms Control data. There are also codes for setting the physical page, loading logical page tables, and controlling the job. Since most of these functions are handled by the spooler when using an environment file, these functions will not be discussed here, but it is possible to have total control over the print environment even if no environment was specified for the job.

One very important concept is that of logical pages. The physical page, typically 8 1/2 x 11, can be divided into as many as 32 logical pages. A logical page is nothing more than a portion of, or a window on, the physical page. Each logical page carries its own set of specifications. They can be oriented in different directions. They can each have a different form associated with them. They can even have different character sets as their base set. Logical pages can also overlap. By simply having 4 logical pages all the same size as the physical page, but all with different orientations, you have the ability to write text on the physical page in any of the 4 possible orientations.

Now that all of the basic information has been given, let's discuss some of the function codes.

Contrary to popular belief, the laser can only have 2 character sets available at any given time. Everything else is done with slight of software. The character sets for use are selected with a control code of 128. Paraml and param2 bits 8-15, are used to define the primary and secondary character sets. Paraml holds the ID for the primary character set, while param2 holds the ID for the secondary character set. The IDs for the character sets are the font numbers as described in the environment file. The SO (%16) and SI (%17) ASCII control codes are used to switch to and from the secondary character set. (An important note: If you were using the secondary character set before execution of a control code 128, you will still be using the secondary character set after the execution, even if the secondary character set was changed.)

A control code of 129 allows the activation and deactivation of logical pages. Param2 is divided in half. The upper byte holds the logical page number to be deactivated, while the lower byte holds the logical page number to be activated. Paraml bit 0 indicates deactivate the logical page specified in the upper byte of param2, while bit 1 indicates activate the logical page specified in the lower byte of param2. Therefore it is possible to activate a logical page and deactivate a logical page at the

FDEVICECONTROL control code summary

Control Code (in decimal)	Definition
1	Print Data
2	File Control
3	File Open
4	File Close
128	Primary,Secondary Font Selection
129	Select/De-select Logical Pages
130	Move Pen Relative
131	Move Pen Absolute
132	Define Job Characteristics
133	Define Physical Page
134	Download/Delete Character Set
135	Download/Delete Form
136	Download Logical Page Table
137	Download Multi-copy Form Overlay Table
138	Download/Delete VFC
139	Download/Delete Picture
140	Page Control
141	Clear Portions of the Environment
142	Job Open
143	Load Default Environment
144	Print a Picture

Controlcode = 139		Download/Delete Picture
paraml	(0:1)	0 - Load a Picture 1 - Delete a Picture
param2	(0:1)	0 - First record of a load 1 - Continuation record of a load
	(8:8)	Picture identifier (0-31)

Controlcode = 144		Picture Print
paraml	(0:1)	0 - Temporary Picture 1 - Addressable Permanent Picture
	(1:1)	0 - Co-ordinates X and Y are relative to current pen position 1 - Co-ordinates X and Y are absolute pen positions on the logical page
param2	(0:1)	0 - First record of a temporary picture load 1 - Continuation record of a temporary picture load
	(8:8)	Picture identifier (0-31)

Table 1

same time with the same command. Further, all 32 logical pages can be active simultaneously. One logical page must be active at all times. Switching between active logical pages will be described later.

The command more frequently used to move between logical pages, without having to worry which ones are active, is the control code of 140. This is the page control function. It allows for a physical page eject and/or a change of logical pages. If paraml is set to a 1, a physical page eject occurs. If it is set to a 0, no page eject occurs. Bits 8-15 of param2 are used to specify the next logical page to be used. It can be the current logical page, or it can be a new logical page. Whichever logical page is specified in this parameter becomes the active logical page upon completion of the intrinsic call.

There are two pen movement commands. These are important for the positioning of data and graphics on a page. For those that are familiar with plotters, the first thing you look for is a pen up/down command. Well, there is none for the laser. The laser printer is not a plotter, and the pen command is only used to position information on the page. A control code of 130 moves the pen relative to its current position, while a control code of 131 moves the pen absolute with reference to the upper left hand corner of the logical page. The laser printer always views the logical page in its readable direction, so rotating the logical page moves its upper left hand corner (See Figure 3). The logical page is referenced in a logical page co-ordinate system with the upper left hand corner being 0,0. The X axis runs across the top of the logical page, increasing in value as you move to the right of the upper left corner, or to the right of the current pen position in a relative move. The Y axis runs down the side of the logical page. Its value increases as you move down from the upper left corner, or down from the current pen position in a relative move. The co-ordinates actually represent dots, so an X co-ordinate of 12 is actually the 13th dot along the X axis. It is the 13th dot because the co-ordinate system starts at 0. The same applies to the Y axis.

LOGICAL PAGE ORIENTATION

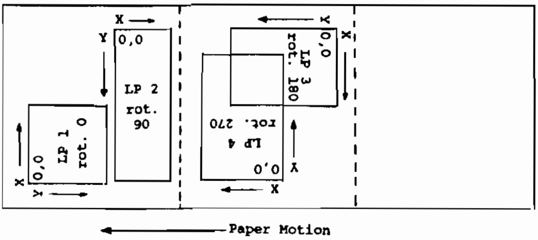


Figure 3

In the pen commands, param1 is the X axis co-ordinate or offset, while param2 is the Y co-ordinate or offset. These co-ordinates are expressed in 16 bit signed radixed integers. The radix point falls between bit 13 and 14 of the word. Since we cannot move the pen in increments of less than a single dot, bits 14 and 15 will always be 0. To try and cut through the fog, if you wanted to move one inch, at 180 dots per inch, the parameter would be set to 180 x 4 or 720.

Now we can see how to select a logical page, select a character set, and position the pen to any point on the logical page. To print text, you simply use the FWRITE intrinsic. The text will be printed on the page starting at the current pen position. The current pen position is the lower left hand corner of the first character. Typically, a file is opened to the laser with carriage control. This allows the use of the control parameter in the FWRITE intrinsic. If mostly text positioning is being done, a code of %320 is used. If only text is being printed, any of the carriage control codes defined with the FWRITE can be used to control text positioning.

When sending a page eject to the laser printer, using the FWRITE intrinsic with a control code of %61, it actually performs a logical page eject. Whether a physical page eject also occurs is dependent upon what logical page you are currently on, and how many logical pages are active. We have seen that you can activate and deactivate logical pages on command. We also know that all 32 logical pages, if defined, could be active at the same time. Logical page information is held in the laser printer in a Logical Page Table (LPT). The table has 32 entries, one for each possible logical page. When a page eject is received by the laser, it starts scanning the LPT from the current entry, looking for an active logical page. If it gets to the end of the LPT without finding one, it performs a physical page eject, and starts scanning the LPT entries from the beginning. When it finds an active LPT entry, it sets the current logical page to that entry and positions the pen to the upper left hand corner of the logical page. Of course, using the FDEVICECONTROL intrinsic with a control code of 140 insures that a physical page eject will occur.

Another method of printing data is to use the FDEVICECONTROL intrinsic with a control code of 1. This code works much the same as the standard FWRITE. When using this control code, paraml, bits 8-15 are used to determine the vertical format for printing. The codes used here are the same as %0 - %377 for a standard FWRITE with the following exceptions:

Code	Meaning
%1	Use the first data byte of the record for VFC
%61	Conditional logical page eject
%62	Physical page eject
%63	Unconditional logical page eject

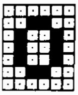
Param2 bit 14 is used for Autoeject mode. A 0 indicates set Autoeject, while a 1 indicates reset Autoeject. When Autoeject is set, if carriage control causes the pen to move off the logical page, a logical page eject is automatically done and the text is printed on the next logical page. If Autoeject is not active, no text is printed and an error is written to indicate that the pen movement was off the logical page. Bit 15 is used to determine what the spacing mode will be. A 0 indicates postspacing while a 1 indicates prespacing. Prespacing causes the carriage return line feed to occur before the printing of the text, while postspacing causes it to occur after the text is printed.

The final topic to be covered is the formatting, downloading and printing of graphic data. In order to work with graphics on the 2680A, you must understand the formats of dot-bit memory words and triplets.

DOT-BIT Memory images

A dot-bit memory image is just what the name implies, a bit map of the image to be printed. Each bit of the dot-bit image represents one dot of the actual figure to be represented. Figure 4 shows an image that is 6 dots by 7 dots. In a dot-bit image, a 1 represents no dot, and a 0 represents a black dot. This means that the dot-bit image of the graphic in figure 4 would consist of 42 bits. Since data is stored in the HP-3000 in 16 bit words, this image is packed into 16 bits per word, with any remaining bits being a 1. Looking at the dot-bit map in figure 4, this can be seen. Word 0 bits 0-5 contain the dots for row 1 of the image. Bits 6-11 contain the dots for row 2 of the image. Word 0 bits 12-15 are the first 4 dots for row 3, while Word 1 bits 0-1 contain the last 2 dots for row 3. The dot image continues with Word 2 bits 4-9 being the dots for the last row of the image. Bits 10-15 of Word 2 are set to one to fill out the word. These bits will not be used by the laser printer.

DOT-BIT MAPPING



word	Dot-bit Image	
0	1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1	%017747
1	0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1	%066667
2	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1	%037777

Figure 4

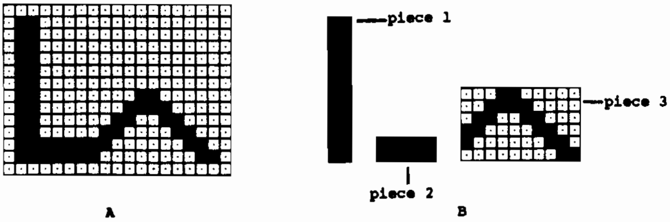
This method of image representation is referred to as a "Bit Raster", or simply a Raster, image. Since the laser resolution is 180 dots per inch, an 8 1/2 by 11 image would require a bit map of (8.5*180) * (11*180) or approximately 4 million dots. That would require 1/2 megabyte of memory for storage of a single graphic. Since the 2680A uses memory to store all character fonts, logical pages, and other process information, 1/2 megabyte storage for a single graphic is unacceptable. To download 32 graphics would require 16 megabytes of memory for storage alone.

Consider an 8 1/2 by 11 page with a single line bordering all sides. If a dot-bit map were generated for this graphic, we would see that over 99% of the dot-bit map was describing white space where no image was to be printed! Now that is a real waste of memory. In order to overcome this deficiency, "Partitioned Raster" dot-bit maps were developed. In a Partitioned Raster Dot-bit map, only those dots required to define the image are used.

Partitioned Raster Dot-bit images

Figure 5A shows a graphic that is 19 dots wide by 14 dots high. In standard raster format, this would require 266 bits, or 17 words. Figure 5B shows the same image broken into 3 pieces that can be used to describe the entire image. Piece 1 requires 24 bits, piece 2 requires 10 bits, and piece 3 requires 60 bits. The "partitioned" dot-bit map in figure 5C shows word 0 bit 0 thru word 1 bit 7 hold the image for part one. Bits 8 thru 15 are padded with ones to fill out the word. Each part of the image is treated as a single image, and the dot-bit memory image must be represented in full words. Word 2 bits 0 thru 9 hold the image for part 2, and word 3 bit 0 thru word 6 bit 11 hold the image for part 3. The entire image now requires only 7 words for storage instead of 15. That is less than half that required for a standard "Raster" dot-bit image.

PARTITIONED DOT-BIT MAPPING



Printer Image would appear as ⌞

word	Dot-Bit Image	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	%000000
1	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1	%000377
2	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1	%000077
3	1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0	%163760
4	1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0	%174634
5	1 1 1 1 0 0 1 1 0 1 1 1 1 1 1 0	%171576
6	0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1	%077717

C

Triplet Words

	piece 1	piece 2	piece 3
0	%176414	%175002	%172406
1	%000000	%000002	%000003
2	%000020	%000060	%000200

D

Figure 5

Now we have a much smaller dot-bit image of our graphic, but we need a way to tell the laser printer how to put the "pieces" together on the page. This is done by using what is known as "triplets". A triplet is a

```
                         TRIPLET WORD LAYOUT

              MSB                                      LSB
             ------------------------------------------------
     word    | 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 |
             |---------------------------------------|
      0      | - Hor Dot Cnt. | Vertical dot count    |
             |---------------------------------------|
      1      |      LSB  Dot-bit memory pointer       |
             |----------------------------           |
      2      | X Coordinate of left edge|  MSB        |
             ------------------------------------------------
```

Figure 6

group of three words describing each "piece" of the graphic. Figure 6 shows the layout of a triplet. Word 0 bits 0-7 are used to indicate the number of dots there are horizontally in this piece of the graphic. This number is the l's complement of the count. Word 0 bits 8-15 indicate the number of dots vertically in this piece of the image. We can now see that one restriction of a partitioned raster format is that each piece of the graphic cannot be any greater than 255 by 255 dots. Word 1 contains the 15 least significant bits of the memory pointer and word 2 bits 12-15 contain the 4 most significant bits of the memory pointer. The memory pointer is used to indicate which word, starting from 0, in the partitioned dot-bit map that the image for this piece of the graphic begins. This pointer is 20 bits long, because dot-bit maps for large graphics can exceed 65,000 words. This 20 bit pointer allows us to have a graphic that requires over a million words to represent. Word 1 bits 0-11 are used to indicate the position along the X axis of the entire graphic, starting from 0, that this piece of this image is to be placed.

Figure 5C shows the 3 triplets that would be used to describe the image shown in Figure 5B. Piece 1 has a horizontal count of -2 (one's complement), a vertical count of 10, the first bit describing this part of the image starts at word 0 in the bit map, and this piece of the image starts at dot 1 on the x axis of the graphic. Looking at the triplet for piece 3 of the image, its horizontal count is -10, the vertical count is 6, it starts at word 3 in the bit map, and its X coordinate is 8.

DOWNLOADING OF GRAPHIC DATA

Now that the pieces to the graphic are formatted, they need to be downloaded to the laser printer. An FDEVICECONTROL with a control code of 139 is used. This is the download/delete picture function. Paraml bit 0 is set to a 0 to delete a picture, or set to a 1 to download a picture. Param2 bits 8-15 are the picture identifier. This can be an integer number between 0 and 31, since the laser can hold 32 pictures. When downloading a picture, if a picture currently exists with the same id, it is deleted, and replaced with the one being loaded. Since picture data can span more than one record, it is necessary to be able to indicate continuation records for download. Param2 bit 0 is used to indicate if the current record is the first record of this picture, or if it is a continuation record for a picture in the process of being downloaded. Bit 0 is set to a 0 for the first record and set to a 1 for all subsequent records. All records for a picture must be downloaded without interruption. There cannot be any other commands sent to the laser printer until all of the picture has been downloaded.

Figure 7 shows the layout of picture download data. The data is divided into 3 functional parts:

1. Picture Descriptor Block
2. Triplet data information
3. Dot-bit map data

PICTURE LOAD RECORD LAYOUT

```
                 ----------------------------------------
                 |                                      |
                 |       Picture Descriptor Block       |
                 |                                      |
            ---  |--------------------------------------| ---
           /     |          Raster Line Number          |    \
          /      |--------------------------------------|     \
         /       | Number of triplet words in scan line |  triplet
                 |--------------------------------------|  group 1
                 |        All triplets for this         |     /
      All        |           Scan Line                  |    /
    triplet      |--------------------------------------| ---
     data        |      Next Raster Line Number         |    \
  information    |--------------------------------------|     \
                 | Number of triplet words in scan line |  triplet
                 |--------------------------------------|  group n
                 |        All triplets for this         |     /
         \       |           Scan Line                  |    /
          \      |--------------------------------------| ---
           \     |  All 1's  (-1)  Data separator       |
            ---  |--------------------------------------|
                 |           Dot-Bit  Memory            |
                 |             information              |
                 |--------------------------------------|
                 |  All 1's  (-1)  End of valid data *  |
                 ----------------------------------------
```

*added by the spooler

Figure 7

The first part encountered is the picture descriptor block. This data block is used to describe the characteristics of the picture being loaded. This will be covered in detail later. The next part contains all of the triplets, in groups, that are needed to describe all pieces of the picture. Triplets are grouped together according to their starting location in the picture. The first word in a triplet group represents the raster scan line where the following triplets will begin. Recall, that when the picture was broken into pieces, each piece had a triplet describing it. The triplet described the size of the picture piece, horizontally and vertically, where the data for that piece could be found in the dot-bit map, and the X co-ordinate of the left edge of that piece in the entire picture. The only thing missing that would be needed for proper placement of the piece in the entire picture was the Y co-ordinate. The Y co-ordinate is referred to by raster scan line number, so the raster scan line number for a particular triplet would be the Y co-ordinate, starting from 0, of the top line of this piece in the picture. The next word in the triplet group indicates how many triplet words there are for this scan line. If there are 2 triplets for picture pieces starting on this scan line, then this number will be 6 (3 words per triplet, 2 triplets). Following this are all of the triplets that start on this scan line. Due to limitations of the laser printer, the maximum number of triplets, or pieces of the picture, that can start on a single scan line is 255.

Following this triplet group would be another triplet group for the next raster scan line. The triplet groups are loaded in order by ascending Y co-ordinate (raster scan line number), starting from the top of the picture. Since the Y co-ordinate starts at 0 at the top of the picture, and goes thru some Y co-ordinate N, the triplet group for raster scan line 0 will be the first group and the group for raster scan line N will be the last group. If there are no pieces of the picture that start on a given raster scan line, then there is no triplet group required for that scan line. After the triplet data information there is a -1 (all ones) data separator. This is used to signify the end of the picture description, and the beginning of the dot-bit memory map for the picture. Following this data separator, is the dot-bit memory map for the entire picture.

PICTURE DESCRIPTOR BLOCK LAYOUT

```
            MSB                                 LSB
           -----------------------------------------
word       | 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 |
           |---------------------------------------|
 0         |      not used              | MSB      |
           |----------------------------           |
 1         |  Number of Triplet words in Picture   |
           |---------------------------------------|
 2         |      not used              |  MSB     |
           |----------------------------           |
 3         |  Number of Dot-bit memory words       |
           |---------------------------------------|
 4         | Offset to left edge of Picture        |
           |---------------------------------------|
 5         | Offset to right edge of Picture       |
           |---------------------------------------|
 6         | Offset to top edge of Picture         |
           |---------------------------------------|
 7         | Offset to bottom edge of Picture      |
           -----------------------------------------
```

Figure 8

The picture descriptor block is 8 words long and is used to describe overall information about the picture being downloaded. Figure 8 shows the layout of the picture descriptor block. Word 0 bit 12 thru word 1 bit 15 is a 20 bit integer which indicate the total number of triplet data information words in the download record. This includes all triplets, all scan line numbers, and also the data separator as shown in Figure 7. Word 2 bit 12 thru word 3 bit 15 is a 20 bit integer indicating the number of dot-bit memory map word used for this entire picture. Words 4 thru 7 are used to describe the location of the picture origin. The picture origin is used to determine where to place the picture on a page. When a picture is printed on the laser printer, it is positioned with the origin at the current pen position. In most cases, the origin is placed in the upper left hand corner of the picture. This allows the corner of the picture to fall at the current pen position.

Figure 9 shows the data buffer that would be used to download the graphic shown in figure 5A. This image would be downloaded using the FDEVICECONTROL command with a control code of 139, paraml set to a 0, and param2 set to a 2. Buffer length for the download is 29 words. At the completion of the command, this picture would be stored as picture 2 in the laser printer.

To print the picture, an FDEVICECONTROL command with a control code of 144 is used. Paraml bit 0 is used set to a 1 to indicate an addressable picture, or set to a 0 to indicate a temporary picture. An addressable picture is one which is downloaded with a control code of 139. A temporary picture is one which is downloaded with this command, is printed, and then deleted from memory. Temporary pictures will be covered later. Paraml bit 1 is set to a 0 to indicate that the picture is to be printed with its origin at the current pen position. It is set to a 1 to indicate that the origin is relative to position 0,0 on the logical page. Param2 bit 0 is a 0 for permanent pictures, and bits 8-15 are used to identify the picture. If we wanted to print the picture just loaded at the current pen position, then parameters 1 and 2 would be %100000 and %000002 respectively.

TEMPORARY PICTURES

When a picture is to be printed only once, it is better to download the image as a temporary picture. This prevents the memory in the laser printer from being used to hold images that are printed only once. It also

```
                 PICTURE DOWNLOAD RECORD for
                   PICTURE in FIGURE 5A
                  (Permanent Addressable)

              word   buffer data

       ----    0     %000000        # triplets
      /        1     %000016          in description
     /
    /          2     %000000        # dot-bit memory
 Picture       3     %000007          map words
descriptor
 block         4     %000000          offset to left
   \           5     %000016          offset to right
    \          6     %000000          offset to top
     ----      7     %000023          offset to bottom

     ----      8     %000001          scan line 1    ----
    /          9     %000006          6 triplet words    \
   /          10     %176414          -                   \
 triplet      11     %000000          triplet 1            \
 group 1      12     %000020          -                     \
   \          13     %175002          -
    \         14     %000002          triplet 2          triplet
     ----     15     %000040          -                   data
                                                          infor-
     ----     16     %000006          scan line 6         mation
    /         17     %000003          3 triplet words
 triplet      18     %172406          -                      /
 group 2      19     %000003          triplet word 3        /
    ----      20     %000200          -                    /
                                                          /
              21     %177777          data seperator ----

              22     %000000
              23     %000377
              24     %000077          dot-bit memory
              25     %163760          map
              26     %174634
              27     %171576
              28     %077717
```

Figure 9

keeps the laser printer memory free to hold 31 permanent images. (One of the 32 image ids must remain free for the download of the temporary picture, but the picture is deleted after printing.) This means, that if many images are to be used only once, there is no limit to the number of pictures that can be printed during a logical job. A temporary picture is actually downloaded with the print picture command, control code 144. When downloading a temporary picture, the buffer is formatted the same as with a permanent picture except the picture descriptor block is preceded by 2 extra words, as shown in Figure 10. The first word is an X co-ordinate, and the second word is a Y co-ordinate for placement of the temporary picture. The co-ordinates are expressed in radixed integer format, the same as those used for pen movement. When downloading a temporary picture with a control code of 144, param1 bit 1 is used to indicate whether the X and Y co-ordinates preceding the picture are absolute pen positions on the logical page or relative to the current pen position. A 0 indicates they are relative, while a 1 indicates that they are absolute pen positions. As with the download picture command, control code 139, download information can span more that one record. Param2 bit 0 is used to indicate the continuation of a picture download, just as it was with the control code of 139.

```
                    PICTURE DOWNLOAD RECORD for
                      PICTURE in FIGURE 5A
                          (Temporary)

                 word   buffer data

                  0     %000100          X coordinate (radixed)

                  1     %000100          Y coordinate (radixed)

          ----    2     %000000        # triplets
         /        3     %000016          in description
        /
       /          4     %000000        # dot-bit memory
   Picture        5     %000007          map words
 descriptor
   block          6     %000000          offset to left
      \           7     %000016          offset to right
       \          8     %000000          offset to top
          ----    9     %000023          offset to bottom

          ----   10     %000001          scan line 1   ----
         /       11     %000006          6 triplet words   \
        /        12     %176414          -                  \
   triplet       13     %000000          triplet 1           \
   group 1       14     %000020          -                    \
      \          15     %175002          -
       \         16     %000002          triplet 2          triplet
          ----   17     %000040          -                   data
                                                             infor-
          ----   18     %000006          scan line 6         mation
         /       19     %000003          3 triplet words
   triplet       20     %172406          -                    /
   group 2       21     %000003          triplet word 3      /
         ----    22     %000200          -                  /
                                                           /
                 23     %177777          data seperator ----

                 24     %000000
                 25     %000377
                 26     %000077          dot-bit memory
                 27     %163760          map
                 28     %174634
                 29     %171576
                 30     %077717
```

Figure 10

EXAMPLE

Figure 11 shows a sample program, written in pseudocode, to download the picture described in figure 5. It is downloaded using a pictue ID of 2, the pen is positioned and the picture is then printed. Next, a temporary copy of the same picture is printed. Notice that a picture ID is used to download the temporary picture. If a picture was currently using that picture ID, it is first deleted, and then the temporary picture is downloaded. Once the picture is downloaded, it is printed and deleted. It is important to note that the temporary picture is not truly deleted until a physical page eject occurs. Therefore, if more than one temporary picture is to be downloaded on a physical page, they must all have different picture IDs. Once a physical page eject occurs, the picture IDs of all the temporary pictures may be reused. Figure 12 shows the resulting output from the program. For clarity, the size of the image was increased. The text at the bottom of the second picture was printed using the FDEVICECONTROL 131 and FWRITE intrinsics.

EXAMPLE of PICTURE
DOWNLOAD and PRINT

The following is a pseudocode example of how to download and print the image shown in Figure 5A. All buffer offsets start at 1. This example shows both a temporary and permanent addressable picture load and print.

```
     Buffert = Logical 31 word buffer initially loaded with the
               temporary picture data in Figure 10

     Bufferp = Logical 29 word buffer equated to Buffert(3), which
               would make it the same as the buffer in Figure 9

     Textbuffer = a logical buffer used for text data

     Laser = "*Laser "

***** Open the spoolfile to the laser as ASCII,NEW,CCTL,WRITE ONLY
      Laserout = FOPEN(Laser,%404,%1)

***** Download the permanent picture with ID 2
      FDEVICECONTROL(Laserout,Bufferp,29,139,%0,%2)

***** If there was a continuation record, the buffer and count would be
***** changed and param2 would be set to %100002

***** Move the pen to an absolute location
      X = %002640     **** 2 inches (180 dots per inch) radixed
      Y = %001320     **** 1 inch   (180 dots per inch) radixed
      FDEVICECONTROL(Laserout,Notused,Notused,131,X,Y)

***** Print the permanent picture at the current pen position

      FDEVICECONTROL(Laserout,Notused,Notused,144,%100000,%2)
***** The picture is now printed with the origin at X = 2 inches
***** and Y = 1 inch.

***** Now print the picture as a temporary picture using the
***** X and Y in the download file to position the orgin at
***** absolute  X = 4 inches and Y = 4 inches
      Buffert(1)=%005500
      Buffert(2)=%005500
      FDEVICECONTROL(Laserout,Buffert,31,144,%040000,%3)
***** The laser will temporarily use Picture ID 3 to download
***** and print the picture. Once the picture is printed, it
***** is deleted.

***** Move the pen under the picture and print text

      Move "This is the figure title" to Textbuffer
      FDEVICECONTROL(Laserout,Notused,Notused,131,%5500,%5764)

***** The pen is now 1 inch under the origin of the tempory picture

      FWRITE(Laserout,textbuffer,-24,%320)

      FCLOSE(Laserout,1,0)

      END of pseudocode
```

Figure 11

CONDENSING DOT-BIT-MAPS

Since triplets are used to point to a location in the dot-bit map where a portion of a picture begins, if there are several parts of the picture with identical data, the same locations in the dot-bit map can be used. A very good example is shown in Figure 13. All pieces of the graphic can be described as an all black image. If we describe the largest black area in the image, in this case piece 2, we see that it takes 32 bits to describe. All of the other pieces of the image are smaller, and can be described using the same dot-bit map area. All of the triplets in figure 13 point to word 0 for the dot-bit map data. The only difference is the horizontal and vertical counts, as well as the X coordinate.

Example Program Results

Please note that the picture size has been increased for clarity

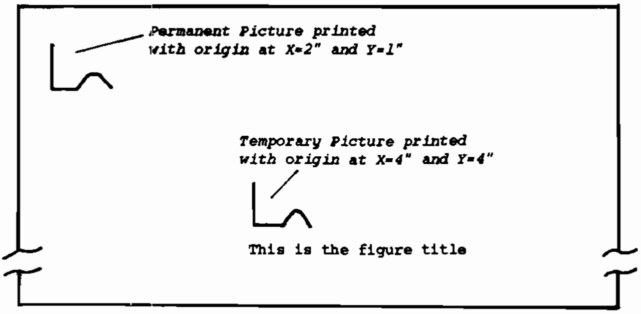
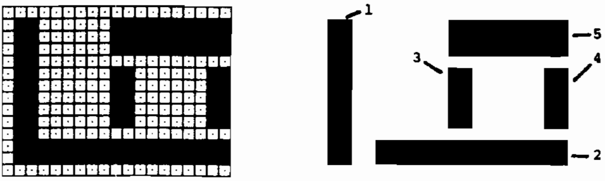

Permanent Picture printed
with origin at X=2" and Y=1"
Temporary Picture printed
with origin at X=4" and Y=4"
This is the figure title

Figure 12

CONSERVING DOT-BIT MAP SPACE



```
word       Dot-Bit Image

  0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     %000000
  1    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     %000000
```

Triplet Words

	1	2	3	4	5
0	%176414	%175002	%176405	%176405	%172403
1	%000000	%000000	%000000	%000000	%000000
2	%000020	%000060	%000220	%000420	%000220

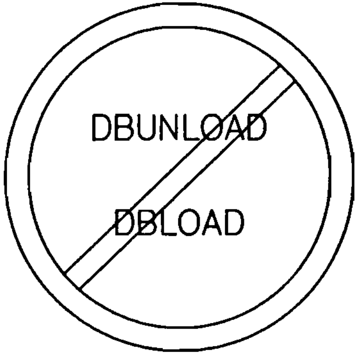
Figure 13

BIOGRAPHY

Richard Oxford has been involved with computer hardware and software for the past eleven years. His experience includes teaching computer hardware, developing software, and maintenance of hardware and software on various computer systems. He has worked with HP computers for the past seven years, and with the 2680A laser for four years. Presently he is with MCI Digital Information Services as system manager for their network of HP-3000s and laser printers.

TurboIMAGE DBchange

A restructuring utility for TurboIMAGE databases

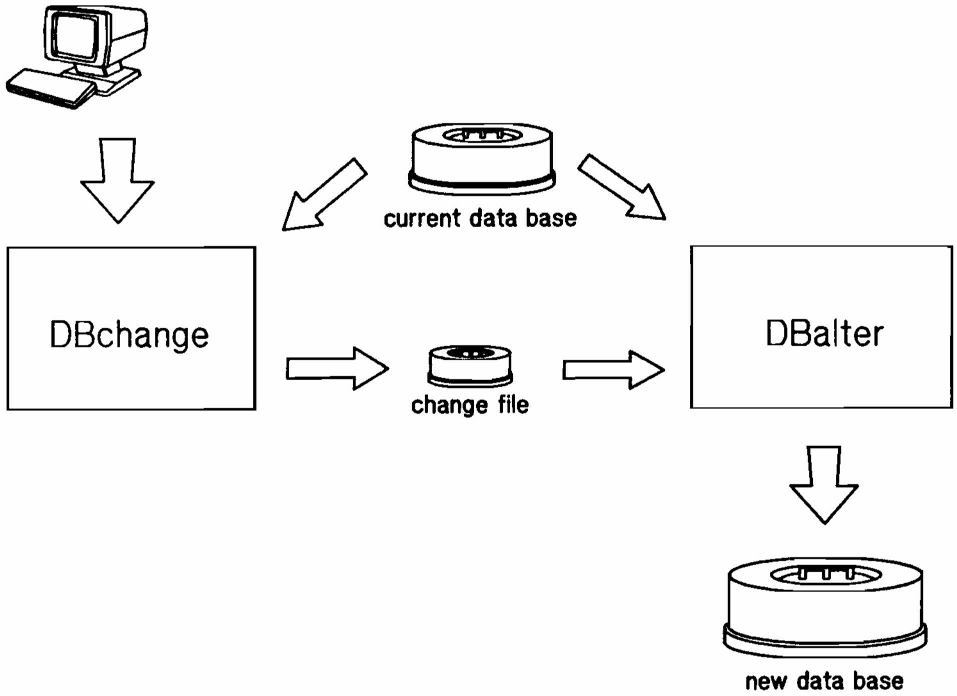


DBUNLOAD

DBLOAD

HEWLETT PACKARD

DBchange



DBchange

current data base

change file

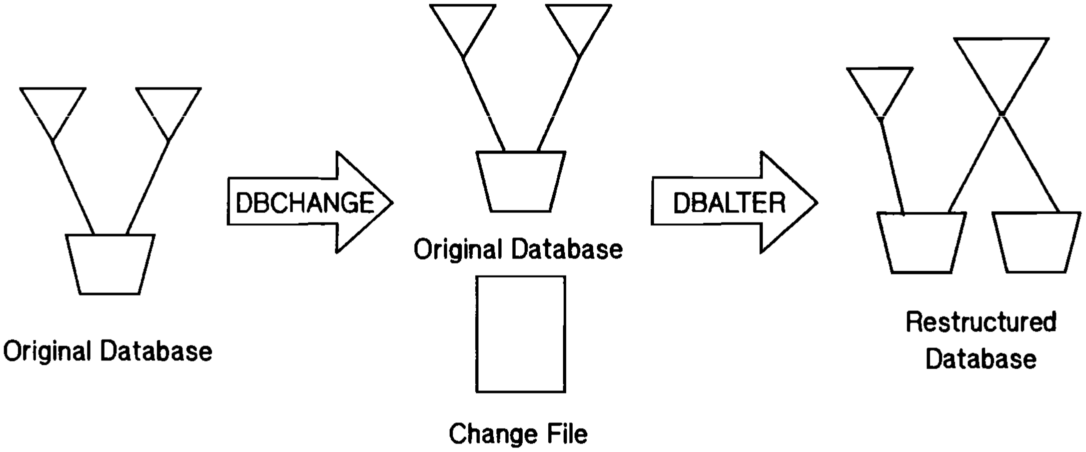
DBalter

new data base

DBC0106

HEWLETT PACKARD

DBchange



- * DBCHANGE creates a change file containing all database modifications
- * DBALTER actually does the restructuring
- * DBALTER can either be invoked through DBCHANGE or run separately

HEWLETT PACKARD

DBchange: Capabilities

ADD

- Passwords
- Data Items
- Data Sets
- Paths
- Data Set Fields
- Sort Items

CHANGE

- Passwords
- Data Items
- Data Sets
- Paths
- Data Set Field Order
- Sort Items
- Data Set & Item Access
- Data Item Schema Order
- Data Set Schema Order
- Capacity
- Blocking Factor
- Device Class
- Data base, Set, Item Names

DELETE

- Passwords
- Data Items
- Data Sets
- Paths
- Data Set Fields
- Sort Items

DBchange: Capabilities (Continued)

REVIEW

Data Sets
Data Items
Paths
Data Set Fields

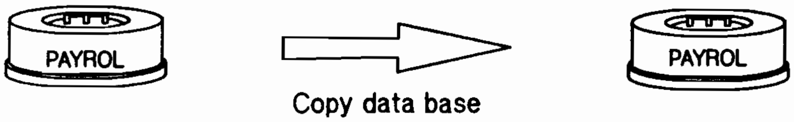
COPY

Data Bases

PRINT

Schemas

DBchange: Copy a data base



Creator (DORIS.ADM,PAYDEPT)

- * Run DBChange to grant copy access to BOSS.COMPANY,PLANNING
- * Release the change file
- * Release the data base

Copier (BOSS.COMPANY,PLANNING)

- * Run DBalter to copy the data base

DBchange: Item Type Conversion

1. I1, I2, J1, J2, K1, P, Z → U, X

2. I1, I2, J1, J2, K1 ⇄ P, Z, R2
   I1, I2, J1, J2, K1

3. X → U

5. U → X

AN INTRODUCTION TO HEWLETT-PACKARD'S IMPLEMENTATION OF SQL

Lynn Barnes
Hewlett-Packard Company, Rockville Maryland, USA

Summary

SQL is the language used for database operations with Hewlett-Packard's new relational database, HPSQL. This paper describes what SQL is and its capabilities with particular emphasis on data manipulation and data definition. Topics covered include:

- Definition of relational terms
- Who uses SQL
- How SQL is used (including examples)
- SQL capabilities
  - data retrieval operations
  - data change operations
  - multiple-row manipulation
  - transaction management
  - data definition
  - authorization definition
  - database maintenance and management
  - application programming

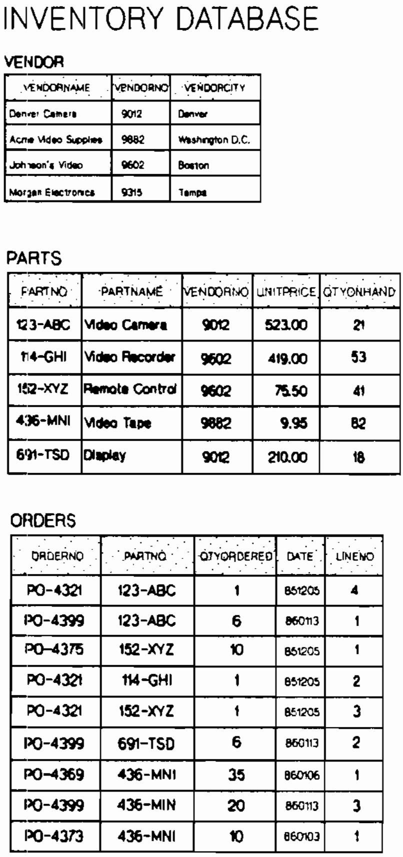
Introduction

Throughout the computer industry, there are several relational database management systems (DBMS), each with its own language for database definition and access. SQL (Structured Query Language) is the language that is emerging as the de facto industry standard for relational data bases.

Hewlett-Packard Company has chosen SQL as the language to be used for database operations with its relational database management system called HPSQL. HP's implementation of SQL is similar to other implementations in the industry, and HP is currently represented on the committee that is defining an ANSI standard for SQL.

SQL is an easy-to-learn, English-like language that may be used both interactively or programmatically. SQL is completely command-driven and performs a broad range of functions from data access to data definition to database maintenance. For these reasons, the same language is used by programmers, DBA (database administrator), and ad hoc query users.

An HPSQL database logically consists of tables which appear to be like flat MPE files, but actually are logical views of the available data. The INVENTORY database illustrated in Figure 1 is the sample database which will be used throughout this paper. This database consists of three tables: VENDOR, PARTS, and ORDERS. All access to the information in this database is done through these tables or through *views* (filters) which are derived from these tables.



INVENTORY DATABASE

VENDOR

VENDORNAME	VENDORNO	VENDORCITY
Denver Camera	9012	Denver
Acme Video Supplies	9882	Washington D.C.
Johnson's Video	9602	Boston
Morgan Electronics	9315	Tampa

PARTS

PARTNO	PARTNAME	VENDORNO	UNITPRICE	QTYONHAND
123-ABC	Video Camera	9012	523.00	21
114-GHI	Video Recorder	9602	419.00	53
152-XYZ	Remote Control	9602	75.50	41
436-MNI	Video Tape	9882	9.95	82
691-TSD	Display	9012	210.00	18

ORDERS

ORDERNO	PARTNO	QTYORDERED	DATE	LINENO
PO-4321	123-ABC	1	851205	4
PO-4399	123-ABC	6	860113	1
PO-4375	152-XYZ	10	851205	1
PO-4321	114-GHI	1	851205	2
PO-4321	152-XYZ	1	851205	3
PO-4399	691-TSD	6	860113	2
PO-4369	436-MNI	35	860106	1
PO-4399	436-MIN	20	860113	3
PO-4373	436-MNI	10	860103	1

Figure 1.

Since all access to data is through these "logical" tables, there is no need for the user to actually know how the data is physically stored on disc. Instead of intrinsic calls, the programmer and interactive user use SQL commands to retrieve and manipulate data. The SQL commands allow the user to specify **what** data he or she wants and not **how** to get it. The actual data access is done on the user's behalf by the DBMS.

SQL commands may be entered programmatically by embedding the SQL commands in a COBOL or Pascal program. Before compiling the program, the source is run through an HPSQL preprocessor which checks the syntax of SQL commands, comments them out, and inserts procedure calls. The modified source can then be compiled and prepared using the standard COBOL or Pascal compiler and segmenter.

These same commands, with the exception of a few application program dependent commands, may also be entered interactively using the program ISQL. ISQL is a utility program which comes standard with HPSQL. It accepts SQL commands from a user at a terminal and executes the commands on the user's behalf. ISQL is particularly useful for ad hoc queries, database maintenance and management, and program development and debugging.

As shown in Figure 2, all SQL commands, regardless of their origin, are processed by HPSQLCORE. HPSQLCORE checks command syntax and performs functions related to accessing and safeguarding data. The actual database access is done by DBCORE.

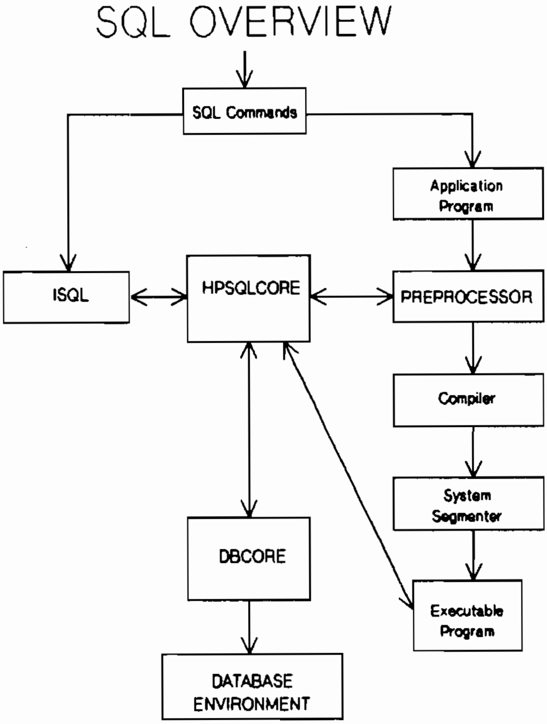


Figure 2.

As previously mentioned, SQL is a multifunctional language. There are commands for data retrieval operations, data change operations, data definition, authorization definition, database maintenance and management, and some special commands for application programming.

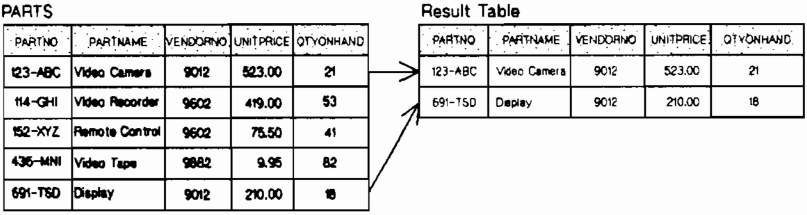
Data Retrieval Operations

Data retrieval operations can be broken into three primary operations: *selection*, *projection*, and *join*. All three of the above operations are accomplished with the SELECT command. The user of this command must have SELECT authority on the table or view from which the data is to be extracted.

A *selection* is the simplest of all retrieval operations. It is the retrieval of a subset of rows that satisfy certain criteria. For example, in the INVENTORY database, if we want to see all rows in the PARTS table whose vendor number equals "9012", we would use the following SELECT command:

```
SELECT *
FROM PARTS
WHERE VENDORNO = '9012'
```

The output from this selection is the result table shown below.



PARTS

PARTNO	PARTNAME	VENDORNO	UNITPRICE	QTYONHAND
123-ABC	Video Camera	9012	523.00	21
114-GHI	Video Recorder	9602	419.00	53
152-XYZ	Remote Control	9602	75.50	41
436-MNI	Video Tape	9882	9.95	82
691-TSD	Display	9012	210.00	18

Result Table

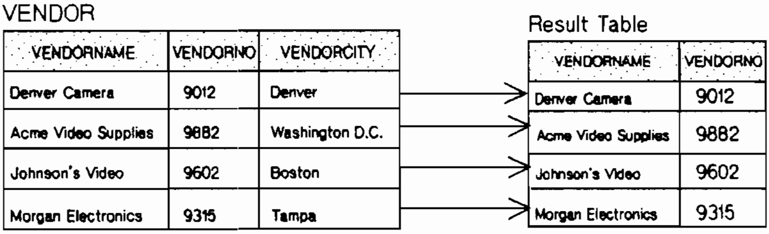
PARTNO	PARTNAME	VENDORNO	UNITPRICE	QTYONHAND
123-ABC	Video Camera	9012	523.00	21
691-TSD	Display	9012	210.00	18

Figure 3.

In the above example, we were able to choose which rows in the PARTS table we wanted to see. If, on the other hand, we wanted to retrieve all the rows in a table but only wanted to see certain columns, the operation would be a *projection* and might look like the following:

```
SELECT VENDORNAME, VENDORNO
FROM VENDOR
```

Here we are asking to retrieve only the vendor name and city for all rows in the VENDOR table. The result table for this command is shown below.



VENDOR

VENDORNAME	VENDORNO	VENDORCITY
Denver Camera	9012	Denver
Acme Video Supplies	9882	Washington D.C.
Johnson's Video	9602	Boston
Morgan Electronics	9315	Tampa

Result Table

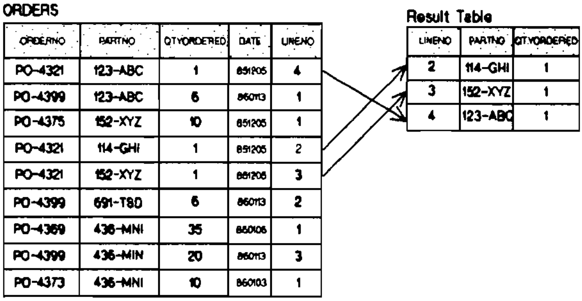
VENDORNAME	VENDORNO
Denver Camera	9012
Acme Video Supplies	9882
Johnson's Video	9602
Morgan Electronics	9315

Figure 4.

By combining the *selection* and *projection*, we can further narrow down our window of selected data.

```
SELECT LINENO, PARTNO, QTYORDERED
FROM ORDERS
WHERE ORDERNO = 'PO-4321'
ORDERED BY LINENO ASC
```

In this example, we are asking only for the columns containing line number, part number, and quantity ordered from the ORDERS table whose order number equals 'PO-4321'. In addition, we are asking for the data to be returned to us in ascending order by line number. The following is the result table for the above command:



ORDERS

ORDERNO	PARTNO	QTYORDERED	DATE	LINENO
PO-4321	123-ABC	1	851205	4
PO-4399	123-ABC	6	860113	1
PO-4375	152-XYZ	10	851205	1
PO-4321	114-GHI	1	851205	2
PO-4321	152-XYZ	1	851206	3
PO-4399	691-TSD	6	860113	2
PO-4369	436-MNI	35	860106	1
PO-4399	436-MIN	20	860113	3
PO-4373	436-MNI	10	860103	1

Result Table

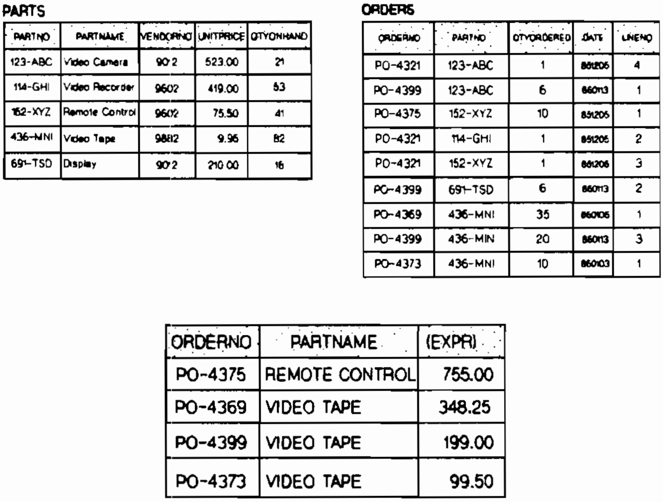
LINENO	PARTNO	QTYORDERED
2	114-GHI	1
3	152-XYZ	1
4	123-ABC	1

Figure 5.

The previous examples illustrated data retrieval from a single table. It is very likely, however, that it would be desirable to retrieve data from more than one table with one SQL command. This type of retrieval is call a *join*. An example of a join is shown below.

```
SELECT ORDERNO, PARTNAME, QTYORDERED*UNITPRICE
FROM PARTS, ORDERS
WHERE QTYORDERED >= 10
AND ORDERS.PARTNO = PARTS.PARTNO
```

In the above example, we are asking for the order number from the ORDERS table, the part name from the PARTS table, and we are asking that the total cost be calculated for us by multiplying the quantity ordered from the ORDERS table by the unit price in the PARTS table. Part number, which is common to both the PARTS and ORDERS tables, is used as the link to determine which rows in the respective tables will be used together. Figure 6 illustrates the result from this join.



PARTS

PARTNO	PARTNAME	VENDORNO	UNITPRICE	QTYONHAND
123-ABC	Video Camera	9012	523.00	21
114-GHI	Video Recorder	9602	419.00	53
152-XYZ	Remote Control	9602	75.50	41
436-MNI	Video Tape	9882	9.95	82
691-TSD	Display	9012	210.00	16

ORDERS

ORDERNO	PARTNO	QTYORDERED	DATE	LINENO
PO-4321	123-ABC	1	851206	4
PO-4399	123-ABC	6	860113	1
PO-4375	152-XYZ	10	851205	1
PO-4321	114-GHI	1	851205	2
PO-4321	152-XYZ	1	851206	3
PO-4399	691-TSD	6	860113	2
PO-4369	436-MNI	35	860106	1
PO-4399	436-MIN	20	860113	3
PO-4373	436-MNI	10	860103	1

ORDERNO	PARTNAME	(EXPR)
PO-4375	REMOTE CONTROL	755.00
PO-4369	VIDEO TAPE	348.25
PO-4399	VIDEO TAPE	199.00
PO-4373	VIDEO TAPE	99.50

Figure 6.

As shown in Figure 6, SQL can do calculations on data retrieved. In addition to evaluating an arithmetic expression during data retrieval, SQL also allows the user to request that an aggregate function be performed against the data selected. The built-in aggregate functions are:

AVG - calculate the average
MAX - display the maximum value
MIN - display the minimum value
SUM - calculate the sum
COUNT - count the number of occurrences

In all the examples shown so far, the search condition was based on a comparison, e.g., WHERE QTYORDERED >= 10. In addition to comparisons, the search condition may include a BETWEEN condition where only the rows which have a value between two given values will be selected. The search condition may also include a LIKE condition which will cause a partial match to be done on the specified column. Or the search condition may be for only those rows which have a NULL value in a particular column or NOT NULL (column must contain data).

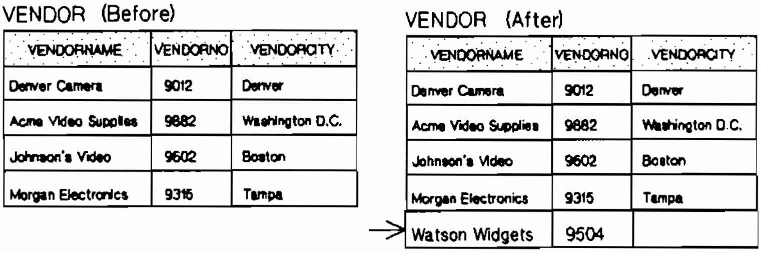
Data Change Operations

SQL has three commands which accomplish data change operations: INSERT, DELETE, and UPDATE. As with the SELECT command, the user must have INSERT, DELETE, and UPDATE authority respectively to execute the aforementioned commands.

The INSERT command is used to add a new row into a single table. The user specifies the columns in the row and their values. Note in the example below that all columns need not be included in the command; any column defined as NOT NULL, however, must be included in the list.

```
INSERT INTO VENDOR(VENDORNAME, VENDORNO)
VALUES ('WATSON WIDGETS', '9504')
```

In this example, we are inserting a row into the VENDOR table with vendor name = WATSON WIDGETS and vendor number = 9504. Figure 7 shows VENDOR table before and after the INSERT.



VENDOR (Before)

VENDORNAME	VENDORNO	VENDORCITY
Denver Camera	9012	Denver
Acme Video Supplies	9882	Washington D.C.
Johnson's Video	9602	Boston
Morgan Electronics	9315	Tampa

VENDOR (After)

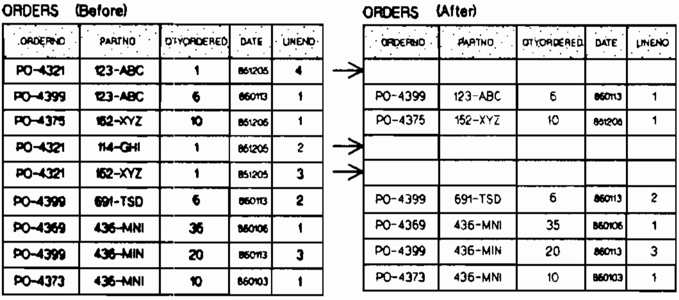
VENDORNAME	VENDORNO	VENDORCITY
Denver Camera	9012	Denver
Acme Video Supplies	9882	Washington D.C.
Johnson's Video	9602	Boston
Morgan Electronics	9315	Tampa
Watson Widgets	9504	

Figure 7.

To delete one or more rows from a table, the DELETE command is used. This command will delete all rows in the table which meet the search condition.

```
DELETE FROM ORDERS
WHERE ORDERNO = 'PO-4321'
```

In the sample database, the above command would delete three rows as indicated in Figure 8.



ORDERS (Before)

ORDERNO	PARTNO	QTYORDERED	DATE	LINENO
PO-4321	123-ABC	1	851205	4
PO-4399	123-ABC	6	860113	1
PO-4375	152-XYZ	10	851206	1
PO-4321	114-GHI	1	861205	2
PO-4321	152-XYZ	1	851205	3
PO-4399	691-TSD	6	860113	2
PO-4369	436-MNI	35	860106	1
PO-4399	436-MIN	20	860113	3
PO-4373	436-MNI	10	860103	1

ORDERS (After)

ORDERNO	PARTNO	QTYORDERED	DATE	LINENO
PO-4399	123-ABC	6	860113	1
PO-4375	152-XYZ	10	851206	1
PO-4399	691-TSD	6	860113	2
PO-4369	436-MNI	35	860106	1
PO-4399	436-MIN	20	860113	3
PO-4373	436-MNI	10	860103	1

Figure 8.

The INSERT and DELETE commands are used to insert and delete entire rows in a single table. To change the values of columns in existing rows in a single table, the UPDATE command is used. There are no restrictions as to which columns may be updated, i.e., even if a column is used as a key it may still be updated with the UPDATE command. In the following example, we wish to lower the unit price by 25% for all parts in the PARTS table which have a quantity on hand between the values of 20 and 50.

```
UPDATE PARTS
SET UNITPRICE = UNITPRICE * .75
WHERE QTYONHAND BETWEEN 20 AND 50
```

The new table is shown to the right:

PARTS (Before)

PARTNO	PARTNAME	VENDORNO	UNITPRICE	QTYONHAND
123-ABC	Video Camera	9012	523.00	21
114-GHI	Video Recorder	9602	419.00	63
152-XYZ	Remote Control	9602	75.60	41
436-MNI	Video Tape	9882	9.95	82
591-TSD	Display	9012	210.00	18

PARTS (After)

PARTNO	PARTNAME	VENDORNO	UNITPRICE	QTYONHAND
123-ABC	Video Camera	9012	392.25	21
114-GHI	Video Recorder	9602	419.00	53
152-XYZ	Remote Control	9602	56.63	41
436-MNI	Video Tape	9882	9.95	82
591-TSD	Display	9012	210.00	18

Figure 9,

Multiple-Row Manipulations

Programmatic data manipulation can only operate on a single row at a time unless a cursor or the BULK option is used. That is, data retrieval and insertion by a program can only occur a row at a time unless one of the above options is used (DELETE and UPDATE can be applied to multiple rows without any special options).

A *cursor* is a pointer that allows you to advance one row at a time through a set of rows retrieved with a SELECT command. A cursor is declared and associated with a SELECT command. Once the rows are selected, the FETCH command is used to retrieve the current row pointed to by the cursor.

The BULK option is used to retrieve or insert multiple rows with a single execution of the SELECT, FETCH, or INSERT commands. When using the BULK option, an array must be used to hold the rows to be retrieved or inserted. Using BULK access improves performance by reducing disc I/O and access time.

Transaction Management

A transaction can be defined as a unit of meaningful work. There are several SQL commands which may be used for transaction management. With SQL, a transaction is delimited by BEGIN WORK and COMMIT WORK commands.

All database changes are logged to a log file until a COMMIT WORK command is issued. At that time, all changes made since the last BEGIN WORK are posted against the database. Up to this point, a user may use the ROLLBACK WORK command to undo changes that have not yet been committed.

The SAVEPOINT command may be used to define save points within a transaction and can be used with the ROLLBACK WORK command to undo part of a transaction. The following is an example of transaction management:

```
BEGIN WORK
    SQL Command 1
    SQL Command 2
SAVEPOINT
  save point number is 1
    SQL Command 3
    SQL Command 4
    SQL Command 5
SAVEPOINT
  save point number is 2
    SQL Command 6
IF error THEN ROLLBACK WORK TO 1
COMMIT WORK
```

In this example, if an error occurs, only **SQL Command 1** and **SQL Command 2** will be posted against the database. If no error occurs, then all six SQL commands will be posted against the database.

Data Definition

Data definition for a database is also accomplished using SQL commands. The CREATE and DROP commands can be used to add and delete *tables*, *views*, and *indexes* in the database. In addition, the ADD command can be used to add columns to the end of a row in a table. All three of these commands (CREATE, DROP, and ADD) can be issued while users are accessing the database.

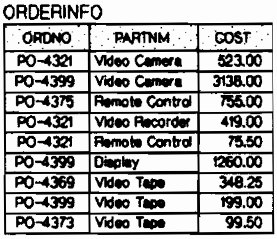
The following command was used to create the PARTS table:

```
CREATE TABLE PARTS
(PARTNO     CHAR(10) NOT NULL,
 PARTNAME   VARCHAR(30),
 VENDORNO   CHAR(4),
 UNITPRICE  DECIMAL(8,2),
 QTYONHAND  SMALLINT)
```

For security reasons or for convenience, we might want to create a *view* derived from columns from the PARTS and ORDERS tables. An example of the command to create such a view follows:

```
CREATE VIEW ORDERINFO(ORDNO, PARTNM, COST)
AS SELECT ORDERNO, PARTNAME, QTYORDERED*UNITPRICE
FROM ORDERS, PARTS
WHERE ORDERS.PARTNO = PARTS.PARTNO
```

The following shows the resulting view:



ORDERINFO

ORDNO	PARTNM	COST
PO-4321	Video Camera	523.00
PO-4399	Video Camera	3138.00
PO-4375	Remote Control	755.00
PO-4321	Video Recorder	419.00
PO-4321	Remote Control	75.50
PO-4399	Display	1260.00
PO-4369	Video Tape	348.25
PO-4399	Video Tape	199.00
PO-4373	Video Tape	99.50

Figure 10.

Indexes may also be added to a table to improve performance. The CREATE INDEX command can be used to create an index that contains a B-tree type structure of key values and pointers to rows in a data table. Up to 15 columns may be concatenated to create one index. Below is an example of adding an index for part number in the PARTS table:

```
CREATE INDEX PARTINDEX
ON PARTS(PARTNO)
```

As with the data manipulation commands, data definition commands require the proper authorization to be issued.

Authorization Definition

Most SQL commands require users to have some type of authority granted to them before they can execute the commands successfully. The different types of authority are:

select - authority to retrieve data
insert - authority to insert rows
delete - authority to delete rows
update - authority to change data in existing rows

alter - authority to add new columns to a table
index - authority to create and drop indexes on a table
run - authority to execute a specified SQL module
connect - authority to start a session in a database environment
resource - authority to create tables and authorization groups
DBA - authority to use any valid SQL command
owner - owning a table, view, module, or authorization group

Authorities may be granted to individual users or authorization groups. An authorization group is a named collection of users or other groups. For instance, all the users in the accounting department could be added to an authorization group called *Accounting*. Authority can be granted to the group *Accounting* with one GRANT command giving all members of the group the specified authority.

There are SQL commands available to GRANT and REVOKE authorities, TRANSFER OWNERSHIP of a database object (table, view, etc.), and CREATE and maintain authorization group. All of these commands require *DBA*, *owner*, or *resource* authority.

Database Maintenance and Management

Database maintenance and management are usually jobs for the DBA. SQL provides a variety of commands to allow the DBA to accomplish these tasks.

It is the job of the DBA to control access to the database environment. With the START DBE NEW command, the DBA can define the start-up parameters for the database environment, such as multi-user mode or single-user mode, maximum number of concurrent transactions, logging information, etc. In addition to the START DBE NEW command, the DBA can use the GRANT command to control what users are allowed to CONNECT to or use the environment.

The DBA is also responsible for creating and controlling the physical database environment. Physically, the database environment is made up of DBEFILES (MPE files) assigned-to DBEFILESETS in the DBENVIRONMENT. It is up to the DBA to create these entities and allocate the DBEFILES to the appropriate DBEFILESET.

Logging is always activated and maintained by HPSQL when an HPSQL database is in use. The DBA can, however, control the size of the log buffers, whether dual logging is to take place, the names of the log files, and whether HPSQL should do rollforward or rollbackward logging. If rollforward logging is used, there are additional SQL commands for the recovery process.

Statistics are maintained by HPSQL which contain up-to-date information on the database environment, structure, and database use. SQL provides the DBA with commands to update these statistics and reset set them when desired.

Application Programming

As previously mentioned, there are certain SQL commands that are only appropriate for application programs. These commands include commands for BULK access and cursor management, preprocessor directives, and dynamic preprocessing.

Dynamic preprocessing allows an application program to preprocess and execute SQL commands at run time that are not known at programming time. The application program accepts the command at run time and stores it in a program variable. With the use of SQL dynamic preprocessing commands, the stored command is processed and stored as a module, executed, and erased after execution.

Conclusion

As we have seen from the examples, SQL is an easy-to-use language. Its English-like structure allows users to specify what they want and not how to get it. Because there are identical commands for programmatic and interactive access, the same language can be used by programmers, DBA, and ad hoc query users.

With the easy-to-use language and the interactive facility for testing and debugging programs, programmer productivity is increased. The implementation of new systems will be much faster, making SQL a cost-effective and powerful solution for your database needs.

Biography

Lynn Barnes has been with Hewlett-Packard Company for 12 years. She began her career with HP as a programmer, writing COBOL application programs for the Region Information System Group. After three years as a programmer, she moved into the Systems Engineering Organization as a systems engineer for the HP 3000. For 6 years, her duties included teaching a variety of customer and HP internal training courses; customer consulting, both on site and over the phone; acting as the local software update and patch coordinator; providing customer account management functions; and serving as an area resource in the fields of data management, performance, and MPE internals. For the past 3 years, Lynn has been a system specialist, working on benchmarks, consulting, and special projects.

MAPPING IMAGE DATABASES TO A RELATIONAL ENVIRONMENT

Michele Dingerson Hewlett-Packard Company, Cupertino California, U.S.A.

Summary

With the introduction of Hewlett-Packard's relational database product, HPSQL, many existing IMAGE/3000 users may want to study the issue of mapping their IMAGE/3000 databases to HPSQL databases. Mapping an IMAGE/3000 database to a relational database system is a relatively easy task.

It is important first to understand the differences in terminology. For example, relational databases have tables and views, whereas IMAGE/3000 has datasets. It is also important to consider which types of applications are better suited for a relational environment than a nonrelational environment. Stable applications with well-defined data structures are well-suited to a nonrelational system.

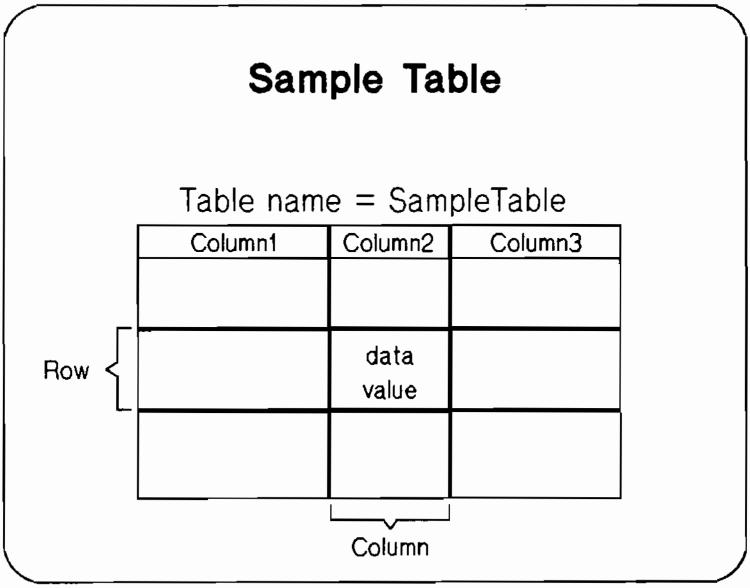
The differences in naming conventions, and item type and length must also be studied. It is then relatively easy to map each master and detail dataset to relational structures. You must take into consideration sort items, maximum record size, capacities, and security.

Introduction

Since the 1970s when the first database management systems were developed, their use has become widespread. Initially, database management systems were either network or hierarchical. In more recent years relational databases have entered the arena. Both network and hierarchical technologies require data access to be defined when data structures are defined. Relational technology removes that major limitation. Data relationships in the relational model are determined solely by the data, not by pointers or other connectors. This means the user does not need to specify how to access data, rather what data is desired.

Relational Terminology

A relational database is composed of one or more tables. All tables are made up of columns and rows (sometimes called attributes and tuples, respectively). The intersection of a column and a row is called a data value, or field.



Relationships between data are defined using data values when you access the data. For this reason, the order in which rows are physically stored is not important. Likewise the order of the columns is unimportant.

There are special tables in the relational model called views. A view is a subset of one or more tables called base tables. Data is not stored in a view. Only the definition of the view is stored. The data is retrieved from the base tables when you use the view. Views are useful for limiting the visibility of data or for combining data from several tables.

Which Model for Your Application?

Both relational and nonrelational models have distinct benefits. Most applications could be implemented on either relational or nonrelational systems. However, certain characteristics of applications make them more suitable for one model than another.

Let's look at the characteristics that make an application more appropriate for relational technology.

- Applications that require a high degree of data access flexibility are better suited for a relational system.

- Applications in which the data structures themselves will change are well-suited to a relational system. A relational system allows tables and views to be added or removed from the database at any time. New columns may be added to existing tables at any time.

- Applications with data access requirements that are unknown at the time of database design are also well-suited for a relational system.

The characteristics of applications that are better served by nonrelational systems are generally related to stability and well-defined structures. If access methods are very predictable at database design time, a nonrelational model would serve the application well. In addition, applications with a high volume of transactions that access a single data record are typically better candidates for a nonrelational system.

IMAGE Environment Versus Relational Environment

The remainder of this paper explores the mapping of an IMAGE/3000 database (referred to as an IMAGE database in the remainder of this paper) to a relational system. For the purpose of illustrating this concept, one specific implementation of relational technology is used, Hewlett Packard's HPSQL. Prior to reviewing the actual mapping process, let's review some key differences between IMAGE and a relational system.

Data Storage

IMAGE stores data in data sets. In a relational system, all data is stored in two-dimensional tables. A data set maps to a table. Records in IMAGE are known as entries. In a relational system a record is known as a row or tuple. In IMAGE terminology, a field of a record is called an item. In relational terminology, this field is called a column or attribute.

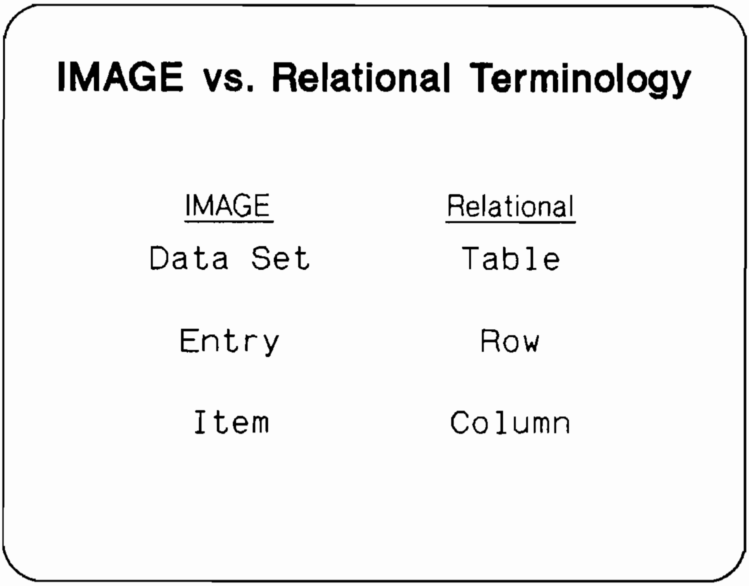
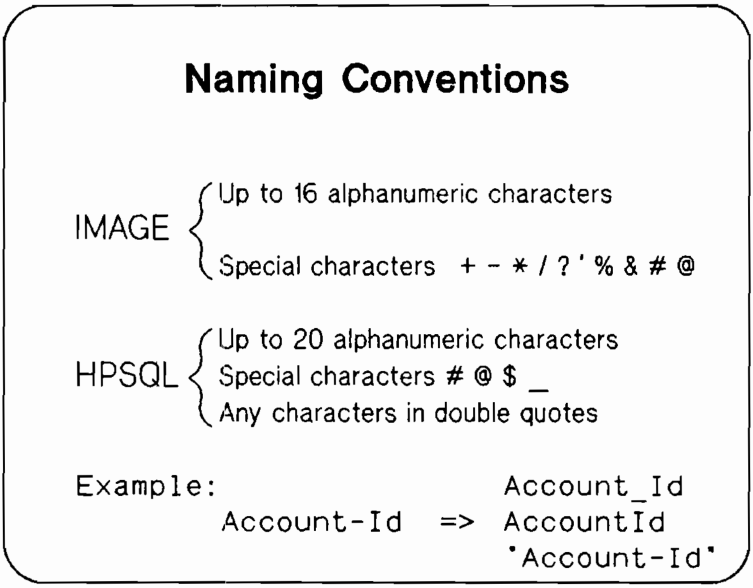


IMAGE vs. Relational Terminology

IMAGE	Relational
Data Set	Table
Entry	Row
Item	Column

Naming Conventions

When an IMAGE data item is defined, its definition contains four parts: name, type, count, and length. The figure below shows the differences in naming conventions between IMAGE and HPSQL.



Naming Conventions

IMAGE
- Up to 16 alphanumeric characters
- Special characters + - * / ? ' % & # @

HPSQL
- Up to 20 alphanumeric characters
- Special characters # @ $ _
- Any characters in double quotes

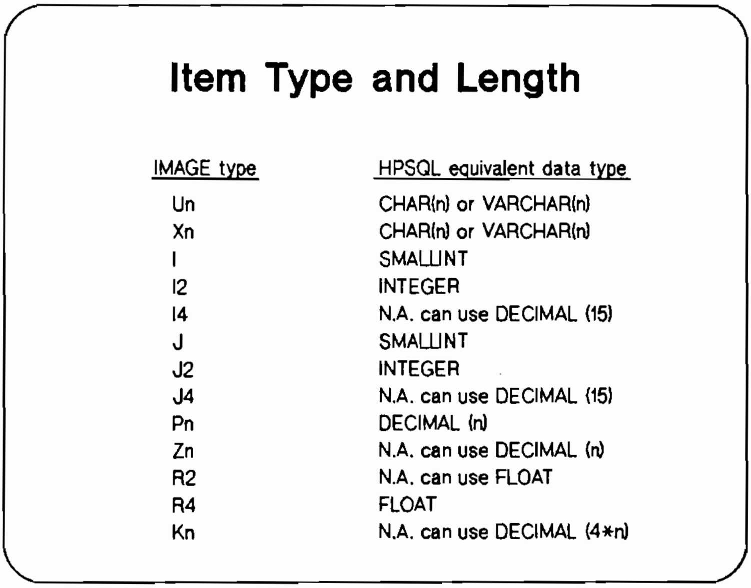
```
Example:                      Account_Id
          Account-Id   =>     AccountId
                              "Account-Id"
```

HPSQL allows names to be longer than IMAGE; therefore, length is not really an issue when mapping IMAGE data item names to HPSQL names.

The figure shows an example of mapping a data item name, Account-ID, to an HPSQL column name. Since a dash (-) is not a supported special character in HPSQL, it can be translated to an underscore (_), it could be dropped, as in the second example, or it could be retained by enclosing the name in double quotes, as in the third example.

Item Type, Length and Count

Not all data types that are allowed in IMAGE are allowed in HPSQL. The figure below shows the supported IMAGE data type and the HPSQL data types that are most closely equivalent.



Item Type and Length

IMAGE type	HPSQL equivalent data type
Un	CHAR(n) or VARCHAR(n)
Xn	CHAR(n) or VARCHAR(n)
I	SMALLINT
I2	INTEGER
I4	N.A. can use DECIMAL (15)
J	SMALLINT
J2	INTEGER
J4	N.A. can use DECIMAL (15)
Pn	DECIMAL (n)
Zn	N.A. can use DECIMAL (n)
R2	N.A. can use FLOAT
R4	FLOAT
Kn	N.A. can use DECIMAL (4*n)

When mapping item types from IMAGE to HPSQL, the following points need to be considered:

- There can be a loss of accuracy in mapping I4 to DECIMAL, since I4 handles as many as 19 digits and DECIMAL handles as many as 15 digits.
- Mapping J4 to DECIMAL or Zn to DECIMAL does not result in any loss of accuracy.
- Mapping R2 to float results in a gain in accuracy from 7 significant digits to 15 significant digits.
- Mapping Zn to DECIMAL means that you will need to use DECIMAL in any COBOL applications.
- To map Kn to DECIMAL, multiply n by 4. For example, a K2 item can be represented as a DECIMAL (8,0).

IMAGE allows you to create compound data items using item count. This cannot be directly mapped in Hewlett-Packard's relational system. You can create the same effect as a compound data item in one of two ways. You may create a different column for each occurrence of the field; for example, a data item defined as Semester, 2X6 could be mapped to two columns defined as Semester1 Char(6) and Semester2

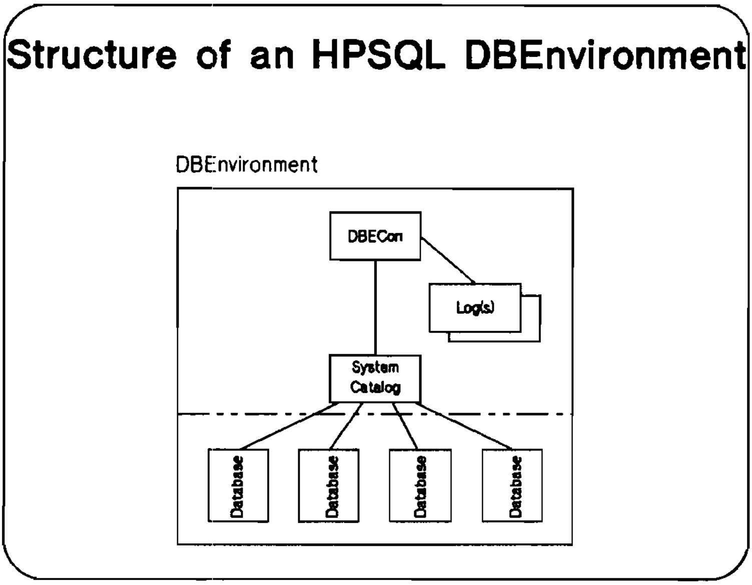
Char(6). You can also map a compound data item to a table with a row for each occurrence of the field. An index could be defined on the table for easy access.

Mapping IMAGE to a Relational Database

The process of mapping an IMAGE database to a relational environment is a relatively easy task. This section shows a method to accomplish this process. For a further example of this process, please refer to the appendices of this paper: Appendix A contains a schema listing for the IMAGE Orders database that is used in the IMAGE documentation. Appendix B contains the "schema" for that database after it has been mapped to a relational system, HPSQL.

Database Environment

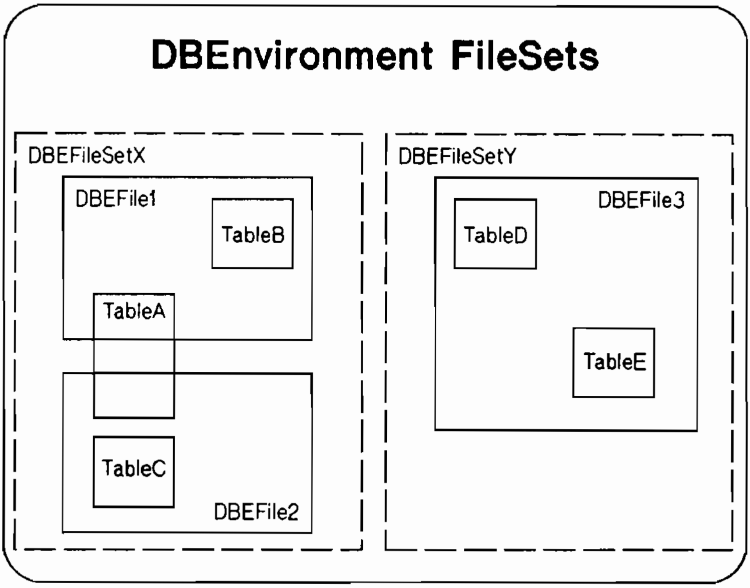
HPSQL tables, views, and indexes are grouped into logical databases. Databases are grouped into database environments (DBEnvironments). A DBEnvironment consists of one or more databases, a system catalog, a DBECon file, and one or two log files. The system catalog is a set of tables used by HPSQL to store the structure of the databases and to access the data. The DBECon file contains global parameters for the DBEnvironment. The log file(s) are used to maintain the physical and logical integrity of the DBEnvironment.



Structure of an HPSQL DBEnvironment

DBEFileSets and DBEFiles

When HPSQL tables are defined, they are assigned to a DBEFileSet. A DBEFileSet is a logical grouping of space. It is a collection of DBEFiles. A DBEFile is an actual system file. To allocate space to a DBEFileSet, a user simply adds DBEFiles. By allocating space in this manner, users can expand tables at any time by adding another DBEFile to a DBEFileSet.



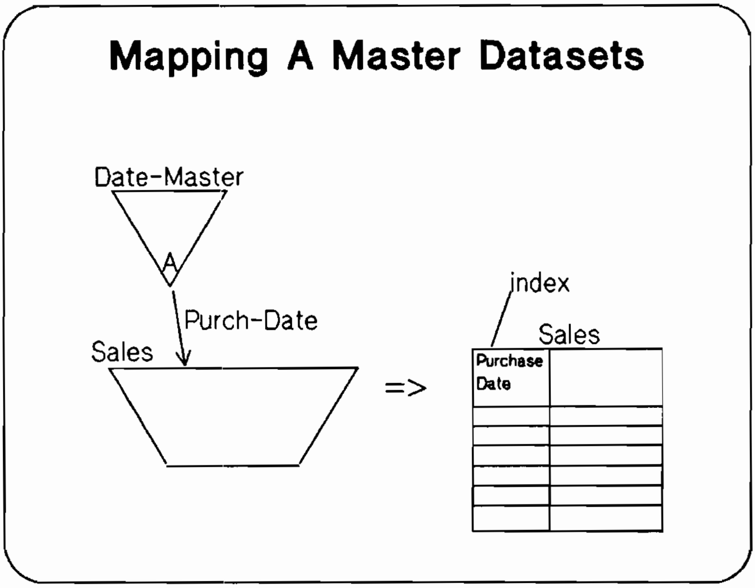
In IMAGE, a dataset is associated directly with one system file that can be moved at will to a specific disc device. In HPSQL, more than one table can be associated directly with one DBEFileSet to which DBEFiles can be added. DBEFiles can be moved at will to specific disc devices. However, when more than one table resides in a DBEFileSet, it is not possible to move only one table to a specified device. To obtain an effect similar to IMAGE in HPSQL, it is possible to create only one table in a DBEFileSet. Any DBEFiles added to the DBEFileSet will house data for that particular table only. This will make it possible to move DBEFiles to a particular device guaranteeing that only one table is moved.

Mapping Master Datasets

IMAGE datasets contain entries that are of fixed length. Therefore, they can be mapped directly to tables. In a relational system, indexes can be defined on any column or columns in a table. These indexes allow data to be accessed more quickly. Key and search items in IMAGE datasets can be defined as indexes in a relational system.

A manual master dataset can be mapped directly to a table with a UNIQUE index defined on the column corresponding to the key field. A UNIQUE index, like a key value, allows only one data item of a particular value. To preserve a higher degree of compatibility, the key column can be defined as NOT NULL. The NOT NULL clause requires the column to always contain a data value.

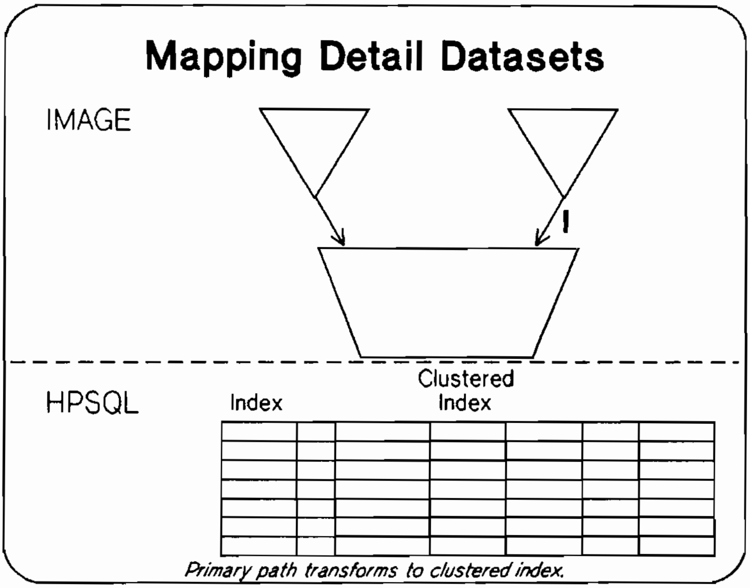
Automatic master datasets can be mapped by creating an index on the table that corresponds to the associated detail dataset(s). For example, when mapping an automatic master consisting of a date field with a path into a Sales detail dataset, create an index on the PurchaseDate column of the Sales table.



Mapping A Master Datasets

Mapping Detail Datasets

Relational systems draw no distinction between master and detail datasets. All tables are treated at the same level. To map a detail dataset to a relational system you need to consider the paths into the dataset. Detail datasets can be mapped to tables by converting the search items to indexes. The primary path must be defined as a clustering index. A clustering index will place the data physically close whenever possible. Since only one clustering index may be defined on a table, any other search items must be specified as non-clustering.



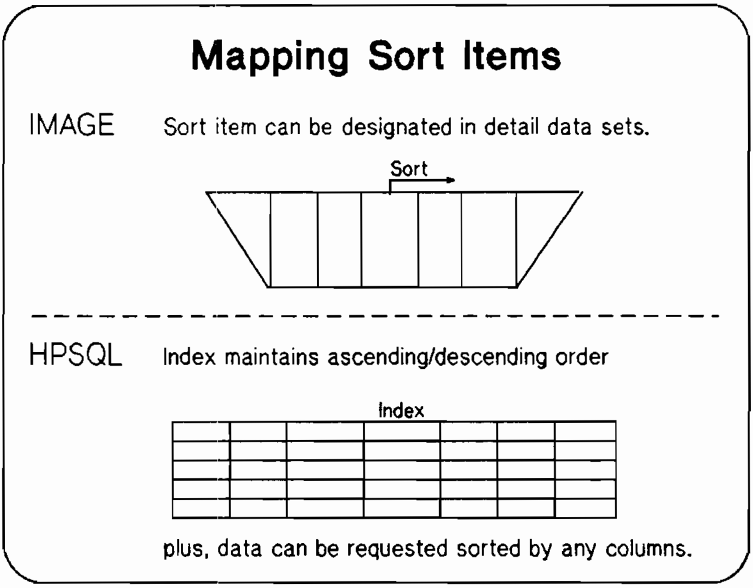
Primary path transforms to clustered index.

Zeros Versus Nulls

In IMAGE, binary zeros are stored for items that are not listed in the DBPUT intrinsic. HPSQL does not store anything for columns that are not listed in an INSERT command. HPSQL uses an indicator to specify that a given column does not have a value specified for it. In other words, that column is NULL.

Sort Items

IMAGE allows sort items to be specified in detail datasets to maintain sorted data chains on search items. Search items can be defined as indexes in HPSQL which maintains the index in both ascending and descending order. The rows that are being selected can be ordered in ascending or descending fashion, with minor sort columns specified in any order. In IMAGE, items that have default values sort low. In HPSQL, columns that are null sort high.



Mapping Sort Items

IMAGE Sort item can be designated in detail data sets.

HPSQL Index maintains ascending/descending order

plus, data can be requested sorted by any columns.

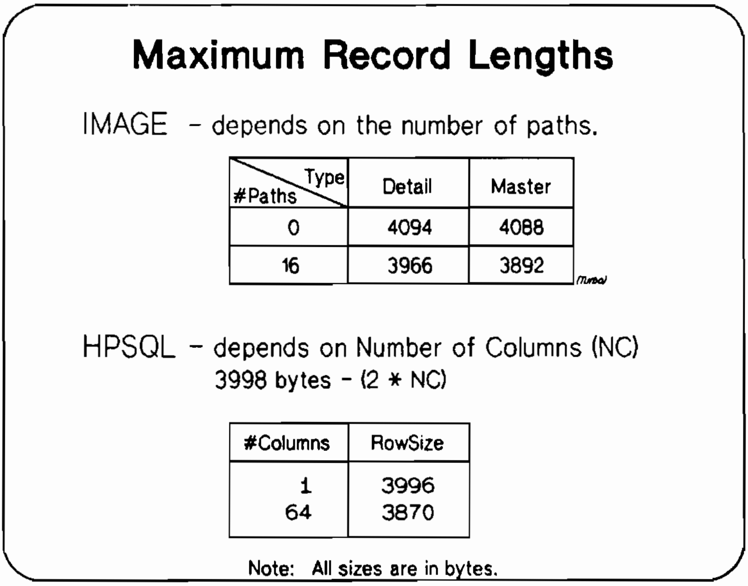
IMAGE Integrity Checking

IMAGE allows for integrity checking of values. This is accomplished within manual masters; an entry without a key or duplicate key value is prevented from being added to the dataset. To do this in HPSQL, create UNIQUE indexes on critical columns and disallow null values.

Another way of preserving integrity of data in IMAGE is provided by detail datasets. An entry in a detail dataset cannot be added if the associated manual master datasets do not contain the key value. This type of integrity checking is not part of HPSQL's feature set. The integrity checking has to be performed programmatically.

Maximum Record Size

In IMAGE, the data entry size depends on the number of paths defined on the dataset and whether the dataset is a master dataset or a detail dataset. The size of a data entry decreases when the number of paths increases. For example, the maximum data entry size is achieved in IMAGE when there are no paths defined. The minimum data entry size is achieved when the maximum number of paths is defined.



Maximum Record Lengths

IMAGE - depends on the number of paths.

#Paths \ Type	Detail	Master
0	4094	4088
16	3966	3892

(Turbo)

HPSQL - depends on Number of Columns (NC)
3998 bytes - (2 * NC)

#Columns	RowSize
1 64	3996 3870

Note: All sizes are in bytes.

As the illustration above shows, the maximum entry size of a detail dataset with no paths is 4094 bytes. For a standalone master dataset, the maximum entry size is 4088 bytes. For a detail dataset with 16 paths defined, the maximum entry size is 3966 bytes. For a master dataset with 16 paths, the maximum entry size is 3892 bytes. The discrepancy in size for detail datasets and master data sets is due to the fact that the data chains and bit maps are stored with the data.

In HPSQL, the data pointers are stored separately from the data. The number of columns in a table (NC) is the decisive factor in determining the actual maximum row size. The illustration shows the formula for calculating the size of a row in HPSQL. This formula yields a maximum row size of 3996 bytes for a one-column table and a maximum row size of 3870 bytes for a 64-column table.

Some measures that can be taken to decrease the column size are:

- truncate some columns to reduce the overall size
- combine some columns to reduce the total number of columns in the table.

IMAGE Capacity

The capacity of IMAGE datasets must be specified at creation time. The maximum size that can be specified for a dataset is 2^{23} minus one. HPSQL does not require capacities to be specified for tables. Instead, storage structures, DBEFiles, can be created and added any time more storage space is needed. The maximum size a table in HPSQL is (32767^2 * 3996) bytes.

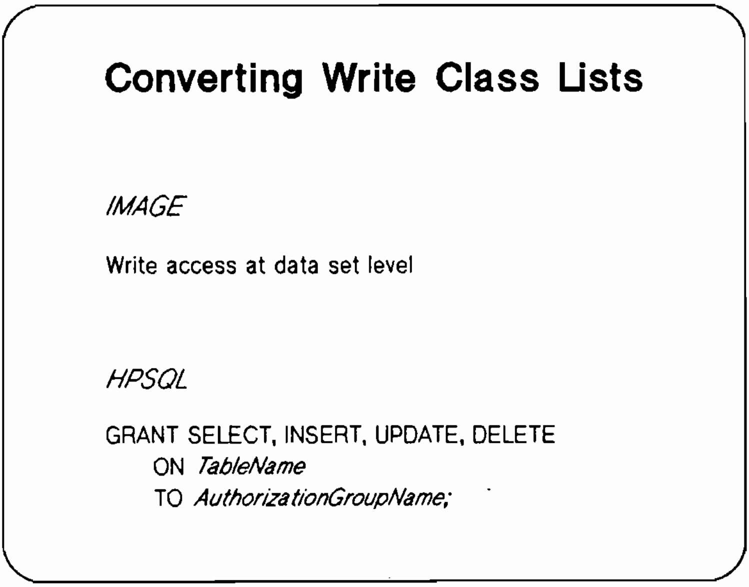
Data Security

In IMAGE, passwords are the basis for establishing security within a database. Passwords are given different types of access to the database. To access the data, the user need only know the password. HPSQL offers an extensive security system that is maintained by the database administrator (DBA). In HPSQL, the same effect as an IMAGE password can be achieved by creating an object known as an authorization group with the same name as the password and granting all of the authorities necessary to accomplish the same tasks as an IMAGE password. An authorization group is a named collection of users who all have the same authorities.

Write Class List

IMAGE security can be specified for read access and/or write access. Security is specified both at the dataset level and at the data item level. To convert IMAGE security to HPSQL, it is necessary to look at the read and write class lists specified for the items and datasets.

Write access at the dataset level overrides any security specified at the item level and implies read access on all of the items within the dataset. Write access at the dataset level means that the dataset can be modified by the user with that particular password. Data set modification means that the user can GET, PUT, UPDATE, and DELETE in IMAGE terms. In HPSQL terms, the user can SELECT, INSERT, UPDATE, and DELETE. The illustration below shows how an IMAGE write class list would be converted to HPSQL.



Converting Write Class Lists

IMAGE

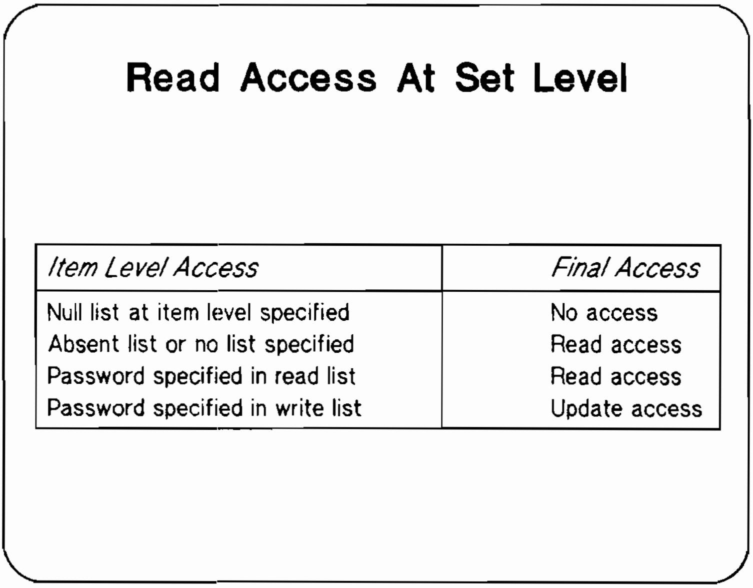
Write access at data set level

HPSQL

GRANT SELECT, INSERT, UPDATE, DELETE
    ON *TableName*
    TO *AuthorizationGroupName;*

Read Class List

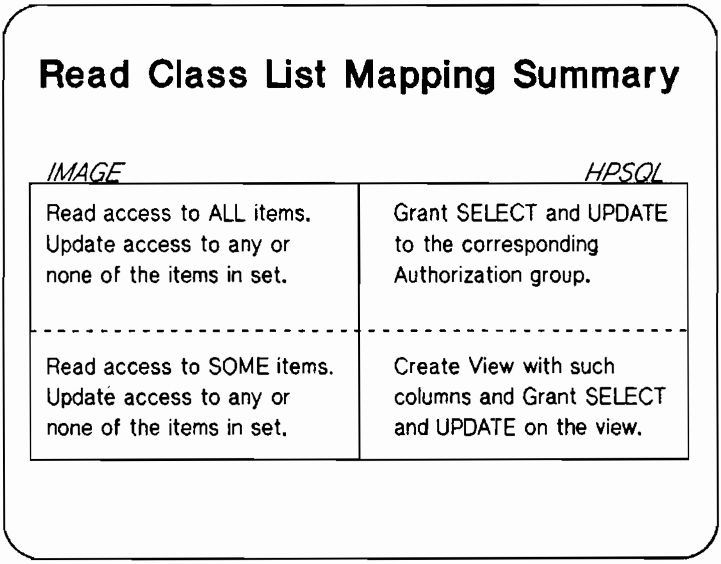
Converting the read class list at the set level is somewhat more intricate since read access at the dataset level does NOT override the item level specifications of security. Instead, read access at the dataset level implies that item level security should be combined with the dataset security to obtain final access to the item. IMAGE security specifies that whenever a password appears only on the read class list at the set level, the item level security is also used to obtain the final access. There are several combinations of security when read access is allowed at the set level that can be summarized as shown in the table below.



Read Access At Set Level

Item Level Access	*Final Access*
Null list at item level specified	No access
Absent list or no list specified	Read access
Password specified in read list	Read access
Password specified in write list	Update access

Whenever a password has read access to all of the items in a dataset, it is sufficient to grant SELECT authority on the corresponding table to that particular authorization group. However, if a password does not have read or update access to all of the items in the dataset, a view must be created with only those columns that can be accessed; then SELECT and/or UPDATE authorization must be granted on that view to the appropriate authorization groups. This is summarized in the table below.



Read Class List Mapping Summary

IMAGE	*HPSQL*
Read access to ALL items. Update access to any or none of the items in set.	Grant SELECT and UPDATE to the corresponding Authorization group.
Read access to SOME items. Update access to any or none of the items in set.	Create View with such columns and Grant SELECT and UPDATE on the view.

User Class 0

In IMAGE, user class 0 (zero) is a special class that is not allowed a password and can be specified in the read and write class list of an item or a dataset. This user class can be used to allow read and/or write access to nonrestricted portions of the database to any user. The equivalent to this user class in HPSQL is the special designation PUBLIC. In HPSQL, authorities can be granted to PUBLIC in the same manner that user class zero is given read/write access.

Conclusion

The task of mapping an IMAGE schema to HPSQL data definition commands is fairly simple as was shown. However, the user performing the task must be well aware of the differences described in this paper. It should be kept in mind that many IMAGE applications are well suited to a nonrelational environment and should not converted to a relational environment. For those users who do decide to map their IMAGE applications to HPSQL, the steps are summarized below:

- Create a DBEnvironment

- Transform the IMAGE database(s)
    - Create authorization groups
    - Create DBEFileSets
    - Create DBEFiles
    - Create tables
    - Create views
    - Create indexes

Biography

Michele Dingerson is currently a Marketing Engineer for Hewlett-Packard working in the Data Management Support Group of Information Technology Group's Technical Marketing Department. As a member of the documentation and training team for Hewlett-Packard's new relational database product, HPSQL, she has developed a customer training course for database administrators and participated in training systems engineers on HPSQL.

Prior to her current position, Michele worked as a Financial Systems Analyst for a large California-based telecommunications firm. Prior to that position, she worked as a Programmer/Analyst at the Corporate Headquarters of Hewlett-Packard in Palo Alto California. Ms. Dingerson initially joined Hewlett-Packard in 1980, and most recently has been with Hewlett-Packard since 1984.

Appendix A: IMAGE Schema

```
$CONTROL TABLE, NOROOT

BEGIN DATA BASE ORDERS;

PASSWORDS:
  14 CLERK;       << SALES CLERK >>
  12 BUYER;       << BUYER - RESPONSIBLE FOR PARTS INVENTORY >>
  11 CREDIT;      << CUSTOMER CREDIT OFFICE >>
  13 SHIP-REC;    << WAREHOUSE - SHIPPING AND RECEIVING >>
  18 DO-ALL;      << FOR USE BY DB MGMT >>

ITEMS:             << IN ALPHABETICAL ORDER FOR CONVENIENCE >>
  ACCOUNT,         J02 ;                << CUSTOMER ACCOUNT NUMBER      >>
  BINNUM,          Z02 (/13);           << STORAGE LOCATION OF PROD     >>
  CITY,            X12 (12,13,14/11);   << CITY                         >>
  CREDIT-RATING,   R02 (/14);           << CUSTOMER CREDIT RATING       >>
  DATE,            X06 ;                << DATE (YYMMDD)                >>
  DELIV-DATE,      X06 (/14);           << DELIVERY DATE (YYMMDD)       >>
  DESCRIPTION,     X20;                 << PRODUCT DESCRIPTION          >>
  FIRST-NAME,      X10 (14/11);         << CUSTOMER GIVEN NAME          >>
  INITIAL,         U02 (14/11);         << CUSTOMER MIDDLE INITIAL      >>
  LAST-NAME,       X16 (14/11);         << CUSTOMER SURNAME             >>
  LASTSHIPDATE,    X06 (12/ );          << DATE LAST REC D(YYMMDD)      >>
  ONHANDQTY,       J02 (14/12);         << TOTAL PRODUCT INVENTORY      >>
  PRICE,           J02 (14/);           << SELLING PRICE (PENNIES)      >>
  PURCH-DATE,      X06 (11/14);         << PURCHASE DATE (YYMMDD)       >>
  QUANTITY,        I   (/14);           << SALES PURCHASE QUANTITY      >>
  STATE,           X02 (12,13,14/11);   << STATE -- 2 LETTER ABB.       >>
  STOCK#,          U08 ;                << PRODUCT STOCK NUMBER         >>
  STREET-ADD,      X26 (12,13,14/11);   << NUMBER AND STREET ADD        >>
  SUPPLIER,        X16 (12,13/);        << SUPPLYING COMPANY NAME       >>
  TAX,             J02 (14/);           << SALES TAX                    >>
  TOTAL,           J02 (11,14/);        << TOTAL AMOUNT OF SALE         >>
  UNIT-COST,       P08 (/12);           << UNIT COST OF PRODUCT         >>
  ZIP,             X06 (12,13,14/11);   << ZIP CODE                     >>
```

```
SETS:

  NAME:        CUSTOMER,MANUAL(14/11,18); <<CUSTOMER MASTER INFO>>
  ENTRY:       ACCOUNT(1),
               LAST-NAME,
               FIRST-NAME,
               INITIAL,
               STREET-ADD,
               CITY,
               STATE,
               ZIP,
               CREDIT-RATING;
  CAPACITY:    200;


  NAME:        DATE-MASTER,AUTOMATIC;       <<DATE INDEX>>
  ENTRY:       DATE(3);
  CAPACITY:    211;


  NAME:        PRODUCT,MANUAL(14,13/12,18);<<PRODUCT INDEXT>>
  ENTRY:       STOCK#(2),
               DESCRIPTION;
  CAPACITY:    300;


  NAME:        SALES,DETAIL(11/14,18); <<CREDIT PURCHASE INFO>>
  ENTRY:       ACCOUNT(CUSTOMER(PURCH-DATE)),
               STOCK#(PRODUCT),
               QUANTITY,
               PRICE,
               TAX,
               TOTAL,
               PURCH-DATE(DATE-MASTER),
               DELIV-DATE(DATE-MASTER);
  CAPACITY:    500;


  NAME:        SUP-MASTER,MANUAL(13/12,18);<<SUPP MASTER INFO>>
  ENTRY:       SUPPLIER(1),
               STREET-ADD,
               CITY,
               STATE,
               ZIP;
  CAPACITY:    200;
  NAME:       INVENTORY,DETAIL(12,14/13,18);<<PROD SUPPLY INFO>>
  ENTRY:      STOCK#(PRODUCT),
              ONHANDQTY,
              SUPPLIER(!SUP-MASTER),       <<PRIMARY PATH>>
              UNIT-COST,
              LASTSHIPDATE(DATE-MASTER),
              BINNUM;
  CAPACITY:   450;

END.

 DATA SET       TYPE  FLD  PT  ENTR  MED  CAPACITY  BLK  BLK   DISC
   NAME               CNT  CT  LGTH  REC            FAC  LGTH  SPACE

CUSTOMER           M  9    1   41    51    200      10   511   84
DATE-MASTER        A  1    3   3     23    211      22   508   44
PRODUCT            M  2    2   14    29    300      13   378   75
SALES              D  8    4   19    35    504      14   491   148
SUP-MASTER         M  5    1   31    41    200      12   493   72
INVENTORY          D  6    3   20    32    450      15   481   124
                            TOTAL DISC SECTORS INCLUDING ROOT: 560


NUMBER OF ERROR MESSAGES: 0
ITEM NAME COUNT: 23       DATA SET COUNT: 6
ROOT LENGTH: 729     BUFFER LENGTH: 511     TRAILER LENGTH: 256
```

Appendix B: HPSQL Schema

```
/* ************************************************************* */;
/*      Converted IMAGE schema to HPSQL (from ORDERSSC)           */;
/* ************************************************************* */;

START DBE 'OrderDBE' MULTI NEW
    DBEFILEO DBEFILE OrdersRoot
        WITH PAGES = 50,
        NAME = 'Orders',
    LOG DBEFILE OrdersLog1
        WITH PAGES = 200,
        NAME = 'OrderLog';

CREATE GROUP Order.Clerks;
CREATE GROUP Order.Buyers;
CREATE GROUP Order.Credit;
CREATE GROUP Order.ShipReceive;
CREATE GROUP Order.DoAll;

GRANT CONNECT TO
        Clerks, Buyers, Credit,
        ShipReceive, DoAll;
```

```
/* **************************************************************** */;
/*         CUSTOMER TABLE                                           */;
/* **************************************************************** */;

CREATE DBEFILESET OrdersSet;
CREATE DBEFILE OrdersData01
        WITH PAGES = 50, NAME = 'OrdersD1', TYPE = TABLE;
CREATE DBEFILE OrdersIndx01
        WITH PAGES = 50, NAME = 'OrdersX1', TYPE = INDEX;
ADD DBEFILE OrdersData01 TO DBEFILESET OrdersSet;
ADD DBEFILE OrdersIndx01 TO DBEFILESET OrdersSet;

CREATE PUBLIC TABLE Order.Customer
     (Account           INTEGER     NOT NULL,
      LastName          CHAR(16),
      FirstName         CHAR(10),
      Initial           CHAR(02),
      StreetAddress     CHAR(26),
      City              CHAR(12),
      State             CHAR(02),
      Zip               CHAR(06),
      CreditRating      FLOAT )
     IN OrdersSet;
REVOKE ALL
      ON Order.Customer
      FROM PUBLIC;
CREATE UNIQUE INDEX CustomerAccount
      ON Order.Customer (Account);
GRANT SELECT,
      UPDATE (CreditRating)
      ON Order.Customer TO Clerks;
GRANT SELECT, INSERT, UPDATE, DELETE
      ON Order.Customer TO Credit, DoAll;
```

```
/* **************************************************************** */;
/*          PRODUCT TABLE                                           */;
/* **************************************************************** */;


CREATE DBEFILESET ProductSet;

CREATE DBEFILE ProductData01
        WITH PAGES = 50, NAME = 'ProdD1', TYPE = TABLE;
CREATE DBEFILE ProductIndx01
        WITH PAGES = 50, NAME = 'ProdX1', TYPE = INDEX;

ADD DBEFILE ProductData01
      TO DBEFILESET ProductSet;
ADD DBEFILE ProductIndx01
      TO DBEFILESET ProductSet;

CREATE PUBLIC TABLE Order.Product
     (StockNum          CHAR(OB)             NOT NULL,
      Description       CHAR(20)             NOT NULL)
     IN ProductSet;

REVOKE ALL
      ON Order.Product
      FROM PUBLIC;

CREATE UNIQUE INDEX ProductStockNum
      ON Order.Product (StockNum);

GRANT SELECT
      ON Order.Product
      TO Clerk, ShipReceive;

GRANT SELECT,
      INSERT,
      UPDATE,
      DELETE
      ON Order.Product
      TO Buyer, DoAll;
```

```
/* ************************************************************ */;
/*        SALES   TABLE                                         */;
/* ************************************************************ */;

CREATE DBEFILESET SalesSet;

CREATE DBEFILE SalesData01
        WITH PAGES = 50, NAME = 'SalesD1', TYPE = TABLE;
CREATE DBEFILE SalesIndx01
        WITH PAGES = 50, NAME = 'SalesX1', TYPE = INDEX;
ADD DBEFILE SalesData01 TO DBEFILESET SalesSet;
ADD DBEFILE SalesIndx01 TO DBEFILESET SalesSet;

CREATE PUBLIC TABLE Order.Sales
     (Account          INTEGER             NOT NULL,
      StockNumber      CHAR(08)            NOT NULL,
      Quantity         SMALLINT            NOT NULL,
      Price            INTEGER             NOT NULL,
      Tax              INTEGER             NOT NULL,
      Total            INTEGER             NOT NULL,
      PurchaseDate     CHAR(06)            NOT NULL,
      DeliveryDate     CHAR(06)            NOT NULL)
     IN SalesSet;

REVOKE ALL
         ON Order.Sales
         FROM PUBLIC;
CREATE INDEX SalesAccount
         ON Order.Sales (Account);
CREATE CLUSTERING INDEX SalesStockNum
         ON Order.Sales (StockNumber);
CREATE INDEX SalesPurchaseDate
         ON Order.Sales (PurchaseDate);
CREATE INDEX SalesDeliveryDate
         ON Order.Sales (DeliveryDate);
GRANT SELECT,
      INSERT,
      UPDATE,
      DELETE
      ON Order.Sales
      TO Clerk, DoAll;
CREATE VIEW Order.SalesCredit
      AS SELECT Account, StockNumber, Total, PurchaseDate
      FROM Order.Sales;
GRANT SELECT
      ON Order.SalesCredit
      TO Credit;
```

```
/* **************************************************************** */;
/*         SUPPLY  TABLE                                             */;
/* **************************************************************** */;

CREATE DBEFILESET SupplySet;

CREATE DBEFILE SupplyData01
        WITH PAGES = 50, NAME = 'SupplyD1', TYPE = TABLE;
CREATE DBEFILE SupplyIndx01
        WITH PAGES = 50, NAME = 'SupplyX1', TYPE = INDEX;

ADD DBEFILE SupplyData01
       TO DBEFILESET SupplySet;
ADD DBEFILE SupplyIndx01
       TO DBEFILESET SupplySet;

CREATE PUBLIC TABLE Order.SupplyMaster
     (Supplier          CHAR(16)             NOT NULL,
      StreetAddress     CHAR(26)             NOT NULL,
      City              CHAR(12)             NOT NULL,
      State             CHAR(02)             NOT NULL,
      Zip               CHAR(06)             NOT NULL)
     IN SupplySet;

REVOKE ALL
      ON Order.SupplyMaster
      FROM PUBLIC;

CREATE UNIQUE INDEX SupplyMasterSupplier
      ON Order.SupplyMaster (Supplier);

GRANT SELECT
      ON Order.SupplyMaster
      TO ShipReceive;

GRANT SELECT,
      INSERT,
      UPDATE,
      DELETE
      ON Order.SupplyMaster
      TO Buyer, DoAll;
```

```
/* ***************************************************************** */;
/*         INVENTORY TABLE                                            */;
/* ***************************************************************** */;

CREATE DBEFILESET InventorySet;

CREATE DBEFILE InventoryData01
        WITH PAGES = 50, NAME = 'InventD1', TYPE = TABLE;
CREATE DBEFILE InventoryIndx01
        WITH PAGES = 50, NAME = 'InventX1', TYPE = INDEX;
ADD DBEFILE InventoryData01
        TO DBEFILESET InventorySet;
ADD DBEFILE InventoryIndx01
        TO DBEFILESET InventorySet;

CREATE PUBLIC TABLE Order.Inventory
     (StockNumber       CHAR(0B)             NOT NULL,
      OnHandQty         INTEGER              NOT NULL,
      Supplier          CHAR(16)             NOT NULL,
      UnitCost          INTEGER              NOT NULL,
      LastShipDate      CHAR(06)             NOT NULL,
      BinNumber         SMALLINT             NOT NULL)
     IN InventorySet;
REVOKE ALL
         ON Order.Inventory
         FROM PUBLIC;

CREATE INDEX InventoryStockNum
         ON Order.Inventory (StockNumber);
CREATE CLUSTERING INDEX InventorySupplier
         ON Order.Inventory (Supplier);
CREATE INDEX InventorLastShipDate
         ON Order.Inventory (LastShipDate);
GRANT SELECT, INSERT, UPDATE, DELETE
      ON Order.Inventory
      TO ShipReceive, DoAll;

CREATE VIEW Order.InventoryBuyer
      AS SELECT StockNumber, OnHandQty, Supplier, UnitCost
      FROM Order.Inventory;
GRANT SELECT,
      UPDATE (OnHandQty, UnitCost)
      ON Order.InventoryBuyer TO Buyer;

CREATE VIEW Order.InventoryClerk
      AS SELECT StockNumber, OnHandQty
      FROM Order.Inventory;
GRANT SELECT
      ON Order.InventoryClerk
      TO Clerks;
```

SQL The Defacto Standard for Relational Databases.

This paper addresses the features of the SQL as a DBMS, and why its popularity is making it an industry standard. To address this topic the features of SQL, specifically HP's implementation, HPSQL, are discussed in detail.

HPSQL Features and Benefits

HPSQL is a fully relational database management system that is based on SQL. SQL is an acronym for Structured Query Language. This language is rapidly becoming the industry standard for relational databases. A true standard for SQL does not yet exist in the industry. However, an ANSI standard committee formed by representatives of different computer companies, including HP, is considering approving a set of standards for SQL. A description of the major features offered by HPSQL follows. The remaining of this paper goes on to explore the applicability of these features in more detail.

Relational Structure

HPSQL supports a relational data structure that is easy to visualize and understand, making it possible for users to start using this product at once. Conceptually the database is composed of flat files or 'tables'. A table basically consists of data records which are known as 'rows'. Each row contains a number of data fields or 'columns'. The tables do not contain any predefined relationships. Instead relationships between tables are established at the time of inquiry, by comparing values in columns common to the tables. These simple concepts are the nucleus of the relational database.

Structured Query Language (SQL)

The Structured Query Language (SQL) supported by HPSQL consists of a simple set of commands for data definition and data manipulation. That means that both functions are provided through a common language interface which is easy to understand and learn. Additionally, the basic SQL command set has been extended to support such functions as database administration and maintenance.

Database Security

A database security system, more extensive than ever before offered by an HP data management product, is available with HPSQL. Data security can be so fine- tuned that access can be restricted to only certain rows within a table or to only certain programs that manipulate data within the DBEnvironment. The database administrator (DBA) is the person responsible for establishing security within the DBEnvironment. A DBA can designate alternate DBA's to help with the task of managing security.

Transaction Management and Recovery

HPSQL is a transaction oriented database management system where the transaction is handled as a complete unit of work. The user is responsible for defining the beginning and end of a transaction and HPSQL ensures that a transaction is never partially executed; either all or none of it is made permanent. Thus, the DBEnvironment is always in a logically consistent state. HPSQL also maintains the physical integrity of the DBEnvironment in a consistent state at all times regardless of the type of failure that may occur to the system. The logical and physical integrity is maintained by a log file which is part of the DBEnvironment. To achieve an even greater degree of integrity, HPSQL also supports an optional dual log file in a separate disc from the first log file. When dual logging is enabled HPSQL writes identical log records to both log files. Multiuser access is accomplished by a sophisticated algorithm that handles all locking automatically, at the same time resolving deadlocks. Explicit locking can also be performed for those applications that require it.

Interactive Interface

Included in the HPSQL product is an interactive interface, ISQL, which is an acronym for Interactive SQL. ISQL is a multifunction program that supports all of the SQL command set along with commands of its own. Also supported in ISQL are a number of facilities to ease command entry and execution. A friendly help facility along with redo functions, make ISQL very easy to learn and use.

Programmatic Interface

HPSQL also has a programmatic interface which allows application programmers to insert SQL commands in programs. As soon as the users learn SQL, they are ready to start programming with it, without having to learn a secondary method to access the DBEnvironment programmatically. Every clause and parameter variation allowed in an SQL command is also allowed programmatically. This approach to programming allows users to start accessing DBEnvironments programmatically in a very short time. Embedding SQL commands in source programs reduces the size of the source programs while at the same time increasing its maintainability.

HPSQL Components

As mentioned earlier HPSQL is the relational interface of DBCore. This section describes the different objects and components of an HPSQL DBEnvironment. Subjects that will be covered include tables, data types, views, indexes, DBEnvironment, DBEFilesets and DBEFiles.

Table

A table is the basic object in a relational database. Data is represented to the user as being contained in a table, also said to be a flat file, which consists of records (called 'rows') and fields (called 'columns'). Each row in a table contains the same number of columns. That does not necessarily imply that each row is of the same size since columns can be of variable length.

Data Types

The data types allowed for column attributes are sufficient for any kind of applications that are developed using HPSQL.

A list of types with a short description of them follows:

Decimal - This numeric data type is typically used in commercial applications. It has a range of 15 digits plus a sign; optionally, the number of digits to the right of the decimal period can also be specified.

Character - This alphanumeric data type stores any ASCII character type data of fixed length.

VariableCharacter - This alphanumeric data type is the first of its kind offered in HP's data management products. It allows storage of data that is usually variable in length, such as an address or comments. The main feature of this data type is that saves disc space storage since only the actual length of data is stored.

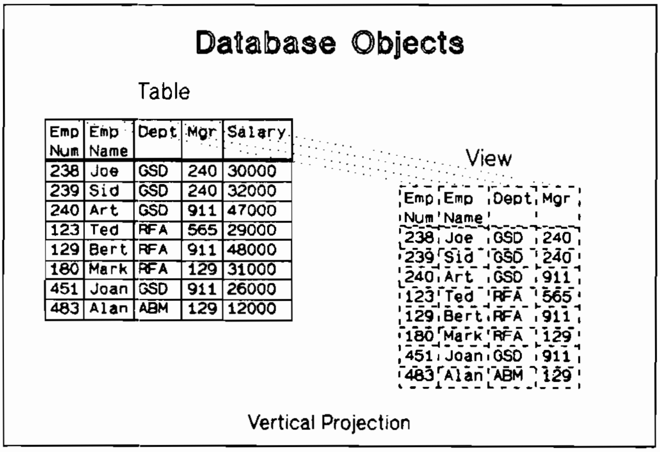
SmallInteger - This numeric data type is a 16 bit signed integer with a range of -32,768 to 32,767.

Integer - This numeric data type is a 32 bit signed integer with a range of -2,147,483,647 tɔ 2,147,483,647 or just over 2 billion.

Float - This numeric data type is a double precision real value with 16.9 digits of precision that is most useful in scientific applications.

Views

Views are table projections used to filter data for user access. Figure 1 shows a view used to filter out one column of an underlying table. The underlying table is known as a ′base table′. In this case, a view is used to prohibit access to the Salary column. The user accesses the view as if it were the actual table. The view itself does NOT cause the data to be stored twice, but acts merely as a filter for the base table. Thus a view is another way to describe the data that resides in storage.

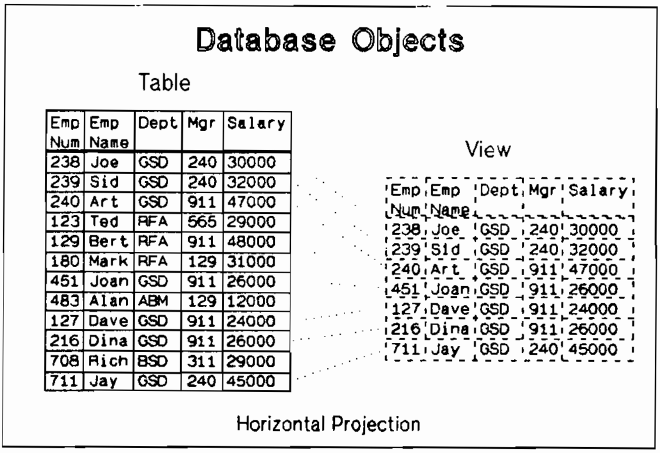


Emp Num	Emp Name	Dept	Mgr	Salary
238	Joe	GSD	240	30000
239	Sid	GSD	240	32000
240	Art	GSD	911	47000
123	Ted	RFA	565	29000
129	Bert	RFA	911	48000
180	Mark	RFA	129	31000
451	Joan	GSD	911	26000
483	Alan	ABM	129	12000

Emp Num	Emp Name	Dept	Mgr
238	Joe	GSD	240
239	Sid	GSD	240
240	Art	GSD	911
123	Ted	RFA	565
129	Bert	RFA	911
180	Mark	RFA	129
451	Joan	GSD	911
483	Alan	ABM	129

Figure 1. Vertical Projection

Another feature of views is horizontal projections from a base table. Consider, for example, a table that contains employee information for the entire company with users from different departments accessing this data. Users from one department are not allowed to access employee data from a department other than their own. To solve this situation and still have all of the employee data in one table, a view can be created for each department, allowing access only to data in a single department. Figure 2 shows an example of using a view to allow access to certain parts of a table. This is a very useful feature since it allows the same set of data to be accessed by many different users with different needs. This use of views minimizes data storage requirements since the data is only stored once.



Database Objects

Table

Emp Num	Emp Name	Dept	Mgr	Salary
238	Joe	GSD	240	30000
239	Sid	GSD	240	32000
240	Art	GSD	911	47000
123	Ted	RFA	565	29000
129	Bert	RFA	911	48000
180	Mark	RFA	129	31000
451	Joan	GSD	911	26000
483	Alan	ABM	129	12000
127	Dave	GSD	911	24000
216	Dina	GSD	911	26000
708	Rich	BSD	311	29000
711	Jay	GSD	240	45000

View

Emp Num	Emp Name	Dept	Mgr	Salary
238	Joe	GSD	240	30000
239	Sid	GSD	240	32000
240	Art	GSD	911	47000
451	Joan	GSD	911	26000
127	Dave	GSD	911	24000
216	Dina	GSD	911	26000
711	Jay	GSD	240	45000

Horizontal Projection

Figure 2. Horizontal Projection

Views can also be created by joining rows and columns from other tables and views. This capability makes the usefulness of views one of the most important features in a relational database. A view derived in this form comes in useful in situations where reports are to be produced by joining rows from multiple tables. Figure 3 depicts this concept.

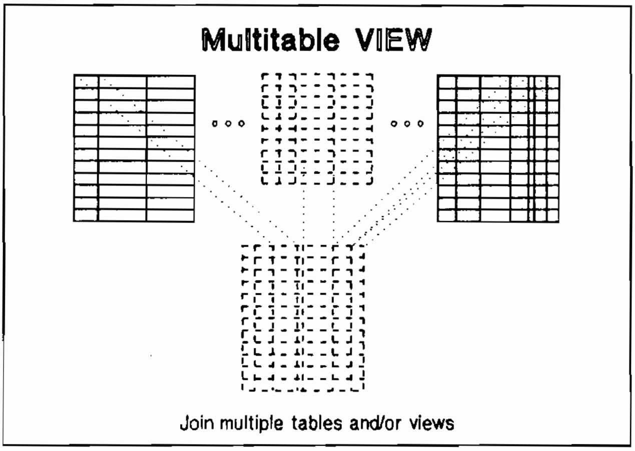


Figure 3. Multitable View

Indexes

To access the data rapidly, indexes can be defined for any column in a table. An index can be defined as containing one or more columns. For instance an index can consist of the concatenation of several columns in any order. Also, an index can be defined as unique or clustered. When an index is defined as unique, rows with duplicate key values cannot be inserted into the table. This index property is useful in tables that contain data such as social security numbers or employee numbers.

On the other hand, when an index is defined as being clustered, HPSQL inserts new rows with similar key values in storage locations that are close to each other whenever possible. This permits faster access of data at the time of inquiry. HPSQL also allows creation of indexes with both properties, unique and clustered. The indexing mechanism used is a modified B-tree structure. This B-tree is characterized by having doubly linked leaves to speed up sequential access in both ascending and descending order.

Database Environment

HPSQL objects and components reside in what is known as DBEnvironment. Figure 4 illustrates an HPSQL DBEnvironment. The DBEnvironment contains a System Catalog, a DBECon file, a log file or optionally, a dual log file. The System Catalog is a set of tables used by HPSQLCore to store the structure and location of the databases. The DBECon file contains global configuration parameters for the DBEnvironment. The log file and the optional dual log file are used to maintain the physical and logical integrity of the DBEnvironment. Details on the use of the log file(s) are given in the HPSQL Transaction Management part of this paper.

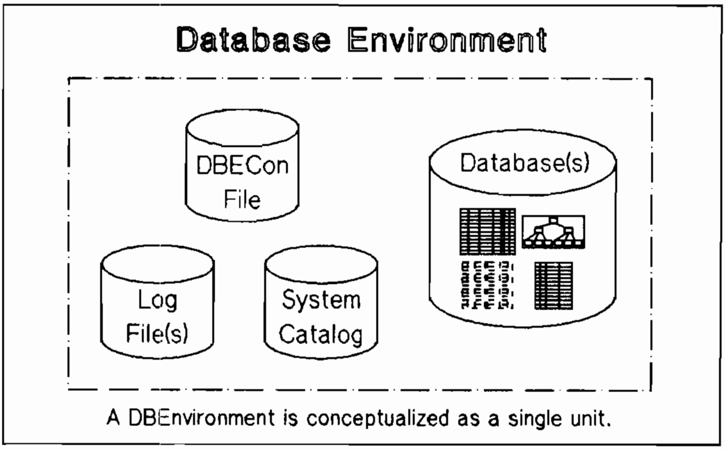


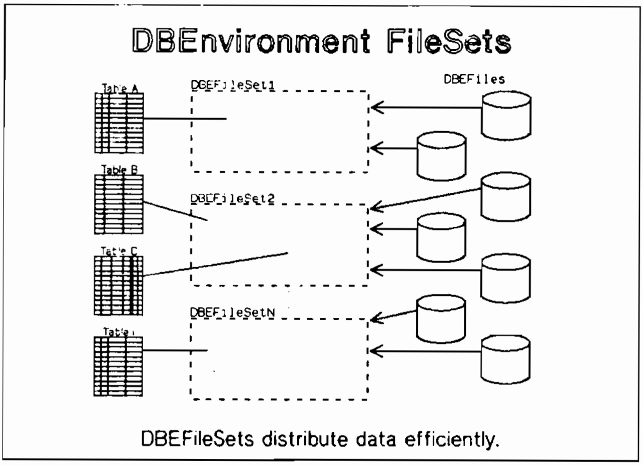
Figure 4. HPSQL DBEnvironment Conceptual Overview

DBEnvironment DBEFileSets

In HPSQL tables are defined to reside in what is known as 'DBEFileSets'. A DBEFileSet is a logical definition of space where a table can reside. This is shown in Figure 5. To obtain actual disc space a 'DBEFile' is added to the DBEFileSet. A DBEFile corresponds to an actual MPE file. Allocating space in this manner users can expand tables in size at any time that space is required by simply adding a DBEFile to the DBEFileSet, ultimately making the table of infinite size.

A DBEFileSet can also be used to cluster data from tables. For instance when two or more tables are accessed together it is possible to assign them to the same DBEFileSet. Tables that reside in the same DBEFileSet share the same DBEFiles causing the data from those tables to be stored physically close to each other. This physical proximity allows for faster access whenever the data from those tables is accessed.

A feature of DBEFiles is that they can be defined to contain data for tables, indexes or to contain mixed data for both. This feature allows for versatile placement of indexes and data from tables since a DBEFile can be moved to a specific disc device.



DBEFileSets distribute data efficiently.

Figure 5. DBEFileSet

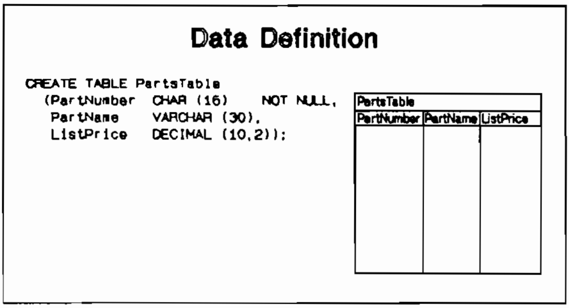
Structured Query Language (SQL)

SQL is an all purpose language that can be used to perform the functions of data definition, data manipulation and database administration. Data definition commands can be used to create DBEnvironment objects and components such as tables, views, indexes, columns within tables, etc. Data manipulation commands are used to modify data in tables, such as adding new rows, deleting rows, updating column values and retrieving data from tables. Data administration commands are used to maintain the DBEnvironment and administer security. Examples on the use of data definition and data manipulation commands are given below, data administration commands will be considered beyond the scope of this paper.

Data Definition

Data Definition commands allow the user to create all of the different DBEnvironment objects such as: tables, views, authorization groups, indexes, DBEFiles, DBEFileSets and Modules.

Figure 6 shows an example of an actual data definition command. This command creates a definition for a table named "PartsTable" with 3 columns.



Data Definition

```
CREATE TABLE PartsTable
  (PartNumber  CHAR (16)     NOT NULL,
   PartName    VARCHAR (30),
   ListPrice   DECIMAL (10,2));
```

PartsTable		
PartNumber	PartName	ListPrice

Figure 6. Data Definition

The first column is named "PartNumber" and consists of 16 alphanumeric characters. The NOT NULL designation forces a value to be specified for this column when new rows are added. The second column is named "PartName" and is defined as a variable character type column of up to 30 characters. This data type, as described in the previous section, means that for every PartName containing less than 30 characters space will be saved. The third and last column is named "ListPrice" and can contain up to 10 digits with the last two digits being to the right of the decimal point. Any arithmetic operations performed with SQL commands will maintain the decimal period in the correct place.

Dynamic Restructuring

Data definition commands can also be used to dynamically restructure a DBEnvironment at the time it is being accessed by other users. The following dynamic restructuring operations can be performed: columns can be added to any table; tables and views can be defined or dropped; and indexes can be added or dropped. When an index is added dynamically, it is available for use as soon as it is created.

Data Manipulation

Data Manipulation commands are non-procedural commands which allow the user to access data in tables or views. Non-procedural means the user does not have to specify HOW the data access is to be performed. All the user needs to specify is WHAT data needs to be accessed and HPSQL optimizes the data access. Data manipulation is categorized in four different operations that are implemented via four different commands: SELECT, INSERT, UPDATE and DELETE. The SELECT command is used to retrieve partial data rows, entire data rows or rows combined from multiple tables (join operation), the INSERT command is used to add data rows, the UPDATE command is used to change column values in rows, and the DELETE command is used to remove data rows.

Each of these commands is described below in some detail.

SELECT Command

The SELECT command is the basic command for data access. Its simplest form is

```
        SELECT columns
        FROM tables/views
```

The user only specifies which columns are to be retrieved from which tables or views. The simplest form of the command shown above retrieves only the columns specified and all of the rows that belong to the specified tables/views. The user can also to retrieve only some of the rows by adding a WHERE clause. The order in which the desired rows are to be retrieved can be specified in the SELECT command by adding an ORDER clause. An example using these clauses follows.

The objective is to obtain an alphabetical listing of the parts and their prices for those parts which cost $10,000 or less.

The SELECT command is:

```
       SELECT PartName, ListPrice
       FROM PartsTable
       WHERE ListPrice <= 10000
       ORDER BY PartName ASCENDING
```

In this example the requested data (PartName and ListPrice) is returned in alphabetical order as specified by the ORDER BY clause. The qualification of rows is given by the WHERE clause, which specifies only those rows that have a ListPrice less than or equal to $10,000.

Qualification operators that can be specified in the WHERE clause are classified as follows:

Comparison:

< <= > >= = <>

Logical:

NOT, AND, OR

Arithmetic:

+ - / * ()

Aggregate Functions

Functions that perform calculations on a column, such as obtaining an average value, are also supported by HPSQL. These are known as Aggregate Functions. A list of these functions along with a short description of them follows:

SUM - Obtains the sum of the rows selected.

COUNT - Obtains the count of the number of rows selected.

AVG - Obtains the average value for a specified column.

MAX - Obtains the maximum value for a specified column.

MIN - Obtains the minimum value for a specified column.

Other Clauses

Other clauses allowed in SQL commands are:

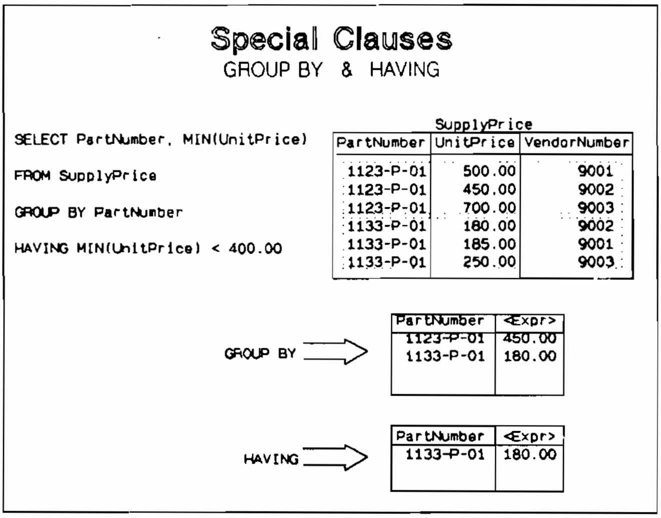
DISTINCT - This clause eliminates any duplicate rows from the resulting table.

ALL - This clause prevents the elimination of any duplicate rows. It is the default; however, it can be included for documentation purposes.

GROUP BY - This clause allows the grouping of rows. It can be used to apply aggregate functions per department, date, etc.

HAVING - This clause is used in conjunction with the GROUP BY clause; it further discriminates the resulting rows after grouping.

An example of using GROUP BY and HAVING clauses is shown in Figure 7.



Special Clauses

GROUP BY & HAVING

```
SELECT PartNumber, MIN(UnitPrice)

FROM SupplyPrice

GROUP BY PartNumber

HAVING MIN(UnitPrice) < 400.00
```

SupplyPrice

PartNumber	UnitPrice	VendorNumber
1123-P-01	500.00	9001
1123-P-01	450.00	9002
1123-P-01	700.00	9003
1133-P-01	180.00	9002
1133-P-01	185.00	9001
1133-P-01	250.00	9003

GROUP BY ⟹

PartNumber	<Expr>
~~1123-P-01~~	~~450.00~~
1133-P-01	180.00

HAVING ⟹

PartNumber	<Expr>
1133-P-01	180.00

Figure 7. Special Clauses

Joining Tables

The SELECT command can also be used to join rows from multiple tables. Table joins are easily accomplished using the same syntax shown earlier. As in a simple SELECT, the user specifies which columns are to be retrieved; the FROM clause lists the tables involved; and the WHERE clause specifies which column values from one table are to match column values from other tables for the join criteria. An example is shown in Figure 8.

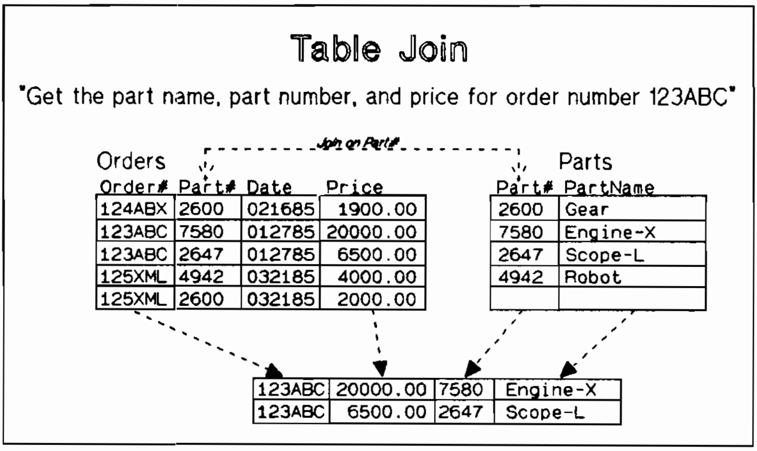


Table Join

"Get the part name, part number, and price for order number 123ABC"

Orders

Order#	Part#	Date	Price
124ABX	2600	021685	1900.00
123ABC	7580	012785	20000.00
123ABC	2647	012785	6500.00
125XML	4942	032185	4000.00
125XML	2600	032185	2000.00

Parts

Part#	PartName
2600	Gear
7580	Engine-X
2647	Scope-L
4942	Robot

123ABC	20000.00	7580	Engine-X
123ABC	6500.00	2647	Scope-L

Figure 8. Table Join

In this example the required data exists in two tables. The table Orders and the table Parts need to be joined to obtain a resulting table. The common join column is PartNum (shown as Part# in Figure 8). This column can be used in the WHERE clause of the SELECT command to satisfy the join criteria.

The SELECT command for this query is:

```
SELECT OrderNum, Price, Parts.PartNum, PartName
FROM Orders, Parts
WHERE Orders.PartNum = Parts.PartNum
```

In this example the table name is also used in the column list to further eliminate any ambiguity because the PartNum column is found in both tables.

INSERT Command

The INSERT command is used to add rows to tables. This command permits the user to specify the columns for which values are to be added to the table. It also allows specification of null values for any column that accepts them. This command is also used to add rows to views that are derived from one table. The simplified syntax of this command is:

```
[BULK] INSERT INTO Table/View
       [ColumnName1, ColumnName2, ...]
       VALUES {ColumnValues | BulkValues}
```

The syntax for this command allows for bulk insertion of rows. This capability, allowed only programmatically, is discussed in more detail in the programmatic access section of this guide.

DELETE Command

The DELETE command is utilized to remove rows from a table. Once a row has been deleted, the space that it was using is released and used by other rows being inserted. The simplified syntax of this command is:

```
DELETE FROM Table/View
  [WHERE SearchCondition]
```

Note that this command contains an optional WHERE clause to specify which rows are to be deleted from the table/view.

UPDATE Command

This command is utilized to change column values. The simplified syntax is:

```
UPDATE Table/View
  SET ColumnName1 = {Expression | NULL}, ...
  [WHERE SearchCondition]
```

Note that, as the syntax shows, column values can be set to a value of an expression or to NULL. The WHERE clause can be used to specify which rows in the table/view are to be updated. The absence of the WHERE clause implies that all rows should be updated.

HPSQL Security

HPSQL has one of the most comprehensive security schemes ever offered by any of HP's data management products.

Security Overview

HPSQL security is seen in terms of 'authorities'. An authority is a right to access a given DBEnvironment or DBEnvironment object in a specific way. To read or modify a given object, the user must first be 'granted' the authority to do so. In other words, a user cannot perform any functions on the database until permission is given to do so. Authorities to a user can also be taken away, or 'revoked.'

DBEUserIDs and Authorization Groups

Within HPSQL each system user has a corresponding 'DBEUserID' which is derived from the logon id assigned to them. The DBA can assign authorities to each user on individual basis or through 'Authorization Groups' which are logical groupings of users. Desired authorities can be granted to the authorization group and any users that are members of the group will receive them automatically. Any new DBEnvironment users can be added to the authorization groups to obtain the required authorities. Conversely, when authorities are dropped from the authorization group, the users belonging to the group loose the corresponding authorities

Types of Authorities

There are four types of authorities that can be granted in HPSQL:

- Special Authorities: DBA, CONNECT, RESOURCE

- RUN Authority

- Owner Authority

- Table/View Authorities: SELECT, INSERT, UPDATE, DELETE, ALTER & INDEX

A detailed description of each of these authorities follows.

Special Authorities

These authorities allow the user to perform a variety of functions within the DBEnvironment, from accessing it to performing database administration tasks. There are three special authorities: DBA Authority, Connect Authority and Resource Authority. DBA Authority (database administrator authority) is given automatically by HPSQL to the user that configures the DBEnvironment. Having this authority means that a user can perform the tasks of resource management (allocating disc space), archival and recovery, security management, as well as enable the DBEnvironment for single or multiuser access. This authority can be granted to other users to share the database administration functions and responsibilities.

Connect Authority is required for any user that needs to access the DBEnvironment. The DBA can revoke this authority from any user or users to prevent them from accessing the DBEnvironment without having to revoke their other authorities. Having connect authority does not imply any other authorities. The user still has to be granted other authorities to perform any functions in the DBEnvironment.

Resource Authority allows a user to create database objects such as tables, views and authorization groups. This authority can be granted to users to help the DBA with the task of defining the databases in the DBEnvironment.

Run Authority

This authority is utilized to control the access to the application programs that access the DBEnvironmert. With this authority the DBA or the owner of the program can specify which users can run which application programs. Having run authority for a specific program allows a user to run the program. The user running the program can perform any of the functions that the program is allowed to do, but only through the program.

Table/View Authorities

These authorities can be grouped into two categories. The first category consists of authorities that allow the user to modify data in a table or view; the second category consists of those authorities that modify the table itself. In the first category the authorities are: insert, select, update and delete. These authorities are independent of each other, which means that select authority only allows the user to select data, insert authority only allows data insertions, etc. The users must be granted each authority explicitly, since having one authority does not imply other authorities. In the second category, authorities which modify the table itself, there are two authorities that qualify: Alter and Index. Alter authority allows the user to add columns to a given table; Index authority allows the user to create or drop an index from a table.

Owner Authority

Owner authority is not an authority that is granted to users as are the other authorities. A user becomes the owner of an object at the time the object is created. For the owner of an object certain authorities are implied automatically. Those authorities are: select, insert, update, delete, alter and index. Another way of obtaining ownership of an object is by having it transferred explicitly via an SQL command. Only the owner of the object or the DBA can transfer object ownerships.

Object Ownership

Ownership of an object is an important concept in HPSQL since the owner can grant access privileges on the object itself. In HPSQL, object ownership can be established in such a way that multiple users or just the DBA can own any object in the DBEnvironment.

Transaction Management

As in any information system one of the most important concerns is that of data integrity. HPSQL offers a very comprehensive scheme to ensure data can be recovered from any type of catastrophe that may occur. A transaction in HPSQL is considered a user defined unit of work that is indivisible, meaning that either the entire transaction or none of it occurs.

Transaction Definition

A transaction in HPSQL basically consists of a number of SQL commands delineated by a 'BEGIN WORK' and a 'COMMIT WORK' pair of commands. Refer to Figure 9. The BEGIN WORK command specifies the beginning of the transaction and the COMMIT WORK specifies the end. If the user forgets or does not specify a BEGIN WORK command, one will be issued automatically, thus ensuring that all work is defined as transactions within HPSQL.

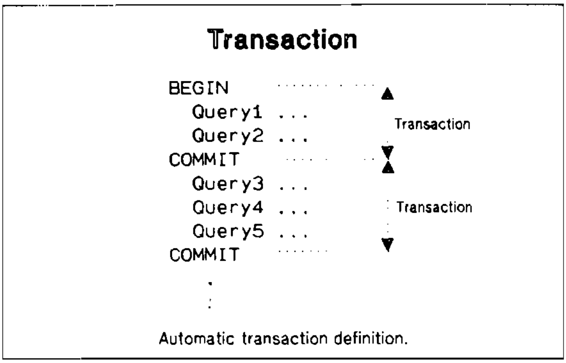


Figure 9. Transaction Definition

Committing Transactions

As stated earlier, a COMMIT WORK command specifies the end of a transaction. Once a transaction has ended, HPSQL ensures that it becomes permanent in the DBEnvironment by flushing all of the modifications to the log file. If the transaction does not end with a COMMIT WORK (due to a failure) none of the transaction is recorded in the DBEnvironment.

Transaction Rollback

The modifications made by a transaction in progress can be undone before it is committed. This is known as 'transaction rollback'. This feature comes in handy when certain conditions occur in the course of a transaction that invalidate any of the modifications that have occurred. When a transaction rollback occurs, all of the modifications that occurred during it are taken out, leaving the data in the same condition that it was in before the transaction started. Thus the DBEnvironment remains in a perfectly consistent state.

Transaction Savepoints

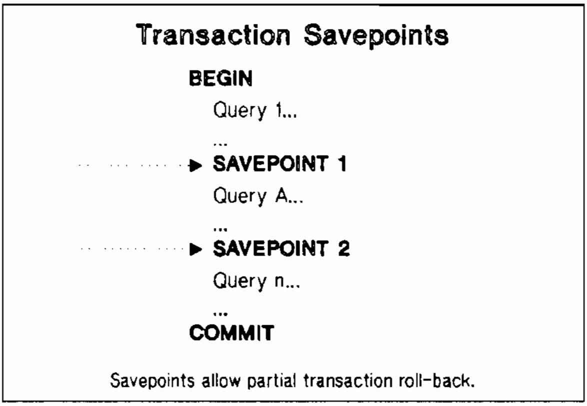
HPSQL also supports partial transaction rollback by allowing the user to issue savepoints within the transaction., Refer to Figure 10. The user can rollback the transaction to any savepoint previously established in that transaction. The feature is helpful in situations when only part of a transaction needs to be undone. In the event of a system crash, DBCore rolls back to the beginning of the transaction, regardless of any savepoints, to ensure logical consistency of the data.

Transaction Locking

HPSQL supports Implicit or Explicit locking. Implicit locking is accomplished via the actual data manipulation command issued. The command directs DBCore to lock the pages of data that are being accessed, allowing multiple users to read the data but only one user to modify it at a time. Explicit locking can be accomplished by issuing an SQL lock command to lock any table that will be used in exclusive mode. All data locked by either of these two methods gets unlocked automatically at the time the transaction ends, by either being committed to disc or rolled back.

Deadlock Protection

In any system that provides automatic locking, some deadlocks are bound to occur. HPSQL provides a mechanism to resolve this situation. Since HPSQL performs all locking automatically, it first verifies that no deadlocks would occur. If any potential deadlock situations are detected, HPSQL utilizes a mechanism to isolate the offending transaction and rolls it back. This frees up the resources and allows the processes involved to continue their transaction.



Transaction Savepoints

BEGIN
Query 1...
...
SAVEPOINT 1
Query A...
...
SAVEPOINT 2
Query n...
...
COMMIT

Savepoints allow partial transaction roll-back.

Figure 10. Transaction Savepoint

Transaction Recovery

HPSQL also offers a mechanism which permits recovery of the DBEnvironment from any type of failure. The failures that HPSQL can recover from are categorized as soft and hard. A soft failure is any software failure causes an HPSQL abnormal termination such as program aborts, operating system failures and extended power blackouts that drain the reserve battery. A hard failure is one that causes permanent loss of data, such as disc head crashes.

Transaction Logging

HPSQL utilizes a permanent log file to perform transaction recovery. This log file can be placed on a device different from the device or devices where the databases reside to improve transaction throughput. To further ensure recovery, an optional dual log file can also be specified for the DBEnvironment. The second log file should be placed on a different disc device from the first log file. The dual log file, when enabled, is utilized automatically by HPSQL to recover the DBEnvironment in the case that the first log file becomes damaged.

Soft Failure Recovery

Recovery from soft failure is simply accomplished. Recovery from program aborts occurs automatically by a special HPSQL process that rolls back the transaction in progress. All the user needs to do to recover from system failures and power blackouts is access the DBEnvironment. Recovery is performed automatically by HPSQL by rolling back all of the transactions that were in progress at the time of the failure.

Hard Failure Recovery

Recovery from a hard failure is still simple but requires manual intervention. This type of recovery requires a DBEnvironment backup or 'archive' and log file backups. The log file or dual log files being used at the time of the failure can also be utilized for recovery if they were not damaged. Using all of these files for recovery restores the DBEnvironment to the last logical and physical consistent state it was before the failure. Figure 11 depicts the different files that can be used for recovery.

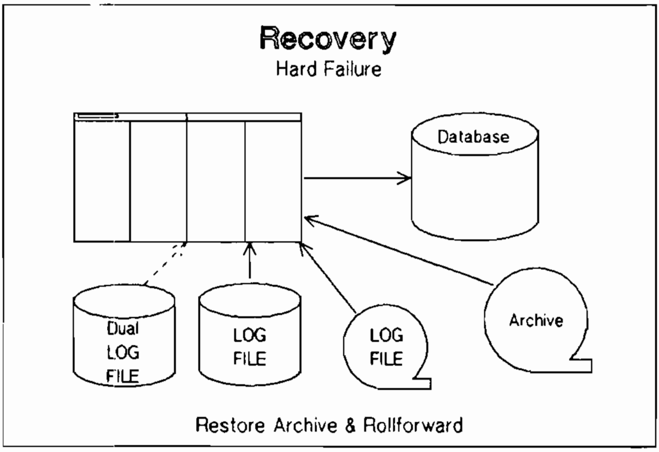


Figure 11. Hard Failure Recovery

ISQL Interactive Interface

HPSQL also offers an interactive interface named ISQL. This interface is designed to allow the user to perform multiple functions in one program. ISQL can execute all of the SQL commands (with the exception of those commands that are allowed programmatically only), plus some commands of its own for utility and maintenance functions. With this interface program all of the data definition, data manipulation and database administration functions can be performed via an easy-to-use command-driven interface.

Help Facility

ISQL has a friendly help facility which provides information on the use of commands along with syntax and examples. This help facility is similar to help facilities offered in other HP software products.

Profile File

ISQL also supports the use of profile files. These files contain commands to set up the ISQL session such as the subsystem prompt, the output line width and others. A Profile file can also be set up in a way that will automatically connect ISQL to a predefined DBEnvironment, thus permitting more control over the users.

Prompting Mode

One of the most helpful features of ISQL is that it provides a prompting mode for ISQL commands. The prompting works as follows: whenever an ISQL command is left unfinished, ISQL shows the different options for the user to choose. At this time the user can type the desired option or cancel the command. The user can proceed in this way until the options for all of the parameters for the entire command are entered or canceled. This allows the user to quickly start using ISQL without spending time memorizing commands or thinking about command syntax.

Command Files

The use of command files is also supported by ISQL. A command file is a file containing either ISQL or SQL commands. These files can be used to store command sequences which are used often. This saves the user from having to type the same commands every time.

Command History Buffer

Another attractive feature of ISQL is the fact that it holds the last five commands that have been issued in a Command History Buffer. These commands can be listed, edited and reissued if desired. The facility for editing commands is very similar to the MPE V REDO command, but with several enhancements.

Synonym Files

Another type of file supported in ISQL is a synonym file. This type of file can be used to designate alternate names for SQL and ISQL commands. This means that any command can be known to ISQL by more than one name. This allows users to utilize terminology that is familiar to them, speeding up the time that it takes to use ISQL. An inherent feature that results from the use of this file is that certain commands can be disabled for some users of the DBEnvironment, permitting tighter security.

Data Loading/Unloading

ISQL also has a facility for loading and unloading data to or from the DBEnvironment. This facility supports two file formats, internal and external. Internal format files are specially formatted files that are managed by ISQL only and can be utilized to perform maintenance functions in the DBEnvironment. Such as unloading and loading a table to achieve a higher degree of clustering, thus speeding up data access. Externally formatted files are ASCII files that can be managed by users and ISQL. This type of file can also be used to load tables. The load command allows the user to specify the relative position of each column within the external data file, permitting the use of data that is not in the same order as the table being loaded. Data can be easily unloaded from any table or view or by optionally specifying a tailored SELECT command with any of the options that were discussed earlier. The load/unload facility of ISQL permits the user to perform many maintenance functions in the DBEnvironment without the need for writing application programs.

HPSQL Programmatic Interface (Preprocessors)

The programmatic interface of HPSQL features a simple interface that involves the use of preprocessors. The user does not have to learn a separate set of procedures to access an HPSQL DBEnvironment programmatically. Programmatic database access and manipulation of data can be performed using the same easily mastered SQL language which is used for interactive access.

Preprocessors Overview

As mentioned above, the programmatic interface uses preprocessors. The user embeds SQL statements in source programs. The source programs are then submitted to a preprocessor which reads the source and produces a modified source program. The modified program still contains the SQL statements in comment form. Following these comments, the preprocessor inserts new statements which consist of actual procedure calls to perform the functions that the SQL statements were requesting. Additional compiler include files are also generated containing ancillary parameters supporting the newly created procedure calls. Figure 12 illustrates this process.

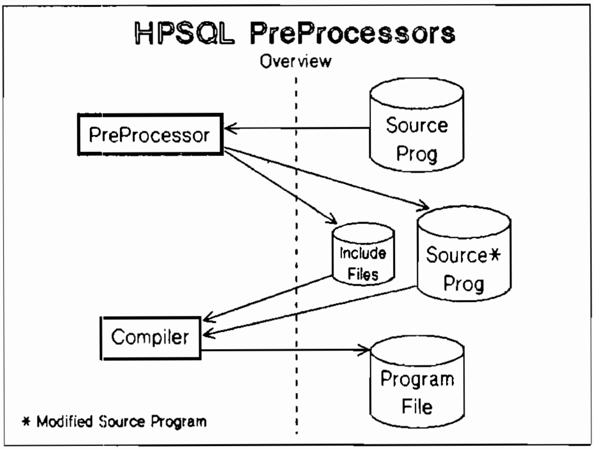


Figure 12. HPSQL Preprocessor Overview

Embedded SQL

The embedded SQL statements in the source program are written in the same easy-to-learn language that was described earlier, with a few elements added. Consider for example the following SQL statement:

```
SELECT PartName
FROM Parts
WHERE PartNum = '123ABC'
```

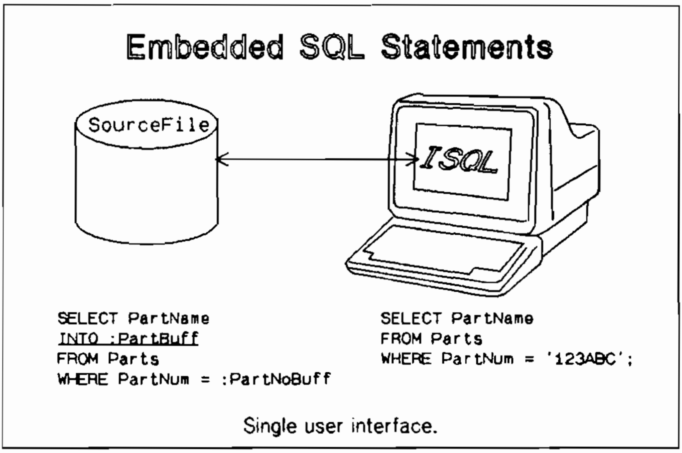
This statement, to be used programmatically, requires that a buffer be specified to receive the information that is being selected (PartName in this case). The buffer parameter is specified via a program variable (also known as host variable) in the special INTO clause of the SELECT command as follows:

```
SELECT PartName
INTO :PartBuff
FROM Parts
WHERE PartNum = '123ABC'
```

The only difference between this command and the previous shown is the addition of the INTO clause with a buffer designation of :PartBuff to indicate where the requested data is to be returned. While this SQL statement returns the PartName for PartNum = '123ABC' only, it is possible to request information for other PartNames. The value setting for PartNum can also be substituted for a host variable to be able to request information for any PartNum other then ' 123ABC'. The SELECT statement with a host variable will now look like this:

```
SELECT PartName
INTO :PartBuff
FROM Parts
WHERE PartNum = :PartNoBuff
```

As this example indicates, the same basic SQL statements that are used interactively can be embedded in source prog ams. Thus, the user needs only one language to accomplish both types of access.



```
SELECT PartName
INTO :PartBuff
FROM Parts
WHERE PartNum = :PartNoBuff
```

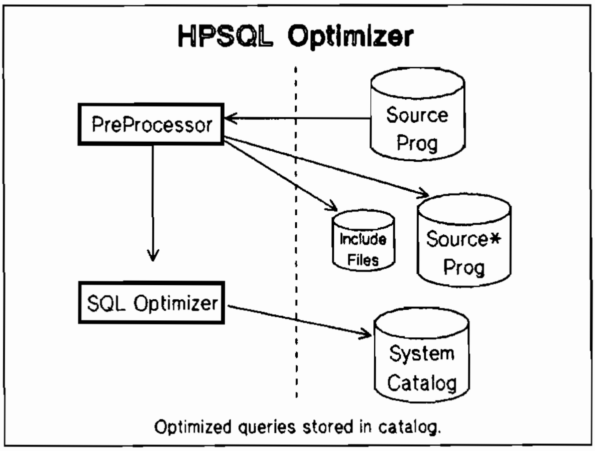
```
SELECT PartName
FROM Parts
WHERE PartNum = '123ABC';
```

Single user interface.

Figure 13. Embedded SQL

HPSQL Optimizer

The HPSQL preprocessors perform several tasks during the preprocessing phase. One task is that of parsing the embedded SQL statements and verifying that no syntax errors are present. Another task performed by the preprocessor is to optimize the data access to speed up data manipulation. The preprocessor accomplishes this task by calling a query optimizer which is a component of HPSQL. This is shown in figure 14. The optimizer calculates a cost function for the different ways to access the data, taking into account the indexes defined on the tables being accessed through the query. After calculating the best cost function, a data access module known as a section is stored in the System Catalog. The preprocessor generates the code in the modified source program to utilize the stored sections at run time. HPSQL makes sure that any changes made to the structure of the data -- whether indexes are added, dropped or security is changed -- are reflected in the stored sections to maintain the optimum access module at all times.



Optimized queries stored in catalog.

Figure 14. HPSQL Optimizer

Bulk Access

Another programmatic function provided by HPSQL is that of 'Bulk Access.' This type of access can be used for data input or output. To perform bulk access, the user defines an array (also known as Row Buffer) in the data area (or stack) of the program. Through this row buffer the user can request to SELECT or INSERT multiple rows at the same time, thus minimizing disc access. Figure 15 summarizes this type of access.

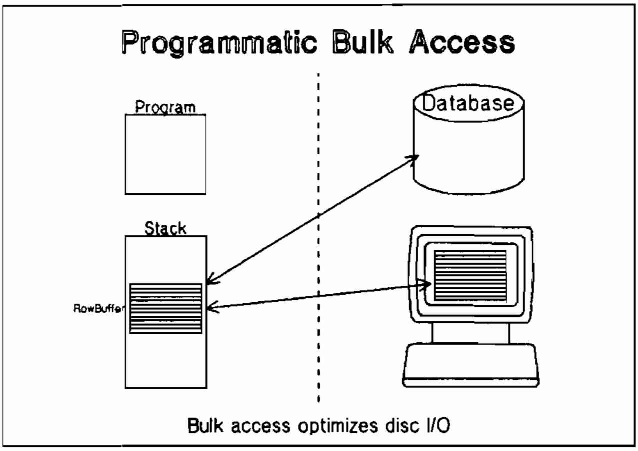


Figure 15. Bulk Access

Summary

The capabilities offered by HPSQL can be summarized as follows:

- Database environment (DBEnvironment) that supports multitable views with built-in logging and is treated as a unit.
- Powerful multipurpose language that can be used dynamically in interactive or programmatic modes.
- Comprehensive security mechanism that assures that only authorized user are allowed to access the DBEnvironment.
- Transaction management mechanism that ensures physical and logical integrity.
- Multifunction interactive interface (ISQL) that facilitates tasks of database administration, maintenance as well as satisfying ad-hoc query use.
- Programmatic interface that improves programmer productivity with a built-in optimizer.

HPSQL is a versatile DBMS that offers a set of features that are suitable for information systems that are very dynamic in nature.

Memory Mapped File Access on the HP3000 900 Series
Exploiting HP Precision Architecture

by
Winston Prather
Hewlett Packard
Information Technology Group

Abstract

The MPE XL File System was designed to provide high speed file access with low overhead. These goals are achieved by taking advantage of design features of the HP Precision Architecture. This presentation will address the methods used by the new file system to maximize performance while maintaining source and object code compatibility with MPE V. A new file access method, known as **User Mapped File Access**, will also be discussed.

Memory Mapped File Access on the HP3000 900 Series
Exploiting HP Precision Architecture

by
Winston Prather
Hewlett Packard
Information Technology Group

Presentation Overview

The MPE XL File System was designed to take full advantage of the HP Precision Architecture and (at the same time) ensure source and object code compatibility with previous versions of MPE by supporting the MPE V intrinsic interface. The new file system also provides an additional file access method known as User Mapped File Access. This access method, which also exploits the HP Precision Architecture, eliminates file system overhead to increase file access speed.

HP Precision Architecture

Before understanding how the file system maximizes performance it is necessary to become familiar with some basic features provided by HP Precision Architecture. Specifically, an understanding of the large address space and how it is utilized must be achieved. The architectural features mentioned in this presentation are only a small subset of the total architectural design.

First implementations will provide for a 48 bit virtual address. This allows 256 trillion bytes of information to be assigned virtual addresses (mapped into the virtual address space). Once this mapping has occurred, the data can be accessed directly by memory LOAD and STORE instructions. This means the LOAD and STORE instructions use a virtual address as the source or target of the operation.

Obviously the machine does not have 256 trillion bytes of physical memory. Therefore, a mechanism must be provided to manage the swapping of data between virtual and physical memory. Virtual memory is defined as all code and data that has been mapped into the virtual address space.

In the HP Precision Architecture, physical memory is allocated on a demand basis. This means that when a LOAD or STORE instruction executes, the hardware determines if the requested data is in physical memory. If the data is not present, it is moved into physical memory (via disc I/O) and the instruction is restarted.

In order to facilitate the swapping of data between physical and virtual memory, both the virtual address space and physical memory are logically divided into 2Kb sections called pages. When a reference is made to a virtual address, the entire page that contains the referenced data is brought into physical memory. This type of virtual memory management system is known as demand paging. Disc caching is inherent in this type of architecture.

MPE XL File System Exploits the HP Precision Architecture

Having explained a few of the fundamental concepts upon which the HP Precision Architecture is based, techniques used by the MPE XL file system to to increase file access speed can now be examined.

The MPE XL File System exploits the large address space and virtual addressing of the new architecture by mapping all open disc files into this virtual address space. Once this mapping has occurred, the file system uses machine LOAD and STORE instructions to access the file. No explicit buffer management is necessary. The overhead involved in file access is substantially less, in most cases, than that of MPE V.

The first time a reference is made to a page within a file, a page fault (absence of a page) will occur. The Virtual Memory Management System will move the correct virtual page into physical memory. Processing can then continue. The next time a reference is made to the same file page the data will already be present in physical memory. No subsequent I/O will be needed.

An example can be used to illustrate this. Consider a program that opens a file using FOPEN, reads the first record using FREAD, updates record 10 using FWRITEDIR, and then closes the file with FCLOSE. How the file system handles each of these intrinsic calls can now be examined.

First, FOPEN is called. As usual the file system validates parameters, checks security, etc. In addition, FOPEN will request the Virtual Memory Management System to map the file into the virtual address space (assign every byte of data in the file a unique virtual address). The file system will retain the starting virtual address of the file for use by later intrinsic calls.

Next, the user calls FREAD to read the first record of the file. Again, the file system validates parameters, etc. At this point, in a traditional architecture, the file system would ask the I/O system to read the data from the file into a local file system buffer. This is not the case in a virtual memory demand paging environment. In this environment, the file system will calculate (using the current record pointer, record structure, and starting virtual address) the virtual address of the requested record. The file system will then do LOADs from this calculated virtual address and STOREs to the virtual address of the target buffer passed to FREAD.

A call to FWRITEDIR works similarly. The target address is calculated by multiplying the record number by the record length and then adding this value to the starting virtual address of the file. LOADs are then performed from the virtual address of the source buffer, passed to FWRITEDIR, and STOREs are performed to the calculated virtual address within the file. Page faulting could occur during these LOADs and STOREs and would be handled by the Virtual Memory Management System.

Lastly, a call to the FCLOSE intrinsic would ensure all file pages currently in physical memory were posted to disc and request the Virtual Memory Management System to un-map the file from the virtual address space. The address space that was used for the file is now free to be used for other files, data structures, etc.

Prefetching

The MPE XL File System further increases file access speed by prefetching file pages into physical memory prior to their actual reference. This is accomplished by determining if a future LOAD or STORE to the file is going to cause a page fault. If a page fault is inevitable, the file system will notify the Virtual Memory Management System that a needed page is not present in physical memory. The file system then treats this as an opportunity to request additional file pages. For example, if a file is being accessed in a sequential manner (FREAD), it is likely that accesses to file pages directly following the current file page will occur. If the operating system brings not only the currently needed page into memory, but also the next few pages, the time waiting for absent pages will be reduced thus increasing the program execution speed.

Not only does the new file system decrease file access time with this technique, but also adjusts to the programs demand via heuristics. For example, if a program is consuming a file at a high rate of speed, a larger prefetch will be done. Prefetching is adjusted based on application demand, system loading, and available cpu.

User Mapped File Access

A new file access method, known as User Mapped File Access, will be available to users of the MPE XL operating system. This method builds on the demand virtual memory environment discussed previously, but eliminates file system overhead. User Mapped File Access does not use intrinsics to access files. Instead, FREADs and FWRITEs are replaced by high level language assignment statements for which compilers generate machine LOADs and STOREs. Additionally, this new access method also allows programs to treat files as shared virtual memory, providing an extremely flexible migration path for extra data segments. With User Mapped File Access programmers will be able to define multiple views of the data in files with no concern for the physical file structure.

Examples of this technique will be shown in the presentation.

MPE XL File System

The MPE XL File System takes maximum advantage of the foundation laid by HP Precision Architecture to provide high speed file access. Inherent caching, along with the elimination of explicit buffer management decrease the overhead involved in file access. Prefetching and heuristics provide the ability to maintain the optimum number of file pages in physical memory. These features, combined with the new file access method, User Mapped File Access, provide high performance and extreme flexibility of file access and management for users of the MPE XL operating system.

HP 3000 Object Code Compatibility
on Future HP 3000 Systems

by

R. Gregory Stephens
Hewlett-Packard
Information Technology Group

Abstract

With the introduction of HP Precision Architecture based HP 3000 systems the question of Object Code Compatibility between HP 3000 systems based on differing architectures arises. This lecture addresses this topic by presenting an overview of the technology developed to provide compatibility between these systems at the object code level.

As a foundation for addressing this topic the lecture will provide a technical definition of Compatibility Mode and Native Mode followed by a description of the technology itself. The technology consists of an Emulator and Object Code Translator which will be discussed in detail.

HP 3000 Object Code Compatibility on Future HP 3000 Systems

by

R. Gregory Stephens
Hewlett-Packard
Information Technology Group

SYNOPSIS

MPE V Programs on MPE XL

To provide compatibility between MPE V based systems and MPE XL based Precision Architecture systems Hewlett-Packard has invested a significant amount of effort in providing a **Compatibility Mode environment** that allows programs written on MPE V based systems to run without changes. It is this Compatibility Mode environment that will allow Customers, Third Party Software Developers, and Hewlett-Packard itself, the ability to execute applications without changes. (Note: In order to distinguish between the two different architectures that now make up the HP 3000 line of computers I will use the term "Stack 3000" to refer to that series of machines 37-70 using the stack architecture and the term "Register 3000" to refer to the 900 Series of machines using the HP Precision Architecture.) Turbo Image and SPL are two examples of HP software which take advantage of Compatibility Mode.

In contrast to Compatibility Mode, **Native Mode** allows users the ability to take advantage of many of the new features the HP Precision Architecture affords including 32 and 64 bit addressing and mapped files as well as providing the best performance with source code compatibility. By re-compiling source code with the MPE XL version of the compilers user can take advantage of Native Mode features and performance. ALLBASE and the majority of the MPE XL Operating System are examples of HP software which take advantage of Native Mode.

Programs executing in Native Mode can also call procedures which reside in Compatibility Mode SLs via the MPE XL Switch Subsystem. The Switch Subsystem also provides the ability for Compatibility Mode programs to call procedures which reside in Native Mode Libraries. ALLBASE' TurboWindow and the MPE XL File System are examples of HP software that takes advantage of the MPE XL Switch Subsystem to perform dual mode execution.

To achieve the high degree of compatibility that MPE XL provides, the 16 bit stack with DL, DB, Q, S, and Z pointers is created for use by MPE V programs executing in Compatibility Mode. The MPE V program structure is also maintained including use of the P, PB, and PL registers as well as the structure of the STT.

Emulator

The first method of supporting the Stack 3000 instruction set on the Register 3000 machines is the Emulator. When MPE V programs are restored on MPE XL based systems the user types the :RUN command at the MPE prompt and the program will run under the Emulator, transparent to the user.

The Emulator interprets the Stack 3000 instructions at run time in a manner similar to what the microcode of a Stack 3000 does. A detailed discussion of how the Emulator interprets MPE V programs will be presented in the lecture.

Object Code Translator

The second method of supporting the Stack 3000 instruction set is via the Object Code Translator (OCT). Just as the Emulator was analogous to an interpreter, the Object Code Translator is analogous to a compiler. Inherent in this approach is the requirement of a translation step just as a compiler requires an initial compilation step before execution of the program.

The Object Code Translator will accept as input an MPE V program, it creates a duplicate of this MPE V program, translates the Stack 3000 instructions in the input file into Register 3000 instructions and appends the output of the translation to the output MPE V program.

When the translated program is run, it still operates in Compatibility Mode with the familiar 16 bit stack but the translated code which was appended to the program file is executed. This approach saves the decoding and interpreting of the Stack 3000 instructions at run time, as the Emulator does, plus it allows optimizations to be made, all of which results in significantly better performance

A detailed discussion of the Object Code Translator will be presented in the lecture.