HP 3000 IUG COPENHAGEN

1982

HP 3000 INTERNATIONAL USERS GROUP

1982 HP 3000 INTERNATIONAL CONFERENCE
COPENHAGEN OCTOBER 25–29
SCANDINAVIA HOTEL
COPENHAGEN, DENMARK

PROCEEDINGS

STAFF

William Crow	Executive Publisher/ Association Manager
Renaye Lee	Conference Manager
John Knapp	Publisher
Sandra Hawker	Managing Editor

CONTENTS

HP-3000 COMPUTER NETWORK MANAGEMENT

Haye Swart Ir ,Information Manager Thomassen &
Drijver - Verblifa N.V. Deventer The Netherlands

ABSTRACT:

Management, organization and control of a network of 8 HP-3000 minicomputers located at 7 different locations within the Netherlands and Belgium, serving 10 different production plants. The network operates 'in a decentralized environment with central control and development. It supports on a highly interactive way the day to day operations and it is used for (central) batch processing as well. This paper deals with the functional and technical control of the application systems ,the hardware at the different locations and the organisational consequences of same. It goes into the back-up and security measures and the way the internal usage charges and resource utilization is administrated.

INTRODUCTION:

- company overview

THOMASSEN & DRIJVER -VERBLIFA N.V.,TDV is a major Benelux manufacturer in the packaging field. The main products are tinplate containers and glassclosures for the beer and beverages, food and non-food markets. Besides it manufactures steel drums and pails as well as all types of plastic containers and vending machine cups. TDV is a daughter of the Continental Can Corporation, the packaging division of the Continental Group Inc. Stamford,USA. The headoffice of TDV is located in Deventer the Netherlands with a fiscal Belgium headoffice near Antwerp.TDV has 9 production plants within the Benelux and a Deventer based machine shop. Its personnel numbers over 5800, while net sales in 1981 passed the 1 billion dutch guilders mark.

- network (plan) 1976-1982

In 1976 a plan was approved to gradually install a minicomputernetwork with roughly as main objectives:

to replace the central batch processing and the decentral magnetic ledger processing,

to introduce where practical interactive processing throughout the company as a whole ,and

to decentralize responsibility for non-redundant input.

A network concept was chosen instead of a mainframe concept, because it would allow to store and handle bulkdata at the place of origin.In addition we would have a gradual hardware expansion(sliding investment)and we did not have to close our shop because of a gigantic conversion effort. As a matter of fact the old batch processor will be released for retirement in the first quarter of 1983.

result: We have more than benefited from the new improvements in hardware and software during the birthcycle.We installed for instance last may the HP- 3000-64 instead of the originally planned two HP-3000-III computers. For 4 years we had nearly time to do more than necessary maintenance on existing systems because we were always busy redesigning the old batch systems for the new environment.

- typical decentral operation

The raw materials and finished goods movements are monitored by computer from planning into production (orders),quality control and warehouses.Storage locations are registrated with aid of printed barcode labels. Personnel attendance is monitored by timereporting terminals at the plant entrances. This provides input to the personnel and the payroll systems. Furthermore data input preparation is done locally for all other central or decentral systems, such as financial reporting.

Typical hardware configuration: HP-3000-III with tape and 1 MB internal- and 120/240 MB external - memory, 35-40 video-,printer-,barcodereader-, timereporting -terminals, DS-3000, INFORM-3000 and future MM-3000 software.

- strategies

The philosophy behind the automation within TDV can be summarized as follows:

.high user involvement,therefore users responsibility .uniform systems,but with a local face .onetime input of data,but simplicity above (super) integration .no local EDP groups,but user friendly retrieval tools

The result of above strategy is that development time tends to become relatively long. However, the degree of user's acceptance is high. Quick short haul successes occur very seldom.

ORGANISATION AND CONTROL

- strategic planning

strategic committee The company has formed a strategic committee that advises the board in all matters concerning information processing ing. EDP is a major part of this. This committee is formed by line- and staff-managers at one level below the board. Chairman is boardmember and V.P Finance and Administration. It convenes at a two to three month interval.

The major tasks of this group are to set rules,to initiate and review (middle) long term planning for investments and application development. It channels decisions thru the lineorganization and forms a possible sounding board for unsatisfied or impatient users.

result: The automation is an integral part of the organisation. It allows EDP to enforce for users impopular decisions. It enforces EDP at the other hand to make more elaborate groundwork before decisions can be made.

- project organization

project groups For all automation projects larger or smaller project teams are formed.They are composed of users and the necessary specialists,including one systems designer. The latter on his turn is EDP internal projectleader of a team of analysts and programmers. The projectgroup is lead in all instances by a user, preferable from the TDV line organization.

result: Users in the group are forced to take responsibility for the system concepts. They are however seldom made sufficiently free from other duties.

steering committee Two steering committees are in operation to monitor the different major projects. The fixed members are chosen from the strategic committee. The systemsdevelopment manager is permanent member. Depending of the project a different user manager is added. The steering group takes decisions towards the project group involved and prepares decisions to be taken by the strategic committee. It reports in the latter about progress.

result: Projectprogress is greatly enhanced since this committees are placed between projectgroups and strategic committee.

- (application) systems organization

systems controllers As soon as an application system is implemented and accepted by the user, the functional control of that system is put into their hands. Somebody within the user organization is made responsible. He guides the users how to work with the system, teaches its (im)possibilities and prepares user documentation. All contacts concerning this system with EDP such as requests for change(development group) and processing requests(data centre/network) are approved and channeled thru him. He contineously needs to evaluate the cost/benefit and the appropriateness to the organization. Yearly he prepares together with EDP the budgets for the next year. Monthly he receives an account of the actual cost breakdown.

data controllers In a decentralized environment with different computers on which run "cloned systems with local faces" ,it is man- datory to install only one system controller per system to prevent divergence.It is physical impossible for him to have a daily control over the processing at the different locations. He therefore delegates part of his respon- sibilities to a local data controller. He in fact has the EDP processing contacts and guides the local users.

result: Correct use of a system is made a user responsibility For every system there is only one focalpoint between EDP and the users. There are many focalpoints with different attitudes and background. System control is an additional task and does not always gets the attention it deserves.

- network organization

  network controller He is the "geographically stretched" datacentre manager. His main task is to maintain an optimal functioning network He acts in case of failures and/or breakdown of any component within the network. He is responsible for adequate backup and recovery procedures. He is functional responsible for the local network coordinators.

  network coordinators There is no need for local computeroperators, but at every location with a computer, a network coordinator is appointed. This coordinator is responsible for operational tasks. such as physical security measures, file reorganisations. He at the same time supports the local datacontrollers and the users on technical matters concerning the hardware. He reports hierarchical to line management,but for the for the network tasks functionally to the networkcontroller.

  result: The network is centrally controlled by decentral non-EDP personnel, so a responsibility of the plant management. It is found sometimes difficult to convince local management for the need, but in practice it proves them to be an absolute necessity. The organisation structure from the outside looks "non- transparent", but it proofs to be very "user friendly".

- software control

  The software control is centralized in the Technical Support Group.They support the network organization as a whole.They alone have SM capabilities and are responsible for datasecurity. They prepare users UDC's and install or change (on systemscontrollers request) user passwords.

  result: Network security is brought down to a small group of centrally controlled people.

SECURITY AND BACK-UP

- account structures There are some basic rules applied to TDV's account structure,such as:

  . do not allow application production from the systems account . allow in the production accounts data files only . put locked programs into one separate group in the sysaccount . donot allow development people to the production accounts. . provide for a separate development account . have a separate "play account" for the software group . and allow non-production/development personnel on a harmless "demo account" only

interactive systems There are only a few large accounts within the computer systems. Every systems controller has access to his particular part of the account. He controls his users and the facilities they are entitled to. The systems manager e.g.also accountmanager, translates this into UDC's at logon time. By means of this UDC the user can choose from a menu of programs he is allowed to use.

batch systems With exception of the development computer,all (production) jobstreams for all computers in the network are made by the schedulingsection of the central datacentre. Operators and networkcontrollers (or in special cases users) have an execution capability only.

result: User capabilities within the accounts are" program defined" therefor no file- or data element-locks are necessary. The necessary separation of functions is achieved. The accountstructure is not very transparant, but probably an optimum between processing and datasecurity is reached.

- systems back-up and recovery

systemdumps Besides the normal generation procedures for application systems in the batch environment, every night short sysdumps and every week total sysdumps are made of all on-line files at all locations. The tapes are stored in fireproof vaults near the computer and at least once a week in a vault at a location separated from the computer.

manual datalogging Recovery is done from dump and renewed manual input. There is not provided for automatic logging. All input documents are filed for at least one week, besides in some instances a simple sequence store of specific input.

result: Simple backup and recovery procedure,without complicated resource(overhead) intensive programs. Users responsiblity for "manual" logging is necessary and not always proven to be watertight.

- hardware back-up

datacom fallback The datacom installation at the central site is equiped with a Racal Tester. This allows the network controller to determine at failures the actual source,wether it had to be the computer,modem or the leased line. In the latter case an alternate dial-up connection to the central site is always standby.

computer back-up The central site development computer is a general standby in case of a computerbreakdown for more than 24-48 hours. Every decentral site can be connected by a standby multiplexor to the development computer for 8 predetermined crucial terminals in the plant. For the centralcomputer it is assumed that inp's and peripheral devices can be rearranged. In the event of a real disaster in which both the central and the development computer brteak down,we will have a problem that can be felt for weeks to come.

result: Cumbersome procedures and constructions are made which have to be tested. It gives users management and the automation manager a feeling of security at the expense of the conscience of the software specialists.

RESOURCE UTILIZATION CHARGES AND ADMINISTRATION

- capital investments capital investments are budgetted and made by the EDP department.Line management has approval authority on users requests for hardware expansion e.g.terminals,disks a.s.o. All costs such as depreciation,maintenance, insurance of hardware are incurred by EDP. The total EDP costs (budgets) must balance against the (internal) EDP charges to the users.

  variable charges The network as a whole including lines,datacom and com- puters up and thru the (async) terminal ports are con- sidered to be one source of costs in the costprice of the unitoperation. Charged are actual unit operations such as CPU sec.,spool lines,disk and tape I/O's which are on a weighted basis converted to "Computer Units" (CE's).

  fixed charges Terminals are purchased by EDP and leased on a fixed monthly basis to the users. The latter have to account for that in their costbudgets.

  result: The users get billed there own unit operations,which can be billed,regardless on which computer they are made. EDP is free to alter the hardware configurations at the locations.Users are not always costconscious enough about their hardware demands from the EDP departement.

- monthly EDP accounting

  Once a week the logfiles of every computer within the network is decentrally condensed and stored. Once a month this information is processed centrally. Some of the information stored per job and session is:

  time in/out,computer,account,user,device no, appl. system code,tape/disk I/O's,spoollines, CPU sec.

  computer utilization It is obvious that from this detail information several reports can be made such as computer utilization per system, unit operations per system per computer, elapsed time, terminal utilization per device port etc.

billings The invoice is prepared on the basis of :
computer/account/user" converted to "systemcode /client".
Costreporting is per system and per department. In the same invoice project development costs and testing CE's are merged.

result: The system responsible sytems controller and the budget responsible user receive all information necessary to at least monitor and/or influence costs/benefits. The network controller assesses the usage of "his" computer and the terminals within "his" location.

RESUME

Within Thomassen & Drijver - Verblifa the automation is a very user controlled activity. This is proven to be very effective to achieve full supported user oriented systems. In such an environment it is necessary to have an organisation in which responsibilities are defined and controlled and where the necessary security steps are provoked. A fulltime professional EDP auditor within our company scrutinizes all our actions, even this paper.

MANAGING JOBS AND OPERATING IN A UNIVERSITY COMPUTING CENTER.

G.-P. Raabe, G.-F. Rueck, R. Knott

Universitaet Passau, Rechenzentrum
Postfach 2540, D-8390 Passau, Fed.Rep. of Germany

INTRODUCTION

The University of Passau is the youngest University in Germany, it started by the wintersemester 1978/79. So it is the latest and as it looks like the last foundation of a new university in Germany. At present the university of Passau has about 1800 students and four faculties: Theology, Law, Humanities, and Economics. A Faculty for Mathematics and Informatics will start with research and lectures by the end of next year.

The computing center serves as a central service department for the whole university and at present runs a HP 3000/III.

Due to the variety of usage of the computer by students and research staff it has become more and more necessary to have a better means for managing batch jobs and the operating. What we need is the possibility to impose several rules on the processing of jobs.

So we planned and wrote some programs which are started by the operator service program called OPSER. This program has to run on the operator's console and serves for system supervision. OPSER gets information about other programs and system status from other programs, such as BATMAN, which inspects appropriate system tables. OPSER handles a variety of commands which are forwarded to MPE or to subsystems. As a speciality all commands, also MPE commands, may be abbreviated to any degree, the abbreviation only must be unique in that context.

OPSER starts other programs as son-processes: These are BATMAN, HAMMER and planned are several spool-processes.

BATMAN and UBATCH

BATMAN manages all batchjobs by collecting them, classifying them into queues, starting them for execution and keeping track of them until execution will be terminated successfully.

Users create batchjobs by writing a control sequence as usual under MPE. These files may be EDIT/3000 keep-files or QEDIT-files. The new command SUBMIT starts the program UBATCH. UBATCH corresponds with BATMAN by Message-Files. All UBATCH-processes write their needs for service into a single MSG-file, which is read by BATMAN. When an UBATCH process, associated with an individual user, requests for service BATMAN creates a special Job-File into which UBATCH copies the job-sequence. UBATCH checks all job-commands for correctness. The coordination between BATMAN and each UBATCH process is maintained in the direction from BATMAN to UBATCH by individual MSG-Files, for each UBATCH-process one, which are established by UBATCH.
After UBATCH closed the jobfile it can only be accessed by BATMAN. A user may obtain information about his jobs and their status by the command LISTJOBS. This command gives information only about his own jobs, resp. about all jobs according to his capabilities. With the command CANCEL a user may abort his jobs.

BATMAN manages several queues: up to 10 freely defined and 3 standard queues which exists allways. These 3 standard queues are:

AFTER:

Jobs which are to be executed after a certain time, defined by a timestamp with date, hour, minute.

DEPENDENCY:

The user defines a dependency counter in the range of 0 to 127. The dependency counter must be managed by the user himself. Only jobs with a dependency counter equal to 0 will be executed. This is a mechanism for planning and managing a series of jobs with interdependencies.

READY:

Into this queue all jobs are classified for which no suited freely defined queue could be found. (e. g. requiring Mag Tape when no such access is allowed). Jobs in the READY queue must be explicitely waked up for execution, either by the user or by the operator.

DEFINED QUEUES:

With the program QDEFINE up to ten queues may be established with the following parameters:

CPU-time :	min - max
Printed Lines :	max
Operator-Action:	YES/NO (e.g. REPLY for Mag Tape Access)
Schedule :	YES/NO (NO means that no more jobs will be accepted for this queue; queue closed).
Run :	YES/NO. (YES says that the jobs from this queue will be executed. NO says that jobs are only collected into this queue).
Max-Jobs :	max. number of jobs from this queue which are allowed to run at the same time.

According to the parameters of the queues and according to the attributes of a job BATMAN decides about the classification of each job.
The attributes for a job are set up in this sequence:

1. Standard Attributes
2. Parameters from the JOB-card.
3. Parameters of the SUBMIT-command.

Jobs may have the following attributes:

CPU =	CPU-time in seconds (or unlimited).
AFTER =	date, time of day
DEPENDENCY =	0 ... 127
OPERATOR =	YES/NO
RESTART =	YES/NO (if necessary a restart is managed by BATMAN)
UNIQUE =	YES/NO (If YES any other jobs of the same user must not run at the same time; e.g. when the access to data files is EXC.)
PRIORITY = (planned)	0 ... 127, with 0 highest. (These priorities will only have effect on jobs from the same user, no influence on the overall execution.)

According to the queue parameters BATMAN selects jobs for execution and starts them by the sequence

```
STREAMS ON
STREAM job-file
STREAMS OFF
```

BATMAN knows all jobs it started. If anybody would succeed in streaming a job with the MPE-stream-command within this small window between STREAMS ON and OFF, BATMAN will find a job in the system tables he did not know and all these jobs will be kicked out by BATMAN.
Every 0.5 sec (may be changed easily) BATMAN looks for job requests in the MSG-File. After n1 such cycles (normal n1 = 120 or 1 min.) the queues are inspected. After n2 cycles the AFTER queue will also be inspected.
From the CPU-time demand a priority is derived which in each queue defines the order in which jobs become candidates for execution.
After successful execution the jobs will be removed from the queue and the jobfiles will be purged.

OPSER

As mentioned above OPSER runs as the operators session and starts all other processes. It forwards commands to several subsystems and to MPE. Via special commands OPSER acts on BATMAN: aborting jobs, putting jobs to higher priority, suspending and continuing jobs. To a certain extend the parameters of the queues may be modified: Changing the number of maxjobs, setting the RUN and SCHEDULE parameters.
A new SHOWJOB command gives more and better readable information, such as CPU-Time consumed by a job or session. The list is sorted first sessions then jobs and in each group sorted by number. Also a new SHOWOUT command produces a more informative listing of spooled output.
OPSER will act on the planned Spoolers,too, for aborting, putting forth, suspending, and continuing output and for managing the devices.
Another feature is a modified WELCOME, which reads the welcome messages from files. So these texts can be better prepared and often used texts may be kept in files.

HAMMER

This program is also started by OPSER as a son-process. The HAMMER aborts all sessions and/or jobs at a specified time. With the EXCLUDE or INCLUDE lists this may be restricted to specified groups, which are defined by ldev-numbers or by user.acct, whereby wild card characters are allowed. It is planned to provide these features by a OPSER procedure.

CONCLUSION

The programs are written in PASCAL 3000 with the exception of a few SPL routines and the HAMMER which was written in FORTRAN about 1 1/2 years ago, when PASCAL 3000 was not available. At present the system is in the test phase and it is planned to get it released for final usage by october/november of this year.

Low-Budget Site Renovation

by Wayne E. Holt
Director of Computer Services
Union College

Introduction

The Union College Computer Center was established in 1962 to service the needs of the students, the faculty, and the administration. During the 20 years of its existance, it has grown and expanded in a relatively unplanned manner. While the intentions of those responsible for each phase of growth were reasonable, the end result over time has produced a facility with noticeable problems in physical site security, office work flow, and general working conditions. The problems would have been exacerbated by the acquisition of three new mini-computers unless immediate steps were taken to provide the proper environment.

The situation at Union is not unusual for the industry. Whether large or small, new or old, it seems that many sites suffer from the malady of poor planning. When this problem is recognized, there is usually a further problem: namely, that of little or no budget to accomplish improvements.

This paper will outline the issues used to evaluate the Union College site with regard to changes in layout for work flow improvement, increased site security, better utilization of existing space, and adequacy of air and power supplies for new and existing computers. A limiting factor in the evaluation was the budget for such rennovation. Hence, the thrust of the paper will be to emphasize what can be done with a minimal budget.

The problems in the Union Computer Center can be broken into two general areas; that is, those of the actual layout, ie. arrangement of walls and doors, and those of the environment, ie. the power and air. The former can lead to serious personnel problems, while the latter can cause difficulties with the various computer vendors.

The attached diagram "A" shows the layout of the Center as it appears before the renovation, while diagram "B" details the current configuration. Various elements have been noted in order to assist in following the textual discussion.

The Center is located in an older building, originally used as an engineering laboratory. Thus, all interior walls are of sheetrock construction, and are not load-bearing. This fact (one common to many small shops) means that decisions to move walls are primarily restrained by cost only.

Structural Problems

Like most older shops, Union's was layed out with a distinct operations area separate from systems and programming. The times changed, but the facilities did not. The following problems are representative of those almost any shop might encounter:

- o Growth in systems and programming overflowed into operations. Walls were not changed to account for this, resulting in wasted space and decreased security of output listings (A3). Further, the two entrances (A1)(A2) confused the public, since it was difficult to determine where someone would be located.
- o The traffic pattern through data entry (A4) included activity not at all related to operations, but rather to offices located beyond this "bottleneck".

- There were no windows in the facility, even though the original structure contained them.

- Each desk had a phone, but with a unique number, and no central intercom or switch unit. This coupled with the layout, made it very difficult for internal or external communications.

- Since all of the chaff producing peripheral equipment, such as printers, bursters, and decollaters, were located near the main equipment frequent media problems on both tape and disc would occur.

Figure B shows the results of moving a minimum number of walls:

- All access to the Center is controlled through one point, where a receptionist can direct the flow of visitors (B1). This also allows the other old entrance to be used as an office, and the duties of that receptionist to be re-assigned.

- The new location of Keyentry (B2) prevents unnecessary traffic from passing through. The output bins for Staff(B3) also mean better flow control. The service window (B4) provides access to the public who need operations assistance without creating traffic inside the shop.

- The new offices(B5)(B6)(B7) provide better working space for programmers.

- The new Print Room (B8) isolates all chaff-producing equipment from the computers, and the bins and lockers (B9) prompt distribution of output in a secure manner.

- The new Lobby (B10) provides good access to view the computing facility for public relations purposes, but still provides as good security as before.

- The removal of two closets allowed the creation of an interior passageway (B11), thus uniting the two halves of the staff. This provided an enormous moral boost, as well as sloving a long existant problem: in Winter, all access from one side of the shop to the other was via the main public corridor, which was unheated.

- A window was re-opened in a lounge area, and all coffee dispensers and other refreshments were moved to this single location. This, too, improved moral considerably.

The total cost for this renovation, including materials and labor, was $US 12,000. The sum is so small only because extreme care was taken to minimize wall removal and construction. The cost of output bins was minimized by ordering stock items, rather than having custom installation. The impact of the cost was softened by having the work done over time in pieces. Thus, steady, affordable progress was the result.

Environmental Problems

As computer systems mature, growth tends to occur in an ad-hoc fashion. A disc drive, a printer - surely this would not bother the environment of the center?

Of course, the answer is yes, it will.

"Creeping consumption" is a plague that afflicts most computer sites, especially in the mini-compuer environment. Power is usually provided for in the best manner, although even this is not always true. Most systems actually use less power than the rated consumption; thus, most sites have built-in slack if the original power supply met power specifications for the original equipment.

This is less true of the air handling equipment. Problems with inadequate air are frequently manifested in unusual hardware failures, typically intermitant in nature. Rarely does the problem reach up and slap you with an alarm bell. Most sites measure temperature and humidity close to the air handling unit rather than around the equipment or below the floor where the air is drawn up into the computer.

The situation at Union was typical. Over the years, equipment changes had placed a "rated" 90Kva load on a 75Kva isolation transformer. Worse yet, the main campus feed was loaded with heavy compressors and other equipment that caused "brownouts" for the Center, resulting in frequent computer halts.

The original plan for air handling called for two units, each with two compressors. The full original load only required two of the four compressors, so the site was considered fully redundant. However, over the years the equipment changed - three compressors were eventually needed to handle the load. That meant that if a compressor failed, the unit had to be taken down to fix it, thus rendering both compressors in the unit inoperable. So, with only two compressors left, the computer had to shut down.

Because of the growth in the number of computers, Union was able to solve these problems as part of the "growth" budget, rather than as a special problem. The most noteworthy aspect of the effort to improve the environment was the discovery that new technology has driven the cost of "conditioned" power down to levels affordable by sites that have mini-computers. Good quality motor generators or magnetic power management devices can be purchased for $US 10,000 to $US 15,000.

With the change in computer technology, it is becoming clear that the simple isolation transformer is inadequate. Sites should better protect the new generation of mini's and resist following the "plug it into the wall" syndrome. If your power supply is not adequate, it will cause untold problems for future maintenance.

Conclusions

Many sites are content to suffer with "minor" problems in the layout of their shop, their power supply, and air supply. It is usually felt that change will be expensive and hard to justify. However, sites can be renovated at a minimum cost if management is willing to tackle the problem in a reasonable way.

The environment affects the way in which people work; better productivity usually results when people are more comfortable with where and how they work. Even if a renovation project must be developed in stages because of costs, it can still pay off if each step results in improvements.

The key is planning.

FIGURE A

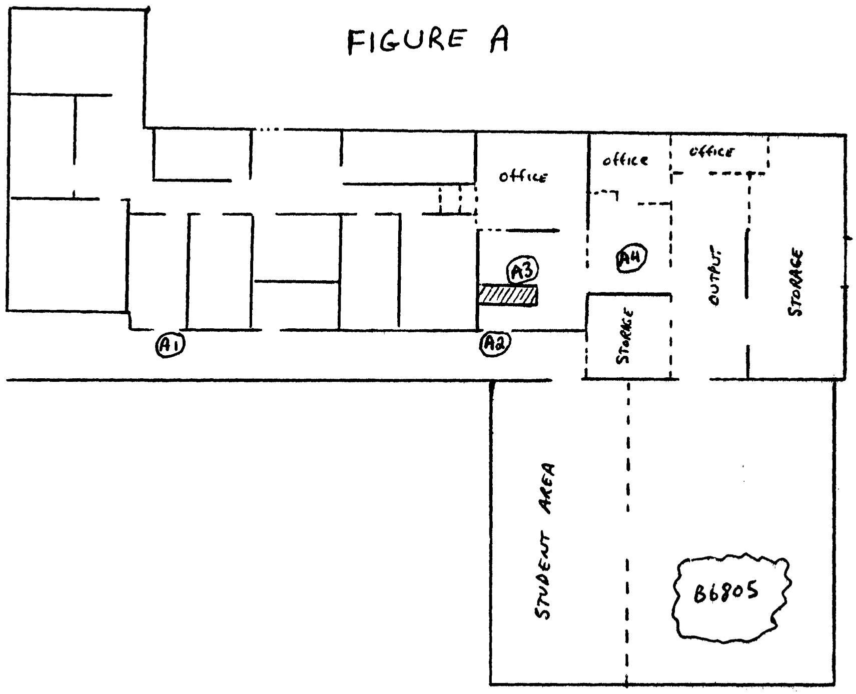
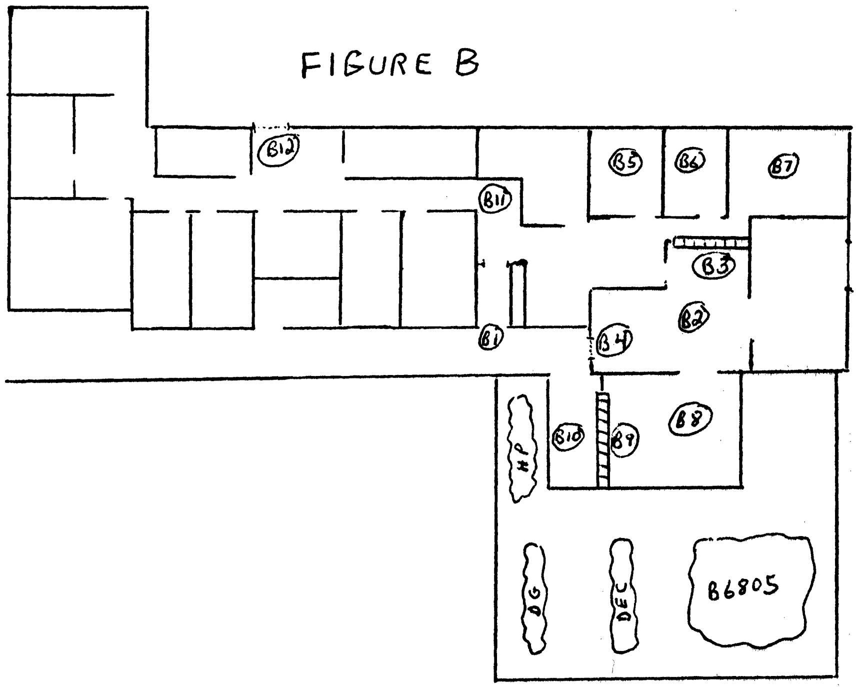


FIGURE B



LOCAL AREA NETWORKING: A TUTORIAL OVERVIEW

Paul T. Antony
Hewlett-Packard
Business Computer Group
19447 Pruneridge Avenue
Cupertino, California 95014

A B S T R A C T

Although Local Area Networking technology is still at an early stage, it is already enjoying rapid growth. Computing environments increasingly require the interworking of many different data processing devices, each with its own degree of intelligence, each with its own advantages in terms of functionality and cost, each capable of operating more or less independently of any other system component, and, increasingly often, each supplied from a different source in order to take advantage of the rapid developments in information processing technology. Providing such an environment through integration within a local area network will offer the users of mini computer based systems new power and versatility, with which they will be able to offer a real price/performance challenge to traditional mainframe based systems. An integrated system based on a LAN also offers the hope of a smoother path for upgrading overall systems capability through replacement of individual functional units as newer and more powerful facilities become available.

This paper provides a tutorial introduction to Local Area Networking concepts (broadband, baseband, etc.) and will illuminate some of the important issues and challenges related to local area communications networks.

Few areas in the data communications world have seen as much recent technological innovation and new commercial offerings, as the area of Local Area Networking.

As the cost of individual processing components falls, organizations are aquiring greater numbers of separate "specialized" computer-based systems. Each system focused on meeting user needs for higher worker productivity, better computer accessibility, and faster responsiveness. This in turn is leading to a greater awareness (at both the individual and organizational level) of the benefits of convenient interconnection of systems to achieve coordinated access both to common resources (such as databases, analysis programs, development tools, and office style memos and reports) and to sophisticated or specialized (and often expensive) resources, such as high speed printers, large plotters, etc.

A local area network is a data communications system that can be used to provide the level of interconnection described above. Some general characteristics of this type of network are:

- SINGLE ORGANIZATION OWNERSHIP - LAN's are usually wholly owned by a single organization, with gateways to allow communications to other organizations.
- LIMITED GEOGRAPHICAL COVERAGE - Distances can vary from a few hundred feet to several miles. Generally, LAN's span less than two kilometers.
- FLEXIBLE TOPOLOGY - You can readily modify a network as the organization expands. You can attach, disconnect, and delete stations without operational changes.
- HIGH DATA RATES - One megabit/sec or higher. Usually, near 10 megabits/sec. To allow fast, multiple station access with transparent network operation.
- LOW TRANSMISSION ERROR - Networks realiably accomodate heavy data transmission traffic. Should an error occur, a network station can detect it and institute a recovery.
- SUPPORT LARGE NUMBER OF USERS - LAN's can support hundreds of users and are not constrained by organizational growth.
- ECONOMICAL CONNECTION - Connection cost should not exceed 10 to 20 per cent of station equipment cost.
- RELIABLITY AND HIGH AVAILABILITY - Networks can remain unaffected by individual failures or removals for service.

There are a broad spectrum of LAN products available, and finding the one that best suits your particular needs can be difficult. In practice, LAN's fall into three distinct cost/performance categories:

1) Low cost, low performance systems using twisted wire pairs as the transmission medium.

2) Medium cost, medium performance systems using shielded BASEBAND coaxial cable as the transmission medium.

3) High cost, high performance systems using shielded BROADBAND coaxial cable as the transmission medium.

Anticipated use will determine which network type best matches your application requirements. For example, some applications such as office automation, entail text editing and file processing, which can occur at adequate rates over short (room) connection distances using low cost desktop computers. On the other hand, general purpose data and message communication applications, with or without office automation, demand more expensive minicomputers interconnected over relatively long (floor) distances. Finally, for very high speed, complex engineering and scientific computations, you might need costly superminicomputers interconnected to nodes scattered throughout one or more buildings.

Whatever the application, you must evaluate several key LAN technology factors:

- Geographical layout (2.0 km or less, in most cases)
- Transmission topology (bus or ring)
- Transmission medium (twisted wire pair, baseband coaxial cable or broadband coaxial cable)
- Operation type (asynchronous or synchronous)
- Maximum data rate (0.1M to 50M bps)
- Traffic load utilization (bursty or regulated)
- Maximum number nodes (computers and/or intelligent workstations)
- Maximum and minimum node to node separation
- Maximum number of data channels
- Transmission delay restrictions (bounded/deterministic, or unbounded/probabilistic)
- Access control scheme (token passing or collision sense multiple access with either collision avoidance or collision detection)
- Software requirements
- Maintenance, test and error detection/correction
- Safety conditions (cable flammability, radiated interference, and grounding rules)
- Interactions or gateways with other local networks, as well as long haul networks.

You must define your application thoroughly and select the products that will meet your present needs, and allow you to plan for future growth.

LOCAL AREA NETWORK TOPOLOGIES

A) STAR

A star network consists of a central node to which each of the host system devices are connected. The central node acts as a routing switch for data arriving at the central node from each of the host connections.

A star network simplifies access control and routing decisions required within attached hosts. However, throughput performance and reliablility of the entire network relies on the operation of the central node.

B) RING

A ring topology seeks to eliminate the dependency on a central, controlling node of the star network without sacrificing the relative simplicity of the other nodes.

In a ring network a single communications path is shared amongst, all attached nodes. This path is unidirectional and provides for the transfer of discrete packets of data. Each packet of data is injected into the ring from one node to the next in a predefined direction. There are no routing decisions to be made in this topology. The sending node simply transmits its message towards its next neighbor node in the ring and the message then passes round the ring until it reaches the node for which it was intended.

Each node acts as a regenerative repeater of receiving packets, generally introducing one or more bits of delay as it does so. The only routing requirement made of each node is that it be able to recognize those messages that are intended for it, by examination of the node address contained in each data packet. Then, dependent on the control strategy and implementation, the receiving node may remove the packet from the ring or pass the packet on back to the transmitting node with an acknowledgement field marked to indicate whether or not the packet was accepted. In this case, the original transmitting node removes the packet from the ring.

The inherent weakness in this structure is that the transmission path relies not only on the integrity of the tranmission medium, but also on that of the ring interface which is an active component of the network. A failure in either of these two areas could paralyze the entire network. Recently, techniques have been devised which detect repeater failures and switch those units out, while allowing the remainder of the network to function normally.

C) LOOP

Loops are essentially ring style networks with centralized control. The topology of a loop is identical to that of a ring but a polling technique is generally used to provide the appropriate degree of access control.

D) BUS

The bus or broadcast network structure is conceptully simpler than that of the ring. The basic structure is linear and derives essentially from traditional computer architecture of input/output channels.

The bus medium itself is passive and allows bidirectional transfer of messages. Like the ring, the bus structure does not require any of the attached nodes to make routing decisions. A message simply flows away from the originating node in both directions towards the ends of the bus. The intended destination node must be able to recognize messages that are intended for it and then read the message as it passes by.

Each node is attached to the bus medium in a 'T' fashion so that the message signal continues to propagate down the bus whatever the action in or by the attached nodes; there is therefore no requirement for a bus node to absorb and regenerate the message, and no modification or delay is imposed on the information in transit.

It is this intrinsic feature of the bus topology which offers the great attraction of simplicity and 'fail safe' operation.

In practice, most LAN's employ the serial BUS topoplogy, with a few manufacturers prefering the RING topology for higher transfer speeds. They are used primarily because they utilize decentralized or distributed control. In contrast to the centralized approach of the STAR and LOOP topologies.

NETWORK ACCESS METHODS

Currently, two dominant but widely varying network access control schemes have proved successful: carrier sense multiple access with collision detection (CSMA/CD) and token passing.

Although used predominantly in ring networks, token passing also finds use in some bus networks. By design only the node that holds the token at any time can transmit on the network; therefore, no access control conflicts can arise.

In a ring network, a message token passes from node to node in one direction during the idle network periods. Under strict timing rules, any node wanting network access must acquire the token within a defined interval by altering one of the token's bits on the fly. This node then transmits its message.

Additionally, in ring networks, the sending node also serves as the receiving node for token acknowledgement purposes and recreates the token for recirculation to furnish network access to other nodes.

In bus networks, the token goes from node to node in predetermined fashion. Each node knowing from which node a message comes from and where it is going.

Token passing is highly deterministic and predictable. One can calculate the maximum delay that a station will encounter in gaining network access (this is not possible with CSMA methods, whose channel access times fluctuate randomly).

Another token passing advantage stems from the high transmission efficiency acheived for varied message/packet sizes and data rates. Other key advantages include guaranteed maximum access times to accomodate real time applications, reliable operation under all load conditions and media independence without collision detection mechanisms.

However, tokens travel only in one direction in a ring network; if a node misses a token, it must wait until the token makes a complete ring revolution. Additionally, a break in the enclosed ring opens the circuit and destroys the token. The asynchronous operation of token passing can allow message tokens to get lost, destroyed, or degraded in a distributed control topology. Furthermore, strict timing requirements translate into design complexity.

Just as token passing dominates in control accessing ring networks, the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) method governs virtually all bus networks. It involves two major operational rules. First when the bus or data channel is busy -- attending to a nodes's message transmission needs -- all other nodes must wait until the channel clears before trying to send their own messages. Then, multiple nodes might attempt to transmit simultaneously, resulting in message interference or collision. To avoid this situation, the second rule stipulates that these nodes stop transmitting and wait for varying delay times before transmitting again.

This access method's chief advantages lie in 1) its simplicity - reflected in lower costs per node because the scheme needs no complex or expensive priority access circuits, and 2) its variable length message handling efficiency - more than 99 percent of all messages get through in bursty or intermittent applications (such as office automation). Conversely, for heavy traffic applications such as busy data or reservation processing systems, CSMA/CD incurs higher message collision levels, longer access delays and reduced thoughput as maximum node to node distance increases (for a given data rate). Moreover, it doesn't suit real time needs because no priority mechanism exists: Messages with different degrees of importance compete equally for bus access. In the ultimate worst lockout condition, a node's message continually collides with those of competing nodes and never gains bus access.

Several bus networks do not use the collision detection aspect of CSMA/CD because their lightly loaded applications result in extremely low collision rates. Using collision avoidance instead, these networks save cost and complexity in their bus interface boards.

In collision avoidance, message collisions get detected by the sending and receiving nodes' circuits. In a common detection technique, the receiving node delivers an acknowledgement signal back to the sender. Should the sender not receive the acknowledgement, it usually retransmits until successful.

TRANSMISSION MEDIA

As mentioned earlier, transmission media vary from low cost twisted pair, to expensive fiber optic cable. In some applications, even low cost telephone or ac-power wiring prove adequate. But the systems that use such wiring can be complex, expensive, and offer limited performance. For example, digital PBX systems use telephone wires for both voice and data switching, but their high cost ($100,000 or more), low data rates (56K bps) and centralized control limit their usefulness in most local networks -- except for large busy corporations.

Other applications employ shielded twisted pair wiring. This inexpensive, easily installed medium effectively ties together limited numbers of microcomputers (normally under 100) operating at moderate speeds (250K to 2M bps) over distances to several thousand feet.

Although currently exhibiting the highest transmission medium price fiber optics possesses inherent point to point performance capabilities that outclass all other transmission media. Its key characteristics include virtually unlimited bandwidth, high gigabit per second data transmission speeds, insensitivity to electromagnetic interference, complete electrical ground isolation between transmitter and receiver, high voltage isolation, nonelectronic radiation, small size and light weight. Despite these impressive properties, though, fiber optics is still too costly for multitapped connections.

Compared with low cost twisted pair wiring, shielded coaxial cable's moderate cost, high performance, ready availability, easy configurability and wide bandwidth make it the most popular transmission medium for interconnecting local networks. Because it maintains low level capacitance in lengths to several miles, coax allows high megabit per second data rates without signal regeneration, echoes or distortion. These features reflect a field proven technology with more than 20 years of use in data communication networks and CATV applications.

Two major types of coax exist, baseband and broadband. Physically, and in price, they differ only slightly. Connections to baseband coax can be made two ways -- inline and via a clamping tap. When a clamping tap is used, systems can be easily and quickly connected to or disconnected from a baseband coax at any location without disturbing network operations. To make an inline connection to baseband coax, one must shut down the network in order to sever the cable -- or else attach to the end of a cable segment.

Currently, connections to broadband coax must be inline. Therefore, one should plan ahead by installing extra interface units. Otherwise, the cable must be cut while making new connections, resulting in segment shutdown.

Broadband's principle advantage lies in its immense information handling capacity: One braodband cabled local network accomodates many thousands of connected nodes, handling nearly all the word, data, voice, video, and image communications generated by a busy high traffic system. These communications might include broadcast and closed circuit TV, video surveillance, telephone calls, facsimile, word messages, and data transactions.

However, broadband has two major shortcomings. In addition to requiring network and station interfaces, it uses expensive fixed frequency or frequency agile modems costing $500 to $1200. Not only does this double the broadband interface cost, but tunable RF modems prove difficult to check, maintain, and adjust because they are usually installed behind walls and above ceilings. They also limit the broadband coax's data throughput.

The second major shortcoming centers on the need for incorporating a central transmission or head end in a single cable network. This facility acts as the network's technical control center. It filters incoming RF signals from multidropped sending nodes and retransmits them at higher frequencies to receiving nodes. This central transmission facility represents a point that, if it fails, can deactivate the entire network.

Overall a baseband network exhibits a considerably lower price tag both in network startup and in network interface costs, than a broadband network.

FUTURE ISSUES FOR LOCAL AREA NETWORKS

NETWORK MANAGEMENT - As larger numbers of systems connect to the LAN, having the tools to allow effective resource management and planning will become increasingly important.

SECURITY - Most organizations want to protect the confidentiality of traffic flowing over the network. In some ways a LAN provides more security than conventional office communications, such as phone or interoffice mail. Converting voice and paper information to electronic signals makes them less accessible to the ordinary office worker. On the other hand, a communications network is a powerful tool in the hands of a sophisticated information thief or saboteur.

TECHNOLOGY IMPROVEMENTS - The encapsulation of LAN communication systems onto VLSI silicon chips will contribute significantly to the promotion and general acceptance of LAN facilities. THe VLSI will 1) reduce the LAN component prices and 2) enforce a level of LAN standardization through market volume.

STANDARDIZATION - ANSI? ISO? IEEE? XEROX "ETHERNET"? Ring or Bus... or both? Broadband and baseband standards... how compatible? The number of organizations involved in working with LAN specifications is large and their interests and motivations are diverse.

BRIDGES AND GATEWAYS - The creation and use of LAN facilities will demand the development of mechanisms to 1) interconnect various LAN's to each other and 2) interconnect LAN's to long haul networks to allow long distance informational exchange and resource sharing.

SUMMARY

The creation and effective implementation of a local area network is the key to unlock the full potential of the "information age". A LAN should be used to bind the distributed systems within an organization into a unified resource. The effectiveness of this total resource will then be measured by added capability and the degree of coherence that it achieves. This is turn will depend on the care and foresight put into the design of the network and the development of standards for interworking of systems at all levels.

R E F E R E N C E S

1) Heard, K., Local Area Networks, Gartner Group Special Report, February 1982.

2) Hopkins, G. and Meisner, N., "Choosing between broadband and baseband local networks," MINI MICRO SYSTEMS, June 1982, pp 265-274

3) Kinnucan, P., "Local Networks Battle for Billion Dollar Market," HIGH TECHNOLOGY, November/December 1981, pp 64-72

4) Kotelly, G., "Local Area Networks - Technology", EDN, February 17, 1982, pp 109-119

LOCAL AREA NETWORKING SIMPLIFIED WITH A DATA PABX

By C. H. "Bill" Skaug

Micom Systems is a rapidly growing manufacturer of data communications equipment for minicomputer users. Two years ago, when the company was less than a third of its present size, we met our data processing requirements by using service bureaus. Today, we have six in-house systems in a local network, another system on order, and we still use outside services.

Soon after we began installing our in-house systems, we recognized the need for local networking. We chose a data PABX approach, and time has shown that it was the right choice, as the system has easily grown to accommodate more machines and users. We think our experiences are typical of many growing organizations, and of other computer users with dynamically changing systems.

We also believe that our evolution into local networking--from dial-up lines, to data concentrators, to dedicated in-house links, to a data PABX, and finally to multiple interlinked local nets--is typical of the direction others will take over the next several years, in spite of many glamorous alternatives.

Beginning with dial-up access

Starting in 1980, we used two major computing services, Xerox Computer Services and General Electric Information Services Company (GEISCO). Xerox supplied standard "canned" business applications, while GEISCO was used on a more ad hoc basis for engineering and management applications. Due to the different usage patterns of the two services, different communications techniques proved appropriate.

For the quick or intermittent problem-solving performed by a large user base accessing GEISCO, we used dial-up lines, allowing the engineer or manager to reach the system from a terminal in his (or her) office. Dial-up access is one of the simplest forms of data communications, requiring only a modem to supplement the user's terminal, and this is the way many organizations begin their dp. We were no different.

Concentrating on leased lines

We used the Xerox service bureau for the day-to-day commercial applications, which typically were accessed by a smaller group in the accounting and manufacturing departments. These users were on-line all day, or at least a great part of it. This meant that a leased, rather than dial-up, line would be most cost-effective.

To further keep telephone line costs under control, we installed 8-channel (later 16-channel) data concentrators at each end of the leased line. These concentrators (sometimes called statistical multiplexors) let us run up to 16 terminals while paying for only one telephone line.

In some ways, using concentrators and a leased line is at least as simple as dial-up access, since the amount of data communications hardware is reduced. Instead of 16 modems and 16 phone lines, our communications link had one concentrator with an integral modem at each end of the single telephone line. (And the concentrators could pay for themselves in a matter of months in savings on telephone line charges.) This worked well for us--for awhile.

Driving an in-house system

Then, about two years ago, when we were a $15 million company, our growing size and DP requirements finally sent us looking for our first in-house computer. We hoped for a single system that could handle all our dp needs: administrative, financial, engineering software support, customer service, and CAD/CAM. We didn't choose an HP 3000, as this audience might expect; we installed a Prime 750, mainly due to software considerations.

However, by the time the Prime was up and running, we had already outgrown it. We proceeded to order our first HP 3000, a Model 44 (again chosen due to its software), along with another, smaller, Prime to handle CAD/CAM on a dedicated basis. By this time, we had decided to put financial and manufacturing applications, including MRP and Bills of Material, on the HP, leaving the Primes for use by our customer service and engineering departments.

Fortunately for us, with the installation of our first computer, the Prime, we made a data communications policy decision which is with us today: unless the terminal is in the same room as the computer, we use line drivers to connect the terminals to their hosts. For those who are not familiar with them, line drivers are functionally analogous to modems but much less expensive. Capable of operating over distances of a few miles, they condition the signals going between computers and terminals in order to ensure reliable transmission beyond the EIA RS232C-specified maximum cable run of 50 feet.

While people routinely exceed the 50-foot limit without modems or line drivers, we didn't want to risk the problems of cross-talk or electrical interference when our walls and ceilings got more crowded with active data links. In fact, we even use line drivers to connect terminals to nearby data concentrators if there is any chance of a problem.

All of this worked well while we had only a single computer supporting those rapidly multiplying terminals.

Accessing multiple computers

When our second computer, the HP 3000, arrived, we faced a new problem, one which would compound as we added more machines. While we had successfully managed to split applications between HP and Prime systems, many users needed access to both. So much for "best laid plans."

Faced with providing one user access to two computers, we had several choices. We could have added extra wires through walls and ceilings for each multi-machine user, and asked the user to plug his terminal to the appropriate set as needed, or we could have--even more magnanimously--installed a second terminal for each of these folks. Neither solution was economically viable, particularly when we envisioned the consequences of adding more people and more machines to cover our continuing growth.

Instead, we left the wiring and terminal situation as it was, and made our move in the computer room, where we installed a data PABX. With the data switch, as it is often called, we have in essence a private telephone system that can connect any authorized user to any available computer port. And, if a port isn't available--say all the HP ports are tied up with accounting business at the end of the quarter--then the user asking for an HP port is told (in effect) "They're all busy. Wanna wait? You'll be number three in line." This is the data PABX equivalent of camp-on busy in a voice exchange.

The advantages of a data PABX

The primary advantage of a data PABX, although it has many, is its ability to connect any terminal to any requested resource (subject to security considerations programmed into the switch). This means we can hardwire a terminal to the data switch, and from the terminal keyboard a user can request a port on one of our HP systems, one of the Primes, a Zilog development system our engineers use, or whatever we may acquire in the future. We can add more computer systems and terminals as we grow. We can even let users call outside, through the switch, to reach one of the service bureaus.

We also benefit from a "statistics log" feature of the switch which provides detailed reports of all switch activity (a function which would require a dedicated processor in other proposed networking plans). And our network easily supports remote users and remote computers.

The switch also lets us save money by using leased lines, while still providing "dial-up like access" to several computers. And nothing keeps us from "remoting" some of the CPU's, heightening that dial-up parallel.

An example may help to illustrate the degree of flexibility we realize by combining data concentrators, leased lines, and the switch. Say one user at Micom Caribe, our Puerto Rican facility, needs to use one of the HP systems. He simply turns his terminal on, and the switch, which is in our headquarters in Chatsworth, California, automatically asks him where he wants to be connected. He can answer with a symbolic name, in our case with an "H" for one of the HPs or with an "M" for the HP running MANMAN, or whatever, and the data PABX makes the connection.

A few minutes later, an engineer in Puerto Rico may turn on his terminal and ask for one of the Primes. Although both active terminals are on the same leased line, each user gets the same service he would on his own unshared line. For that matter, either of these two users can log off one machine and ask for another without affecting the transmission of the other user on the line.

Perhaps it's inevitable that our version of local networking using a data PABX is continually compared to others which use more exotic technologies. We don't mind. We show up well in the comparisons. For example, the new proposed local networks which use coaxial cable or fiber optics offer one very appealing feature in their ability to support many users on the same physical medium. But we can do the same by adding a few new twists to plain old telephone technology. In our corporate headquarters we've begun using a new type of line driver that can multiplex up to eight asynchronous terminals over the same two pairs of wire that might otherwise be used for a single telephone or terminal. This saves us time, money, and effort. We already have our offices wired for terminals, so now we can use the same wiring to connect several terminals in one room to the data PABX. In addition to saving us the time and cost, and disruption of stringing new wires, it also reduces the number of line drivers we need by as much as a factor of eight.

Growing the network

As might be expected, as our data processing capabilities have grown, so has the data PABX. Luckily, the switch is easy to expand with the addition of simple plug-in interface card modules. These provide four line or port interfaces per card slot, and each bay has up to 32 slots, which works out to 128 lines or ports (intermixed) per 19-inch bay.

As we fill a bay, we simply add another. Once we had our first HP 3000 and Prime pair connected to the switch, it took about 30 days before we added our second bay; 60 days later we put in bay number three. We've already ordered our fourth, and we can continue to put them in until the raised floor collapses from the weight.

All of this can lead to a massive set of cables for local and remote terminals, we found. We have about 250 terminals in our Chatsworth facilities, and another 30 to 40 in the field. To simplify the wiring of the nearly 300 RS232C connections coming into the computer room, we have adopted a technique called "group termination." All incoming lines go to a wall-mounted telephone block; cables with 50-pin connectors attach the block to the switch. Each of the cables handles either six terminals (with EIA control signals) or 12 terminals (data only).

Of course this greatly reduces the snarled wiring behind our switch, and decreases the cabling under our flooring. It also makes connecting to the data PABX quicker, as we can attach as many as 12 terminals when we plug in a single 50-pin connector. (The group termination technique we use should not be confused with another that uses similar wall-mounted blocks but omits the 50-pin connectors, leaving the user with a fistful of loose wires that must be connected to screw terminals on the switch.)

Growing new networks

Another advantage of basing local networks on data PABXs comes from the transparency of the network and the flexibility for adding new network components. While our switch grew, we acquired a pair of used HP 3000 Series III systems. One has been traded up to a Model 64. The other has been earmarked for Micom Caribe, with installation there slated for the first few months of next year. The machine destined for Caribe is already operating in Chatsworth, and users in Puerto Rico already access it through the switch. This allows them to follow our progress in implementing their applications, and to offer feedback. It allows us, in turn, to maintain centralized control over the system without flying liaison staff members back and forth between the two sites.

When it comes time, we see two ways to make the cut-over. We may temporarily move the Caribe applications to one of our other HP 3000s, tell the switch to put Caribe users on the new host, and ship the Series III to Puerto Rico. We could then fly the latest SYSDUMP of their work to Puerto Rico over a weekend, and complete the cut-over. The real location of the hardware, thanks to the data PABX, is transparent to the users. The second alternative would be to acquire still another HP 3000, endearing us to our friends in Palo Alto, and as in the first case, pull a SYSDUMP at the start of a weekend, fly it to Puerto Rico, and complete the cut-over in time for the start of a new week.

In either case, we could retain a leased telephone line between Puerto Rico and our California data PABX, which would give users at each site access to each other's computer systems and data bases. Should the workload in Puerto Rico ever warrant it, we could also install a data PABX there, to attain still another degree of flexibility.

And there's more. We may also bring our British subsidiary, Micom-Borer on-line soon, and one day could plant a small network there as well.

In spite of the many miles involved, including possibly a trans-oceanic hop, we'd still have "local" networks in most senses of the term; those local networks would just happen to be interlinked.

Networks within networks

We're also expanding the scope of our networking in another way. Our fourth HP 3000 will be expected to support DS/3000. The new machine will connect to our data PABX and through a high speed link to one of the other 3000s. Our reason for going to DS/3000 is one of response time for our many users in finance, manufacturing, and sales support who make inquiries against a large IMAGE data base now residing on one of the 3000s.

Initially we expect to put MANMAN and Accounts Payable on one of the linked 3000s, General Ledger, Accounts Receivable, and Order Management applications on the other. Files unique to each host will become more readily available to our users, and common files will be kept synchronized through the facilities of DS/3000.

Because of the transparency of the data PABX, we don't foresee any problems with the two networking systems operating together. Additionally, if someone needs to switch applications, say from A/R to A/P, it will be more efficient to reconnect to the appropriate host through the switch, as opposed to using the high speed interprocessor link and stealing capacity from DS/3000.

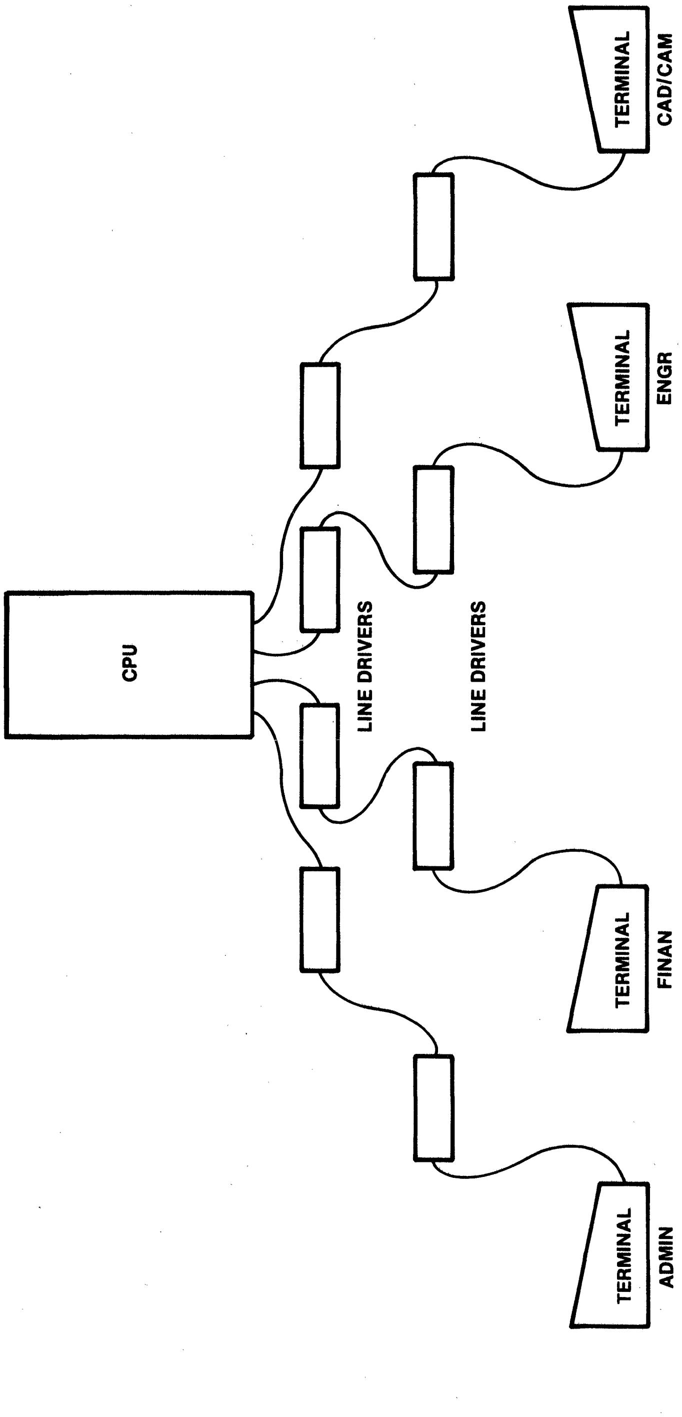
Conclusions

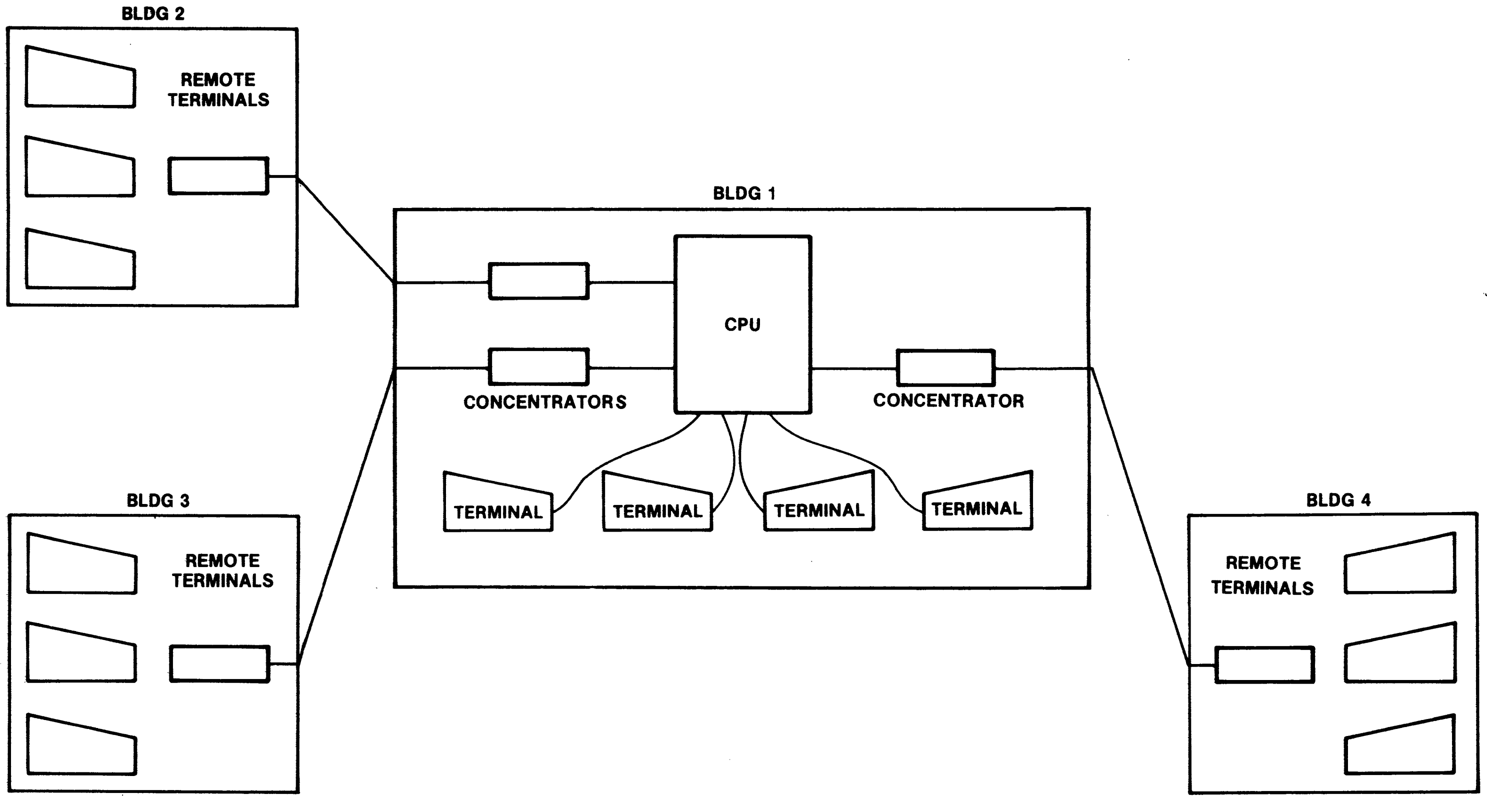
As we've grown our network, we've learned a few things, the most important of which is just how right we were to start with a data PABX. Unless your users stick to a single computer, or don't mind marching off to a centralized terminal room, it's difficult to see how to manage without one.

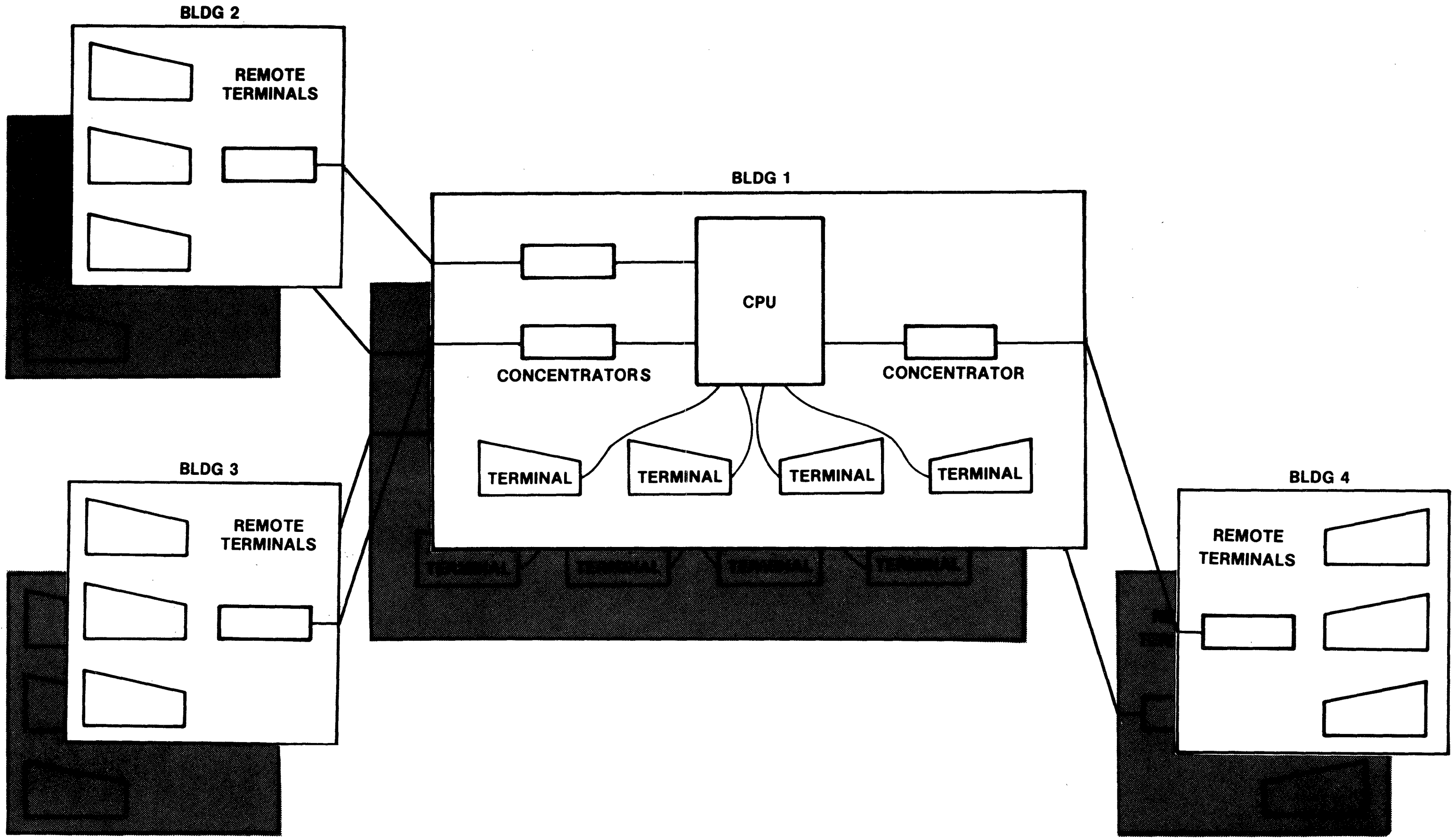
Granted, telephone technology is not as exotic or glamorous as working with coax or glass fibers, but it more than makes up for its lack of sexiness. Its technologies are proven, and relatively standardized in a de facto sense. Twisted-pair wiring or in-house telephone wiring is inexpensive, and in most offices, it's already in place. Likewise, using line drivers with RS232C interfaces provides a standard method of connecting to the network and avoids any special programming considerations.

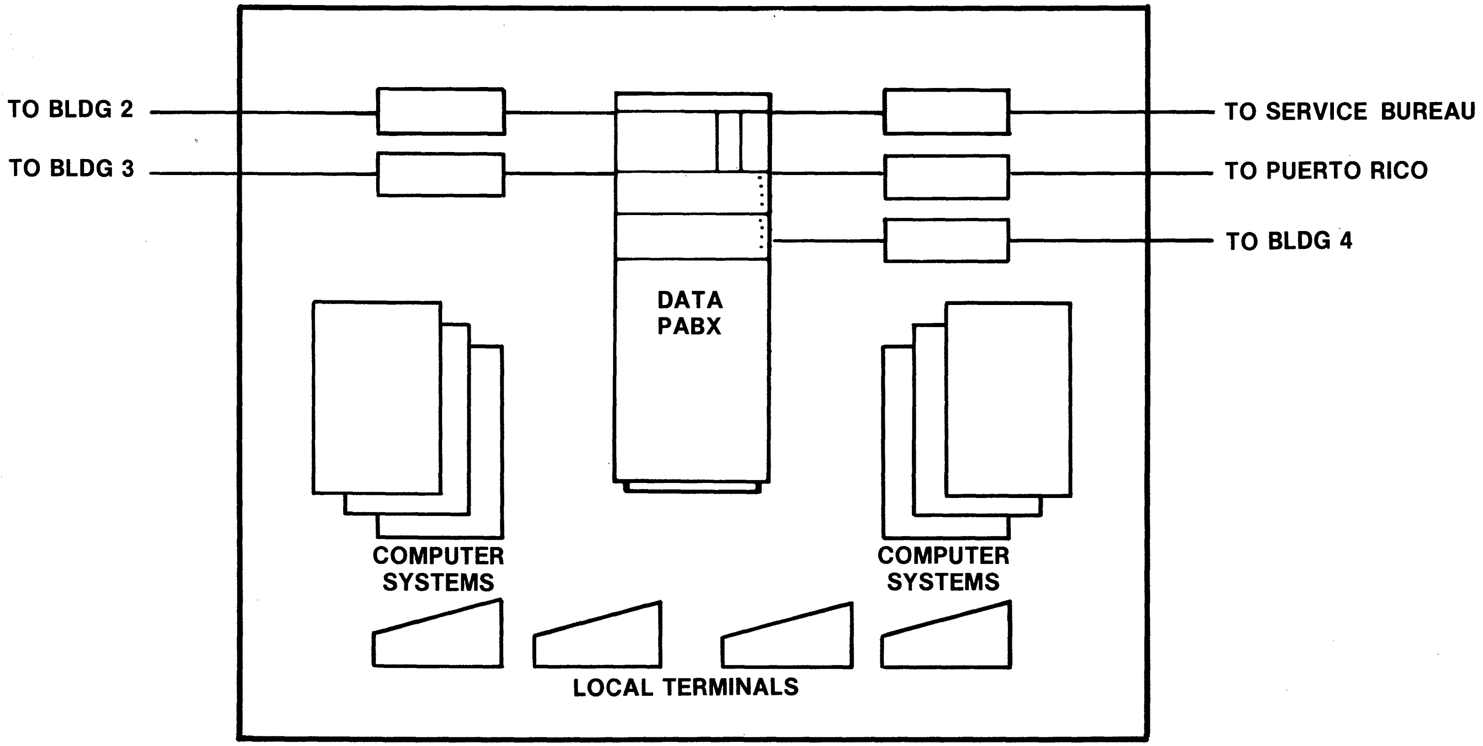
In contrast, networks using coaxial cable or fiber optics run up increased costs due to the expense of the medium. In most buildings the installing of the broadcast medium also runs costs up very quickly, as well as disrupting everyone's work. Then too, the interfaces to the media are more expensive--by an order of magnitude--than line drivers, and are basically unstandardized, incompatible devices today.

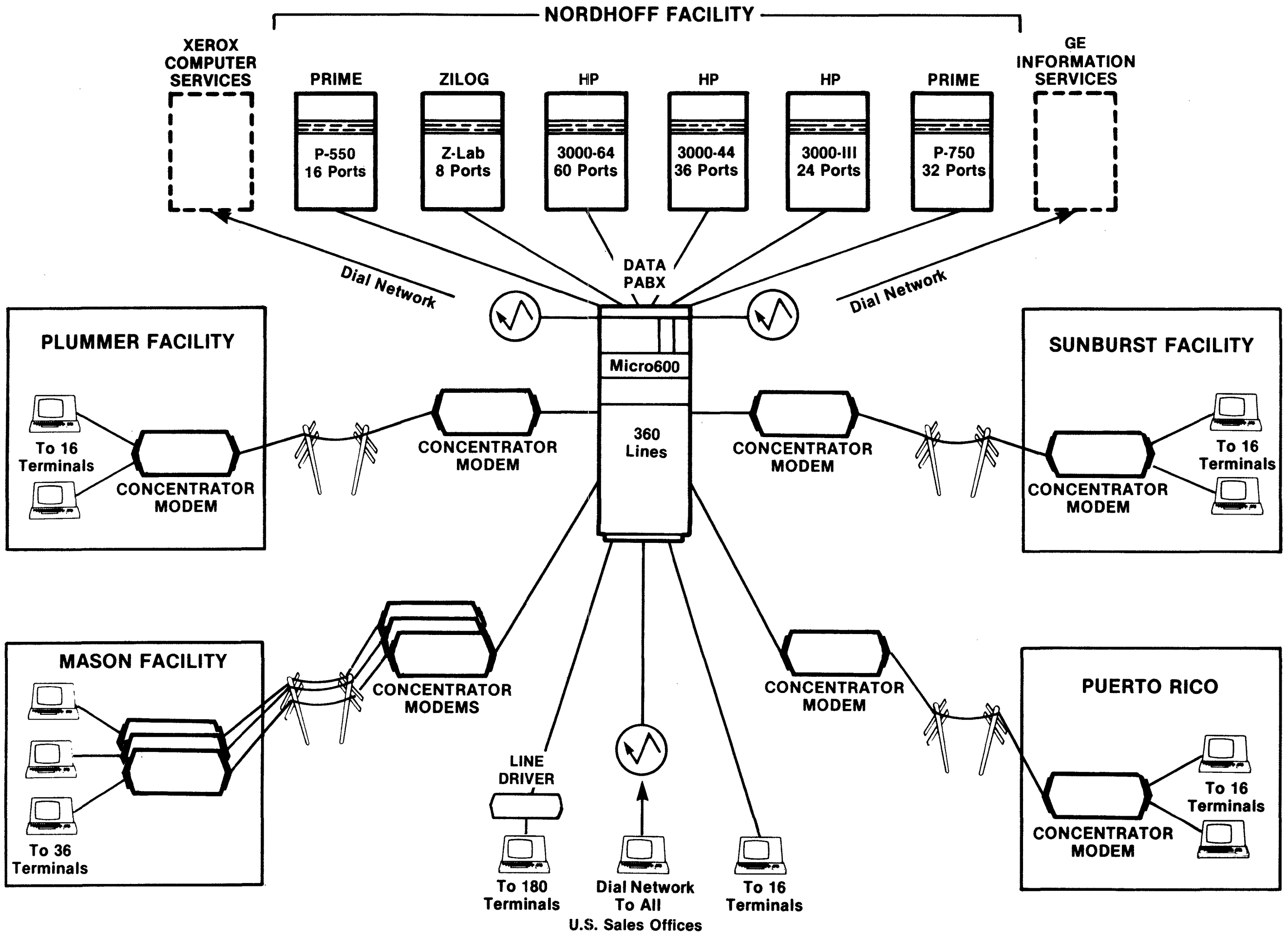
A pioneering user who adopts one of these new and exotic networking technologies may well be casting his decision in concrete before the industry is ready for that. A mistake on his part is likely to become an expensive embarassment down the road. We're not going to have that problem. When those new technologies mature and users can install them with confidence, we won't be left behind; we can connect to them too.


CPU
LINE DRIVERS
LINE DRIVERS
TERMINAL
CAD/CAM
TERMINAL
ENGR
TERMINAL
FINAN
TERMINAL
ADMIN


BLDG 2
REMOTE TERMINALS
BLDG 1
CPU
CONCENTRATORS
CONCENTRATOR
TERMINAL
TERMINAL
TERMINAL
TERMINAL
BLDG 3
REMOTE TERMINALS
BLDG 4
REMOTE TERMINALS


BLDG 2
REMOTE TERMINALS
BLDG 1
CPU
CONCENTRATORS
CONCENTRATOR
TERMINAL
TERMINAL
TERMINAL
TERMINAL
BLDG 3
REMOTE TERMINALS
BLDG 4
REMOTE TERMINALS


TO BLDG 2
TO BLDG 3
TO SERVICE BUREAU
TO PUERTO RICO
TO BLDG 4
DATA PABX
COMPUTER SYSTEMS
COMPUTER SYSTEMS
LOCAL TERMINALS



MICOM'S "LOCAL" NETWORK

Synergy Computing (Pty) Ltd 1119
Cartwrights Corner Adderley
Street Cape Town 8001 South
Africa

Telephone (27) (21) 46-2167 Telex
57-27566

August 1982

For the Proceedings of the Hewlett-Packard General Systems
International Users Group Congress Copenhagen, October 1982

THE CONTROL OF THE INTEGRATED OFFICE

by

Christopher M. Spottiswoode

INTRODUCTION

As D.P. applications multiply, especially with word-processing being integrated, the following problems become more important: access security; coordination of end-user activities, whether computer-based or manual; resource-usage efficiency; operator and program interaction, especially at the system-wide level; adaptation of the system as requirements change. There are many partial solutions, requiring skills in various sub-systems and tending to result in a mix which is difficult to maintain or learn.

In this paper a simple conceptual framework for a total approach is built up, resulting in a consistent and general-purpose way of addressing many of the above-mentioned system-level problems. In fact, a wholly new computer language is born: not just another programming language, but a system design and control language for work-processing systems.

The approach adopted in this written version is rather theoretical, being intended to emphasize the scope and well-foundedness of the concept. The talk on the other hand will revolve more around a practical example, copies of which will be available at the time.

Slides will be shown (though are not included here) which take an integrated data- and word- processing application from its initial definition and simple implementation, through various enhancement steps, to a degree of easy sophistication that would not be practical without this new language and control system.

The structure of a product containing the compiler, its run-time system, and various utilities is described. Some design considerations are mentioned. The relationships with other current products and directions of software development are discussed. The launching strategy for the product is explained, which, because of the potential of the language as a standard, is rather unconventional.

PROBLEM AREA

The overall problem area is system management, where the data and program components have to be fitted together in a flexible way. The following aspects are now briefly discussed, with reference to the shortcomings of some current methods on the HP3000.

(A) Design for integration (B) Shared database design (C) Control of processing (D) Integration of components (E) On-line help (F) System changes

(A) DESIGN FOR INTEGRATION

The analysis and design of a complex system produces a welter of facts and proposed methods that are difficult to organize and consult. It is difficult to steer between the extremes of too much and too little: what you are looking for may be there but you can't find it in all the detail, or else you can read the specifications but what you need simply isn't there.

Then the consequences of the proposed methods are not clear at all: there are probably holes in the design that will only come to the fore when the system is being implemented. There tend to be mismatches between the requirements of one procedure and the output of a prior operation. The synchronization of manual and computerized activities is particularly difficult to picture so that one might foresee potential problems.

Work Study, O & M, and Critical Path Methods can be applied, but the first two are frequently rather hit-or-miss, and CPM is usually based on specifications that are difficult to set up, even more difficult to maintain, and bear no relation to the eventual computer system specifications.

(B) SHARED DATABASE DESIGN

There is just one aspect of shared database design that I must mention here, namely that which concerns the availability of the right information at the right time (This is in fact almost a restatement of the timing problem of the last two paragraphs above). Now at first glance there is no problem: if the database is maintained in real time, everything is always up to date, so there can be no unavailability of information. One might call this "naive real-time". It has one major and fatal problem, which also leads to one major and saving opportunity.

The problem is the painfully obvious one: to maintain all the needed views of data in real-time places an intolerable load on a computer. Adding chains to a detail file in IMAGE is notoriously bad for performance (though for efficient provision of access paths, we are quite well off on the HP3000). Increasing computer power is no way out of the general problem: it sooner or later creates new needs, as Parkinson's Law would lead us to expect (Requirements expand as fast as opportunities). So views cannot be maintained in real-time and must be created when they are required. Even though we can do this super-efficiently with aids such as SUPRTOOL from Robelle Consulting, we have a design and control problem, as in (A) above and (C) below.

The saving opportunity is to be found not in any clever implementation technique, but by considering the real nature of integrated office applications. The fact is that though real-time is seductive, good old batch processing will always be required by all but the simplest data-processing applications. Why? The first reason that comes to mind is that exception reports form a significant portion of the output requirements from so-called real-time systems, and they are most often best produced by batch processes.

Now this comment points to the tip of an iceberg: an exception report only has meaning if its reader knows what data has gone into it (e.g. unless you know that all stock receipts up to a defined point have been entered into the system, what can you do with the out-of-stock report?). So a cut-off point must be part of the end-user's understanding, and for us computer people that means a close. Such a close is a synchronization of end-user processing and computer processing: we have a batch run (a technical concept) that has end-user meaning, so the end-user must reject naive real-time (even though he did originally expect everything to be at his fingertips in his new computer!).

This point can fruitfully be taken further. A cut-off point may also be quite independent of the computer: for example there must be a cut-off point to reconcile a monthly physical stock count with the accounting records. So when such areas are computerized and become integrated data-processing systems, we become involved in the synchronization of the activities of three departments: warehouse, accounts, and computer. Cut-offs and their associated batch processes are thus an opportunity for us to survive well despite limited computer power.

We end up with a requirement/opportunity that will never go away as long as organizations are divided into separate departments whose activities must be organized, programmed, and coordinated: we will have batch runs associated with the interfaces between departments. These we think we know how to handle efficiently (At least we are better at it than at real-time updates). But we will have to plan and control the mix of processing that results on the computer.

This last point clearly leads us on to (C) below, but first a restatement of this apparently rather artificial argument about real-time will be a start to a picture that will be useful to us later. Our "naive real-time" results in a database that exists in space but not in time: it retains the same structure as time advances (Just look for a time component in any of those data-base diagrams!). In fact each department of an integrated office works in its own real-time, largely independent of other departments' real-times. It is only during cut-offs and closes that the different real-times are synchronized and coincide. (It was precisely this issue of synchronization of otherwise independent departments' requirements that led to the splitting up of the once integrated database recounted in "A DataBase Story" by J.J. Sobozak in Datamation of Sept 1977). Now if the specification and control of this strange time component of data can be improved, then clearly we will end up with better organization-wide shared data-bases and hence better applications.

(C) CONTROL OF PROCESSING

There are several large classes of problem here:

* Access control * Serial processing control * Concurrent processing control * Optimization of physical resource usage * Robustness, restartability, recovery.

Access control is traditionally implemented using passwords. Systems like SECURITY/3000 from VESOFT do it a better way. But it is largely immaterial how it is implemented. The problem is how the access is structured: in whatever way the access paths are opened, they must lead further in a logical fashion, determined by application considerations and not by physical structures such as groups and accounts and physical database subsets. Validated access paths may also have to vary with time after the path has been opened by sign-on or other gateway.

Serial processing control (e.g. do not clear the transaction file before the audit trail has been printed) is easily managed by jobstreams, but not if interactive activity is also involved. So systems such as SLS exist in the Contributed Library, and programs start having control routines built in to them to communicate between themselves to ensure that everything happens in the appropriate sequence.

Concurrent processing control (e.g. do not post transactions while the transaction file is being cleared) is sometimes solved by locking of physical resources such as files and building in tests for the resulting unavailability. More often, however, the possibly competing programs have to have yet more control routines built into them, and control files, whether core or disc, start getting more complicated.

Optimization is a very broad field, and major areas such as program language and disc management are certainly not addressed here - our target is the system level! We want to optimize concurrency of processing by launching just the right amount of simultaneous processing, within application constraints. Under MPE IV we know that 2 or 3 jobs might profitably be run together, but queue management is sadly deficient in MPE. So systems like BEACON and MBQ (and many others!) are to be found to help out. But we still need to take account of potential application conflicts. And a standard job, when running alone, should frequently be decomposable into parallel parts, but no handy way of doing so exists.

Optimization of interactive functions by increased concurrency is also frequently reduced because it is too easy, for example, simply to lock out all interactive functions while a close or recovery is in progress.

Optimization by better use of idle capacity can be obtained by time-launching of processing with systems such as SLEEPER from the Contributed Library. Once again, one must comment that a single physical resource such as time should not be the only factor in the decision: other, probably application-related, factors such as the up-to-dateness of data-entry should also be brought into the decision.

Robustness, restartability, and recovery are frequently the poor cousins of end-user specified requirements: it may even be left to those midnight calls to bring home the fact that these qualities should be built into applications right at the beginning.

Robustness is the ability to withstand shocks such as system failures, disc errors, program crashes, manual aborts, and of course terminal operator inventiveness. No prior planning will provide complete robustness, but a system with restartability and controlled recovery should eliminate most of those midnight calls. How? The RESTART option on the JOB command is hardly even the start of a solution. A total physical database recovery using standard IMAGE recovery or other conventional approaches is time-consuming and seldom necessary after a crash. The tools should be there to address the problem during the early stages of an application design, and the directions for the provision of robustness right down to the lower program levels should be provided.

The Application Monitor of MM/3000 is directed at many of the problems addressed here, and strongly shows up the need for an Automated Application Control System. As a report of the Technical Interface Committee of the IUG showed, there is some considerable pressure on HP to make this Application Monitor available for other applications. The response from HP was however that there is no way this Monitor can be generalized for any application.

Another approach to automated control is not to do it at all! This has the advantage of K.I.S.S. and may be quite valid in a small centralized shop with technical skills permanently on hand. But the requirements of continuity as staff changes make such an approach a very short-term solution.

(D) INTEGRATION OF COMPONENTS

The obvious area here is menu specification and presentation. Some of the more common techniques on the HP3000 use V/3000 or MPE commands to interpret or load soft-keys, needing programs or UDCs to specify the options. But these methods get very clumsy when one starts building variability into the menus: menu options and/or allowed choices should depend on user, security level, state of processing, and other system-level factors.

The next problem is the effecting of the choices: one finds multiple RUN commands, sub-programs, process-handling, and no doubt a hundred ways of managing each method by means of parameter files.

The troubles of system management really start, however, when these issues join up with the problem area of (C) above, and control variables of different kinds have to be passed between menu and program and between programs themselves. Disc files, mail, message files, global or local RINs, IMAGE or file locks, STREAM-file creators to plug in parameters, variables passed or received, each with their own special subroutines or copylib coding, soon clutter up programs and technical and standards manuals. A newcomer to an established D.P. department has to spend more time learning the local peculiarities in this area than learning about the HP3000 (the usual poor documentation of such home-grown specials perhaps being the major reason!). And this is quite apart from his problems of learning about the end-user's needs.

(E) ON-LINE HELP

Pity the poor end-user! If systems departments so frequently have problems knowing how a whole system ties together, how is the end-user to make out? The usefulness of reading the specifications was questioned in (A) above, and by the time the system is implemented the specifications are probably out of date.

On-line help is clearly a fashionable solution to the problem, but how is it to be provided? The usual way is to build information displays into programs. This is fine for help on the immediate situation (e.g. "Y = OK, update the record, N = I want to change my mind"), but program help is no good for problems that have to do with system-wide issues (e.g. What will be the consequences if I don't balance my batch now?).

The next step is to have an external file which may be consulted by means of calls to subroutines from within programs. The QHELP sub-system from Robelle Consulting is one neat way of structuring and accessing such an external file. But it remains difficult to provide for the needs of the various types of user, and there is always the danger that the help-file will diverge from the system it is meant to describe.

(F) SYSTEM CHANGES

End-user documentation or help, technical documentation or help, jobstreams, parameterized control files: how are all these system-level descriptions to be kept streamlined, accessible, and up to date? Various types of cross-reference (field/file, file/program, etc) may be available, though they are probably incomplete and inaccurate. Data dictionaries (now usually "generalized") are increasingly accepted in principle (though the degree to which they are actually used, especially in a generalized form, is probably still very low indeed), but appropriate structural principles have yet to be invented.

So the first problem in changing a system is finding out what the current system is. The next problem is establishing the implications of desired changes. A method of making Impact Analyses systematically is desirable. It would be even nicer to be able to ask "what if" questions and model considered changes before being committed to their implementation. I am not aware of any generalized system-level solutions here which do not force the designer into a straightjacket, though there are ways of easily prototyping changes in limited areas (V/3000, various high-level macro or non-procedural programming languages).

Then let us assume that the appropriate changes are planned and are ready for implementation. Now some existing files may need to be reformatted or otherwise transformed. (And what a pity, but how understandable at the moment, that ADAGER requires exclusive access to the database being transformed!) New versions of programs have to be activated at the right moments. New documentation as well as information on the changes must be made available. Especially if the system is distributed, it is sometimes no easy matter to schedule and control this phasing-in of changes.

Clearly a system designed to ease the problems of (A) to (E) above should be capable of assisting with this formidable task.

SOLUTION

(I) THE BASIC IDEA

It is convenient to pretend that a top-down development of this proposed solution took place (though of course it did not): therefore let us start with the fictional idea that it looks as if we need AN APPLICATION-ORIENTATED JOB-CONTROL LANGUAGE INTEGRATED WITH A GENERALIZED DATA-DICTIONARY. Such at least is what I hope the above problem-statement can be seen to be aiming at...

But this is too much of a mouthful, so let us start with something familiar: a plain vanilla job-control language (JCL). Just think of your own favourite JCL (or, if you have worked on IBM mainframes, your most hated one). A command interpreter reads your JCL and executes your commands, the most interesting of which invoke your programs. After program execution the Command Interpreter is reactivated and is ready to interpret more commands.

Generally what happens is that the simple commands define some aspects of the execution environment for your programs (e.g. a FILE command may specify what type of access your program has to an MPE file and what its real name on disc is). It is only a short step to the idea that a more complete system-wide application environment of a user program could be specified in a JCL of some sort. Concurrent application program compatibility could then be deduced and managed.

Such an extended JCL, with embedded comments, could easily be made available, if suitably structured, as on-line system-level help. This is in fact what is proposed. The extended JCL, suitably structured, becomes the system design, specification, and documentation for end-user and technical staff alike.

The immediate advantage of the integration of the documentation and on-line help with execution control specifications is that the two may not diverge so easily.

The structure of the system documentation and help, set up non-redundantly, is based on a coherent set of cross-reference and conditional mechanisms (this is where the data-dictionary features enter the picture). In order to link together what belongs together, and to provide for efficient execution, the JCL is compiled. The compilation of the JCL takes place independently of the live application and at the right moment the compiler output is linked to the live system under application supervisor control. The command interpreter reads the complete compiled form and invokes user programs as specified.

Because the control system can now know more about the run-time environment of user programs, it can provide many more services than are normally available to address the problems of processing conditioning, scheduling, recovery, restarting, multi-processing, etc.

The command interpreter also has many menu and help functions, while user programs can call simple system subroutines to make the on-line documentation available as required.

Error and help messages may now be meaningful and relate to the application. In fact, this is where the name of the language comes from: it is IDIOM, which denotes a language tuned to a particular situation. We will see in due course how the system designer in a very fundamental way sets up the IDIOM to match the idiom of the end-user organization, and the computer system will talk in that idiom. (At the same time, of course, the programmer may conduct his sessions in the normal way and continue to talk to the computer in computer-orientated terms.)

As might be expected, IDIOM is also an acronym: it stands for Interpretable Design for Integrated Operation and Management. The two meanings of the word "idiom" are so appropriate that the name has been kept (despite the fact, as my partner in Synergy Computing, Robert Gibson, laughingly suggested when the name was first proposed, that it may be thought to lead naturally to "interpretable design for integrated operation and tears!"). The name IDIOM has also been trademarked.

We now have a setup where the system definition is written in a heavily-commented language, called IDIOM. The IDIOM for an application, or parts thereof, is compiled using IDCOMP (pronounced "eye-dee-comp") and linked into the live application system model called IDSYS ("eye-dee-sys"). The linking is performed by an Application Supervisor using the many-facetted program IDMGR. Any required transformation jobs, containing their own scheduling information, will be linked in at the same time. IDSYS is interpreted by a super-command-interpreter called IDEXEC ("eye-dee-exec") which interacts with the terminal operators. The operators' commands will usually result in the execution of user programs which will then function in the usual fashion.

This setup has system management advantages whose general nature is, I hope, by now fairly clear. The main advantage to the programmer is that his program is more clearly situated in its overall system context and all the system control and help coding may be removed. This particularly simplifies the use of so-called fourth-generation programming techniques using non-procedural languages (such as QUICK, QUIZ, and QTP from Quasar Systems). And conversely, since programming using such techniques is so spectacularly easy, we end up with more and bigger and more integrated applications, which in their turn require better system management tools!

So far there is nothing striking in these ideas. They are in fact so banal that one must only conclude that they have not yet been made into a widely-used system because the approach after a while usually tends to become messy, causing more bother than benefit. To turn them into a useful system requires some important concepts that will lead us to the appropriate structures and the big pay-offs in the area of application system operational control and management. These will now be developed.

(II) THE KEY CONCEPTS

So varied are the requirements and possibilities that face us that we must base the whole structure on simple and highly general concepts.

The first important point is that since we are trying to be end-user orientated, our concepts must be independent of the hardware and system software that we happen to have.

We start with the basic concept of an end-user who uses a computer, and thereby performs some work for the user organization. We have a collection of "operators" working at "work stations".

The work is split up into separate and interdependent "jobs", interactive and batch, and the operator and the computer work together at these jobs.

How does the computer perform its portion? On instructions from the operator, it initiates processing on "processors" using "resources".

It is the task of the system designer to identify and define all relevant factors as IDIOM resources so that resource conflict during "multi-processing" cannot occur.

The mapping of IDIOM resources onto the actual physical ones is discussed in (III) below. All that need be said now is that this operation is well isolated, and in the present implementation it is kept to a minimum, precisely because it is machine-dependent and the end-user application is the real object of interest.

The concept of a resource is the key to IDIOM. The system designer will define all kinds of things as resources: processors, classes of processors, peripherals, databases, files, subsets of files, sets of files, queues, file access modes, control parameters, conditions or states of any of the above, events, end-user supervisor directives, security levels, operator capabilities, user views, etc, etc.

Such may be the real physical resources that will be identified as IDIOM or "virtual" resources, but how does IDIOM regard them? After all, we may ignore for the moment what they really refer to (just as MPE does not know much from a FILE command about the data that is in the file), but they must mean something to IDEXEC.

An IDIOM resource may be available or not. This is denoted by a Boolean value, true or false. Thus if the resource .Customer master file. (this is the "title" of the resource) is true, the physical file is available or on-line. If the resource .Want to close for the day. is true, we have a condition whose truth may enable some jobs and hence also be an available resource in an understandable if somewhat unconventional way.

A resource is there to be "used". It may be used exclusively or in shared mode. The false state of a resource may also be used or relied on. So we have the main IDIOM resource status changing verbs CREATE, DESTROY, LOCK, SHARE, and RELEASE. A job to enter data with master-file verification may "share .customer master file.". A job that is run when the engineer wants to work on a real device may "share not .document printer." where the document printer has already been put off-line to IDIOM. Subsequently the C.E. job will "create .document printer."

Other access types, such as MPE's EAR access (Exclusive, Allow Read), are implemented by associating new resources with the basic file-related resource. Thus we define a new resource, say .May add to history file., which may be locked independently of a possibly shared .history file. Such an arrangement does not look physically watertight, and indeed it is not, unless some firm identification of real resources with our "virtual" IDIOM ones can be made. These issues are dealt with separately under "Resource Binding" below.

An IDIOM resource may denote the occurrence of an event, such as .Month-end run is in progress. The resource is merely a description of the state that is true after the event, the state's being CREATEd is the event. Subsequently the state is DESTROYed: this way of representing events fits well with the cyclical nature of work-processing systems.

The execution path of a job may be varied according to any logical combination of resources, or a job may WAIT for such a combination to come true (or false), i.e. for that event. There is also an IDEXEC facility for the "creation" of resources which are dependent only on time.

A resource may consist of a set of other resources. This idea is reasonable, considering that a file, for example, may be regarded as a set of records or other subsets of the file. In IDIOM this raises further possibilities. Thus it is arbitrarily declared that any IDIOM resource may be or become or be regarded as a "set", and any resource may be INSERTed into another (if various conditions are met) or REMOVEd again. Queue management is just one very useful by-product of this structure.

The logical interpretation of the IDIOM resource can now be extended to such sets. The truth-value of a set is the logical product of the truth-values of the members, i.e. the members' truth-values AND-ed together. The logical sum of a set, i.e. the members OR-ed, is true if "one of" the set is true. Thus a process may be run concurrently as a background task if (but not only if) it is possible, for example, to "lock one of .background processors.".

The definition of a set of resources is not only useful for dealing with conventional computer-type sets of objects. It is especially used to relate end-user states of the application to finer details of the application. For example, in a simple stock application the resource .may post stock movements. can be defined as the set of ".stock master. and .stock movement file. and .stock not being counted." If the operator is prevented by the unavailability of this resource from posting stock movements, he may find out why, and if he is curious and allowed to do so, he may find out how long the situation is likely to last or how to change it.

The hierarchy of resource sets may be extended down to more implementation-dependent or physical resources. Clearly, the maintenance on-line of resource statuses up and down the hierarchies might start getting complex and inefficient. However, without going into the details here, it can be stated that there is a simple and efficient solution that derives from the very way IDIOM resource sets are intended to be used: IDEXEC always follows logical consequences upwards, but never downwards.

The final key concept that makes IDIOM as powerful as it should be, given the variety of problems that it addresses, is yet another one based on the IDIOM virtual resource. It is in fact a slight softening of the hard abstract nature of the virtual resource: any resource may have data fields attached to it. At last we can start fleshing-out the dry logical skeleton!

IDIOM data fields are used to contain conventional control data such as close dates, operator names, passwords, control totals, and the like (but not other conventional control data such as processing status!). String and numeric and logical expressions using data fields and/or resource statuses are available, as well as various built-in functions.

A resource that has for example a password associated with it, might not refer to any other real object, but one and the same resource may be associated with a file and contain a file control total in a data field, much as if in a user label on the file. In either case the effect on the resource of the resource status changing verbs (lock, create, etc) is carried through to the associated data fields.

But there is a further use within IDIOM for resource data fields. It is to identify members of a set. Thus to multi-process a job may involve a statement such as "lock one of {priority=batchpri of .accounting department.} of .batch processors." (Such a statement would typically be part of a standard macro-style "define", as in SPL.)

We are now encountering some very powerful statements. It is time for some reassurance that the provision of such facilities, along with all the others mentioned along the way, does not add up to an impossible project.

It turns out that the implementation and use of IDIOM is relatively easy: the entities and structures required by the language, and the routines in IDEXEC to deal with them, are mostly based on an ordered list structure. This is true of resource sets and of the various kinds of cross-reference maintained automatically: resource / job, called-job / calling-job, field / resource, field / job, and a few others. IDIOM shares with IMAGE the advantage of having its logical structure mirrored by its own internal representation. The use of IDEXEC soon makes this very plain.

Since general list handling routines are there, they are also used to provide a few extra documentation and on-line help facilities. Using them, the system designer can better keep to the ideal of non-redundancy of documentation by linking facts together in different contexts. This is part of the task of defining fully and maintainably the application and run-time context of end-user work.

Let us continue with the overall picture. Various flow control statements are available, as well as facilities for system-level checkpoints, restarts, and reruns. The distinction is made between global and local resources. Thus IDEXEC is in control of all of the system-level factors that determine job conditioning, sequencing, compatibility, and rescheduling.

End-user work is invoked by means of one general-purpose IDIOM statement: a "run" statement, which passes control to an implementation-dependent run-statement handler, which in the present HP3000 implementation resides in an SL and may run programs, call dynamic subprograms, execute MPE commands, and is easily expandable to do anything else the system manager may wish to permit. It may also be passed parameters by IDEXEC, which may contain resource or data references or routines written in IDIOM as part of the IDIOM job, for the dynamic changing of the program's application context.

Thus a very general and powerful JCL facility is built up, which is integrated with the data-dictionary-like documentation and help facilities.

Now because a strict system-level top-down approach to system design is encouraged, with user programs being well defined and isolated in their contexts, it is possible to model the application system without the programs.

IDEXEC has a run-statement or user program simulator which can do anything to the IDIOM resources and their data fields that a user program can. Using this, an operator can, during live processing put a WS (i.e. work station) into "what if" mode so that he can, for example, test alternative scheduling strategies. An operator, perhaps specified with more special capabilities, may also model several WS's with only one WS to test their interactions, all in one "what if" session.

In order to have robustness in its own processing, IDEXEC logs all changes it makes to IDSYS. Resource usage statistics are a useful by-product. A further useful facility could be based on them (but is unfortunately not there yet!): the IDIOM model together with the history of job times and resource usage, contains all the information needed for a formal simulation and optimization of all computer-based end-user work. Since it is a trivial matter to add non-processing jobs to represent purely manual activities and their relations with computer-based activities, an entire work-processing system can be modelled. We are hoping to have a project working soon on defining a formal network representation of the IDIOM model of the end-user organization, to be used for formal rationalization and optimization.

Let us now start looking at the technical aspects of the implementation of IDIOM. If you are still with me after this long story, you must be asking yourself how one can implement IDIOM in an existing installation and how it interfaces with systems such as MPE and IMAGE. This raises some important design considerations in the current implementation of IDIOM which are now discussed.

(III) RESOURCE BINDING

Resource binding refers to the association during actual IDEXECution of IDIOM resources with real resources. For example, how do we relate or "bind" an IDIOM resource .Customer master file. with the actual physical object and the actual accessibility we wish to grant to that object?

There are two policies here, which one might call "tight binding" and "loose binding". In a tightly bound IDIOM implementation IDEXEC would be supplied with the information required to, for example, open the file or dataset. An MPE FILE equation goes part of the way towards tight binding, while IMAGE goes a bit further. If IDIOM had some of this FILE or IMAGE information it too could bind a program more tightly to the actual physical resource.

But as long as IDIOM is an "add-on" to an operating system and a DBMS, it would appear that a policy of loose resource binding should be followed. That is, there is no physical connection between a virtual IDIOM resource and the actual physical resource (if indeed there is a physical equivalent). It is the system designer's responsibility to identify and specify all virtual resources that represent the relevant interactions between the jobs, while it is the programmer's responsibility to ensure that his programs perform consistently with the system designer's IDIOMatic specifications.

If this appears to be a pity, if not a shortcoming, it is well to consider the crudeness of existing binding. After all, one of the reasons for DBMS is make more program/data binding alternatives available to the system designer, and IMAGE at least is far from perfect. Having access to an item on an IMAGE dataset may permit more than the program needs: perhaps the program should only have access to the subset of records that has a certain value in that field.

The main justification for the current loose binding is however that the IDIOM is supposed to describe and interrelate the end-user or application objects, not the physical objects. It may even be argued that the IDIOM should certainly not set itself the task of describing the physical equivalents, and should even scrupulously avoid doing so.

Such an approach would be consistent with the aim of the "conceptual schema" of the ANSI/SPARC DBMS proposals. Like IDIOM, the conceptual schema is supposed to describe the end-user organization in end-user terms, but it differs from IDIOM in going right down to the finest level of application data definition. IDIOM in its current form is not intended to go down that far, though it may well be that it does provide a suitable starting-point and structure for such definition, especially in the light of the discussion about the real nature of "real-time" in (B) above. (Charles Bachman in his ACM Turing Award lecture in 1973 mentioned the "Copernican revolution" brought about by DBMS: the programmer is now a navigator in the database instead of being program-centred. Maybe DBMS needs an Einsteinian revolution: a way of describing the database in a way that adequately incorporates a time dimension? IDIOM does not pretend to do so, but the IDIOM structure does cater for the time factor: the potential of Critical Path-type methods using the IDIOM model, the representation of an event by an IDIOM resource, the "real-time" transformations implied by departmental cut-offs - all stress time. However, possible ways of extending the IDIOM approach down to the level of detail data definition are beyond the scope of this paper.)

Given the loose binding policy adopted in the initial implementation, it is easy to IDIOMatize an existing installation. This can be done on an application by application basis. Though IDEXEC with its implementation-independence does not need the MPE JOB facility, the system designer can incorporate the streaming of existing jobstreams, even though probably at the expense of some concurrency and control over background queuing and dispatch. The minimum requirement that IDEXEC places on a user program is that the program must terminate normally by invoking a standard SL routine (though even this could be forced by an intermediate subroutine). Migration from an existing hard-coded control setup to IDIOMatic control could if necessary be done on a gradual basis, but in most cases a more rapid elimination of non-IDIOMatic system control (assuming that there is any automatic control!) would be worth-while.

In the current HP3000 implementation, IDEXEC supports applications in a process structure in one session or working as multiple sessions. It implements the IDIOM "virtual" processors using normal process-handling. There are ways of warning about potential physical deadlocks, but with loose resource binding there is no foolproof way of making them impossible.

CURRENT PROJECT STATUS

The IDIOM language syntax was defined in Backus Naur Form, and checked for ambiguities using GSA1100, Univac's General Syntax Analyser. IDCOMP is written in Standard Pascal, and generates a transportable intermediate code. IDMGR is similar, IDSYS being a plain MPE file. IDEXEC is also in Pascal, though the implementation-independence is a matter of degree. The aim was to keep all interaction with the host software at the lowest practical modular level.

IDIOM is currently in alpha test. Volunteers for beta test are welcome to apply!

It is clear that the move from loose to tighter binding is desirable, but cannot be done without the active cooperation of designers and suppliers of software systems such as code generators, non-procedural languages, and sub-systems for the physical support of peripherals, e.g. flexible terminal spoolers. In fact very little such cooperation is required, since IDIOM interfaces very easily with programming tools, and for device support provides highly flexible queuing facilities, only the physical support functions being missing.

Interfacing with current data-dictionary systems is more complicated. In a loose binding implementation an IDIOMatic design can happily coexist with a lower-level data dictionary such as Dictionary/3000 or Quasar's Dictionary-plus. An integration of the two levels is certainly desirable, but can well be carried out in later stages of development, with due consideration (and no doubt refinement of IDIOM).

What is the benefit of cooperation for these various classes of suppliers? Firstly it must be clearly stated that there is no competition between IDIOM and these products. The main advantage, however, is that IDIOM provides a standard system-level matrix or support system for them. There is tremendous potential for some synergy here.

Our marketing strategy will be orientated to encourage such synergy with other vendors and users. We see such a large eventual market, because of the generality and hardware and software independence of the concept of IDIOM, that we shall aim for quantity at a lower price. Such a strategy will not work without many partners in the venture. Here too you are invited to volunteer!

Eventually we hope that the market pressure will be created to extend the tighter binding by more detailed interfaces with basic software such as machine operating systems and DBMS's. The advantages to the user community of such standardization are clear for all to see: even IBM has announced that they are seeking a standard consistent end-user interface for their small and medium machine range. IDIOM is well suited to be such an interface for work-processing systems, as its name implies.

ACKNOWLEDGEMENTS

I must acknowledge the major roles played in this development by my colleagues at Synergy Computing, especially Robert Gibson for his patient support and encouragement, and Stephen Davies for his contributions to the design and for most of the programming. Our thanks also to Alan O'Regan and his team of Fiona Main and Jon Schapiro at the University of Cape Town, who carried out the compiler project.

RANDOM DATA ENTRY USING A GRAPHICS TABLET

Dr. Wolfgang Matt
Industrieanlagenbetriebsgesellschaft Einsteinstr. 4
D 8012 Ottobrunn

1. Introduction.

When goods requiring detailed specification are ordered by telephone standard data entry solutions cannot be adopted. The problen arises from the fact that the caller normally gives the specification in random order and the operator has to follow the caller's flow of speech. A solution was found using a graphics tablet on which an oversized keyboard is painted. This keyboard is formed by a grid of labelled squares.The operator touches these squares in the caller's order with a pen similar to a ball pen. The graphics tablet is connected to a HP125 which acts as a preprocesseor. It displays the data entered and transmits them in a fixed order to the HP 3000.

In this paper we describe a simplified data entry problem, design the layout of the keyboard on the graphics tablet, and show what kind of preprocessing has to be done by the HP 125. It will be shown that accepting orders can be continued even if the HP 3000 is down. It will be explained how several HP 3000 can share one floppy disc drive, thus reducing hardware cost. We conclude by mentioning two other preprocessing problems now running on a HP 2645 terminal which will be converted for the HP 125 shortly.

2. Description of the data entry problem.

The system described here was developed for a manufacturer of optical glasses. Optical glasses differ from each other so much that the opticians usually do not have them on stock but oder them by telephone from the manufacturer.

Since most of you are not familiar with optical terms the principle of operation is demonstrated using a more common example of a screw manufacturer. Screws are specified by the type of thread, the material, the length, the diameter, the type of head and the quantity ordered. Now imagine you are accepting orders by telephone. Some customers start the specification with the type of thread, others with the material others with the quantity. This is what we mean when saying that data are given in random order.

It is not practical to design a form for V/3000 since the operator does not have the time to position the cursor to the correct field. One could use an identification tag for each field type, but it is questionable whether the operator can enter the data fast enough. The customer usually talks very fast since costs for the telephone call are running away.

Using function keys could be a solution. A terminal has eight function keys, but you need several hundred for a real application. Such a keyboard can be realized using a graphics tablet.

3. The graphics tablet.

A graphics tablet can be visualized as an inverse plotter. You touch a point on a sensitive area with a pen and the instrument gives you the coordinates of this point. The pen has the shape of a ball-pen writer with a thin cable connected to it. The coordinates of the last point pinned are returend on request of a computer via HP-IB bus.

We now can realize our keyboard. We draw a grid of horizontal and vertical lines on a sheet of paper and label the individual squares (see fig. 1). We use one column for the type of thread, the second for the material, the third for the diameter, the fourth for the length etc. Within each column we label each square by the value of it's contents, e.g. all available diameters. Some fields can be enhanced by colour or shade since we are using ordinary paper.

The basic operation of data entry for one item is to pin one square for the type of thread, one for the material, diameter, length, type of head and quantity, i.e. 6 keystrokes for one complete item. This operation is now very fast and the great advantage is that the pinning can be done in the same order as the customer gives the specification. Corrections can also be done very easily. Assuming the customer corrects a diameter of 3 mm to 4 mm, the operator just pins the 4 mm square.

Some specifications which are rarely used, e.g. extremely thick screws, are not explictly listed on the tablet in order to keep its structure clear and comprehensive. For such cases a numeric column is provided to allow individual digit entry. To define the meaning of the digits we need squares labelled "diameter" or "length". These labels can simultaneously serve as column headings. To enter a diameter of 65 mm the operator would pin "diameter", "6" and "5".

There also exists a column of function keys in the usual sense, like "end of item", "end of order" etc. A very useful function key is the square "ditto". It repeats all specifications of the current item, which can then be individually changed. This is the case when the customer says "the same with diameter 5 mm". The operator pins "ditto" and "5"in the diameter column. That is all.

3. The HP 125 as preprocessor.

As mentioned above the graphics tablet sends the coordinates of the pinned point to a computer. Instead of sending the coordinates directly to the HP 3000, we use an "intelligent terminal" as preprocessor which displays the pinned squares in a readable form to the operator, does some plausibility checks, and sends the data in ordered sequence to the HP 3000.

For this purpose the HP 125 was chosen (1). The HP 125 is a combination of a terminal and a CP/M personal computer. As a terminal it operates as a HP 2621, but can be extended in it's capabilities by downloading firmware to work with V/3000. As CP/M computer it can execute stand alone applications like Visicalc, Graphics or Wordstar using a floppy or hard disc as storage medium. Both features can be combined as it is done by LINK/125. LINK/125 uses program to program communication with FCOPY or QUERY running on the HP 3000.

In our application the HP 125 works in the same way. A CP/M program executing the preprocessing functions runs parallel to a communication program in the HP 3000. The CP/M program is written in 8080 assembler language, the communication is written in SPL. While assmbler language is nesseary on the HP125 side to provide the required speed, any programming language can be used on the HP 3000 side.

The preprocessing starts with the interface to the graphics tablet HP 9111A. The hardware connection is done via the HP-IB bus. Though the HP 125 uses HP-IB for communication with disc, printer or plotter, the HP-IB driver is not documented. With the help of HP we got the subfunction 117 working. The HP 125 continuously monitors the graphics tablet for the pen to be pressed on its surface and asks for the coordinates.

The next step is to transform the coordinates into row and column of the square. When the coordinate is found to lie within a small strip along the boundary of two squares, an error beep is generated. The same is done for unused squares. The sound is generated by the graphics tablet. We use a beep for an error and a short blob as confirmation of a good coordinate.

Having determined the meaning of the square the contents are displayed in a readable form to the operator on the 24th row. The screen is rolled up as new items are entered. Each specificatin (e.g. diameter) is shown at a fixed column on the screen independent of the order the data is entered. The 25th row (usually containing the fuction key labels) is used to label the columns. In some cases individual labels are enhanced, e.g. when the square "diameter" is pinned for digit entry the label "diameter" is enhanced.

Parallel to displaying the specifications on the screen, a record is to be transmitted to the HP 3000 later. The record contains the same information in more condensed form, also in fixed format. The record could be transmitted after each item, but we have chosen to delay it until the end of the order for reasons described below.

5. On-line application.

Besides transmitting complete orders two other communications are implemented.The first one is the matchcode, the second a stock inquiery. The match code consists of two letters of the name, the city and the street name and is entered via the normal keyboard of the HP 125. The HP 3000 returns the address to the screen directly and the customer number (which the operator does not need to know) to the CP/M program. (We used the I/O mapping feature of the HP125 to implement this.) The operator makes the correct choice by pressing a function key. This operation cannot be done fast enough while the customer is on the telepone, but it is done immediately afer the end of the call. Since we want each record transmitted to contain the customer number, we delay the transmission until the call is finished.

The other on-line communication, the stock inquiery, can be done at the end of each item. It is only done in cases the customer wants to know the delivery time. In this case he is willing to wait for the end of HP 3000 transaction.

6. Off-line application.

We have seen that one complete order is stored in the HP 125 before it is transmitted to the HP 3000. Since the HP 125 has 64K of memory we can continue to store orders in cases the HP 3000 is not ready for transmission, i.e. in cases of maintenance or component failure. Except for total power failure the operator can work almost as normal. The only difference is that the operator has to enter the customer number instead of the match code using a list of customers. Stock inquieries are not possible. All orders are saved until the HP 3000 is up again. At this point the ordes are transmitted as fast as the HP 3000 can accept them. (By the way, this can be used to measure transaction time.) We estimated that in the application of the optical glasses the workload of one day can be stored.

7. Operation without disc.

Our application does not use any data files on the disc of the HP 125. The only time we use the disc is to load the CP/M operating system and the application program. If this program never terminates the disc is no longer needed (except when a power failure occurs) and can be physically disconnected. This means that several HP 125 installed in one location can share one floppy disc drive, thus reducing hardware cost.

When the operator pins "end of program" a code is sent to the communication program telling it to terminate, but the CP/M program does not terminate. Instead it completely blanks the screen including cursor and function key labels and goes into a wait loop until the return key is pressed. The power of the grapics tablet can be switched off. A message is displayed when at restart the graphics tablet is without power.

A problem arises when we want to use the HP 125 alternatively as dialog terminal with V/3000. The standard procedure is to load the block format program from disc, run the V/3000 application, switch to local mode afterwards and load the local application program. This method is not practical when the disc is not permanently connected. The solution we found is to start with a slightly modified block format program as WELCOME program. After loading the firmware into the terminal part, the HP 125 is not set into local mode but the data entry program is loaded automatically and starts execution. At this point we achieved what we wanted. We have the block mode firmware loaded and a running CP/M program. The disc is no longer needed. At the beginning and whenever the operator pins the square "dialog", the CP/M program sets the HP 125 into remote mode and goes into a waitloop. The HP 125 now works as a dialog terminal and any V/3000 application can be run. When the communication program is run, it sets the HP 125 back into local mode, which causes the CP/M program to exit the waitloop and to accept digitizings.

9. Conclusions.

It was shown in this paper how a special data entry problem can be solved by using a graphics tablet as an input medium and the HP 125 as a preprocessor. The layout of the "keyboard" can be easily adapted for the individual application, since the design is made on paper. The adaptation of the CP/M program is relatively easy since grid coordinates, square labels, starting columns on screen and in the record are kept in tables. Plausibility checks have of course to be programmed individually.

The preprocessing facility of the HP 125 is not restricted to the graphics tablet. It can also be useful for normal keyboard entry to check and manipulate data before they are transmitted to the HP 3000. Two applications realized on the HP 2645 terminal will be rewritten for the HP 125.

The first application is a keypunch replacement. The operater is guided by field labels and checks on field length and type are made on a character by character basis. The program contains a verifying punch facility which makes a comparison with the original punch on a character by character basis. All this is done without any delay caused by transmission or transaction time, since transmission is done parallel to data entry.

The second application realized on the HP 2645 terminal is a formatting routine as part of a text management system. The text entered is reformatted between left and right margins and asks the operater for hyphenation before the text is transmitted to the HP 3000. This reduces the load on the HP 3000 and provides immediate response for the operator.

Literature

(1) MATTHEN O'BRIAN, Distributed Processing - A Hewlett Packard Solution
Proceedings of the 1981 Berlin International Meeting

type of thread	material	diameter mm		length mm		type of head	quan-tity	.	ditto
bolt	iron	1	1.5	3	5	hexa-goanl	10	1	end of item
bolt with shank	nickle plated	2	2.5	8	10	hexag. w. colar	20	2	end of order
wood	chrome plated	3	3.5	12	15	cheese	50	3	end of program
press-wood		4	5	20	30	round	100	4	dialog program
		6	8	40	50	counter sunk	200	5	match code
		10	12	60	80			6	custom. number
		15	20	100	150			7	
		25	30	200	250			8	
		40	50					9	

Fig. 1 Layout of the keyboard.

The Interactive Office - Technology and People

by Jack C. Armstrong

Jack C. Armstrong & Associates
Portola State Park Road
Star Route 2, Box 245
La Honda, California
USA 94020

Abstract

Implementation of interactive office technologies utilizing Hewlett Packard equipment, as viewed by the coauthor of the LARC Editor/Scribe system, (now TDP/3000). A brief survey of interactive office functions, with emphasis on complete information processing -- data processing, text and graphics processing, management decision support systems, communications, and automation of routine office tasks. Examination of currently available hardware and software, and a look at future possibilities. A major discussion of the human side of office automation - the impact on office and DP staff, organization and management changes. Based on experiences consulting at Hewlett Packard sites in the United States and Europe, as well as the Hewlett Packard office automation research and development facility in Pinewood, England.

In 1975, when I first participated in the design of word processing software for the HP3000, I chose the name 'Scribe' for the text formatter I was implementing. In retrospect, I now wonder if I wasn't paying homage to the first word processors -- the medieval scholars who formatted words for those unable to do so themselves.

Careful examination of the role played by the early scribes brings to light a number of startling observations. First, their job was to improve communications. Initially, I'm sure this was viewed as a convenient novelty, but quickly became a necessity as the pace of human interaction increased -- due in no small part to their efforts. Another, more ominous parallel to modern day may also be observed. As the service provided by these individuals became indispensable, they became a new 'elite', jealously guarding their profession and carefully promoting an aura of mystery about their talents. They often elevated themselves to positions as advisors to those in high places, although their primary qualifications were limited to skills in their own narrow field. Sound familiar?

It is interesting to note that the demise of the early scribes was brought about by two things -- education and automation. It is my belief that our modern society has allowed a new elite to develop (we call them data processing professionals), and these same two factors will cause a radical change in their futures. The most competent of the ancient scribes survived, indeed flourished, by joining the new wave of education and automation of printing. Those who chose to resist vanished into obscurity.

Invention of moveable type in the early 15th century was probably viewed as a disaster by scribes put out of work, but the more farsighted accepted it for what it was, a dramatic improvement in human communications -- and moved on to become tutors, authors, or worked within the new printing industry. Every subsequent improvement in human communications -- telephone, voice recording, typewriters, and now modern office equipment -- has had its share of detractors and promoters, but each, where its usefulness has been demonstrated, has advanced to positions of absolute necessity.

The advent of the industrial revolution, with its overwhelming impact on society, often obscured the parallel development of industry's necessary partner -- the office. As the pace of manufacturing increased, so did the need for control of its operations, product distribution, personnel management, finances, and the myriad of other factors which now complicate our lives. As the simple craft shop became a factory, its owner became a manager, whose job quickly required the aid of accountants and additional staff to record data for his use in making management decisions. With increasing growth, the primary function of this additional personnel became buried in what now seems to be their only function -- paperwork.

Great strides in productivity were made in the manufacturing portion of these new industries, but corresponding increases in office staff efficiency failed to appear. In the middle of the 17th century, when Blaise Pascal invented the Pascaline, a quite usable calculator, it failed to gain acceptance, for reasons all too familiar to those attempting to market office automation equipment a few years ago. Some resistance came from clerks and accountants who feared that widespread acceptance of the devices would place their jobs in jeopardy, but the primary factor was economics. The device was relatively expensive to purchase, was mechanical, and required frequent repair and maintenance. Further, the work it would have taken over was being accomplished by very inexpensive office staff. Given the salaries being paid at the time, there was little financial incentive to ease the life of drudge accountants, so why waste money on them?

Employers who persist in the philosophy of using expensive machinery to support inexpensive employees are in for a shock. Due to the staggering advances in efficient fabrication of modern electronic devices, coupled with soaring wage increases, an entirely new set of rules apply. Today, we must provide our office staffs with tools which both increase their productivity, and allow them to return to their primary function -- gathering the data necessary for intelligent decisions, and making these decisions. More than any other single factor, this means providing them with improved communications.

To understand how communications can improve office productivity, one must understand the role of an office in the life of an enterprise, the people in the office, and the work performed there. This understanding is vital, because the key to an organization's overall productivity may well be hidden in the office. Increasingly, productivity problems lie not among production or factory workers, but among office workers. In the United States, there are 52 million office workers -- the largest segment of the work force and growing at nearly 2 million per year. Undertaking to automate an office with an eye towards improving the production of paper output and later reducing employment levels is short-sighted in the extreme. Office workers whose tasks may be amenable to this sort of cost reduction (typists, clerks, etc.) comprise slightly more than half of the total office staff, but account for only one third of all office salary costs.

Of all the advances in office tools, the telephone has unquestionably had the largest impact, primarily because of its dramatic improvement in human communication. At the turn of this century came tabulating machines, usable calculators, typewriters, and later dictation equipment. That the primary function of telephones, typewriters, and dictation equipment is communications should be obvious, but what of the calculators and tabulating machines? Their function was to assist in the collection and assessment of data -- today we would call them decision support tools. It does no harm, however, to also view them as communication devices -- communicating information in a form most useful to the decision makers.

For some time now, advances in office technology have concentrated on one narrow aspect of office life -- word processing. From the first mechanical typewriters we progressed thru electric typewriters, to devices with magnetic storage capability, to modern single function, single station word processors. Economics played a role in the introduction of clustered work stations, but these offered no substantial increased functions beyond those available to the single station units. Along the way, we succeeded in reducing the time required to produce a printed page, but a major goal was missed. Much of the information needed on that printed page existed in another device -- the data processing system which had grown from the early tabulating machines into the modern computer center.

There existed some economy in utilizing the large computer to perform word processing functions -- both mainframe and terminals could be shared with other users, but the major impetus for a shift to time-shared word processing systems was derived from the ability to integrate data processing and word processing. This ability was not always delivered as advertised, and was not always utilized when it did exist, but the rationale is sound, and remains a valid objective.

To meet these objectives at an HP3000 site, there appeared a number of software products. EDIT/3000 and a contributed program (GALLEY), was used with mixed success at many sites, generally proving the concept, but failing to meet day-to-day needs. The LARC Editor/Scribe system was much more suited to this task, as were several competitive editor/formatter systems. In 1980, Hewlett Packard purchased the LARC system and released it as TDP/3000. There quickly followed SLATE and WORD/3000. These products coming from the vendor have not substantially slowed the introduction of other competing systems, which offer full screen editors, formatters, and numerous other useful features. Unfortunately, many of these systems, including those from

Hewlett Packard, exhibit a tower of babble syndrome, with poor communication between themselves, and poor communication with the rest of the system software.

Alongside the rush to word processing, new decision management tools were appearing -- again both from Hewlett Packard and from other software vendors. The introduction of HPMAIL, with electronic mail and message capabilities adds another dimension to the ability to automate office communications. New office automation tools seem to be announced daily, but one is forced to wonder at the seemingly total lack of integration of these systems with any other systems, even with each other.

To add to the confusion, new hardware devices have been introduced -- new terminals, graphics plotters, laser printers... the list grows and grows. All of them are intriguing, and we are besieged with rumors of new technologies just around the corner -- facsimile transmission, digitized voice store and forward systems, touch screens, even voice recognition. The low initial cost of new microprocessor systems (and frequently excellent decision management software available for them), has lured many to establish isolated islands of processing capabilities within their organizations. What we are witnessing is an attempt to push technology into the office -- often simply for the sake of the technology. Meanwhile, back at the office, the paperwork continues to stack up to the ceiling. (Ever higher, as we acquire faster printers!)

The term "Office Automation" is an unfortunate choice of words, but seems destined to stay with us for some time. What is needed is not an automated office, but an 'intelligent' one. Current office procedures have grown out of a combination of tradition and expediency. In the past, there were very real constraints on what an office worker could accomplish, given the limited set of tools at their disposal. These constraints are being removed by new technological advances, but the force of tradition is guiding the application of these new technologies down old, well worn paths.

Several facts seem to have escaped the attention of the designers of modern office systems. First, their perception of who will use their systems appears to be drawn from a Charles Dickens novel. The old stereotypes of "clerks don't make decisions", and "managers don't type", are clearly obsolete in modern offices. Many systems appear to be oriented towards a level of intelligence which borders on insult -- with endless menus, prompts, and an effort to make them 'simple' to the point where they are incapable of performing many necessary, but nontrivial, functions. Modern office workers are increasingly intelligent, and have an ever higher degree of discretion concerning what tasks are to be performed, and in what order.

Another, more subtle factor, is that the shape of office organizations is changing -- in large part because of the possibilities offered by the new technologies, which modern managers appear to be more aware of than many vendors give them credit for. New communications facilities allow offices to be geographically dispersed. Modern offices may be organized along functional lines, and may shift form rapidly, to meet modern dynamic changes.

If we stand back and take a detached, objective view of the functions which an office staff is intended to perform, we see other ways in which people can work. We should note that the primary function of office workers is decision making. What we see them doing today is an inordinate amount of 'paperwork' which is not a primary function, but rather a support function. In examining current office operations, start at the top - what does the manager do? Ask where the manager obtains the information necessary for his decisions, how does he arrive at his decisions, and how are these decisions recorded and acted upon. If the office is viewed as a decision making body, then we may examine how best to support this process.

We will find that what is needed are 'clusters' of capabilities, not in terms of physically proximate hardware units, but access to common capabilities by members of a common task force. This must first and foremost include communications, then the data relevant to their operations, and finally the ability to analyze and manipulate this information. All of this is a support function, to allow the group to perform their primary mission -- make decisions. Use of this system will be nothing more than an ancillary part of their job -- like using a telephone is today.

The central computer facility must change from an isolated data processing center into an integral part of a major communications network, providing a facility for storage and large-scale processing of information, while acknowledging that other, independent processing centers are contributing and withdrawing information. That it will provide a means of controlling and securing information flow should for the most part be invisible to the end user and his system.

Future data processing staffs must function as information managers for the entire organization. Companies must realize that the data processing department should operate as an integral part of all corporate activities, not as an in-house service bureau funded by departments on the basis of services rendered. Internal DP people should become something like an in-house OEM -- providing others with the means to do things with their own equipment and systems -- not providing the total service.

That other, local information processing 'centers' will appear as nodes in this network will discomfit many data processing department managers, but is inevitable. The introduction of microprocessors and other small computers into daily life is not to be denied. Many of the reasons for using these local systems are facetious, and in reality may be for reasons of personal ego, paranoia, and other equally unresolvable human reasons. A signal feature of the new age of office automation is that many decisions will be made for reasons which are political and tactical, not strictly based on rules of logic.

An office is not a factory, but a society, with complex structures and seemingly meaningless rules of human interaction. What must be squarely faced and dealt with is that there exist major differences in cognitive and psychological styles. Different individuals perform better using those modes of operation which suit them best, and demonstrating an alternative which you find preferable may not convert them to your techniques. Forcing them to adopt your techniques will quickly lead to their adopting their own alternative -- abandonment of the system. The new users of office technologies are not the captive audience data processing managers are accustomed to dealing with. Like customers in an open, free market system, they will only 'buy' what they want -- not what they are told to use.

Individual adoption and use of any office technology will soon be based on whether or not the user perceives it to be an enhancement of his or her daily work experience. Increasingly, higher wages will cease to offset undesireable working conditions. More importantly, as individuals become more valuable to an organization, it becomes critical to support their work style in order to not only enhance their work, but to deter high employee turnover. The trend will be towards more important employees, as office technology gives them increased powers. Be cautious of the thought that automation, with greater ease of use, will lead to more easily replaceable employees. Machine 'operators' may be easier to replace, owning to less emphasis on physical skills such as high-speed typing, but the valuable office worker of the future will perform more intellectual tasks, and it is their knowledge, not skill, which will be difficult to replace.

To assure the adoption of new office systems by office workers, they must be allowed to participate in the design of their own tools. It is the office worker who knows the task to be performed, and a systems designer who proves a sympathetic listener will soon learn what these are. Beyond the basic necessary functions, are two more points -- how the system works, and what control the user has over it. Individual preferences must be catered to, and as the user learns to utilize the new capabilities, he or she must be able to 'expand' its functionality, by specifying the order of tasks to be performed. These systems must be capable of levering the intellect, to allow the user to excel in new and more productive ways.

Users are in the best position to improve their own productivity. All too often, it is corporate decisions (or non-decisions) which get in the way. They also can best discriminate between two types of tasks -- those soul destroying, mechanical jobs which quickly become error prone and should definitely be taken over by a machine, and those which require a degree of latitude and intelligent choice which should always remain the option of the user. This is often an extremely difficult decision for a system designer to make. As we shift from mechanizing routine office tasks to augmenting the user's intellect, we will find these users utilizing the system in new and innovative ways never imagined by its designer. If a workable link has been forged between the user and the designer, we will find the user suggesting more and more creative enhancements to the system.

What we will find ourselves doing, in addition to mechanizing some office functions such as text preparation, is extending processing power to all parts of the organization, and linking these functions in an information network. With intellectually acceptable interfaces to this network, each office worker will become a valuable component of an immensely productive information center. As users become skilled in the use of this system, accessing data themselves, formatting their own reports, formulating trial analyses of data, current data processing staffs will be free to pursue the development of still more creative tools for their use.

What I am predicting is a dramatic shift in the application of computers -- from mechanizing old rigid methodologies, to use as an extension of man's intellect: as a general purpose, powerful tool with which to explore and experiment. If this seems too extreme, I would ask you to carefully examine the interactive report generators currently available, the electronic spread sheet programs, and some of the better designed word processing systems which allow users to formulate and store a sequence of frequently used commands. Many of these systems and features are being used today to perform extremely complex operations, by individuals who would be terrified if you suggested that they 'program' something. Yet that is precisely what they are doing -- often in amazingly clever and useful ways.

For some time now, people have spoken of the 'software gap', meaning that the advances of new hardware developments have outstripped our ability to produce new software. A new 'gap' is now appearing -- between the user and the power of the new technologies. We have developed marvelous new applications, which obviously fill real needs of modern industry. However, access to these applications remains out of reach of many users, due to limited and inappropriate user interfaces. The physical aspects of ergonomics are being handled -- nonglare CRT screens, comfortable keyboards, chairs, work stations, and the like -- but the area of psychological ergonomics has barely been touched.

The successful office system will address these needs only after system designers take a very careful look at who the users are, how they chose to work, and what they are likely to use the system for. Even after such a study, a system which cannot be molded and extended, (by the user), will face limited acceptance. The successful data processing manager of the future must view the new users as his most valuable resource. He must study their habits and their needs, and he must cultivate their value as high-level system designers -- by providing them with educational opportunities which train them in the possibilities inherent in the new technologies. They already understand the tasks to be performed, and if they can be shown the possibilities which new tools might provide, they will specify which tools they need, and how they wish to make use of them.

GRAPHICS SYNERGY

Stephen J. Wilk
Business Computer Group
Hewlett-Packard Company
Cupertino, California
USA

OUTLINE

I. Information Explosion

II. Graphics Market

III. Understanding the Functioning of the Brain
(Is a picture worth 1000 words?)

IV. The Different Types of Graphics

V. HP3000's Business Graphics Products

VI. Synergy of Text, Data, Graphics

VII. Implementation Considerations

I. Information Explosion

Millions of pieces of information are created daily. If we were to document all of the information stored since the beginning of mankind, 75% of that information has been developed in the last 10 years. In the United States, for example, over 240 billion pages of computer printout was created in 1980, up 25% over 1979. If your having difficulty coping with the information explosion today, imagine how much greater the problem may become in the future. For the amount of information available today will double in 5 years and quadruple in 10 years!

One of the challenges of the 80's is to take that raw data and convert it into meaningful information for managers and business professionals to make sound decisions. Note the distinction between data and information. There is an excessive amount of untimely and irrelevant data available to most managers today which explains why so many computer printouts go unread. Information, on the other hand, may be a summary of only the most important elements that affect the company's business.

Today, very few organizations view information as a corporate resource. And yet, the ability to manage information will be the measure of success of business organizations during the 1980s. In fact, some consultants predict that information will become the next major asset of a company, surpassed only by people, material, and financing. Looking into the future, I think most businessmen agree that computerization in business will continue and the control of business is going to increasingly depend on how well the manager can interact with that computer to get the information he needs, whether it be for the shop floor, Purchasing, Production, Personnel, etc. The challenge for the data processing professionals, is to provide easier access of data stored on the computer to the decision maker, the manager.

II. Graphics Market

Up until now, businesses have concentrated on automating the routine aspects in business. During the 1970's, corporate expenditures were focused on the increasing the productivity of the blue collar and farm worker. In a study performed by Arthur D. Little, Inc., productivity gains clearly relate to the capital investment in productivity tools:

	Capital Investment	Productivity Gain
Office Worker	$2,000 U.S.	4%
Industrial Worker	$25,000 U.S.	90%
Farm Worker	$35,000 U.S.	185%

The 1980s is the decade of the white collar worker. Already, productivity gains are being realized by the increasing use of word processors and personal computers. And business computer graphics is the next major thrust in increasing the productivity of the business professional. Listed below are summarzations of studies performed by HP's Product Marketing Group at Information Networks Division.

Size

According to Frost and Sullivan, business computer graphics will sustain a 40% annual growth rate moving from a $400 million (U.S.) market this year to $1.6 billion (U.S.) in 1985. Some forecasts predict a 59% annual growth in business graphics with a total computer graphics market reaching $7.5 billion by 1985. In addition, the slide making market will grow to $3.5 billion in 1985. Although market projections vary, none leave any doubt that the growth will be explosive. These numbers compare quite favorably with the growth of the clustered word processing market from $1.5 billion to $6.5 billion in 1985.

Use

A recent survey by International Resources Development, Inc. revealed that 71% of the respondents planned to enhance their graphics capabilities in the coming year. This report found that applications are numerous in corporate and financial planning, market plannning, operations analysis, and the boardroom environment. It is also interesting to note that the most common use of computer generated graphics (85% of respondents) was for presentations. This underscores the demand for high quality output which is one of the three major trends in the business graphics market.

Future Trends

In the future, customers will be looking for:

1.) Higher quality output (more character fonts, fat lines, etc.) on high quality media (35mm) which will reduce their dependence on time consuming and expensive professional graphics services or reduce the cost of these services.

2.) Improved access to data that will make graphics a more valuable tool to non-computer professionals.

3.) Faster, more interactive, lower cost multi-function workstations.

III. Understanding the Functioning of the Brain (Is a picture worth 1000 words?)

The information in this section is discussed in much greater detail in a paper by Marv Patterson entitled "Graphic Representation of Numeric Data". Marv is a R & D Section Mgr. at HP San Diego Division and has been studying how human perceive information as well as helping develop new graphical interfaces between humans and computers. The paper is fairly technical in nature, and provides many references to support the conclusions we will rapidly draw here.

Let's take a look at communications from a human point of view. That is, how can we as humans interact with the computer? In doing this, we would like to talk about our human data channels, not the computers. In others words, our five senses, the capacity of these senses, and the processing power of these channels.

Of the five senses or channels that humans have for receiving information from the outside world, we will not consider - smell, touch and taste. They will not be considered primarily because they are very slow for information transfer. Also technologically, they haven't been utilized yet in computers with the possible exception of touch where there has been some pioneering work done with the blind. Hearing and sight are the two senses that we will be concentrating on. So first let's consider the sense of hearing.

Hearing

We listen at the rate of 150 - 300 words per minute depending, of course, on who's talking and who's listening. Looking at the ability to read at 600 - 1200 words per minute would indicate that the hearing sense may not be the most efficient communications channel. Although listening is not an efficient way to get information from the computer today, it is a very natural and friendly way to communicate. Because of this, voice and musical notes are becoming more and more popular in new computer systems.

Sight

Our visual channel, the eyes, is characterized by high resolution, the ability to detect many colors and shades, and by very high speed. Sight is the most powerful channel that we as humans have. We are able to intake and process a tremendous amount of data in a very short amount of time.

Our visual capacity is established by three characteristics:

- First, is the number of smallest indivisible part of a picture that our eye can detect, the pixel. Each optic nerve has approximately 1 million elements. Two eyes, therefore provide two million picture elements.

- Second is the number of colors that we are able to perceive. Tests show the average person can detect approximately 160 colors and shades.

- Third, is the number of individual pictures per second that we can discern. Tests have shown this to be approximately 7 pictures per second.

Adding these numbers together, 2 million picture elements, plus 160 colors plus 7 pictures per second permit us to draw a startling conclusion. In other words, to store one picture detected by the eye with the same resolution as the eye would require 114,000 words of computer memory. THAT'S RIGHT, A PICTURE IS NOT WORTH 1000 WORDS AS WE WERE TAUGHT, BUT IS WORTH 114,000 WORDS.

Now that we have the information in the brain, let's take a look at how this information is processed. Although there is still some controversy as to how our brain actually processes data, the view expressed here is certainly widely held (as Marv Patterson's paper describes). The findings state that a different class of functions are performed in the left hemisphere of the brain than are performed in the right hemisphere. Also, except for at a very young age, there is no functional redundancy in the brain. That is, each portion of the brain has specific and exclusive tasks to perform that are performed nowhere else. Let's take a look at this lateralization of functions.

The left hemisphere or the left side of the brain controls language functions, math, arithmetic, music theory, reading, logic, and sequential analysis. This is the logic and symbolic center of the brain.

The right hemisphere controls visualization of spactial and geometric images, contains simple language, mostly nouns , musical expression, and non-verbal ideation, a concept we will return to in a moment. Let me give you an example of the use of right hand side of the brain. Picture a 3 inch black cube with an orange strip around it suspended in space. Now rotate this cube slowly in space. In the process of doing this, you are utilizing the function of the right hemisphere of your brain - the spactial side.

On the other hand, it would be difficult to use spactial visualization to represent this symbolic relationship, E=MC2. We must not under emphasize the importance of the left side of the brain. Symbols and numbers are important to us all. We do, however, need to better understand and utilize the right side of the brain as part of this man-computer dialogue.

In conclusion, the ideal computer system needs both text and graphical capabilities - text for the symbolic communication and graphics for the visiospactial communication. Now that man can utilize graphical data, the right side of his brain, to enhance communication with computers, let's talk about how your management can more effectively analyze your data.

IV. The Different Types of Graphics

There are three different kinds of computer graphics available in industry today:

1. Design Graphics - where computers and graphics peripherals are used to aid engineers in the design process. We will not be addressing design graphics in today's session.

2. Real Time Display Graphics - this consists of applications such as radar, and online process control, where operators make decisions while viewing real time graphics. Although HP makes some real time display graphics products, we will not be addressing this aspect of computer graphics in today's session either.

3. Data Display Graphics - this is the area of graphics that we would like to discuss today. By this we mean data that is collected and displayed in any of the following forms:

   a. Bar and pie charts. Both can be used to show total values (by the size of bar or pie), as well as component values, such as breakdowns of (say) 'sources of money received' and 'where the money was spent'.

   b. Scatter diagrams. These show the (imperfect) relationship between two variables, such as the number of air travelers that fly on Mondays, on Tuesdays, etc.

   c. Time series charts. These are perhaps the most widely used form of graphics, showing the value of one or more variables versus time. The value scale can be linear or logarithmic.

In addition to Data Display Graphics, there are many other types of graphics that can be found in business and industry:

- Text - Text plays a critical role in graphics - for listing points that the speaker is discussing, for showing subject titles, and for identifying components and values of a chart.

- Hierarchy charts - such as organization charts and module charts, are widely used.

- Sequence charts - such as flow charts, may not be quite as popular as they once were, but they still have a role to play.

- Maps - both two-dimensional and three-dimensional.

- Layouts of rooms, buildings, shopping centers, etc. convey much infor mation in relatively simple diagrams.

V. HP3000's Business Graphic Products

Hewlett-Packard's user objectives for Buisness Graphics are the same as for our Office Systems. In general, they are to:

1. Increase individual productivity
2. Decrease direct cost
3. Increase effectiveness or value added capability
4. Improve job satisfaction

Since our users span many key professionals in both the office environment and the applications development environment, let's take a look at some of his/her activities which can utilize computer graphics assistance.

Graphics-User Activities

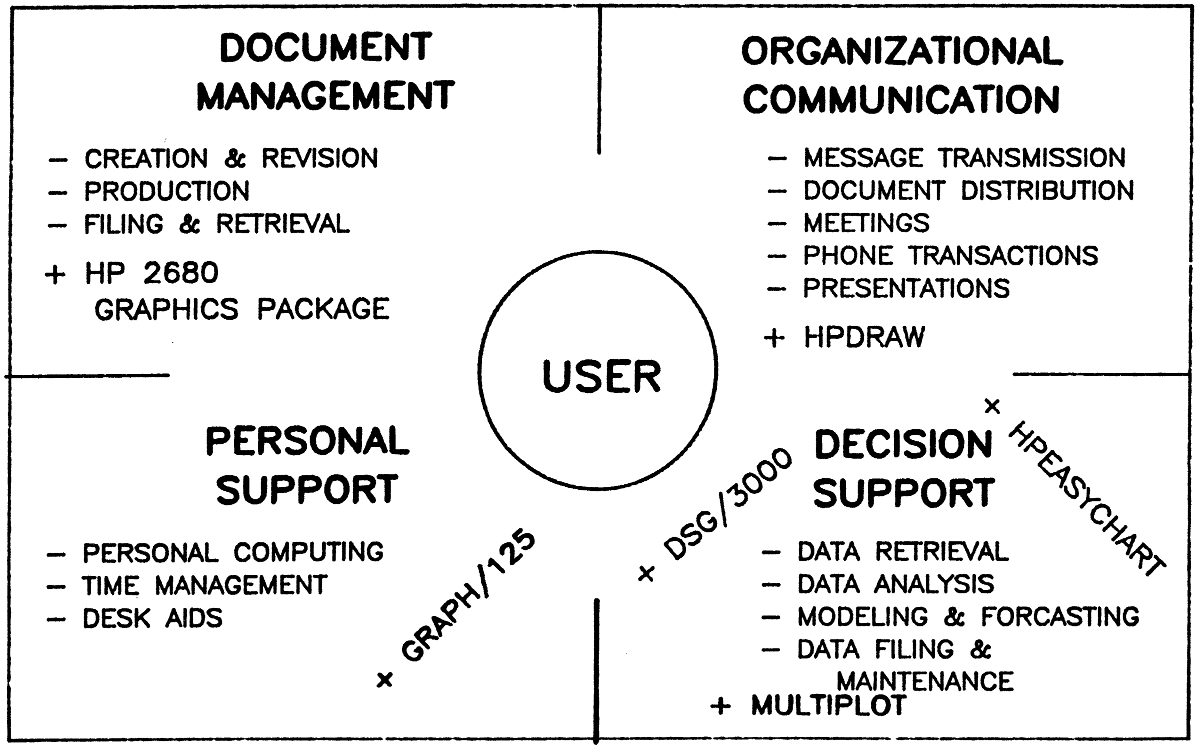
- Create, modify or update a graph, slide, text/graphics document, or organization chart, GANT or PERT chart
- Plot or print a graph, slide
- Store a graph, slide, text/graphics documents
- Send a graph, slide and track it
- Retrieve and update graphs and slides
- Check, annotate, forward and dispose of graphs, slides, text/graphics document
- Present graphs, slides at meetings
- Analyze numerical data through graphs
- Prepare a rough draft of visual aids
- Prepare a manual master with text, graphs, and slides
- Develop graphics application programs using a graphics language
- Querying data files and displaying data in picture form
- Mailing graphs, slides, and text/graphics document
- Display a room layout indicating scheduled meetings

The above list shows the need of computer graphics in each of our four broad office areas of Document Management, Organizational Communication, Decision Support and Personal Management as well as in the four areas of application development. The following figure tell the Hewlett-Packard business graphics story.

As you can see, "HP offers a wide-choice of business graphics products that will improve the productivity of a broad range of users from the smallest,least sophisticated systems to the largest HP3000s available."

THE INTERACTIVE OFFICE

USER PERSPECTIVE



DOCUMENT MANAGEMENT

- CREATION & REVISION
- PRODUCTION
- FILING & RETRIEVAL

+ HP 2680 GRAPHICS PACKAGE

ORGANIZATIONAL COMMUNICATION

- MESSAGE TRANSMISSION
- DOCUMENT DISTRIBUTION
- MEETINGS
- PHONE TRANSACTIONS
- PRESENTATIONS

+ HPDRAW

USER

PERSONAL SUPPORT

- PERSONAL COMPUTING
- TIME MANAGEMENT
- DESK AIDS

× GRAPH/125

× DSG/3000

DECISION SUPPORT

× HPEASYCHART

- DATA RETRIEVAL
- DATA ANALYSIS
- MODELING & FORCASTING
- DATA FILING & MAINTENANCE

+ MULTIPLOT

+ HEWLETT–PACKARD PRODUCTS

VI. Synergy of Text, Data, Graphics

A look to the future

The use of computer graphics is sure to increase, and rapidly. We recently saw projections which indicated that by 1990, 95% of the CRTs for use in business (for terminals, personal computers, etc.) will use color, which implies graphics. Hardware prices are falling and intense competition is occurring in the software markets. So users will find graphics becoming more and more economical.

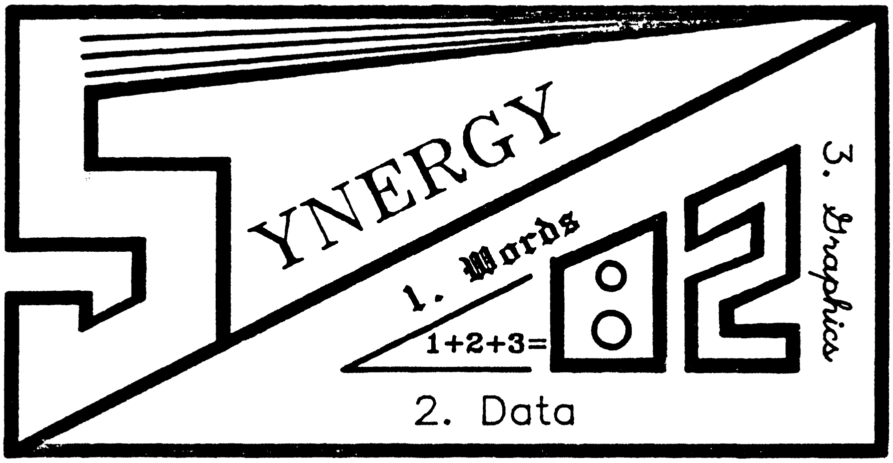
Hewlett-Packard's Direction - Integrated Information Management

In 1982, Hewlett-Packard strengthened its business graphics offerings with the introduction of three new products;

HPDRAW	- a graphics presentation text and figure design package
HPEASYCHART	- a simple, easy to use chart maker for any office user to produce line charts, bar charts, pie charts, and scattergrams interactively within 10-15 minutes - no computer knowledge is necessary
EPOC - G	- the merging of graphics and text on the 2680 laser printer

By allowing text, data and graphics to merge into a single document, Hewlett-Packard has taken the best of both the left and right hemispheres of the brain and make them available for improved decision making. The examples shown on the next pages are actual documents produced at HP during the introduction of HPDRAW, HPEASYCHART, and EPOC-G last June.

SYNERGY EXAMPLES



. . . an NPT Summer Spectacular . . .

. . . a HOT one scheduled for the first day of summer . . . .

. . . dedicated to the proposition that the whole is greater than the sum of the parts . . .

. . . and HP's got it ALL TOGETHER better than the rest . . . .

Now RECRUITING for

the SYNERGY SYNGERS

. . . and SYNERGETTES* . . .

. . . a ribald band of wandering minstrels dedicated to the proposition that . . .

. . . what you lack in VIRTUOSO you can make up for in GUSTO!

Come SIN g (oops!) with us!

See you local RECRUITER today:

Dawn Chesk, 47U, x4291

*Steve Schield, 48N, x3086

Susan Grant, 48S, x4351

Jim Geers, 48N, x4084

Debi Smith, 43U, x3852

HEWLETT PACKARD

INFORMATION NETWORKS DIVISION • 19420 Homestead Road, Cupertino, California 95014, Telephone 408 725-8111

FROM: Marilyn Johnson - Cupertino
Chris Kocher - Cupertino
Shirish Hardikar - Pinewood

DATE: 2 June 1982

TO: Business Computer
Group Sales Center
cc: Divisional Marketing

SUBJECT: **HP Business Graphics -- The Competitive Edge**

Another HP First!

Information Networks Division introduces new business graphics software products which provide the HP 3000 users with the strongest graphics capabilities of any of HP's competitors today — graphics for the manager, secretary and EDP professional. With these products from IND and new firmware for the HP 2680A Laser Printing System from Boise Division, the HP 3000 now has the capability to integrate the processing of words, data and graphics! Everyone in the customer's organization now has simple access to Integrated Information Management.

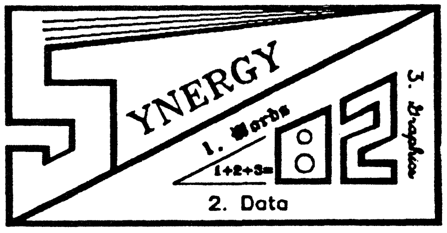
No other vendor can offer this today!

To You Today . . .

The attached Field Training Manual introduces you to the use of business graphics for Integrated Information Management. You'll see the new HP 3000 Business Graphics Package with HPEASYCHART, HPDRAW and enhanced DSG/3000, complete with data sheets. The Sales Reference Manual from Boise Division, mailed separately, will give you the details on *merging words, data and graphics* on the HP 2680 LPS using the new HP 2680 Graphics Package and firmware upgrade. Together these two manuals will provide you with detailed information on selling the HP 3000's newest capabilities. Additional technical product information will be sent in a separate mailing to the SEO prior to announcement.

. . . and Then to the World . . .

HP will make a *world-wide announcement* of these products on *June 15* at the National Computer Graphics Association Exposition in Anaheim, CA. NCGA is a major graphics exposition highlighting the leading edge in graphics technologies and applications.



note: logo done with HPDRAW and printed via TDP

During the *SYNERGY teleconference* on June 22 and the *European NPT* we will discuss and demo the new capability to merge words, data and graphics. You will really see how easily all the products can work together and how quickly they can work for you in selling more HP 3000s.

From: Pat Wilcox Pat
To: BCSC Team
Chris Kocher
Marilyn Johnson
Shirish Hardikar
Ruann Pengov

4/30/82
Subject: DRAW
your own
conclusions

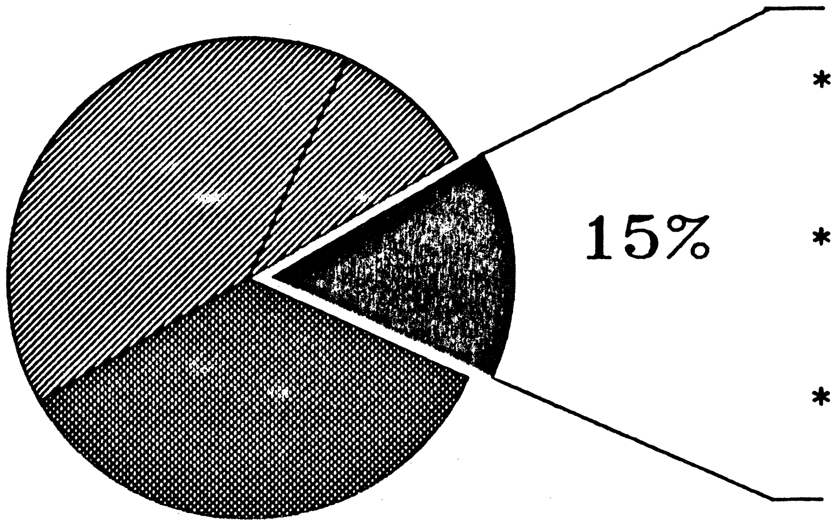
Come hear about our upcoming GRAPHICS introductions! The Office Product Champions (...with a little help from product management...) will fill you in on details on Tuesday, May 18th @ 9:00 in the STANFORD room.

The products we will cover include:

HPDRAW
HPEASYCHART
DSG Update
Laser Graphics

These new products will really fly!

FIXED OVERHEAD COSTS DOWN



With the Introduction of new production line :

* DEPRECIATION ALLOWANCE UP
* MAINTENANCE DOWN TO $90K
* PROPERTY TAXES KEPT AT $100K

. Chart prepared by DSG/3000 . Text prepared by HPDRAW

. Text & Chart merged by HPDRAW

SECTION 4

Production Cost Reductions

4.0 Introduction

The following paragraphs describe the production cost reductions associated with the 2680 Laser Printer toner loader lid. The location of the toner loader lid in the 2680 is shown in the highlighted area in Figure 4-1.

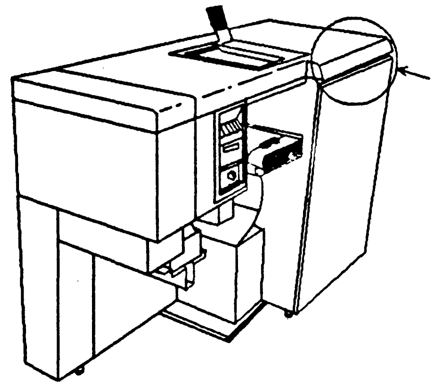


Figure 4-1. Toner Loader Lid Location

4.1 Toner Loader Lid

The Toner Loader Lid (See Figure 4-2) was previously produced using a vendor casting process plus an in-house machining and painting procedure. A recent production change order changed the process to a 100% numerically controlled vendor machined part. Details of the machined part are shown in drawing D-2682-20182-1.

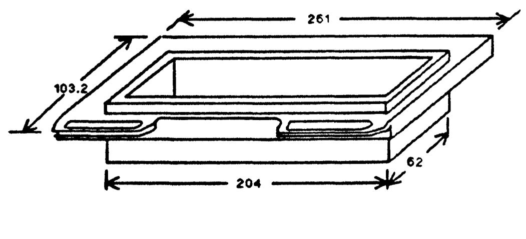


Figure 4-2. Toner Loader Lid

4.2 Cost Analysis

The factory cost for the completed lid using the previous process was \$104.31. In addition there was considerable rework cost due to the 30% reject rate. The new process gives a completed cost of \$60.92, resulting in a cost saving of approximately \$43 per part. Another benefit of the new process is that the reject rate has gone to 0%, providing additional cost savings.

The bar chart in Figure 4-3 shows the breakdown of the costs before and after the process change.

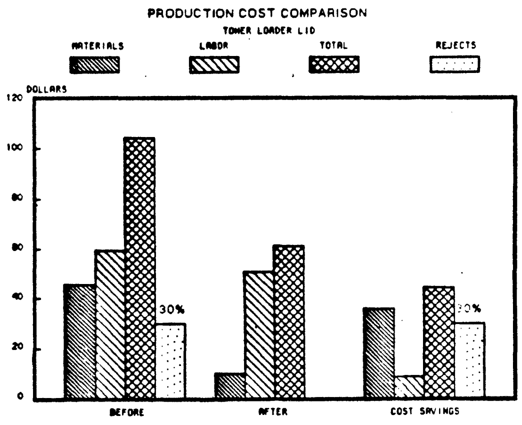


Figure 4-3. Cost Analysis Breakdown

VII. Implementation Considerations

Getting started with graphics

When an organization first considers using computer graphics, from what we gather in our discussions, their initial interest is in continuing to produce the same types of graphics that they have been producing - but faster and at less cost. There is just no comparison in the speed with which drawings, slides, foils, etc. can be produced by computer in contrast to manual methods. And, depending upon volume, the cost of computer graphics can be substantially less than manual graphics.

From a management standpoint, this means that the charts displayed in a management chart room can be much more up-to-the-minute. It also means that different formats, scales, colors, etc. can be tested out economically, to see which format gives the best comprehension of the information, for a particular chart.

There is still another big benefit in computer graphics for management. Their speed and relatively low cost make the answering of many 'what if' questions feasible, such as, "What would our expense picture have looked like over the past year if we had...?" Management can ask questions questions like that and the new charts can be displayed very rapidly.

Hardware

The necessary elements of a graphics system are: 1.) a processor and its memory (generally lots of memory), 2.) disc storage plus a file management or database management system for handling the stored data, 3.) a display (usually a CRT), 4.) graphics software, 5.) probably some provision for hardcopy output, and 6.) possibly a data communications facility for obtaining data from another computing system.

There are several alternatives for acquiring computer graphics capabilities. The graphics system can be a self-contained, stand-alone system with all the elements required, often designed especially for graphics use. Or it can consist of a device connected to a general-purpose host computer. The host can be either a mini-computer or mainframe, and can be in-house or at a time-sharing service bureau.

Graphics Software

Graphics software is another important consideration when looking at a computer graphics system. On some of the larger computer systems graphic software offers many powerful features. Some systems allow customizing of the output to meet the specific needs of the users. Capability of customizing reports is highly desireable because managers from different areas may want to review the same data base but from different points of view, and in different formats. A computer system also should allow data base access and ability to update the information. One must analyze the graphics needs before deciding on a graphics system.

Graphics software can be based on a large computer system or on a computer terminal where the data might be stored in a small cassette.

1. Computer based software consideration.

   These are the large mainframe computers with graphics software packages that allow you to look at the state of your business from your data base graphically. Some things to consider are: initial implementation, hardware flexibility, how easy it is to modify the software application programs to meet your particular needs and how large a staff you need to implement the system.

2. Terminal based software consideration.

   If your needs are for much smaller system where you can input your own data at the local terminal, you may want to consider a terminal based graphics software package. The advantages of those are: a) that they allow local control, b) existing application programs will work and c) your central computer resources are not used. No matter what kind of graphics software you choose, one of the overriding considerations is ease of use. How much programming, formatting and inputting of data is required to turn your tables of numbers into usable graphis and charts?

Supporting business graphics

Summarized below is a very condensed excerpt of a 39 page pamphlet entitled, "Choosing the Right Chart" by Paller, Szoka and Nelson, price $8.50 U.S., available from:

AUI Data Graphics
1701 K Street N.W.
Washington, D.C. 20006
U.S.A.

Paller, Szoka and Nelson say that users' views of their graphics needs can, and often do change quite dramatically once they begin to receive graphic output. They will ask to have more elaborate charts, such as 'exploded' pie charts, stacked bar charts, charts with double scaling, and so on. Many companies that start with a limited package soon find that they must purchase another, more flexible one, because the users really did not know their needs until they started using the system. Thus starting out with a time-sharing service may be wise, just to get needs better defined.

Paller, in his seminar, said that a company planning to start offering business graphics, using their existing mainframe or mini-computer, should be prepared to spend about $30,000 for a flexible graphics package. In addition, they will need to spend about $40,000 for medium to high resolution output devices, plus an additional $40,000 for support for the first year. This comes to a total of about $110,000. A more modest start can be made for about one-half this amount, but the software will probably not provide enough flexibility for longer-term use, he believes.

If an organization chooses to 'start small', a micro-computer system can cost less than $10,000 for the graphics hardware and software. Such a system can be suitable for use within a single department, for instance, and generally no special people are needed. This option provides a limited number of charting options, and the quality may be satisfactory for presentation graphics.

Supporting business graphics is a lot like supporting end user programming. But business graphics has a few differences which we should point out.

As far as graph quality is concerned, what seems obvious when you look at someone else's graph is not always so obvious when creating your own charts from scratch. So companies that are going to let end users create their own charts need to first teach them some rules for generating good graphs. "Choosing the Right Chart", gives some useful ideas on this subject.

Some Objections

Before a computer graphics system is installed, or during the early days of its use, a new user organization is likely to encounter one or more of the following objections to it:

1. Management resistance - some may find it too costly; or they say they prefer to see the actual reports, because they feel more comfortable with numbers than with graphs.

2. Rising expectations - some organizations may start with an inexpensive computer graphics system, but may soon become dissatisfied with it. For one thing, the quality may not be adequate; diagonal lines show up as 'staircases' rather than straight lines, or curves are distorted. Or they may want to make what they feel are simple changes to the system - such as showing two time series with two scales on the same chart, or showing two charts on the same page or screen - only to find that it is impractical to make those changes with their inexpensive system. So they get frustrated with the new system.

3. Need programmer help - creating new graphics formats may require the use of a graphics programmer. This means that managers and other users are back in the familiar scene of waiting for the programmer to become available, waiting for the program to be written, finding that the new format if not quite what was wanted, and ending up with a non-trivial cost.

4. Not familiar with graphics - the problem may be as simple as a poor choice of scales. If the scale is too small, viewers may miss significant variations in values. If the scale is too big, mountains will be made out of molehills. If someone uses a logarithmic scale, a quite-different interpretation of the data may be drawn as compared with the use of a linear scale. Another problem area is the choice of colors for a particular graph. A poor choice of colors can adversely affect the viewers' reaction to the graph, regardless of how important its message is. If used well, color can add to the message, not detract from it. Also, some users may not be too familiar with the common graphical techniques. For instance, one such method is the 'overlay' technique. A chart is shown, giving some information. Then an overlay is added, which adds more information. A second overlay adds still more information, and so on, so that the observer receives information gradually and can thus

comprehend it more easily. Further, blinking can be used on a CRT display, to draw attention to some value. And numbers can be displayed next to critical high or low points on a graph, so that viewers see not only trends and relationships, but also actual values. Thus, while users may be unhappy with their early use of computer graphics, as they get more 'professional' graphics, satisfaction should increase.

REFERENCES

1. "A Brief Perspective on Business Graphics" by Marilyn Johnson and Chris Kocher, November 7, 1981, of HP's Information Networks Division.

2. "Computer Graphics for Business", February 1982, Vol.20, No. 2 of EDP Analyzer.

3. "Business Graphics - An Effective Means of Improving Managerial Productivity", by Chris Kocher of HP's Information Networks Division.

4. "Business Graphics - One Picture May Be Worth 1000 Pages of Computer Printout", by Bill Fuhner, May 1982, of HP's San Diego Division.

Multi-lingual Capabilities of HPWORD

by Alma C. Rodoni

Introduction

As computer solutions expand into the office and to new end user levels we have seen a number of new pressures on vendors who supply products. One of these is the move to integrate all of an organization's information so that it may be easily accessible by anyone, at anytime, in the form in which they need it.

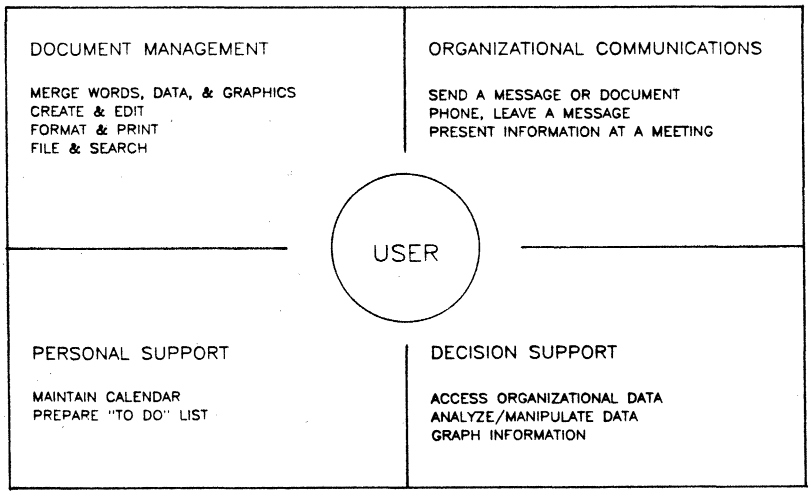
Not only must this information be highly integrated but it must also be easily communicated to the next desk, the next building or to the other side of the world.

As multi-national companies expand into worldwide markets these information needs become more critical. From an end user's view, one can further expand these information needs to a list of office activities for which an innovative end user might consider computer assistance.

These activities can be categorized into four broad areas: Document Management, Organizational Communication, Decision Support, and Personal Support (see the figure below). This discussion will focus on the Document Management area, however it is important to recognize that it is only a portion of the total information solution, that is getting information to all members of the organization, when they need it and in the right form.

THE INTERACTIVE OFFICE

USER PERSPECTIVE



Multi-lingual Capabilities

This paper will focus on what will be referred to throughout as Multi-lingual capabilities of HPWORD. HPWORD is Hewlett-Packard's word processing product targeted for secretarial use for general business correspondence. At this point it is important to define Multi-lingual. Multi-lingual is the capability to run the HPWORD subsystem in different language versions simultaneously on the same HP 3000. This is to be distinguished from simply having a number of native language versions (i.e. English, German, French) which are discrete packages that could not be run simultaneously on the same system.

Let's briefly describe a customer scenario that might require this capability. A company like HP, based in the United States, with organizations in Germany, Switzerland and France must communicate to these entities on a frequent basis. Given that a large portion of the European market speaks at least two languages, this capability is important for communications between countries in Europe, but also for communications to the company headquarters in the United States.

For example our group in Boeblingen, Germany may want to create a document in English to be sent to the United States for review. The Multi-lingual capabilities would allow Germany to create the document in English on the same HP 3000 using their German native language terminal. This would also allow the the United States to receive German and English documents.

As a supplier of Office Systems products, Hewlett-Packard has made a committment to worldwide markets. Worldwide markets are a major focus for our software development and it is HP's philosophy that the software design take place in the country where the product is to be used. We believe the user and marketplace expertise lies in these locations. Therefore the HPWORD development team in the U.S. has worked closely with each software center group in the localization of HPWORD.

Software Considerations

The actual software localization can be made simpler if the product has been designed with worldwide markets in mind. It is important that this be a priority in the design and development. Part of this can be accomplished by identifying those code segments that require language specific changes- these should be isolated for modularity. Whenever possible it is best to use table references with these code segments instead of hard coded instructions.

The following discussion will focus in detail on the two code segments within HPWORD that have been identified as key elements for localization purposes. The first is what is called the **Document Language** and the second is the **Interface Language**.

Document Language- When using HPWORD users at a single installation are allowed to create and edit documents in various languages. The document language field will appear as a new entry on the HPWORD "create document" menu. This gives users the choice of various supported languages. They are allowed to select the desired language right on the HPWORD menu. The primary reason for selecting the document language is to ensure that automatic hyphentation will be performed correctly. The hyphenation exception dictionary maintenance task has been enhanced to allow modification of exception dictionaries in any supported language.

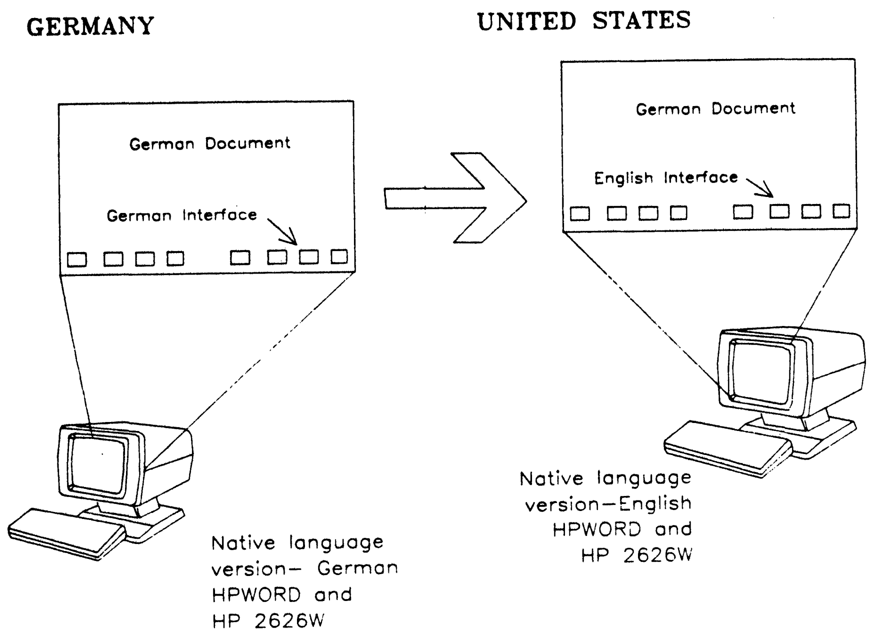
Perhaps the most important benefit is that users can create documents in other languages using their native language terminal. This can be accomplished by using the control (CTRL) key to initiate the special character sequence which allows muted characters for such things as accented words. To use this capability there are no additional hardware requirements.

It may not be possible to view all foreign language special characters on your terminal screen; however, you can produce the relevant characters from the same keyboard. Also the italics enhancement for Roman Extension characters is not visible on your terminal screen (they are shown as underline).

The supported document languages are:

Language	Recommended Keyboard
American English	USASCII
British English	UK
German	DEUTSCH
French	FRANCAIS azM
Canadian French	FRANCAIS qwM

The document language is distinguished from the language of the user interface. Therefore users can use their native interface language (described below) to view or edit any foreign language document.


GERMANY
UNITED STATES
German Document
German Interface
German Document
English Interface
Native language version– German HPWORD and HP 2626W
Native language version–English HPWORD and HP 2626W

In addition to various document languages, an HPWORD user can run in any of several interface languages. This is defined as follows:

Interface Language- The interface language determines the language used for all messages, menus, date formats and softkeys. Users can run in any of the languages listed above.

Each installation has a "primary" interface language, and optional "alternate" interface languages. The primary interface language is selected during installation. All of the HPWORD utilities will run in the "primary" interface language (with the exception of the hyphenation maintenance task which may edit the hyphenation dictionary for any available language). By running HPWORD.PUB.SYS at the primary entry point, the primary interface language will be used.

For each language supported, however, there is an alternate entry point, that can be accessed by typing:

```
                                       run hpword.pub.sys,language.
```

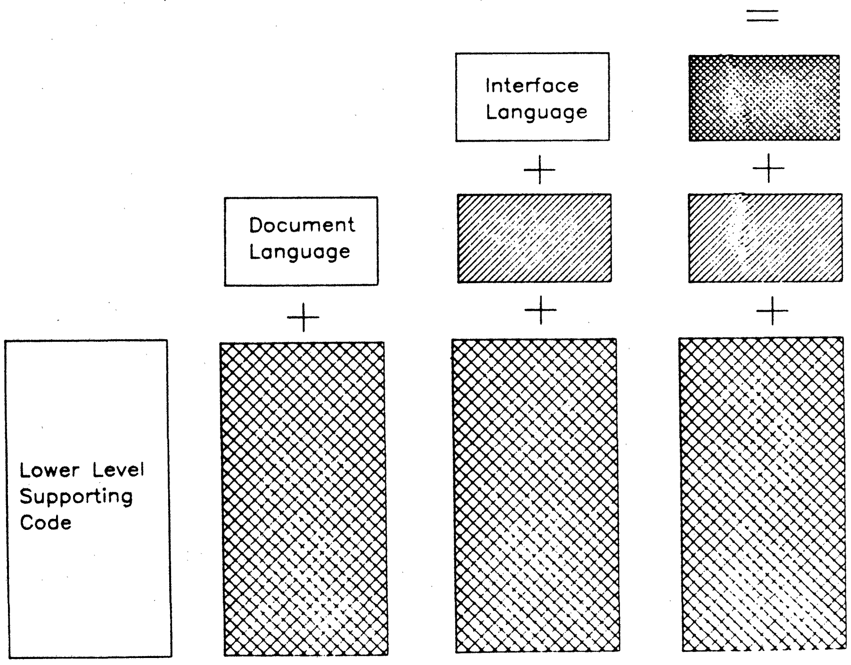
The interface languages and entry points available for HPWORD are shown in the following table:

Language	Entry Point
American English	AMERICAN
British English	BRITISH
German	GERMAN
French	FRENCH
Canadian French	CANADIAN'FRENCH

For localization purposes then, HPWORD can be described by three different modules; the supporting code, the document language and the interface language. Technically the software is structered for maximum flexibility in this regard. The only constraint or dependency is that to run an interface language the document language must also be installed on the system.

The local software centers develop the code for their language version for the document and interface language, and integrate that with the lower level code structure or the supporting code.

HPWORD Native Language Version



Hardware Considerations

To provide a total solution, a word processer must really be a combination of several pieces, including hardware and software. That is, having software alone is not a word processing solution, a user needs a terminal (CRT) and some output device, preferably letter quality.

As other language versions were developed, a number of hardware issues had to be addressed. The terminal must be an integrated piece for a true word processing solution; from an end user point of view it must be accurate in keyboard design, layout and screen presentation. As we began the expansion of HPWORD to other languages we found each language presented unique problems with respect to keyboard translation and layout. These ranged from the simple to the very complex. For example with the German version when we initially recieved the actual keyboard translation from the German software center we found the words (for the syntactic keypad) were too long to fit on the keycaps.

One of the design objectives as mentioned earlier was to allow users to create a document in any language from their native language terminal. This presented some unique problems and required several changes to the HP 2626W to allow access to Roman Extension characters (and displaced USASCII characters from foreign keyboards), these are outlined below:

An invariant mute accent keystation was assigned for all diacritical marks (umlaut, ^, `,', tilde). Each mute accent is accessible using CTRL+[optional SHIFT]+{mute accent key}. The only key stations prohibited are those associated with current "blue dot" HPWORD functions (i.e. discretionary hyphen and required space).

Invariant key stations were assigned to all Roman Extension characters and displaced USASCII characters that cannot be produced using mute accents. Each of these characters will be accessible using CTRL+[optional SHIFT]+{key station}.

This is in addition to any mute or Roman Extension access available on any foreign keyboard. This means that special characters (the 'e' on French keyboards) will be available in two ways: as a single keystroke and as a mute character.

A single template will be printed in the manuals which will document the key stations to use to form Roman Extension or displaced USASCII characters not directly accessible from the keyboard.

The more complex problems came with the keyboard layouts for native language terminals. Many European languages required a different keyboard layout or special keys (French upper case accented characters). This often required some firmware changes in the terminal.

The French version required program ROM changes for the shifted top row numerics the Bold/Bold Roman Extension character set ROMS will be used for all non-USASCII keyboards primarily because of Canadian French needs.

As a result of these hardware issues HP has formed a standards committee to address keyboard layout issues across our product lines. This task force assisted in the standardization of the Roman Extension set for foreign language terminals.

Of course equally important for a true word processing solution, users must be provided with good letter quality output.

The HP 2601A environment files are used to specify what set of print wheels should be used for printing a document to an attended HP 2601. The environment files also contains a mapping from HPWORD internal codes to output codes for each print wheel, so that HPWORD can accomodate foreign and non-standard print wheels without printing nonsense for characters that do not exist, or do exist, but at a non-standard position on the print wheel.

Three environment files are supplied by HP: PICA01, ELITE01, and PROP01. These files describe the most commonly used print wheels for 10-pitch, 12-pitch, and proportional printing on the 2601.

An environment file will describe four character sets, one for each of the Roman, Bold, Italic and Bold Italic fonts known to HPWORD.

Training and Documentation

As we continue to meet new users in markets like those present in the office, training and documentation needs become more and more important, in fact, the self paced training and documentation is an integral part of HPWORD. Very often the translation, typesetting and reproduction of this material required the most time and effort.

HPWORD has a reference guide, a quick reference guide and a self-paced training kit which contains nine training modules. These materials include pictures of the screen showing relevant text and screen labeled keys, all of which must be reproduced with the appropriate presentations in the proper language.

Packaging

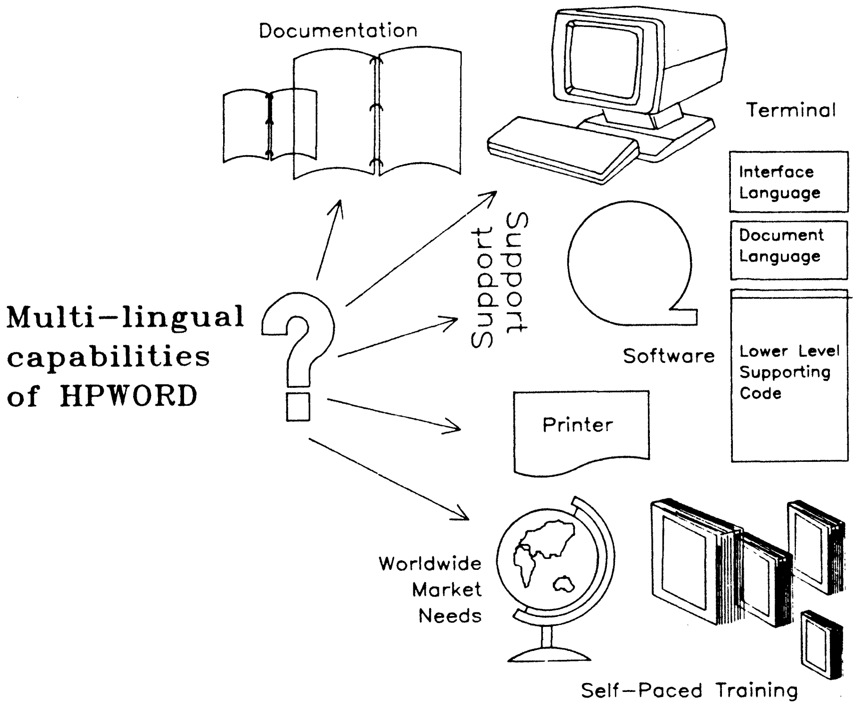
Certainly with this new Multi-lingual capability there are marketing consideratons. How should this Multi-lingual capability be packaged? Where is the percieved value for our customers?

With very few technical limitations, these capabilities could be packaged in a variety of ways. The document languages and interface languages could be packaged seperately for ultimate flexibility or they could be packaged as a unit.

Before discussing the packaging analysis it would be very useful to further define "Multi-lingual capabilities of HPWORD" from a customer's point of view. Given our initial defintion of Multi-lingual, (the capability to run the HPWORD subsystem in more than one language on the same HP 3000), it became necessary to define just what the term "run the HPWORD subsystem" meant. Was it just the ability to run many different document languages? Did it mean different interface languages? Did it imply that both were together? How would support issues be addressed?

As described earlier a word processing solution is really the combination of several elements: the software, a terminal, a printer and the necessary training. If it requires all of these elements to make up a word processing solution the language itself really becomes less relevant. All of the pieces must be translated and useable.

The Multi-lingual capability, therefore is everything that HPWORD represents- the supporting code, the interface language, the document language, the training, the documentation, the terminal and the printer.



Ultimately, then, we believe the customer's perceived value is in the entire solution.

Presently HPWORD is available in English, German, British/ English, French and Canadian French (the latter two have an eight to ten week delivery).

The English version of HPWORD has just been enhanced to include many new capabilities and support many new configurations including the Multi-lingual capability. These capabilities include the integration of text and graphics, four function math, column manipulation, double underline, template documents, asynchronous remote support (using Bell 202/212 modems), and DSN/DS support.

During the first quarter of 1983 those versions listed above will complete localization with the second release features and at that time HPWORD will have complete "Multi-lingual" capabilities (the price of this capability is yet to be determined) as described earlier. Concurrently, additional language versions of HPWORD are under development.

Electronic Mail In the Interactive Office

HPMAIL

Commercial Systems Pinewood
England

IF170982

1982 HP3000 International Users Group Conference,
Copenhagen, 25-29 October.

Title: **ELECTRONIC MAIL IN THE INTERACTIVE OFFICE**

Author: Ian J. Fuller,
Commercial Systems Pinewood,
Hewlett Packard Limited,
Nine Mile Ride,
Easthampstead,
Wokingham,
Berkshire,
England,
RG11 3LL

Telephone: Crowthorne (STD 034 46) 3199.

Abstract:

This paper will explain some of the design considerations behind the implementation of Electronic Mail on the HP3000 and will show how these were implemented in the product. Particular attention will be paid to the design of the User Interface and the thinking behind the facilities provided in the first release.

There will be some discussion of the design and implementation of the HPMAIL Transport System which enables messages to be exchanged in a network of HP3000s with minimal impact on the user.

Topics to be covered:

1. Functional overview of the product:

   Accepting mail
   Addressing users
   Store mail
   Sorting incoming and outgoing mail
   Moving the mail
   Delivering mail to recipients
   Notifying recipients
   Tracking mail progress
   User features - editing, filing etc.

2. Implementation:

   Major Modules in the Product
   Mail Procedures
   IPC Files
   Tracing and Error Reporting

3. The HPMAIL Data Bases:

   Function of the data bases: directory, storage and composition.
   Reasons for using IMAGE.
   Overview of data base design.

4. The User Interface.

   User Interface overview for those not familiar with the product.
   HPMAIL "areas" - the "electronic desk".
   User Interface Structure.
   Reasons for adopting a command based interface.

5. Transport System overview

   How we tackled the implementation of a store & forward system.
   Concepts behind Transport System components.
   Why use PTOP?

6. Conclusion.

Aspects of HPMAIL

1. Functional Overview

2. Implementation

3. HPMAIL Data Bases

4. User Interface

5. Transport System

Figure 0
IF170982

Figures Required:

Title Slide.

0. Summary of what is to be covered.

1. Functions of a Mail System.

2. Major software components of the product.

3. Schema diagram for Local Data Base.

4. Schema diagram for Global Data Base.

5. HPMAIL Object Hierarchy diagram.

6. User Interface structure diagram.

7. Data and control flow for Transport Mechanism

1. Functional Overview of the Product.

HPMAIL is an electronic mail system based on the Hewlett Packard 3000 range computers. It is designed to enable users, who may not be familiar with computers and their associated technology, to interchange information effectively throughout their organisation. This paper provides an overview of the design of the product, the data structures and program code modules that make up the product and some insight into the reasoning behind some of the design decisions that we had to make as we developed the product.

The basic facilities required of any electronic mail system are summarised in Figure 1 and explained further in the following sections.

ACCEPT MAIL

With HPMAIL we made the decision that the product would transmit any information that could be held on the HP3000 system. This could range from a few lines of simple text to a complex report containing, perhaps, reports producted by a word processor such as HPWORD, graphics produced by DSG/3000 or HPDRAW and even MPE program and data files. We decided to adopt as flexible a structure for these items as possible, enabling the user to organise them into "Packages" of logically related items. At the same time, we realised that the majority of HPMAIL messages would be simple text and made it very simple for this type of message to be accepted into the system.

ADDRESS USERS

HPMAIL can be implemented as a multi-computer system, but it is important in the office market that users be shielded from the details of the configuration of their particular network. To this end we implemented an addressing scheme where HPMAIL users are registered with the system in such a way that a sender of a message does not need to know whether recipients are located on their computer or on one many miles away, connected through a complex network. It also allows the administrator of a HPMAIL network to re-configure it without users having to change the way in which they address other users.

STORE MAIL

The system must be able to store mail, before transmission, on intermediate computers in a store and forward network and on the behalf of users. Because of the potentially large space requirements for messages on a system HPMAIL has adopted the strategy of "sharing" information wherever possible; thus when a message is delivered to several users on the same computer only one copy is stored and pointers are set up linking the message to each user's In Tray.

SORT THE MAIL

Mail in transit must be sorted into queues for transmission to the required destination and for distribution at the destination computer, a process analogous to the operation of a sorting office in a manual mail system. This can be a time-consuming process, involving the analysis of large distribution

Functions of a Mail system

1. Accept mail 2. Address users 3. Store mail	User Interface
4. Sort outgoing mail	Mailroom
5. Move the mail	Transport
6. Sort incoming mail	Mailroom
7. Deliver to recipients 8. Notify recipients	User Interface

HPMAIL also:

9. Track mail progress	Mailroom
10. Provide a flexible & powerful user interface	User Interface

Figure 1

IF170982

lists and so we decided to have this function performed by a background process, called the Mailroom.

MOVE THE MAIL

Mail must be transported between computers as economically as possible. HPMAIL uses the Program to Program (PTOP) communication facility provided by DS/3000 for this task. These potentially lengthy and error-prone operations are, again, handled by autonomous processes known as the Transport Manager, Master and Slave Trucks.

DELIVER TO RECIPIENTS

Once arrived at its destination computer, mail must be delivered to its recipients and the recipients must be notified of their new mail if possible. This work is also handled by the Mailroom program.

These are the basic facilities that should be offered by any computer-based messaging system. HPMAIL offers some additional features to the user, amongst which are:

TRACK MAIL PROGRESS

In order for users to have confidence in the operation of the Mail System they must be able to track the progress of messages that they have sent. HPMAIL provides five acknowledgement levels that users can put on a message and the Transport System will keep users informed of the progress of their messages according to the acknowledgement level that they have set.

PROVIDE USER FEATURES

The HPMAIL User Interface provides users with powerful facilities to create and store messages on their "Electronic Desk", making HPMAIL far more than a simple message system. Examples of the facilities provided are a Work Area in which items may be constructed prior to incorporating them in messages, a Filing Cabinet in which users may store messages for future retrieval in Folders named by themselves and a Distribution List Area where they can keep distribution lists of users with whom they correspond for easier compilation of messages. When we designed the HPMAIL User Interface we placed considerable stress on "ease of use" for the non-technical user while maintaining the capability for the product to be used from any terminal that can log on to the HP3000. This enables users to access their mail even when they are away from their office if they have a portable terminal and access to a telephone.

2. The Implementation of HPMAIL

One of our principal design objectives in implementing HPMAIL was that the product should be easy to maintain, test, debug and update. To this end we decided upon a "layer concept" for the software, with each layer communicating via rigidly defined interfaces. This enabled development of the Transport Mechanism to proceed relatively undisturbed while the details of User Interface design were refined through a series of tests, prototypes and evaluations. The task of integrating these components, often a major headache in software projects, proved to be a relatively simple task with HPMAIL and the software has proved to be very robust in use.

Figure 2 shows the major software components of the product. There follows a brief explanation of the functions of each component.

USER INTERFACE

The User Interface program is repsonsible for all the user interaction in the product and provides the user with the "Electronic Desk" described previously.

MAILROOM

The Mailroom is a background job streamed by the system operator and is charged with the sorting and delivery of local mail, that is, mail that is destined for users on the same computer.

TRANSPORT MANAGER

The Transport Manager is reesponsible for scheduling the communication of messages between HPMAIL computers according to the current message load and the availability schedule set up by the HPMAIL System Administrator.

MASTER & SLAVE TRUCKS

The Master Truck is controlled by the Transport Manager and is responsible for the set-up and control of the DS/3000 connection with a remote computer where it uses the Slave Truck program to receive the mail and put it onto the remote data base.

CLOCK TICK GENERATOR

This runs as a son process of the Transport Manager and provides it with a "tick" every fifteen minutes to cause Transport Manager to re-schedule its transmission priorities as necessary.

CONFIGURATOR

The configurator is a VPLUS/3000 application program used by the HPMAIL System Administrator to build and configure the HPMAIL data bases. Facilities are provided so that the Administrator does not need to have any detailed knowledge of IMAGE/3000 to control HPMAIL.

Major Components of HPMAIL

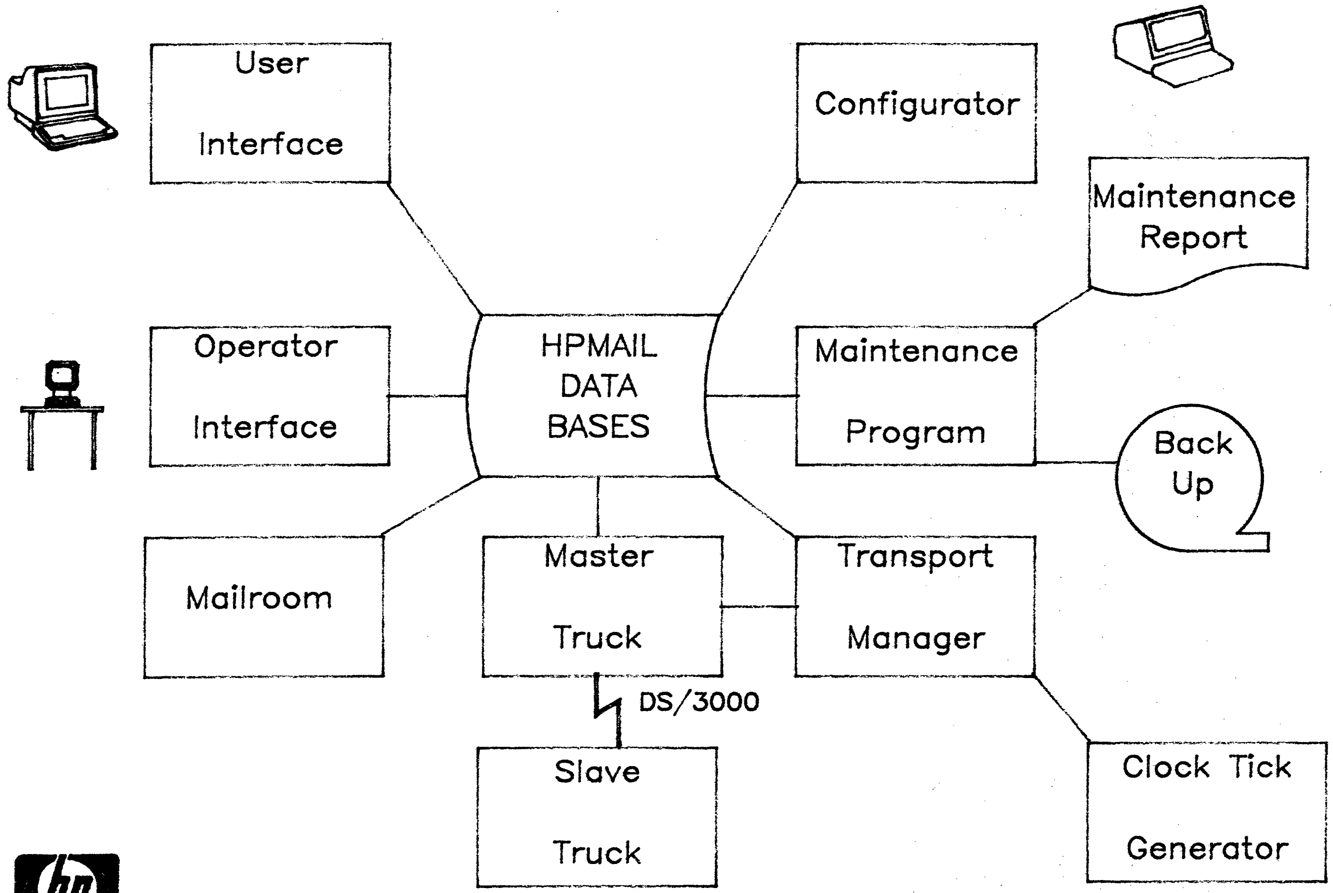


Figure 2
IF170982

OPERATOR INTERFACE

This is a set of functions, invoked by User Defined Command (UDC) to enable the system operator to control HPMAIL and determine its status. Examples of operator commands provided are: MAILON and MAILOFF to enable and disable the system; MAILSTATUS to provide a status display for the system; MAILSHOWNODE to display the number of messages awaiting transmission to remote computers and MAILMAINT to start the Maintenance Program.

MAINTENANCE & REPORT PROGRAM

The Maintenance Program is run periodically to provide verification of the intactness of the HPMAIL data bases, to collect garbage and to generate statistics which are formatted and printed by the Report Program. Regular running of the Maintenance Program is a vital part of maintaining a reliable HPMAIL system.

The programs mentioned were developed simulteneously over a period of eighteen months and in order for this work to proceed reliably a set of techniques was derived before detailed design began to ensure that this work could proceed smoothly.

Three techniques we adopted in the development of the product deserve mention here as being important to the way in which the product evolved. These are the use of "Mail Procedures" to access the data bases, IPC or "pipe" files to communicate between the components of the product and the standardised tracing and error reporting mechanism used throughout the product.

MAIL PROCEDURES

The HPMAIL software is based around two IMAGE/3000 databases, called LOCAL and GLOBAL, which are discussed in the next section. The application programs that access the data bases, for example the User Interface, Mailroom, Transport Manager and the Trucks, do so through a set of intrinsics called "Mail Procedures". These Mail Procedures are on a Segmented Library and so are accessible to all the programs. They perform the low-level data base access functions and insulate the programs that call them from the details of the Data Base structure; in fact the Transport programs do not have a single IMAGE call in them, all interaction is done through these procedures. Examples of the Mail Procedures we implemented include:

MOPEN - to open the Mail Data Bases.

MCREATE - to create a new item.

MEXPLODE - to return information about the contents of an item, for example the messages in a user's In Tray.

MATTACH - to attach one item to another, for example, to attach a new message to a user's In Tray when delivering it.

MDELETE - to delete an item.

MCLOSE - to close the Mail Data Bases.

Other Mail Procedures were implemented to perform common functions required by different modules, for example comparing two names. In all about thirty Mail Procedures exist on the HPMAIL Segmented Library.

These procedures were designed, tested and made available before any coding was done on the programs that would use them, so providing a firm base upon which development could proceed.

IPC FILES

As well as using the two IMAGE data bases, the various components of HPMAIL communicate with each other using the Inter Process Communication (IPC) or "pipe" facility provided by the MPE-IV Operating System. This method of communication has the advantages of being very well defined and "sparse", reliable and provides an "interrupt" mechanism, whereby a program can wait for a message on its pipe telling it what to do next. Once again this convention aided the product implementation as modules were able to be tested using simple test harnesses before attempting to integrate them. In fact early test versions of the product were tested on the MPE-III operating system which did not have the IPC facility by using standard sequential MPE files and the IPC files were introduced as soon as MPE-IV was installed on our development machine.

There are three IPC files defined in HPMAIL:

MRIPCIN - the Mailroom Input Pipe which is written to by the User Interface when a user "MAILs" a message instructing the Mailroom to collect it from the User's Out Tray and deal with it. It is also written to by the Slave Truck when it receives mail from a remote computer bound for the local computer, alerting the Mailroom to deliver it.

TMIPCIN - the Transport Manager Input Pipe which is written to by the Mailroom to inform it about new Mail bound for remote computers which it should attempt to send and also by the Clock Tick process which runs as a "son" of the Transport Manager and send it a message on this pipe every fifteen minutes to cause it to re-schedule itself to take account of any change in the availability of routes out of the local computer.

MTIPCIN - the Master Truck Input Pipe. The Transport Manager sends instructions such as "open a DS line to computer X" to a Master Truck which attempts to perform this operation, then reports back to the Transport Manager on its Input Pipe the success or failure of the operation. The Transport Manager can support up to eight Master Trucks simulteneuosly and each has a separate Input Pipe, referred to as MTIPCIN0, MTIPCIN1 etc.

TRACING AND ERROR REPORTING

Since HPMAIL consists of a number of inter-communicating programs and each program consists of a large number of modules, each separately compiled, it is important that status information about the success of each operation is returned to the caller. It is also important to have a trace facility built into the product so that errors can be traced and diagnosed with minimum trouble.

In HPMAIL a single data structure, called the "Mail Common Area", is used to communicate shared data such as completion status, file numbers and user information between the various procedures in a program and with the Mail Procedures. This is very much in line with products such as VPLUS/3000 or IMAGE/3000.

Each procedure in the product (and there are over 300) takes the Mail Common Area as its first, and sometimes only, parameter. The convention was established that all the procedures call a Mail Procedure called MTRACE immediately upon entry and before exit. MTRACE registers the procedure's identity in the Mail Common Area and can also be made write a log message to a file if required.

Error reporting is centralised in another Mail Procedure, MERROR, which interprets the status words in the Mail Common Area and can produce a customised error message from the message catalogue for each error, together with as much additional information as can be deduced. Since all the trace messages are derived from the message catalogue they can, if necessary, be translated to other languages.

The tracing level of these procedures can be varied by setting a Job Control Word (JCW) or by passing a run-time parameter (PARM) to the program. Thus, when debugging a program, the engineers can easily cause full trace information to be generated and errors can be located more easily. This level of control was particularly important in testing the Transport Programs, where it was not practical to use the MPE DEBUG facility. These tracing facilities have remained in the released product, but cause little performance penalty since they are normally disabled.

3. The HPMAIL Data Bases

HPMAIL uses two IMAGE/3000 data bases to contain the user and network directory information as well as the users' "Electronic Desks", all maintained securely and reliably.

All the HPMAIL programs on a single computer work with the same data base set. Items that are simulteneously "owned" by more than one user, for example a message delivered to two In Trays, are held physically only once per system, with pointers to assert individual ownership.

The HPMAIL "GLOBAL" data base is a directory of all users known to this system. The "LOCAL" data base contains local directory information about the users supported there, network directory information to control inter-computer transmission and mail items. Mail items can be anything from In Trays, messages, queues of messages, individual text items, distribution lists or MPE files "imported" onto the data base - in fact nearly anything a HP3000 can store.

The HPMAIL data bases perform three functions: directory, storage and item composition.

DIRECTORY

The directory function enables HPMAIL to identify a user when they sign on and determine those items that they can access. The directory also identifies users on mail distribution lists and enables user names as entered to be expanded to the full "canonical" form of name, locatoion and sub-location. The network directory functions enable HPMAIL to determine which computers to contact to send mail to a given location, how and when to contact the computers, and what telephone number or Public Data Network (PDN) node name to specify.

STORAGE

The storage functions enable HPMAIL to hold everything in the IMAGE data bases so that the information is always available. Thus HPMAIL will take a HPWORD document and "import" it onto the Local data base so that at some later time the Transport System will be able to send it to another computer without requiring access to the sending user's MPE Group, or having to worry if the user deleted it.

COMPOSITION

The composition functions enable HPMAIL to handle "hybrid" message types and have the same message in several In Trays or Filing Cabinet Folders at the same time, and to operate queues of messages destined for different locations in the network. This leads to the important concept in HPMAIL that the simplest message contains two parts: a Distribution List and some (usually textual) content. The message itself is a single item which "contains" these two others. This split enables the Transport Mechanism to refer to the message as a whole, while the Mailroom can access the Distribution List, its primary concern, without having to scan the text. The end users also derive the benefit that they do not have to read the message distribution list but can go straight to the text.

HPMAIL uses the IMAGE/3000 data base management system for many reasons, of which almost the least important is that it is a data base management system.

The first reason is that it is solid and well-established and provides a firm foundation for a reliable product.

It also offers its own space managment, allocating new space as required and handling direct access problems transparently.

Due to the "hybrid" nature of most HPMAIL items, the chain management system of IMAGE is very important, handling pointer chains very easily.

IMAGE uses shared data storage and shared buffering, adding to efficiency and performance. It has a good locking mechanism to prevent unreliable operation due to multiple updates of the same item.

Finally, an IMAGE data base can be dumped for security purposes very much faster than many MPE files.

The main drawback to using IMAGE is that it is not designed to hold non-structured items such as MPE files, and getting them in and out of the data base ("importing" and "exporting") is a fairly lengthy process. We estimate, however, that the time required to implement the project would have nearly doubled if the facilities of a system like IMAGE had not been available.

HPMAIL LOCAL Database

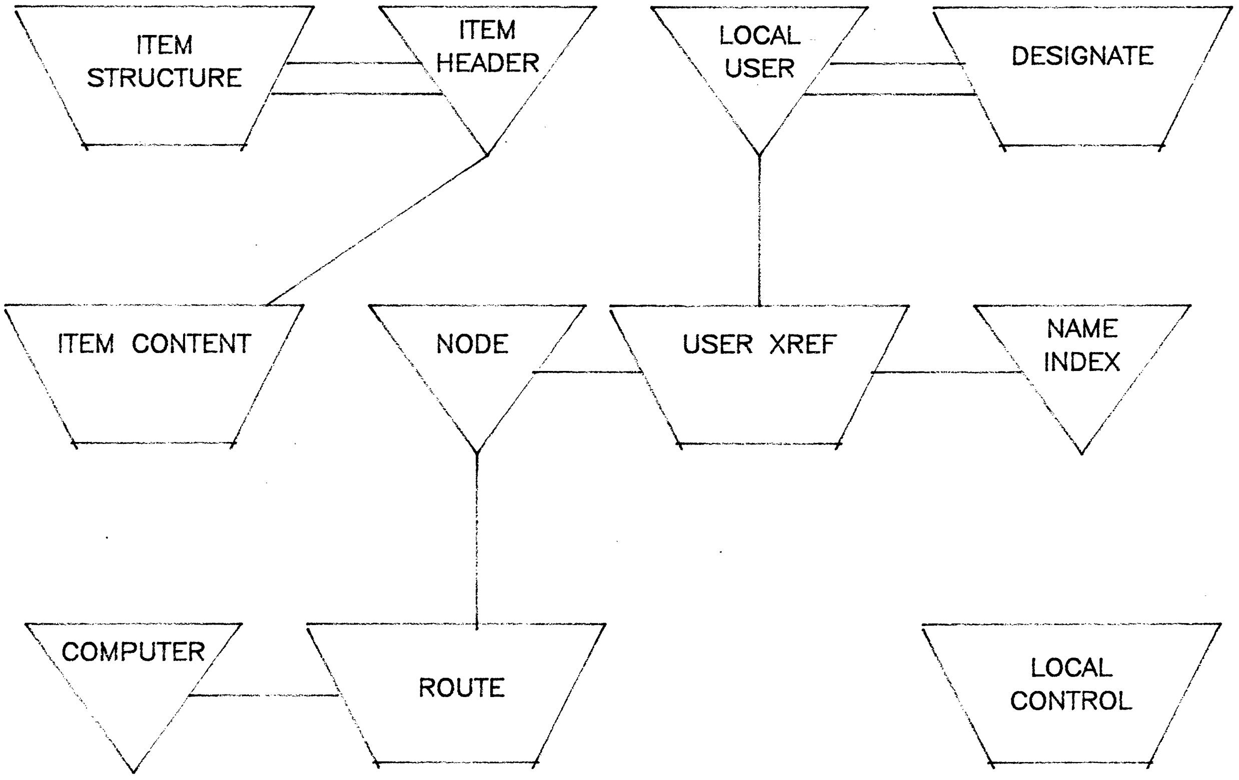


Figure 3

IF170982

HPMAIL GLOBAL Database

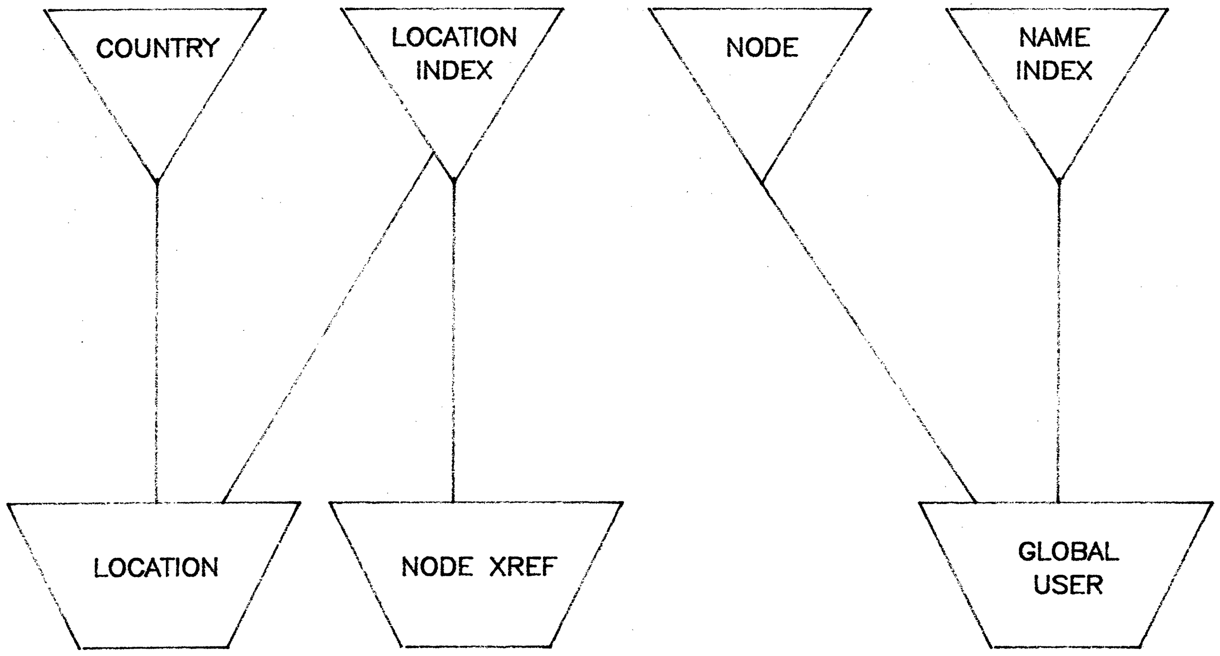


Figure 4
IF170982

HPMAIL Object Hierarchy

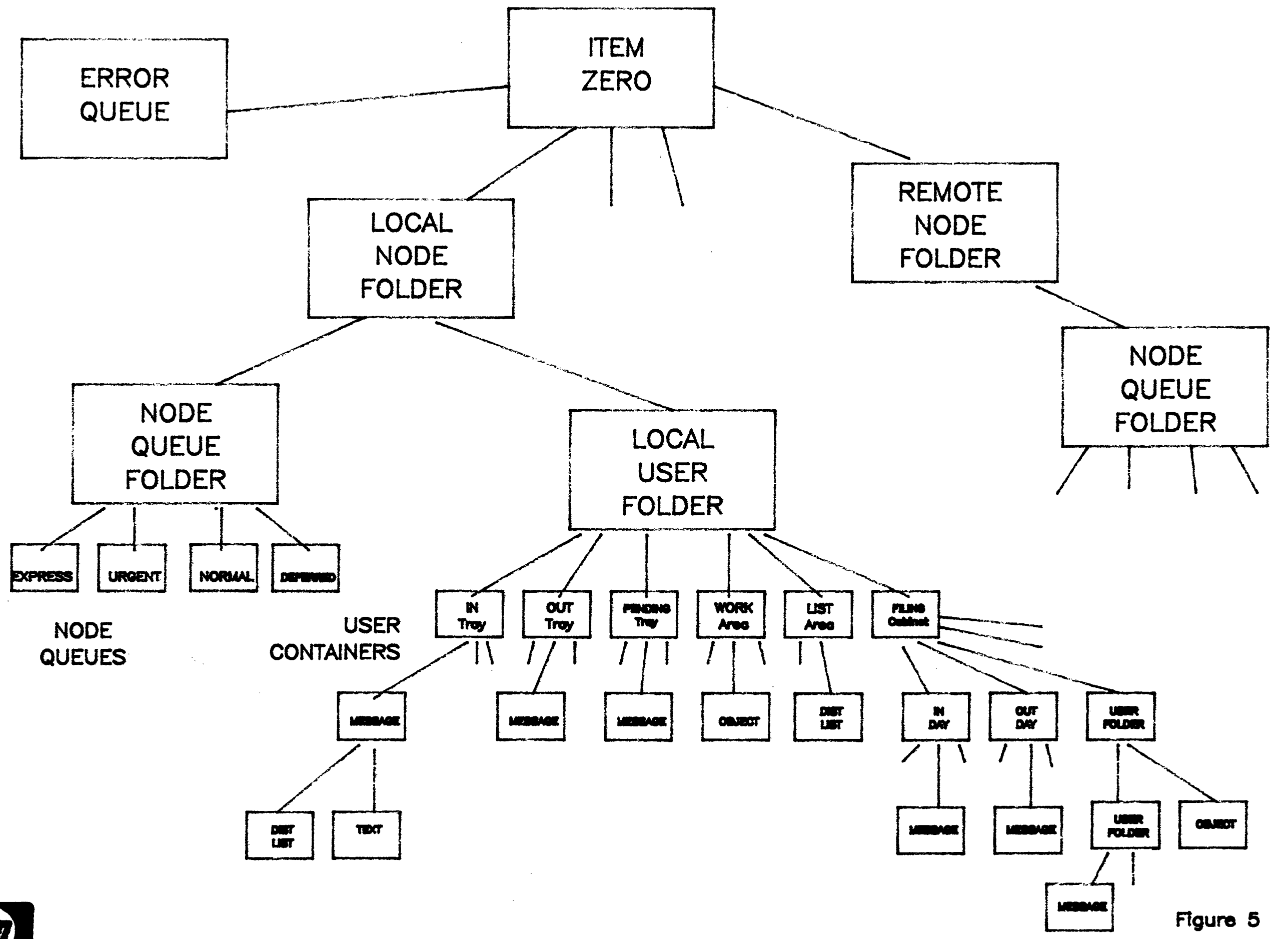


Figure 5

IF170982

DATA BASE DESCRIPTIONS

Figures 3 and 4 show the schemas for the two data bases. There follow two examples of the way in which HPMAIL uses the data bases in the execution of common functions, item storage and name searching.

HPMAIL ITEM STORAGE

The most important data sets to the understanding of HPMAIL's item storage are the ITEM-HEADER, ITEM-STRUCTURE and ITEM-CONTENT data sets on the Local Data Base. Items in HPMAIL are considered to be either "basic", meaning that they can be represented by a single MPE file (for example a Distribution List or a HPWORD Document), or "composite", which have no MPE file form as such but are composed of a number of basic items. An In Tray is an example of such a composite item, consisting of a header linking it to the messages in that particular In Tray.

Every Mail Item, basic or composite, has a Type (e.g. User Folder, Message, Text), a creator and a subject. This information is kept in its Item Header. The Item Structure data set maintains the links between items that go to make up composites. For example, a Message consists of a Message Header, linked by two Item Structure Records to the headers of its Distribution List and Content. The content of a basic item, the MPE file content, is held in the Item Content data set, linked by IMAGE pointers to its Item Header.

In HPMAIL, every item is linked to at least one other, except for the "root" item, which we term Item Zero. If an item becomes detached from all its parents, which happens for example when a message is deleted from the In Trays of all the users who received it, it becomes an "orphan" item and will be physically deleted from the data base by the Maintenance Program.

Figure 5 shows the HPMAIL "Object Hierarchy" in more detail. Although superficially quite complex it is, in fact, quite efficient and lends itself very well to processing by IMAGE, via the Mail Procedures.

SEARCHING FOR NAMES

Every time a user enters the name into the user interface program, whether he is entering his own name during the HPMAIL sign-on, or entering names on a Distribution List HPMAIL searches its Directory for that name, or a similar match. Names in HPMAIL are held in what we call "canonical form", which is:

<SURNAME>,<FIRST NAME> <MIDDLE NAME>/<LOCATION>/<SUB-LOCATION>

where:

surname first name middle name	add up to no more than 36 characters
location	is up to six characters long
sub-location	is two characters

Canonical names are all in upper case, converted by means of an internal translation table to take care of national character sets.

For example, the canonical form of user Ian John Fuller, location HP1600, sub-location 01 is:

FULLER,IAN JOHN/HP1600/01

HPMAIL does not insist that the user types in the full name, just sufficient to identify it uniquely on the data base. Thus if there are no other users with the surname FULLER defined on the data base, the FULLER will be sufficient. If any doubt exists, the User Interface gives a choice of those available and asks the user to choose one. Of course, user names must be unique to a mail node (location + sub-location).

When HPMAIL searches the data base for a name, it does so first by making a "name probe" from the surname supplied. The name probe consists of the first letter of the surname, followed by the next three non-repeating consonents. So, for example, the surname "FULLER" yields the name probe "FLR". This probe is taken as the key to a search of the NAME-INDEX Master Data Set which will give a chain of names that have this probe. The User Interface scans down this chain comparing each with the canonical form of the input name. There may be several degrees of match: all components agree, first names differ, locations differ etc. If a perfect, non-ambiguous match occurs, then the user is not troubled further. Otherwise he is given a choice of those available. The use of the name probe means that a slight mis-spelling of a user's name can sometimes, but not always, yield a choice that reveals the correct spelling of the name the user was looking for. For example, if the user types the surname "COALMAN" in mistake for "COLEMAN", then the program will be able to give a choice of the correct name since both names result in the name probe of "CLMN".

In the case of the user sign-on process, the Local data base is searched for the name, since this holds the name information for the users supported on that computer. For the construction of Distribution Lists, the Global Data base is searched since this contains the directory of all the users known to that computer, but not necessarially supported on it. The user has no need to know where another is supported in order to communicate with him; this is sorted out automatically. The name searches are done in a similar way on both data bases by Mail Procedures.

4. The User Interface.

OVERVIEW

The HPMAIL User Interface provides the user with an "environment" in which to use the product which is designed to mirror the concepts familiar to office users. In broad outline, the user is provided with:

An In Tray - into which his incoming messages are delivered by the Mailroom.

An Out Tray - in which messages the user sends are assembled and from where they are collected by the Mailroom for transmission and delivery.

A Pending Tray - which hold messages awaiting acknowledgements. The user can find the acknowledgement status of any message in the Pending Tray by reading its Distribution List.

A Work Area - which allows the user to create items and assemble them into "Packages" before incorporating them into messages.

A Distribution List Area - which holds lists of users with whom the user may wish to communicate. These lists can be used on messages with a minimum of trouble.

A Filing Cabinet - in which the user can "file" messages for future reference in folders that he has created. The system provides the users with two folders as standard, an "incoming Day Folder" and an "Outgoing Day Folder" which can serve as a complete record of all messages sent and received.

The Administration Area - which gives users extra commands to do such utility functions as defining or changing their password and setting up "Designates" who may work on their behalf.

COMMANDS

The command set of HPMAIL is designed to reflect the terminology used in offices. Examples of HPMAIL User Interface commands are:

READ	to display an item on the user's terminal.
PRINT	to print an item on the system line printer.
SEND	to start to send a message.
MAIL	to commit a message for transmission.
FILE	to "copy" an item to a folder in the Filing Cabinet.
REPLY	to send a reply to a message in the In Tray.

The function keys of Hewlett Packard terminals are used if possible to give the user an easy method of finding his way around the product. Function keys

HPMAIL User Interface structure

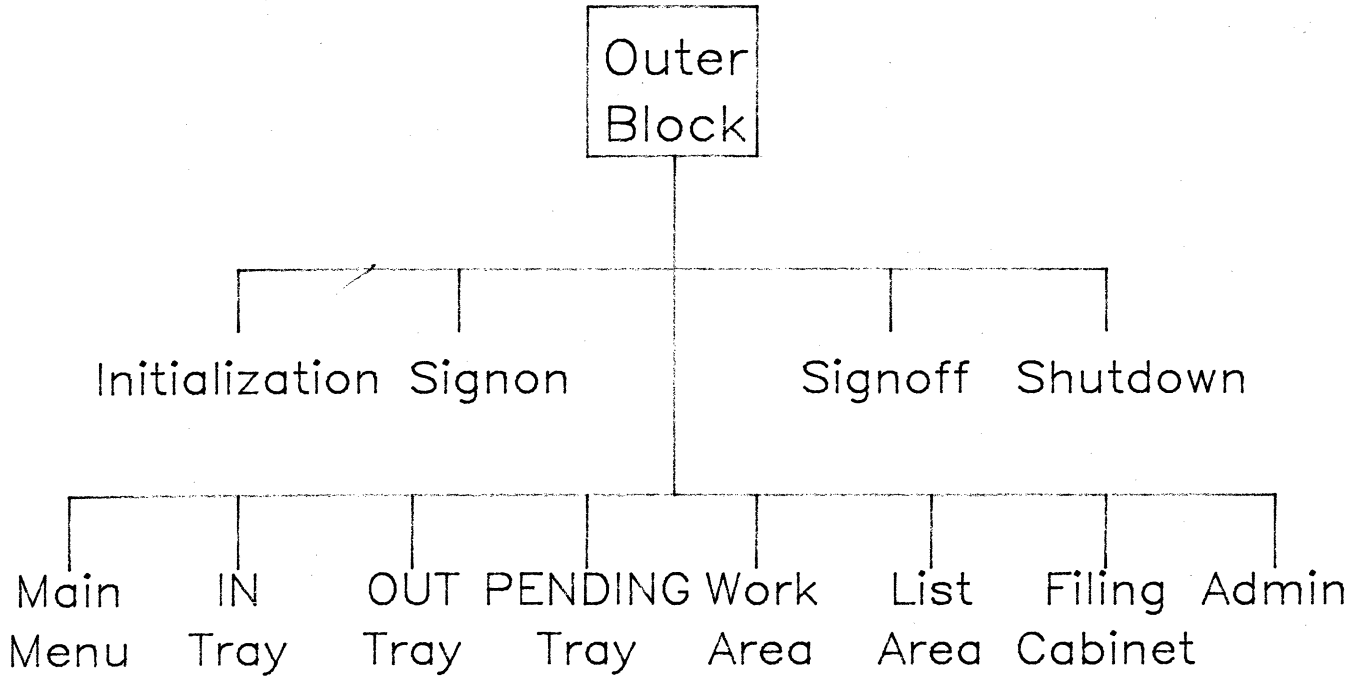


Figure 6

IF170982

<f1> to <f7> select entry to each of the "areas" listed above, with <f8> allowing the user to leave HPMAIL.

The "prompt" displayed by HPMAIL is another aid to the user in finding his way round his desk. This changes according to the area he is in. The In Tray prompt is "IN TRAY >", the Filing Cabinet prompt "CABINET >" etc. When the user has OPENed a folder the prompt changes to the first word of the folder's label, so a folder with the label "User Group Presentation" will, when OPENed, give the prompt "User >".

HELP FACILITY

The User Interface also offers an extensive Help facility which is organised on three levels. Typing the HELP command by itself gives the user general help about the area he is in. For example, HELP in the Filing Cabinet will give the new user some idea of the purpose of this area and suggest some likely commands. The command HELP followed by another command name will give the user specific help about that command as it is used in that area. For example, typing HELP MAIL while composing a message will explain the meaning of the command and how to use it. Finally, typing a question mark (?) in any area will give the user a brief list of the commands available as an aid to memory.

TRANSLATION TO OTHER LANGUAGES

The whole of HPMAIL was designed to be translated easily into other languages without the translator needing to have knowledge of the internal workings of the product or access to the source code. To this end, all messages produced by the product come from a central message catalogue (which is currently about 4,000 lines long) while the HELP displays come from another catalogue which is even longer.

All user input (except the content of messages) is parsed by common parsing routines and commands are recognised by reference to a Dictionary. The development team wrote a special utility to build this Dictionary, which goes under the name of "Webster". This is available to Hewlett Packard Country Software Centres world-wide for them to adapt the product to their local language. Synonyms and common abbreviations for commands can be built into the dictionary as required for each language.

Local character sets are handled by a Translation Table as part of the Dictionary so that users may, in their local versions, use commands containing their local characters. One limitation to the use of local character sets in HPMAIL, however, is that user names, locations and sub-locations have to be in the standard USAASCII code. This is because a HPMAIL network can span countries, not all of which may use the same conventions regarding extensions to the character set.

USER INTERFACE STRUCTURE

The HPMAIL User Interface program is a direct analogue of the data structures it supports in the Local Data Base. It is a direct result of the maxim: "Design the data structures correctly and the applications programs will almost design themselves". The program is a "tree structure", as shown in Figure 6 which follows the object structure quite closely.

The Outer Block of the User Interface controls all the other components which can broadly be broken down into:

Initialisation

User Sign-on

Environment Handlers

Signoff

Shutdown

The Initialisation module opens the data bases, the user's input and output files, the message catalogue and reads the dictionary from a disc file onto the user's stack. It initialises the Mail Common Area will all the relevent data and generally prepares the program for execution. It is also responsible for checking that the software has not been stolen and that incompatible versions of the software are not being used. The last task it performs is to print the HPMAIL "banner" on the user's terminal.

The User Sign-on module is the first that actually solicits input from the user (by asking for his name). The name enables the user's User Record to be located in the LOCAL-USER data set of the Local data base and from that point on the User Interface can latch on to a specific point in the object hierarchy shown in Figure 5.

After a successful sign-on, control is passed to the "Main Menu" processor, the first, and simplest, of the Environment Handlers. This simply displays the list of options available to the user, along with the number of newly delivered messages he has in his In Tray. The user is then prompted for his choice of area (environment) to enter next. This choice can be made by pressing a function key or by typing the corresponding number.

The other environment handlers, for the In Tray, Out Tray, Pending Tray, Work Area, List Area and Filing Cabinet are very similar in structure. They consist of five basic "states":

Initialisation.

Building a "List File" as a temporary MPE file of the current contents of the area (e.g. all the messages in the In Tray).

Producing an initial "LIST" of the area to give the user the information about its contents.

Prompting the user for a command.

Executing the command.

Each command (READ, FILE etc.) has its own "executor" which handles the operation of that command in all areas. Once the environment handler has recognised that a certain command has been entered by the user it passes control to the command executor for that command. It is the responsibility of the executor to determine whether the command is correct, possibly asking the user for additional information if necessary. If the command is correctly formed then the executor passes control to the "action procedure" for the command. This is often just a simple Mail Procedure. For example, the task of the FILE command executor is to determine what the user wishes to file, and where he wishes to file it. This is resolved internally into a pair of internal item numbers (in the ITEM-HEADER data set). These are presented to the action procedure for the FILE command, the Mail Procedure MATTACH, whose task it is to perform the required attachment, linking say, an In Tray message to the Incoming Day Folder in the user's Filing Cabinet.

COMMAND BASED USER INTERFACE

We are frequently asked why we adopted a command based user interface for HPMAIL when many of Hewlett Packard's Interactive Office products have standardised upon "point and push" interfaces, often using VPLUS/3000.

This question caused a considerable amount of heart-searching in the development and marketing teams during the design phase of the product and we finally came down in favour of an "unspohisticated" command based interface for two main reasons.

First, we realised that many of the potential users of our product would be managers, working away from their desks, perhaps in hotel rooms, on hard copy terminals connected to their office HP3000 by a slow speed telephone line. We thought that these users would derive a major benefit from using HPMAIL and so we designed our product to be no more demanding of a terminal than it be "Teletype Compatible". We would not go so far as to claim that the HPMAIL User Interface works well on every terminal that can be connected to HP3000, for we have not tried them all. However, we would be very interested to know of any upon which it does not work.

The other main reason for adopting the command based interface is that, for regular users, such an interface is far quicker to use than one where you have to cycle through several screens just to access the menu you require. Users often comment that, as long as you are not making errors, the User Interface is extremely "terse". It does not usually ask for confirmation of actions, nor indulge in long dialogue about exactly what to do. This squares with our belief that an electronic mail system is of little use to an occasional user and to derive major benefit from it, it must be used regularly. Otherwise users would cease to rely on it as a timely communication medium. Regular users, we believe, do not want verbose or long-winded user interfaces, they know what they want to do and want the software to get in the way as little as possible.

Ideally, of course, we would have written a user interface for every level of user who might encounter our product, tailoring it exactly to his or her specific needs. However, we compromised with one which we thought would please the maximum number of people most of the time.

One drawback with a command-based user interface is that it is often percieved as being harder to learn initially than "point-and-push" interfaces. Many of our users are busy managers who have no time to go on long product training courses or to wade through long manuals just in order to learn how to send a simple message. I was told once while I was testing the product with some Hewlett Packard managers in the United States that if they could not learn the essentials of the product in a maximum of thirty minutes it would not be accepted. To this end we developed an on-line training package for the product, known as the Interactive Training Facility (ITF). This gives users the opportunity to learn about those aspects of the product they need to use at their own speed in their own time. We find this to be an extremely valuable facility in getting the product used throughout the company.

It is interesting to note while discussing User Interfaces that the HPMAIL Configurator program uses an entirely VPLUS/3000 based user interface. We felt that the Configurator was an ideal application for such an interface and makes the potentially complex process of data base build, configuration, modification and expansion far easier to achieve for System Administrators.

5. Transport System Overview.

As stated previously, one of the major product objectives for HPMAIL was to provide a system where the end user did not need to become involved with the intracies of message routing and transmission, understanding DS/3000 or dealing with line failures, message congestion or any of the hundred other problems that can befall a multi-computer based messaging system.

Once a user MAILs a message it is removed from his Out Tray by the Mailroom and handed over to the Transport System for sorting, transmission and delivery. He can track the progress of the message by setting an acknowledgement on the message and monitoring it in his Pending Tray while it is sent to all the recipients, wherever they may be.

Figure 7 shows an overview of the data and control flow required to implement the store and forward system used in HPMAIL.

The Transport System is implemented by background jobs communicating via IPC files. The jobs are known as the Mailroom, Transport Manager, at least one Master Truck, each communicating with a Slave Truck on a remote computer.

MAILROOM

The Mailroom has two basic functions to perform. First, to collect messages from users' Out Trays. Second to deliver incoming messages to recipients on message Distribution Lists where the users are supported on the same computer. It also handles Delivery Acknowledgements, Auto-Forward and Auto-Answer. The Mailroom is structured internally in two halves. The Outgoing half deals with mail collection and sorting, the Incoming half deals with local mail delivery.

Outgoing mail originates from the User Interface via MAILing a message created by the SEND, FORWARD or REPLY commands. The IPC message sent to the Mailroom when a message is MAILed identifies the specific message in an Out Tray that has been commited for transmission. The Mailroom responds by accessing the message and scanning its Distribution List. It builds an internal list of the number of unique mail nodes (local or remote) that the message is bound for and then attaches the message to the node queues for the correct destination nodes. For locally bound messages the Outgoing Mailroom sends an IPC message to its Incoming "half" to deal with the delivery. For remote messages it sends a message to the Transport Manager alerting it of the new mail to shift. If an Outgoing message requires any acknowledgement the Mailroom will attach the message to the sending user's Pending Tray, ready to receive the acknowledgements as they come in. The last action it performs is to detach the message from the sending user's Out Tray.

The Incoming half of the Mailroom deals with mail originating from two sources. First, the local messages sorted by the Outgoing half. Second, the Incoming Mailroom receives IPC messages from a Slave Truck operating on its computer to handle messages received from a remote computer and to be delivered here.

In each case, the Incoming Mailroom examines the message Distribution List to determine to whom the message is to be delivered. It then locates the required users' In Trays on the Local Data base and attaches the message to them. It also has to handle the various types of acknowledgement that can be

HPMAIL Data & Control Flow

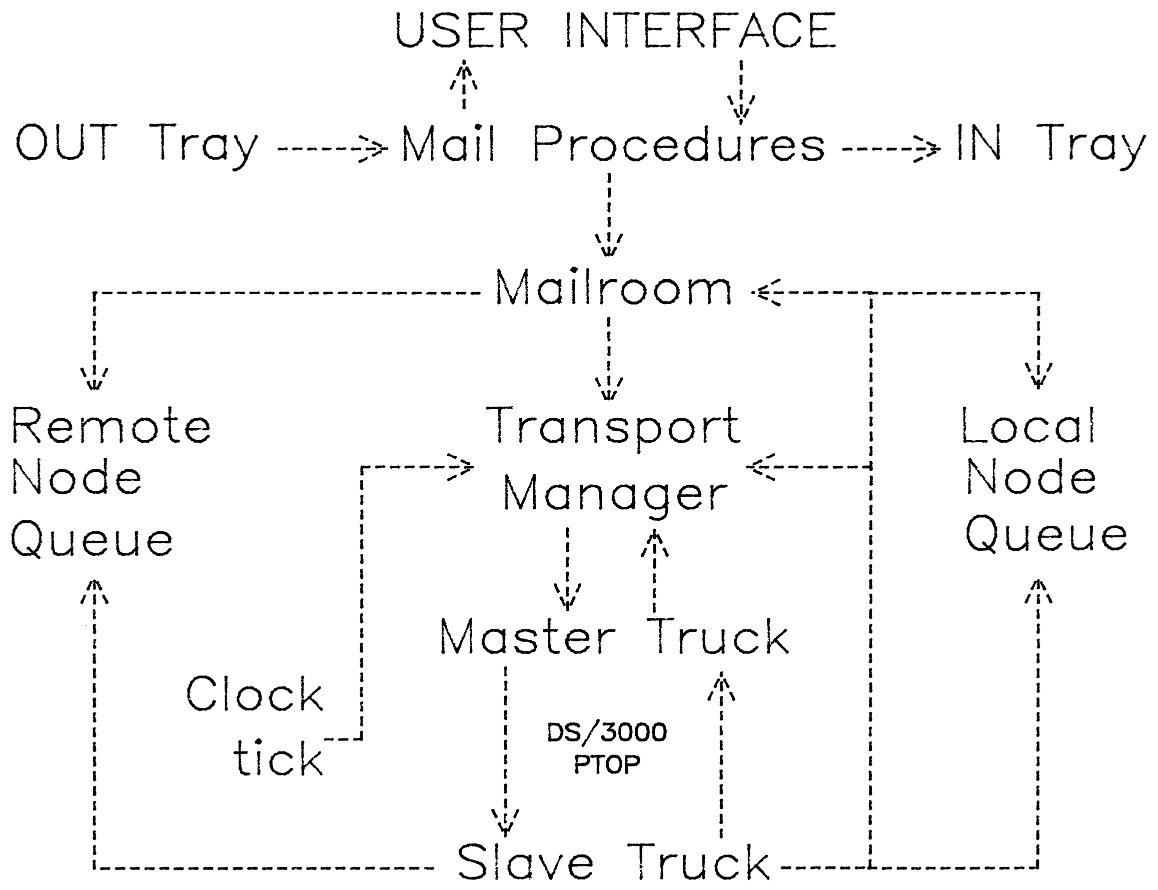


Figure 7

IF170982

received, amending the user's Pending Tray copy of the message with the fact that the acknowlendgemet has been received. Its other main tasks are to generate delivery acknowledgements for the message if necessary and to action any Auto-Forward or Auto-Reply that a receiver has set.

TRANSPORT MANAGER

The Transport Manager is another background job whose function is to control and schedule the operation of HPMAIL in sending mail to remote computers.

It maintains tables of mail-nodes for which mail is waiting and the possible routes that it can use to move the mail. This information is derived from the HPMAIL system configuration specified using the Configurator program. Every fifteen minutes the Transport Manager receives a "Clock Tick" from its son process, the Clock Tick generator. This causes it to re-examine its priorities and re-schedule the Master Trucks under its control if necessary.

The Transport Manager is constantly striving to ensure that the remote mail nodes with the highest priority mail to send are being serviced by the available Master Trucks. Once it selects an available Mail Node, together with an available route to another computer to send it, it will send an IPC message to a free Master Truck instructing it to establish a DS connection with another HPMAIL computer. Once the Master Truck has reported back that it has done this successfully the Transport Manager will instruct the truck to clear a certain remote node queue. Once this has been done the Master Truck will report back, ready to receive further instructions.

The Transport Manager is capable of controlling up to eight Master Trucks simulteneously, although it would be a very ambitious network that required this level of resource.

MASTER AND SLAVE TRUCKS

The Master Truck receives IPC instructions from the Transport Manager and handles all the interaction with DS/3000 and PTOP to physically move mail between computers.

The first instruction received by an idle Master Truck is one to establish connetion with a certain remote computer (identified by name) using a specified DS Line and, if necessary, telephone number or PDN Node name. It uses a MPE :DSLINE command to open a line to the remote computer, followed by a MPE :REMOTE HELLO comand to establish a remote session. It will then attempt to create and activate a Slave Truck on the remote computer, using the DS/3000 intrinsic POPEN. If the Slave Truck is successfully awoken the Master Truck will determine that it is, in fact, communicating with the correct remote computer by interrogating the Slave for its computer name. Once this protocol has been observed the Master Truck will report back to the Transport Manager "truck idle with DS line open" and await further instructions.

The next command received by the Master Truck in the message transmission operation is the "move mail" command. This instructs the Master Truck to transmit mail from a given remote node queue on the Local data base. It is responsible for "serialising" messages from the data base and packing them into transmission buffers for sending to the remote computer via the PWRITE PTOP intrinsic.

At the other end of the communication link the Slave Truck will, if possible, ACCEPT these transmissions and re-create the message on its computer. Once the message has been completely transmitted the slave truck will "deserialise" the message onto its data base. Depending on the ultimate destination of the newly received message the Slave Truck will either alert the Mailroom for locally bound messages, or its own Transport Manager for messages that are destined for onward transmission. Once the Master Truck has sent a complete message successfully it will detach that message from the remote node queue as responsibility for it has now passed to the next computer in the chain.

This process will be repeated for all the messages that the Master Truck has to send. Once the Master Truck has complied with its instructions to move mail it will send another report back to the Transport Manager "truck idle with DS line open". The Transport Manager will either respond with an instruction to begin transmitting messages bound for another mail node, so taking advantage of the open DS line, or it will instruct the Master Truck to terminate the link if there is no more mail to be sent on that route.

The main challenge in designing the Transport Mechanism was to provide for all the failures that can take place during transmissions over communications lines. Examples of the type of problems that have to be handled are:

> Inadvertent connection to the wrong computer, due perhaps to an incorrectly dialled telephone number.
>
> Finding that the remote data base cannot accomodate the message to be sent, either temporarily due to congestion, or semi-permanently due to the data bases not being built sufficiently large.
>
> Configuration errors resulting in transmissions being refused or an infinite loop being set up.
>
> Line failure during any stage of the process.

Since detachment from a node queue is the last operation that the Master Truck performs, and only after it has determined that the message was transmitted successfully, it is unlikely that messages will be lost due to line failure. It is, however, quite possible that HPMAIL will send messages twice in some circumstances.

WHY USE PTOP?

We chose the Program to Program intrinsics provided by DS/3000 in preference to the other methods of transmitting files across DS connections for four reasons.

First, PTOP ensures that there is a communicating process at the other end of the DS line to receive the mail. Without it, we would have had to implement yet another level of IPC communication to ensure that the remote computer was aware of the mail being delivered to it.

Second, the performance of PTOP is superior to, for example Remote Data Base Access (RDBA) since, in good circumstances, the Master Truck can be assembling transmission buffers for the next transmission at the same time as DS is taking care of the transmission of the previous buffer over the line. This makes optimum use of the communications line.

Third, PTOP provides very good control for the application program over the communication activity with the provision of a "tag field" in the PTOP intrinsics which can be used to send status, control and block-level acknowledgements at the same time as the buffer transmissions.

Lastly, the PTOP intrinsics are implemented on other Hewlett Packard computers, for example the HP1000 Technical Computer range, giving us the option of using these machines as mail nodes in a mixed DS network.

By removing the Transport Mechanism to a completely separate layer of the product we will be able to take advantage of enhancements to Hewlett Packard's data communications products as they become available with minimal disruption to the product's users.

It is notable that the HPMAIL Transport Mechanism uses only supported features of DS/3000 and works entirely in user-mode. This, we feel, contributes greatly to the robustness of the product. This remark applies to the product as a whole; in fact the only line of Privileged code in the product is one where the System Cold Load Identity is obtained. This is required to maintain system integrity as we have to know if there has been a system restart since the data bases were last accessed. Needless to say, we actively exploring ways of removing even this excursion from conventional programming practice.

6. Conclusion.

Unfortunately I have been unable to cover all the aspects of HPMAIL in this presentation. Only cursory treatment has been given to such vital areas such as Configuration, Maintenance and Operator Control, but I hope that this paper gives a overview of the sort of problems we faced when we looked at implementing a multi-computer messaging system on the HP3000. The solutions we arrived at were, as always, compromises between what we would like to have done, given infinite time and infinite money, and what was practically possible given finite resources.

We are always looking at ways to improve HPMAIL and make it a more effective tool in the Interactive Office. I would be very pleased to hear any suggestions for enhancements in any area.

BUSINESS GRAPHICS
An Integral Part Of The Electronic Office

Shirish S. Hardikar
Product Manager
Commercial Systems Pinewood
Hewlett-Packard Co.
Wokingham, United Kingdom.

Good morning. Today I am scheduled to talk to you about business graphics; more specifically, "Business Graphics - an integral part of the electronic office" So let me begin by taking a closer look at this title. "Business graphics" is a term which we have all seen used frequently to describe charts, graphs, etc. which represent tables of numbers graphically and aid the decision-making process by summarising facts and focusing on key points. But graphics are not new nor is the realisation of their effectiveness in communicating ideas; but until quite recently they were not easy to create. Computers, new firmware and software has changed this. New and growing number of "graphics solutions" are being offered to meet the organisational need. Graphics costs are driven down and their creation is made easier; complex information, ideas and interpretations can now be communicated with graphic illustrations as naturally and easily as we do with words and data.

This growing awareness of graphics as an essential tool in focusing on and understanding business facts and figures has led to a boom in the graphics industry. The problem now is not whether to use business graphics, but how to implement it. Quite naturally I have some definite views on this, but before I lay them in front of you let me outline how I arrived at these.

Business graphics products are orientated toward commercial applications where it is helpful to display important information visually. But professionals, managers and executives have long been using manually and photographically produced graphics effectively and it is not immediately obvious how to invest in graphics systems to noticably improve productivity and most important, save hard cash : dollars-and-cents, pound-and-pence, DM's-and-pfenings. What we do know is that these people need graphics in their daily work - for presentation, rapid understanding of facts, analysis and decision-making. But, to look for the best way to implement graphics for their use we need to understand three areas : the different types of graphics that are used; why graphics are used; and how (or when) graphics are used.

In general, business graphics is used by all executives, managers and professionals in an organisation whichever department they may be in. Two more common factors are what graphics they use and why they use them. Charts, schematics, diagrams and just plain text are the most common graphics used to better understand and present data. They are used for the benefit of external audiences - during formal presentations and when included in reports and other documents - and here the quality of the graphics is necessarily high. They are also used internally to illustrate data and improve communications; often in such use the graphics production quality need not be very high and expensive.

But the extent to which graphics are used depends very much on their cost-effective production. We need to take a look at who creates the graphics, how they are created and when & how much graphics is wanted. Let's go back to the last slide and add these factors. Let's take a look at where these graphics are created :

Long before the advent of computers, business graphics were produced in the Graphic Art Department. Even today most large companies have this department as part of the organisation. Designers and artists cut and paste artwork together to create graphic images, professional presentations are produced, but this is not all ... a lot of time is spent developing literature - brochures and documents for internal and external use. As a result, they just cannot cater for those rush management presentations that we are often required to make! Frequently I used to spend evenings with a calculator, graph paper, a ruler and multi-coloured pens drawing charts and writing-up my comments for presentation first thing the following morning!

But, if you have time, there have always been alternatives : a large number of custom graphics-art-houses and graphics service bureaux have traditionally complemented the in-house service. New computerised sytems costing upto $100K are being used to help provide faster, better service , but I beleive they still fall short of what a manager, executive or professional wants.

The custom-graphics-art-houses offer full services that can take the rough design to top quality finished products. In addition, they can provide design consulting for sophisticated, high quality graphics. This is particularly useful for formal presentations for external audiences when visual impact and creative design and communication are essential. But I think you will agree that these are 'one-offs' and their high quality is not required for the daily use of business graphics .... the costly revisions, 2-3 week turnaround, and total cost cannot justify it.

The first step for most organisations towards automating the graphics- generation process is usually through a service bureau. Timesharing is a very popular way to implement graphics in an organisation. In addition to the usual production service many bureaux also offer access to Industry databases ... market information, community indexes, price fluctuations, and much more .. all easily accessable. A very valuable service indeed! But this hasn't resolved the question of how a manager can use graphics daily and easily whenever wanted - on demand. This type of timeshare graphics is characterised by hardware leasing - one or two graphics workstations hooked-up through dial-up facilities. But associated with this are connection costs, sporadic turnaround, dependancy on phone lines and host availability.

More commonly, this method is used for trial use of software before purchase. As volume increases and quick turnaround is desired it becomes expensive and cumbersome.

The use of both these alternatives is growing but I beleive strongly that neither can meet the real demands of business graphics. I see these demands as: ease of use, the ability to use personal and community data, and the ability to merge text and graphics (everyone - managers, professionals, executives and secretaries - communicate with written documents. You have to be able to include raphics in the body of these report/manuals/etc. without manual, time-consuming effort ! .... but I'll talk about this aspect in more depth later). But the key demand of business graphics is for immediate access and results.

Now, the need to offer this wide accessability and functionality (in terms of working with, even integrating with, words and data) may not at first glance seem at all necessary. Why not just focus on graphics-made-easy?

Well, let's take a look. One good solution for this is the stand-alone graphics workstation. It is a single-user system much like personal computer/microcomputer systems. However, its operating systems and languages are optimised for graphics and offer powerful graphics capabilities. With costs ranging from as little as $8K to $100K at the top-end, these systems are much favoured. Some offer 35mm camera attachment and highly interactive colour CRT's with light pens. These higher cost systems are often a hybrid alternative to having graphics produced entirely offsite or having a large, expensive production facility onsite. Even nicer, several workstation vendors* maintain service centres to receive transmitted graphics designed at the customer's* workstation. These centres provide for processing into 35mm slides, colour prints or transparencies. The average cost of an overhead transparency produced by such a system is approximately $50.

This is all well and good for presentations, but is expensive if what I want is ten or twelve charts immediately so that I can analyse some data, prepare a report and/or reccomendations based on my analysis, and then throw away the charts ! Business graphics needs also to be disposable !

This thinking maybe runs against the grain today when graphics offerings are full of technology ..... 3-dimensional representations, rotating figures, purpose-built (high cost) terminals, etc. Their use is valuable for periodic analysis and review - like quarterly & annual business consolidations, assessment of corporate achievements, and similar; but I am not sure they are cost-effective for my very real need for simple ad-hoc graphics. On the other hand, why should they be ? The answer is, I maintain, simple. If you beleive, as I do, that business graphics is an invaluable tool in communicating and analysing facts and ideas, then it should

*Xerox, Computer Pictures, Comshare, Dicomed, GE.

be made available to me in my normal, daily working environment. In this, I don't need very high quality graphics, but rather graphics for rough-note projections of trends, quick analysis of changing data, and similar that only circulate, if at all, in-house; a complement to the way I use other tools - when I want, how I want!

More recently, the electronic data processing department has become an organisation's graphics centre. Computer generated graphics began here with large, complex, CPU intensive graphics programs. Even today, the majority of business graphics software is found on mainframe computers. As many of these packages are third-party supplied as are vendor supplied and offer high-quality and high flexibility graphics. In general, all these products are programmer-oriented; offering FORTRAN libraries, high-level languages and simple commands to generate business graphics. But, there are also reports that mainframe supported graphics softare are so CPU intensive that Fortune 5000 type users are running up costs in the area of $25,000 per month to support their use!**

EDP graphics specialists often act as an in house graphics service bureau, providing completed charts and visuals for analysis and presentation. However, with demand increasing these specialists are finding that their major focus is on maintaining existing applications. Ad-hoc graphics requests are getting low priority.

*1981 Survey User Ratings of Graphics Software Packages, AUI Data Graphics, (Alan Paller).

This is not a syndrome unique to graphics. The 'personal' needs of the professional or administrative support staff are increasingly outgrowing the centralised resources made available to them. As a result we have seen the growth of the personal and mini computer distributed throughout organisations to satisfy individual and departmental requirements. The personal computer offers excellent graphics capabilities and interfaces for the 'office user', but this is not its only function. It and business graphics are used and useful because the personal computer can also be used for general applications - graphics is a valuable complement to the other office functions .... this and ease of accessibility, I beleive, ensure the continued and increasing use of graphics by almost all personal computer users in an organisation. But one more step would be nice! Most personal computers cannot offer easy, efficient access (local or remote) to community information - documents, data and graphics are not easily shared.

The other growth area is that of the general purpose minicomputer. With the increasing power of minicomputers many graphics applications are being transferred from mainframes to minis. The biggest problem has been that the software has remained EDP-orientated while the minicomputer has moved into the offices of functional departments, such as the Finance, Personnel, Marketing and other departments

Until recently, these functional departments in an organisation were relying on the graphics artists or the EDP departments to satisfy their graphics needs. Long lead times, high expense and poor response to ad-hoc requests are now making them look for better ways to satisfy these requests whilst maintaining the much-needed service provided by the EDP centre for complex graphics with direct database access and programmatic features to access and manipulate centralised information. With new, easy-to-use business graphics software, vendors such as Hewlett-Packard are beginning to meet this demand by moving graphics directly into the hands of the professionals, as well as offering the foundation of sohisticated business graphics within the EDP centre.

The HP 3000 Business Graphics Package provides a comprehensive working solution for the graphics needs of the manager, secretary and EDP professional. It packages together three products, HPEASYCHART, HPDRAW and HP DSG/3000, which are designed to work and fit together and extend graphics capabilities from the EDP centre to the office end-users.

HPEASYCHART is Hewlett-Packard's totally new "No-experience-necessary" chartmaker for business professionals and secretaries alike. For a long time now we have all wanted a 'easy-to-use/friendly/simple' graphics system, but these much-used cliches are open to interpretation and few systems achieve this. HPEASYCHART was designed to overcome this : a picture-orientated main menu helps users select a chart by its visual appearance on the screen ..... I don't need to know what a vertical cluster bar chart is; simply that "I want to make a chart that looks like that one"! By the philosophy of 'point & push' a chart can be selected. Once selection is made, you can view a pre-defined chart of that type with sample data ..... no need to know where to type the numbers and words..... type in your own data over the example, and then plot the chart. All of this with just three screens (two, if you plot to the screen)!

This tool for the professionals is ideal for the 'scratch-pad' graphics capabilities I spoke about earlier. However, it needs to be backed-up by more complex graphics capabilities to meet growing needs. DSG/3000 introduced in 1980, is HP's "full capability" chart design and production system to satisfy this demand. Its advanced capabilities provide for periodic, automatic generation of charts, data transformation (mathematical operations may be specified and DSG/3000 will perform the calculations and chart the results), and access to data files on the HP 3000. In addition, DSG/3000 provides greater flexibility and quality by offering a selection of fonts, colours, text sizes, legends and the ability to directly enhance charts produced by HPEASYCHART. This last feature is key to the integrated offering and I will outline it briefly a little later.

The third link in the HP 3000 Business Graphics Package is HPDRAW. It is the solution for that other graphics requirement - presentations. HPDRAW comes into its own here by providing the means to produce high-quality textual visual aids in various fonts, illustrate them with charts prepared with HPEASYCHART or DSG/3000 or from a library of commonly used symbols and basic geometric shapes ..... boxes, arrows, machines, plants, flowchart symbols, star shapes, and many more! Ideal for quick, professional overhead transparencies for professional presentations.

In addition, and most important, all three graphics products work together to provide an effective solution for all HP 3000 users. HPEASYCHART charts produced by managers and professionals on an ad-hoc basis can be directly accessed by EDP profesionals for enhancement and inclusion in periodic reports; charts produced by both can be included directly in HPDRAW drawings; and all charts and drawings can be output to a plotter or included in a printed document using TDP/3000 or HPWORD and the HP2680 Laser Printing System without the need for 'cut-and-paste'.

The HP 3000 Business Graphics Package thus distributes graphics to all users. By doing this it should reduce the workload from the EDP department for time-consuming, ad-hoc requests and at the same time offer "graphics on demand" to the manager, professionals and secretaries.

It is here, in the offices (of functional departments), that the full benefits of business graphics will be best realised. The offices are the decision and communications centre of an organisation and graphics is one, but important, complement to the different tools used there. Business professionals, managers and executives interact with information and people daily. Their success relies on effective control and management of people and data. Business graphics is a significant aid in this, but only if it complements the other office communication tools at the disposal of these people - viz. word processing, report generation, telephone, meetings, 'what if' analysis, etc.

Just as Visicalc is used by the professional to "play with numbers", scratch-pad graphics can help him interpret quickly the implications of each change he makes. Also, while new ideas are being developed, these scratch-pad graphics can be used to promote his or her analysis better, professionally, at meetings. Whatsmore, just consider the benefits for a 'project-group' that regularly needs to communicate, but is physically distributed throughout an organisation : for example, the marketeer, the accountant, the design engineers and production people - all involved together in, say, introducing a new product. It is usually hard enough to keep in touch, let alone be able to exchange information and individual analysis without appointments and meetings needing to be scheduled. But help is at hand; the growing use of electronic mail systems is an invaluable aid to solving this problem. With the ability to send graphics and attached analysis for review by any or all members of the group, information flow and understanding is improved. Graphics, with such tools, gives the manager control - control of business facts, of rapid communication of the facts, and most importantly of his time.

But there's more - graphics are also extensively used within many documents. Often the secretary has to use 'cut-and-paste' method to include graphics in reports, and such like. I think it would be nice if this manual, time-consuming exercise is cut-out.

Because of all these uses of graphics, I beleive business graphics is not just a tool for periodic use, but a fundamental part of the working office environment. By distributing graphics throughout an organisation and placing it at the fingertips of the real end-user, Hewlett-Packard's HP 3000 Business Graphics Package, is one example of this. It is a part of the Hewlett-Packard 'Interactive Office' - an integrated set of tools for decision support, document management, personal support and organisational communication. The value there is in integrating the organisation's information needs - not just with data processing, but word processing and graphics, and communications capabilities to tie it all together on a single family of compatible computer systems.

I really beleive that the need for integrated information management is growing. It is of greater need to be able, from a single workstation, for a professional to access information without the need for a programmer, interpret data quickly and present decisions effectively with graphics, prepare reports with the aid of simple word or text processing, (even merge those charts with text !), and send that report/chart/information to others in the organisation! This may sound futuristic, but it is not! With software products to match specific needs, the right capabilities should be distributed to the right people. For successful, cost-effective and acceptable implementation, business graphics cannot stand by itself - it needs to be an integral part of the 'electronic office'.

Business graphics aids the decision-making process, word/text processing documents it, and electronic mail distributes it! All together can offer a working solution with integrated information management.

INCREASED PERFORMANCE AND PRODUCTIVITY FOR 3270-EMULATION

Rolf Frydenberg, M.Sc., Data Comm. Consultant,
Fjerndata A/S, N-1322 Hovik, Norway

1. ABSTRACT

DSN/IMF (Distributed Systems Network/Interactive Mainframe Facility) is Hewlett Packard's system for emulation of IBM3270 control units and terminals. Many users have discovered that using DSN/IMF terminal-emulation (called PASSTHRU) implies long response-times, and that using the DSN/IMF intrinsics means extensive reprogramming on the HP3000. The Interactive Mainframe Access System (IMAS/3000) was developed in order to reduce response time for 3270-terminal emulation. IMAS uses less CPU-time than PASSTHRU, and has a number of enhancements aimed at increasing user productivity. Among the major enhancements are: Multiple sets of user-defined function keys (and labels), user-definable preprogrammed dialogues that take the drudgery out of data entry, print-output to terminal-attached printer, and batch execution of preprogrammed dialogues (in order to use an existing data entry system on the HP3000 together with new online applications on the mainframe). This paper will present IMAS/3000, and discuss this product with reference to DSN/IMF PASSTHRU.

2. BACKGROUND

Fjerndata is one of the largest computer service bureaus in Norway. Our main computers are:

1 IBM3081
1 IBM370/168
1 UNIVAC 1100/62

Our datacommunications network connects almost 2000 interactive terminals and 250 batch terminals to our central site. Most of the batch terminals, and an increasing number of interactive terminals, are really mini- or micro-computers. The total transmission capacity of our data network is 650 to 700 kilobits per second, through a number of timedivision, statistical and intelligent multiplexors.

Fjerndata is a cooperative venture. Among our owners - and users - are some of the largest industrial companies in Norway. The Fjerndata-group (as it is called) consists of the following members:

The Aker Group	- shipbuilding, offshore
Dyno Industries	- explosives, chemicals
Kongsberg Vaapenfabrikk	- manufacturing, metals, electronic products
Kvaerner Industries	- manufacturing, shipbuilding, offshore
Norcem	- cement, chemicals

Aardal og Sunndal Verk	- aluminium
Det Norske Veritas	- shipping and offshore surveyors
Data-Ship	- data processing for shipping companies

Norwegian Petroleum Consultants (NPC)
The Norwegian Scientific Research Council (NTNF)

NPC and NTNF are users only.

The total number of employees in the Fjerndata-companies is 45 thousand, and the expected revenue for 1982 is approximately $ 3500 million.

Fjerndata has during the last two to three years established itself in distributed processing, by supporting three of the most popular local computers among our users: The IBM8100, the ND100 from Norsk Data, and Hewlett Packard's HP3000 series. The Fjerndata users have 10 HP3000's installed as of this moment.

3. HEWLETT PACKARD'S DSN/IMF PRODUCT

Let us take a look at Hewlett Packard's system for IBM3270-emulation, to see what it is, and what it isn't.

IML/3000, the Interactive Mainframe Link, was introduced in early 1980. IML allows programs running on the HP3000 to access an IBM370-type mainframe interactively. This product has later been enhanced and renamed. It is now called DSN/IMF (Distributed Systems Network/Interactive Mainframe Facility). DSN/IMF consists of four basic components. These components are:

The driver - which handles the line protocol, either BSC or SDLC. The driver is in a "download" file for the INP board. It is called CSDIMF.

The monitor - which does the actual emulation of the IBM3274 control unit. This is a special program called TTSMON (ThirtyTwo Seventy MONitor).

The intrinsics - which make it possible for HP3000 programs to access the driver program. These intrinsics access a datasegment which contains the screen image received from the mainframe.

PASSTHRU (in IML it was called IDF) - which performs actual terminal emulation for HP CRTs and printers. PASSTHRU makes HP terminals capable of emulating the functions of IBM3277/3278 screen terminals, or 3284/3286 printing terminals.

These four components may be considered as "layers", such as those standardized by the International Standards Organization (ISO) in its Reference Model for Open Systems Interconnection (OSI). In exhibit A a comparison between the 7 OSI layers and the 4 DSN/IMF layers is shown graphically.

The comparison in exhibit A is not exact, since DSN/IMF internally is quite different from the OSI reference model. The exhibit is intended merely to illustrate the basic "layer-like" structure of most communication systems.

The three lower levels of DSN/IMF (driver, monitor and intrinsics) are difficult to replace by user-developed programs. Nor is there any specific reason for replacing them: They are well-designed components that perform their tasks very well. PASSTHRU, on the other hand, is a user-level application program, and may therefore be replaced by user-developed applications. It is also PASSTHRU that potentially limits user capabilities compared to standard IBM3270-terminals.

Some of the characteristics of PASSTHRU, that could be changed in order to increase performance and productivity, are listed below:

Block Mode terminal-to-HP3000 communications is used, which increases the amount of data transmitted between the HP3000 and the terminal.

The 8 function keys have fixed IBM-functions, which means that all PF keys require two interactions between the user and PASSTHRU.

The address (or terminal number) of the emulated 3270-unit, and the name of the DSN/IMF configuration file must be typed in by the user.

The user must leave PASSTHRU to perform local HP functions, and run other HP3000 programs.

PASSTHRU supports HP block-mode terminals only.

4. THE DEVELOPMENT OF IMAS/3000

Most of the HP3000-users who ordered the DSN/IMF product were primarily looking for the PASSTHRU functions. Quite a lot of them wanted to replace real IBM3270-terminals and cluster controllers with HP-terminals connected to HP3000-systems. DSN/IMF had been "presold" to some extent, by suggesting that it could replace IBM3270s for data entry and other applications. What we, as users were not aware of, was the fact that data entry through DSN/IMF did NOT mean using PASSTHRU, but instead it meant that we would have to write our own data entry programs on the HP3000, and call DSN/IMF intrinsics to access the mainframe.

Rather than writing our own data entry programs for every application that we wanted to reach through distributed HP3000 systems, we decided to write our own IBM3270-emulator, i. e. a replacement to DSN/IMF PASSTHRU.

There were four major participants in the development of this alternative to PASSTHRU. They were:

Aktuelldata Norway (through an ex-HP data comm. expert)
Fjerndata (the coordinator)
Dyno Industries (the end-user representative)

IMAS/3000 - Interactive Mainframe Access System - is a two-part system for replacing DSN/IMF PASSTHRU. IMAS uses the DSN/IMF intrinsics, monitor and driver, and therefore can not replace DSN/IMF.

Design considerations.

When we designed IMAS, two considerations were in focus:

-> The HP3000 is NOT an IBM327X cluster control unit. It is much more complex, and runs significantly more complex software.

-> Data transfer between IBM3270-terminals and control units takes place at very high speed (hundreds of kilobits per second), while it is low- to medium-speed between the HP3000 and its terminals (up to 9.6 kilobits per second).

For these two reasons, an HP3000-based 3270-emulation package must have additional features, that IBM3270-control units do not have.

In order to be able to add these advanced features, we decided to build a two-part system. The first part comprises the "standard" features (i. e. the actual screen terminal emulation), while the second part contains the "advanced" functions that can only be found in IMAS.

Advanced features.

We investigated three possible methods for implementing advanced features in the IMAS/3000 system:

£1. Local screen processing, by storing screen formats locally, possibly in the terminal itself, thereby reducing response time.

£2. Writing data-entry programs on the HP3000 that stored the data in a file while another process transmitted the data to the mainframe (possibly in batch mode).

£3. "Hooks" into IMAS, that could allow flexible programmatic access to the mainframe under user control.

The users told us that method £1 was infeasible, because of the large number of different screens that they used.

Method £2 would force us to write quite a few specific programs, and modifying these whenever the host programs they communicated with were modified. We would also have to write new programs for each new user.

So we were left with method £3. We found that these "hooks" could most easily be implemented if the programmatic access could be performed at a "higher" level than ordinary programming. From this root grew the concept that we now call "automatic dialogues".

5. IMAS/3000 TERMINAL EMULATION

When we designed the screen terminal emualtion of IMAS/3000, we followed the following guidelines:

1. Reduce response-time compared to PASSTHRU.
   Through:
   - Character-mode terminal communications
   - Write updated fields to the screen only

2. Make function key usage more efficient and user-oriented.
   Through:
   - User-assigned functions for the function-keys
   - User-defined function-key labels
   - Multiple sets of function keys

3. Add local functions.
   - MPE interface (including RUN command)
   - Screen copy to system or terminal-attached printer
   - Trace facility

4. Include character translation for native character-sets.

5. Flexibility and adaptability.
   Through:
   - Parameters specified in configuration file.
   - All files backreferenced to :FILE-equations.

Terminal handling.

The terminal handling has been significantly enhanced compared to PASSTHRU, by avoiding HP block-mode data transmission. Character-mode has two significant advantages: First, data input from the keyboard is sent to the HP3000 immediately, so that there is no delay after the user presses <ENTER>. Secondly, data output to the terminal need not erase the whole screen, but can be transmitted as "updated fields only". For these two reasons, data transmission delays are significantly shorter for IMAS users than those experienced when using PASSTHRU.

Input/output operations under MPE require quite a lot of CPU-time. By keeping the amount of data transmitted between the terminal and the HP3000 to a minimum, total CPU-time per transaction is reduced compared to PASSTHRU. This was not a design objective, but is still an important feature in IMAS.

In the bar-chart in Exhibit B, response-times and CPU-times for IMAS and PASSTHRU are compared. The comparison covers 6 different transactions of three basic types, each transaction was run 10 times, and average CPU- and response-times calculated from this sample.

IBM3270 functions and keys.

IBM3278 screen terminals have 24 Program Function keys (PF1 through PF24), three Program Attention keys (PA1, PA2, PA3), and an host of other special-function keys (Erase End Of Field, Erase End Of Input, CLEAR, RESET, SYS REQUEST, etc.). HP262X terminals have only 8 user-definable function keys (f1 through f8), and a limited number of fixed-function keys (clear line, clear-display, roll-down, roll-up, etc.). These differences between IBM- and HP-terminals imply that a direct mapping between HP- and IBM-function keys is infeasible.

In IMAS we have allocated a two-digit function number (from -3 to 99) to each IBM or local function that can be performed. A list of these functions may be found in Appendix A. These function numbers are assigned in groups of eight (a "key-set"), to the eight function keys on the HP262X terminal. There are also special function numbers for switching from one key-set to another. For each defined function, a label may also be defined. This label is displayed by IMAS in the special label-field of HP262X terminals (lines 25 and 26 on the screen).

Local functions.

One of the most important local functions of IMAS is the MPE MODE. This function allows the user to enter standard MPE commands, including the RUN command, without leaving the IBM session. Local programs may therefore be run while the IBM session is still active. The user can return to IMAS by using the RESUME command.

There are also local functions for a variety of other purposes. Some examples are changing key-sets, turning the trace facility on and off, printing the current screen image, and starting automatic dialogues.

Configuration file.

It is of high importance to be able to adapt IMAS to different users, and environments, without having to change the actual program. This is achieved through the use of an IMAS configuration file, where all relevant parameters are defined. The configuration file may be specific for the individual user, or for a group of users. The configuration file is backreferenced to a :FILE-equation, making it extremely easy to implement account- and system-wide defaults through the MPE User Defined Command facility.

Parameters that are defined in the configuration file include: Which IMF configuration file to use, which of the emulated IBM3270-terminals to open, what to trace when the trace facility is invoked, when to use autodialogues, and what functions and labels are in which key-set.

The chosen method for user-oriented programmatic access to the mainframe is through a symbolic "language" for describing user-application dialogues. IMAS interprets and executes the statements written in this language, thereby emulating a user entering data through his keyboard. We call this feature of the IMAS/3000 system "automatic dialogues".

Autodialogues reside in standard TDP or EDITOR files. One autodialogue file can contain up to 9 separate autodialogues. Each autodialogue describes a sequence of operations on one or more screen images received from the mainframe.

Autodialogue functions.

In the current version of IMAS (called ON) the following functions may be performed by autodialogues:

Data may be INPUT to a specific field in the screen image. The source for this input data may be the keyboard, an MPE file or hard-coded in the autodialogue itself.

Data may be OUTPUT from any field in the screen image, either to the terminal, or to an MPE file.

There are a number of control statements, such as IF-type statements, GOTOs, and SUBROUTINEs. There are also statements for defining which files to access on the HP3000, turning the IMAS/3000 TRACE-facility on and off, displaying messages on the user's screen, terminating autodialogues or the whole IMAS/3000 session, etc.

Developing autodialogues

Autodialogues are written and edited using standard editors such as EDIT/3000 and TDP/3000. EDIT/3000 is directly callable from IMAS/3000 for editing the current autodialogue file. File format for autodialogues is standard numbered or unnumbered ascii files, where each autodialogue statement is terminated by a semicolon. Only one autodialogue statement is allowed on each line.

Autodialogues are called by the IMAS/3000 user in the same manner as other IMAS/3000 functions, i. e. identified by a function number. This function could be implemented directly through a soft-key, in one of the key-sets defined in the configuration file.

Autodialogues are in general user-specific, but system- or account-wide defaults may be implemented through the MPE User Defined Command facility, since the autodialogue file is accessed through a :FILE-equation for the file IMASAUTO.

An autodialogue "compiler" is under development. This compiler will make autodialogue development simpler, and will allow the design of more advanced autodialogues than can be achieved at present. The compiler will be able to handle variables (STRINGs and INTEGERs), LABELs and PROCEDUREs. The result of the "compilation" will be an autodialogue file which IMAS will interpret just like any other such file.

Some autodialogue examples.

To illustrate the possibilities that autodialogues give the user, we will describe a set of autodialogues that were developed for users of an electronic message switching system (EMSS), that is available on our mainframes. This system enables communication between the more than 2000 interactive Fjerndata users. The complete listing of these autodialogues can be found in Appendix C.

Dialogue no. 1: Log on to EMSS.

This dialogue will log the user on to EMSS. The dialogue consists of the following steps:

1. Enter EMSS.
2. Query user for ID and password, and enter these.
3. Return control to the user.

Dialogue no. 2: Copy a message to an HP3000 file.

This dialogue copies the current message to an HP3000 fixed record-length ascii file. The user is prompted for the filename. If the file doesn't exist, it will be created for the user. This dialogue performs the following steps:

1. Open the HP3000 file.
2. Copy each page (there are 4 pages, with no END OF MESSAGE delimiter).
3. Return control to the user.

Dialogue no. 3: Copy message from an HP3000 file.

A message is read from an HP3000 file, and written into the EMSS. The user may specify which file the message is in through a :FILE command. If this file doesn't exist, the user is prompted for a filename. This dialogue performs the following steps:

1. Open HP3000 file.
2. Read EMSS destination-id and message title.
3. Read from file, write to EMSS until END-OF-FILE or four pages have been filled.
4. Transmit message, return to user mode.

Dialogue no. 4: Log off from EMSS.

The user is logged off from the electronic mail system, and may access other mainframe applications or leave IMAS/3000 entirely.

When using this message switching system, an IMAS-user will ordinarily only need two IBM3270 function keys, PF7 and PF8, for retrieving the previous/next page of a message. He will also need the four autodialogues mentioned above, and possibly the MPE MODE for entering MPE commands or running HP3000 programs. The In Exhibit C the difinition of this key-set (from the IMAS configuration file), and the resulting labels on the screen terminal, are shown.

7. CONCLUSIONS

This has been a very quick walk-through of the IMAS/3000 system, where we have outlined the basic similarities and differences of DSN/IMF PASSTHRU and IMAS. We have pointed out some of the performance and productivity benefits that IMAS users may experience over PASSTHRU users. Let us try to sum up some of these benefits:

IMAS has significantly better performance than PASSTHRU, making it much more suitable for direct replacement of IBM3270-type terminals.

IMAS is more flexible and adaptable than PASSTHRU, through its more advanced function-key usage, and configuration file parameters.

IMAS can contribute to significant productivity enhancement for data processing users, through the IMAS automatic dialogue facility.

Programmatic access to IBM370-type host systems is made very easy by using IMAS automatic dialogues.

For those users who want these advantages, but are sceptical about bying crucial data communications software from a third party, it is important that there is a large organization behind the product. This organization - Fjerndata - is committed to keeping IMAS compatible with future hardware and software releases from Hewlett Packard.

APPENDIX A: IMAS FUNCTION NUMBERS

Below is the current list of the numbers associated with specific local and IBM-type functions in IMAS.

< -3 - -1 >	PA3 - PA1
< 0 >	ENTER
< 1 - 24 >	PF1 - PF24
< 30 - 33 >	CHANGE TO KEY-SET NO. 0 - 3
< 50 >	PROMPT USER FOR FUNCTION
< 66 >	PRINT SCREEN ON SYSTEM PRINTER
< 67 >	PRINT SCREEN ON 2621P PRINTER
< 68 >	COPY SCREEN TO HP2624/HP2626 PRINTER
< 71 >	ERASE INPUT
< 72 >	ERASE FIELD

< 73 >	CLEAR
< 74 >	RESET
< 80 >	USE 'NEXT' AUTODIALOGUE
< 81 - 89 >	USE AUTODIALOGUE NO. 1 - 9
< 91 >	RESET TERMINAL / REPAINT
< 92 >	REPAINT
< 93 >	TEST MODE
< 94 >	START LOGGING
< 95 >	STOPP LOGGING
< 96 >	SITUATION LOGG
< 97 >	EDITING OF MASAUTO
< 98 >	MPE MODE
< 99 >	TERMINATION

APPENDIX B: THE IMAS AUTOMATIC DIALOGUE SYSTEM

Below is a list of the syntax of IMAS automatic dialogue commands.

```
CHECK     field  "string"    rline  rline ;
ERROR     line ;
EXIT ;
FUNCTION  func ;
GO        line ;
GOSUB     line ;
INPUT     field   "string" ;
MESSAGE   "string" ;
READ      field   Æ"prompt"Å   ÆcharsÅ ;
RECEIVE ;
REOF      line ;
RETURN ;
RFILE     "filename" ;
TRACE ;
TRACEOFF ;
TRACEON ;
TRANSMIT  func ;
USER ;
WAIT      secs ;
WFILE     "filename" ;
WRITE     field   ÆcharsÅ ;
```

func	=	IMAS function number
field	=	Number of field in screen image
line	=	Line number in autodialogue file
rline	=	Relative line number (from the current line)
chars	=	Number of characters to READ or WRITE
secs	=	Number of seconds to wait before proceeding

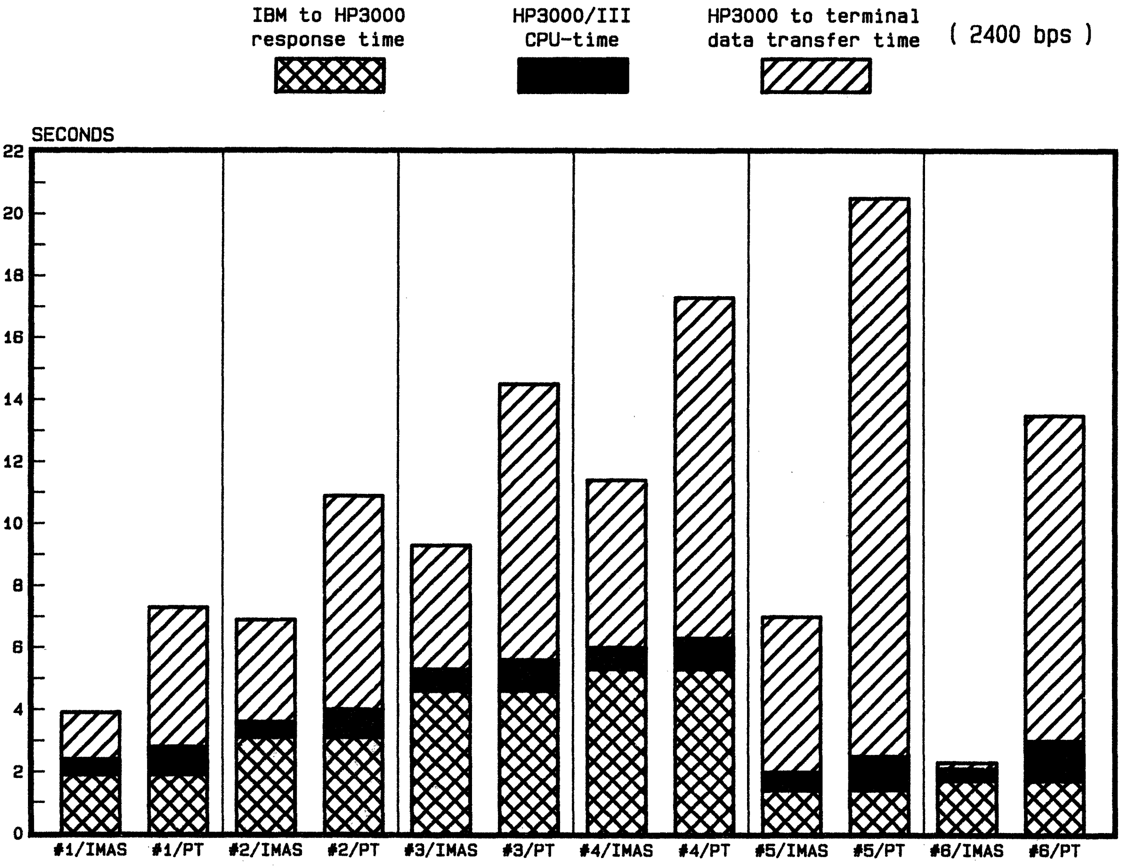
APPENDIX C: AUTODIALOGUE EXAMPLES

Below is the complete listing of the autodialogues used in the example in chapter 6 of this paper. Autodialogue execution starts at line 2 for function number 81, line 3 for function number 82 etc. Lines 6 through 10 are dummies in this example.

```
 1    MASAUTO0; ( test record )
 2    GO 025;      81 >   Logging on to the EMSS application
 3    GO 078;      82 >   Copy a message to an HP3000 file
 4    GO 132;      83 >   Copy a message from an HP3000 file to EMSS
 5    GO 173;      84 >   Log off from the EMSS application
 6    GO 016;      85 >   NO SUCH DIALOGUE
 7    GO 016;      86 >   NO SUCH DIALOGUE
 8    GO 016;      87 >   NO SUCH DIALOGUE
 9    GO 016;      88 >   NO SUCH DIALOGUE
10    GO 016;      89 >   NO SUCH DIALOGUE
11    *
12    *
13    *££££££££££££££££££££££££££££££££££££££££££££££££££££££££££££££
14    *
15    *
16    MESSAGE       "Invalid AUTODIALOGUE number - hit <RETURN> ";
17    *
18    wait   000;
19    USER;
20    *
21    *
22    *PAGE  ££££££££££££££££££££££££££££££££££££££££££££££££££££££
23    *A U T O D I A L O G U E   for logging on to   E M S S.
24    *
25    CHECK    001  "THIS  TERMINAL"   +03 +00;
26    message       " You are not in the 'LOG ON' screen.";
27    wait   010;
28    RETURN;
29    *
30    MESSAGE       "Am entering the EMSS system!";
31    *
32    INPUT    000  "EMSS";
33    TRANSMIT 000;
34    *
35    *
36    CHECK    001  " LOGON ACCEPTED"  +02 +00;
37    RETURN;
38    *
39    RECEIVE;      ---- Get next screen image.
40    CHECK    001  "SELECT FUNCTION"  +02 +00;
41    RETURN;
42    *
43    MESSAGE       "You are now in EMSS";
44    *
45    INPUT   002  "7  ";   Function 7 ( MEMO SWITCHING ) selected.
46    TRANSMIT 000;
47    *
48    CHECK    003  "SIGN ON"          +02 +00;
49    RETURN;
50    *
51    INPUT    008  " ";
52    RFILE         "$STDIN  ";  Get Identification from user
53    READ     023  "What is your user ID? ";
54    READ     026  "What is your password? ";
55    TRANSMIT 000;
56    *
57    RETURN;
58    *
59    *
```

```
 60     *PAGE  ££££££££££££££££££££££££££££££££££££££££££££
 61     *    S U B R O U T I N E    for copying
 62     *    one page of a message to an HP3000 file.
 63     WRITE 031;
 64     WRITE 032;
 65     WRITE 033;
 66     WRITE 034;
 67     WRITE 035;
 68     WRITE 036;
 69     WRITE 037;
 70     WRITE 038;
 71     WRITE 039;
 72     WRITE 040;
 73     *   Have now written one page ( i. e. 10 lines ).
 74     RETURN;
 75     *PAGE  ££££££££££££££££££££££££££££££££££££££££££££££
 76     *    A U T O D I A L O G U E   for copying a 4 "page"
 77     *    message to an HP3000 file.
 78     MESSAGE "I am now copying the message to the HP3000.";
 79     *
 80     CHECK 003 "PRIMARY MENU"  +00 +03;
 81     INPUT 043 "S";
 82     TRANSMIT 000;
 83     *
 84     CHECK 003 "MESSAGE RETR" +02 +00;
 85     USER;
 86     *
 87     WFILE;      ----  Prompt user for filename.
 88     *
 89     WRITE 007;
 90     WRITE 009;
 91     *
 92     GOSUB 063;
 93     *
 94     *Get next page  -  £2
 95     TRANSMIT 008;
 96     *
 97     GOSUB 063;
 98     *
 99     *Get next page  -  £3
100     TRANSMIT 008;
101     *
102     GOSUB 063;
103     *
104     *Get next page  -  £4
105     TRANSMIT 000;
106     *
107     GOSUB 063;
108     INPUT 013 "P";  ---- Return to primary menu.
109     TRANSMIT 000;
110     USER;
111     *
112     *PAGE  ££££££££££££££££££££££££££££££££££££££££££
113     *  S U B R O U T I N E   for copying ten lines
114     *  from an HP3000 file to an EMSS "page".
115     READ 036 072;
116     READ 037 072;
117     READ 038 072;
118     READ 039 072;
119     READ 040 072;
120     READ 041 072;
121     READ 042 072;
122     READ 043 072;
123     READ 044 072;
124     READ 045 072;
```

```
125     *  One page copied from HP3000 to EMSS.
126     RETURN;
127     *PAGE  ££££££££££££££££££££££££££££££££££££££££
128     *    A U T O D I A L O G U E   for copying a
129     *    message of up to 4 pages from an HP3000 file
130     *    ( referenced by FILE READMSG=<filename> ) to EMSS.
131     *
132     MESSAGE "Copying a message from HP to EMSS.";
133     *
134     RFILE "*READMSG ";
135     *
136     INPUT 007 "C";
137     TRANSMIT 000;
138     *
139     CHECK 003 "MESSAGE TRANS" +02 +00;
140     USER;
141     *
142     READ 008 008;  ---- USERID for destination
143     READ 012 014;  ---- TITLE of message.
144     *
145     REOF 163;  Skip to line 163 when EOF is detected.
146     *
147     GOSUB 115;
148     *Skip to next page.
149     TRANSMIT 008;
150     *
151     GOSUB 115;
152     *Skip to next page.
153     TRANSMIT 008;
154     *
155     GOSUB 115;
156     *Skip to next page.
157     TRANSMIT 008;
158     *
159     GOSUB 115;
160     *    Have read page no. 4, and the message may be transmitted.
161     *
162     *    The autodialogue will be directed here on EOF as well.
163     TRANSMIT 000;
164     INPUT 021 "T"; ---- Select option "TRANSMIT".
165     TRANSMIT 000;  ---- "Press" ENTER.
166     *
167     USER;          ---- Return control to USER.
168     *PAGE  ££££££££££££££££££££££££££££££££££££££££££££££££££
169     *     Log off from the EMSS application.
170     *
171     *
172     *
173     MESSAGE "Logging off from EMSS.";
174     *
175     CHECK 003 "PRIMARY MENU"  +03  +00;
176     MESSAGE "You MUST be in PRIMARY MENU.";
177     WAIT 000;
178     USER;
179     *
180     INPUT 007 "X";
181     TRANSMIT 000;
182     *
183     CHECK 001 "SELECT FUNCTION" +03  +00;
184     MESSAGE "Error occurred  -  this screen is unknown."
185     WAIT 000;
186     USER;
187     *
188     INPUT 002 "1";
189     TRANSMIT 000;


190     *
191     RETURN;
192     *
193     *   END OF AUTODIALOGUE FILE FOR   E M S S   USERS.
```



TRANSACTIONS (#1 and #2: data entry, #3 and #4: menus, #5 and #6: editing)

All figures are averages for ten identical transactions

Exhibit B: IMAS and PASSTHRU response times.

Definition in the IMAS configuration file:

```
* LINE #1 | LINE #2
81 LOG ON |  EMSS
82COPY MSG| TO  HP
83COPY MSG|FROM  HP
84LOG   OFF|  EMSS
07   PF 7 |PREVpage
08   PF 8 |NEXTpage
98   MPE  |  MODE
30   MAIN |KEY  SET
```

Resulting labels on the HP262X screen:

```
|LOG ON| |COPY MSG| |COPY MSG| |LOG  OFF|   | PF 7   | | PF 8   | | MPE  | | MAIN    |
| EMSS | | TO  HP | |FROM  HP| |  EMSS  |   |PREVpage| |NEXTpage| | MODE | |KEY  SET |
```

Exhibit C: Function keys for an IMAS/EMSS user.

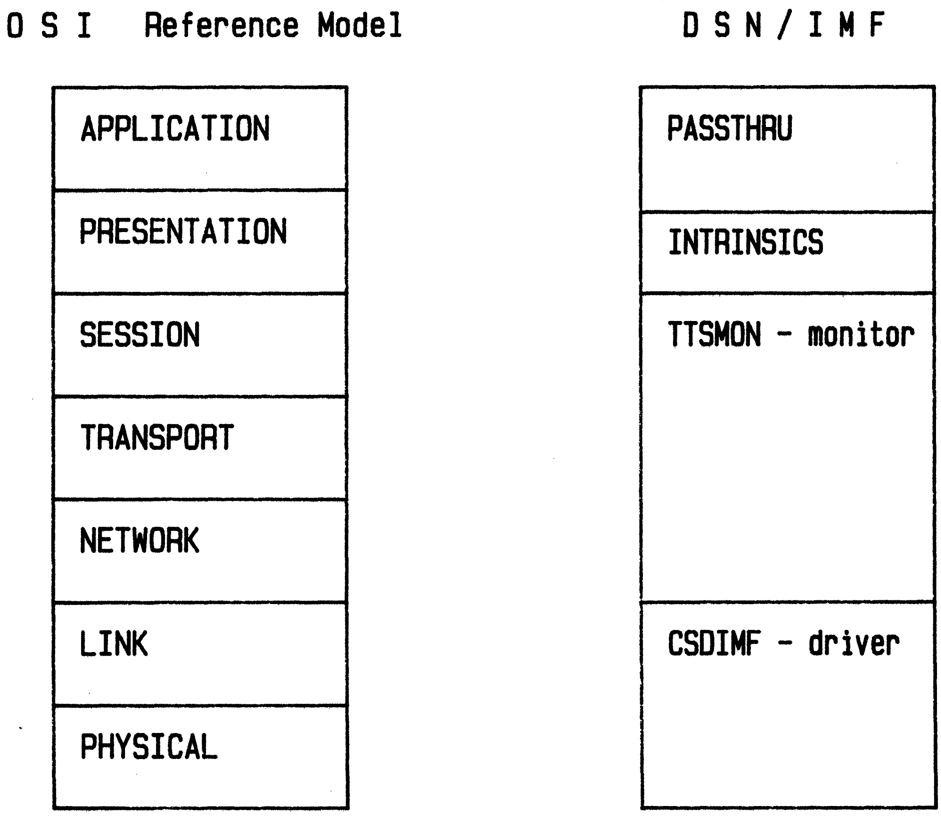


Exhibit A: The levels (layers) of OSI and DSN/IMF.

Character versus Block Mode Terminal Input on the HP/3000

by

Kurt Sager SWS SoftWare Systems AG Schoenauweg 8
CH-3007 Bern / Switzerland

C o n t e n t s

Summary

Most user-friendly online software packages use the screen forms technique to display questions on a CRT terminal. The user then only fills in the blanks or overwrites the default values on the form. Hewlett-Packard offers V/3000 to create and to maintain forms and to use them in the application programs. V/3000 reads ALL input fields in the form at maximum speed each time the user presses the ENTER key: this method is called 'block mode'. The HP-3000 computer architecture has been designed for character mode operation, however. To overcome some serious disadvantages of block mode input several screen form handlers operating in character mode has been developped during the last few years. As an example, the design principles, outstanding features and performance results of the Interactive Terminal Access procedure ITA/3000 are compared to V/3000.

The most important advantages of ITA/3000: - forms design and maintenance using an editor - national terminals equally well supported - account or system wide defined field enhancements - current input field can be highlighted - every input is checked immediately - no peak input rates for the HP-3000 - very easy programming, just one procedure - much better control over user input - typed-in characters are transmitted only

1. Programming techniques for dialogs between terminal users and application programs running on the HP/3000 system

On the HP/3000 system, as on most interactive computer systems, two basically different techniques for the man-machine dialog are used in computer programs:

(1) 'question / answer sequence'

(2) 'show a form & fill in the blanks'

The first method, 'question / answer sequence', is historically the older technique used in interactive programs. This method is quite easy to program and the necessary instructions are part of most programming languages, e.g. DISPLAY/ACCEPT in COBOL, WRITE/READ in FORTRAN, PRINT/INPUT in BASIC etc. The terminal operator sees one question at a time and gives the appropriate answer. Then the program displays the next question on the screen, which usually depends on the previous user input. The 'question/answer sequence' technique is appropriate for:

- short dialogs, e.g. a few parameter values to start a batch program,
- technical and scientific programs with few input variables and a lot of computations,
- one-shot applications, typically such as BASIC programs,
- computer science education,
- very complex dialog structures (see ELIZA [1]), where the actual answer determines the next question.

This way of man-machine interaction gives the terminal user the feeling to be very 'close' to the computer program. It allows the programmer to realise very flexible programs but at the additional risk of more difficult maintenance. It is indeed often not easy to see all possible dialog steps from a program listing.

Method (2), 'show a form & fill in the blanks', has become more and more used for standard data processing applications. It is

- appropriate for well defined and often used applications,
- the typical dialog technique on display terminals for commercial applications,
- similar to the traditional paper work,
- well-suited for unskilled people and occasional users,
- easy to see the relations between many fields to be filled in, e.g. dozens of descriptive and quantitative inputs for a new stock item to be entered.

Using method (2), an entire form is displayed on the terminal screen, typically showing a title, some explanations, a selection of options, and questions followed by distinctly enhanced fields for the user input. The cursor is positioned at the beginning of an input field, ususally the first one. The user has time to read the instructions on the screen, to understand all the questions to be answered. Generally it helps a lot to see all related questions on the same page. The user then answers the first question. If the first input field is completly filled, or if a defined 'termination key' is pressed, the cursor jumps automatically to the beginning of the next input field. If the user types a formally or logically invalid answer, an error message explaining the problem is displayed as soon as possible, and the user is asked to correct his input.

Technically speaking the second method, 'show a form & fill in the blanks', is realized as follows:

- In the first step the screen form is named and designed using a kind of editor, then stored in a forms library.

- The programmer puts a call to the appropriate screen handling procedure in his interactive program for each form to be displayed, erased, read from, ...

- At execution time the forms are displayed and the user input is returned to the calling program as individual fields or as a block of all concatenated field data of the current screen form.

Method (2) has the following main advantages over method (1):

- It looks more professional to show entire forms on the terminal screen and to let the user fill in the blanks.

- It is easier for the terminal user to know what kind of input is expected.

- Possibly the user has the option to check his input in all fields by re-reading his input, and to correct any bad values. Finally he 'sends' the whole block to the program for processing.

- Forms are displayed at once and then remain motionless on the screen until an other form is needed. This is much more agreeable to the operators eye's than the frequent jumping up of lines as with method (1).

How can you use method (2) on your HP/3000 computer? HP says: "That's easy, you just use V/3000, which has been delivered as part of the fundamental system software, and order some of our block mode terminals". BLOCK MODE? What is that? Are there terminals without block mode? Yes, the low cost HP2621. Are block mode terminals always working in block mode? No, if you logon to the system your terminal works in CHARACTER MODE. The same is true if you use EDITOR, TDP, DICTIONARY/3000, the MPE command interpreter, or most HP utilities. Every time you terminate an input by pressing the RETURN key the program you are using works in CHARACTER MODE.

However, for many HP/3000 users the 'show a form & fill in the blanks' technique is synonym to block mode data transmission. Why? Because the tool proposed by HP, V/3000, to realise this technique needs BLOCK MODE terminals.

The V/3000 requirement of block mode transmission does not imply though that method (2) can only be done in block mode. Several screen form handlers, operating in character mode, have been developped by independent software companies and proposed on the HP/3000 software market as a promising alternative to V/3000 ([2],[3],[4]).

Why develop a screen form handler with many man-months effort? Why decide many dp managers to BUY an other screen form handler instead of using HP's V/3000 which is FREE of charge?

The secret is: Thanks to the numerous advantages of CHARACTER mode operation over BLOCK mode the investement for an other screen form handler may be paid back in a few months!

But let us now see what the differences between CHARCATER mode operation and BLOCK mode operation are.

2. Comparison of Character versus Block Mode Terminal Input

Character mode terminal input can be described in a simplified way as follows:

(a) The user sees a question or item name ('PART-NR?') at the left of the blinking cursor. The program has issued a read command to the terminal port.

(b) Each character the user types on the terminal keyboard is immediately transmitted to the computer over the communication line, normally echoed back to the terminal screen at the current cursor position, processed within the computer by the terminal driver (a piece of system software, part of MPE) and stored in a memory resident table called terminal buffer.

(c) If the character count of the read command is satisfied, or if a special termination character (normally the code generated by pressing the RETURN key) is detected, no more characters are accepted. The characters already entered are then moved from the terminal buffer to the programs read buffer.

(d) The user program does the necessary formal and logical checks to verify the input data (e.g. is it numeric? not more than 2 decimals? is there a stock item having this PART-NO already in the data base?)

(e) If the user input is not correct an error message (e.g. "This stock item does not exist") is sent to the terminal, and the program restarts at point (a).

The maximum character input rate added accross all connected terminals is approximately

(number of terminals) times (maximum typing speed),

or for 20 terminals about 100 to 200 characters per second. The terminal driver processes all these incoming characters, and therefor needs to interrupt the CPU about every 5 to 10 milliseconds.

The more terminals send input to the terminal ports at human typing speed the more uniform the character flow to be processed by the terminal i/o driver is (due to statistical reasons).

But how does BLOCK mode input affect the HP/3000 system?

(a) The terminal user types his input, every character typed is shown on the screen and stored in the terminal memory. NO transmission to the computer occurs at this time. If a field is full or if a special termination character is pressed (TAB) the cursor jumps to the next field. Already filled in data can be corrected by tabbing aroung the form and using the local edit keys. Eventually many hundred characters are stored this way in the local terminal memory.

(b) While the user fills in field by field the dumb terminal processor cannot detect invalid input data, except the more expensive HP2624 where some limited local field editing is possible.

(c) When the user finally presses the ENTER key the contents of all input fields including trailing blanks and field separator characters are sent to the HP/3000 at maximum transmission speed (up to 9600 baud for the 30/33/40/44/64, 2400 baud for the older Series II/III).

Once the initial hand-shaking between the terminal controller and the terminal is done there is no possibility for a busy terminal driver to stop the incoming data flow before the end-of-block sign. Worse, the computer hardware/software must be able to digest the AGGREGATE data flow of all connected terminals in the worst case, when all users press the ENTER key within the same one to two second intervall! An interruptable block mode transfer could be imagined, but has never been realised on the HP/3000.

The maximum character input rate in BLOCK mode operation is

(number of terminals) times (maximum transmission speed)

or for 20 terminals about 20'000 characters per second! This is a more than 100 times higher input rate than by CHARACTER mode. BLOCK mode transmission is characterized by peak rates, some times very high, followed by quiet periods. Data overruns are quite frequent, which lead to retransmission of the faulty data block. Worse even: faults may occur (e.g. by transmission over a noisy telephone line) which are not detected and never seen by the user, because the HP/3000 doesn't echo his input when working in block mode. Many low-end models with 10 or more terminals cannot be operated properly at 9600 baud with block mode transmission. Solution: drop down the terminal speed to 4800 or 2400 baud!

The discussion above shows that while the 'show a form & fill in the blanks' technique is often prefered, its implementaion using block mode input (V/3000) has some serious drawbacks on the HP/3000 system.

In fact the HP/3000 system is not built to handle block mode input. The way it reads blocks of data is by simulation of the block mode technique with the existing hardware features and system software.

Every character received at any terminal port interrupts the CPU as in standard character mode but at a much higher frequency. Possibly this is not true any more for the new top model 64.

3. Design Goals for a Character Mode Screen Form Handler

The design goals for a character mode screen form handler as an alternative to V/3000 can be classified in three categories:

(a) Design goals to satisfy the terminal user:

- It is less-error prone to terminate every input sequence by the same 'termination character', the RETURN key, in all situations: in the :HELLO command, :RUNning a program, and to terminate the input data in a long input field of a form. If an input field of visually defined length is completely filled the input operation should automatically terminate.

- It is mandatory that all input data is immediately checked at least for formal correctness (e.g. numeric, number of decimals). If logical checks are necessary (e.g. does this PART-NO exist?) they should be executed by the application program just after reading the input field(s) concerned.

- Error messages should be displayed in a special window line in the local language (implementation parameter), possibly even with different languages on the same system, depending on the actual users mother tongue!

- The following 'standard' functions should be executable by defined control characters: screen refresh, hardcopy to the attached/built-in terminal printer, hardcopy to the system printer, special interrupt with command input, cursor back to previous field.

- Usage of the local terminal memory, today present on EVERY HP display terminal, for very quick change between terminal resident forms.

- Input fields of type 'no echo' for secret input, such as passwords.

- Numeric data should be right justified immediately, possibly missing decimals completed by zeros.

- Different types of form fields (display-only, optional input, required input, current input field) should be enhanced the same way in different forms and programs. This goal will best be achieved by defining system or account wide defaults or standards which automatically take effect.

(b) Design goals to satisfy the dp manager:

- Data transfers between terminal and computer should be in character mode only, to minimize and smooth the average input rate to the terminal controller and terminal driver.

- Response times for the users should be better then by using V/3000: less overhead for forms handling, no transmission of trailing blanks, only data really typed in!

- It should be possible to use ANY HP display terminal without any change in the application program, inlcuding the low cost 2621.

- The development time for interactive programs should be much shorter than with V/3000: if possible have only ONE procedure with few parameters to display the form, read the user input, and with automatic type conversion.

- Except for simple formal checks (numeric/non-numeric, number of decimals) all field processing and logical testing should be in the application program only, and not dispersed partly in the forms definition (field processing in V/3000 forms) and partly in the application program (maintenance problem!).

(c) Design goals to satisfy the programmer:

- Forms should be designed using a simple editor, like EDITOR, QEDIT, TDP, .. by typing in exactly what the final user will see. Special signs can be used for field definitions and field types.

- National terminals (Swedish/Danish, French, German) should equally well be supported.

- Time-consuming forms compilations should be avoided.

- It should be possible to check and demonstrate the forms by a utility program.

- A forms documentation utility would be very helpful.

- Buffer declarations for the major programming languages should be generated automatically.

- Programming should be easy: few procedures, simple parameter constructs.

- High flexibility to control the input process is important to have: control is to be returned to the application program after the data of one, of several, or of all fields is entered.

- The cursor can be positioned at any input field on the form.

- All HP/3000 data types should be supported, and additionally the most common date formats.

- The conversion from the external ASCII format to the specified internal data types should be automatic.

- Function keys must be supported, data entered before pressing a function key must be available.

- A duplicate function for typical data entry applications would be nice.

- The time-out feature (back to program after x seconds without user input), a special command window, and an error message window are important.

- All error messages should be stored in a message catalog, which could be extended by program specific messages.

- Program testing should be easy: additional test output to the same display terminal by standard DISPLAY statements.

- It should be possible to prepare test input on a disk file using an editor. The screen handler procedures should then be redirected to read from this file instead from the terminal keyboard.

- The HP/3000 standard DEBUG facility should be easy to use.

4. Comparison of two Screen Form Handlers: - V/3000 operating in Block Mode - ITA/3000 operating in Character Mode

Based on the design principles presented in the preceeding section the terminal screen handler ITA/3000 (Interactive Terminal Access) has been developed by SWS SoftWare Systems AG, and is available as a fully supported vendor product. ITA/3000 works in character mode and es satisfies most of the features discussed above. The following table shows a comparison to V/3000:

	ITA/3000	V/3000
* transmission mode terminal to computer	character	block
* immediate input checking	yes	no
* error messages in local language	yes	no
* instantanous forms change only	yes	2624
* number of forms resident in local memory	2	1, up to 5 in 2624
* no-echo fields for passwords	yes	no
* use of local edit keys	no	yes
* automatic & immediate right justification numeric data	yes	no of
* softkeys supported	yes	yes

*	time-out feature	yes	yes
*	special 'current field' enhancement	yes	no
*	system/account wide defaults & standards	yes	no
*	'visual' forms design by	QEDIT/EDITOR	FORMSPEC
*	maximum number of fields per form	unlimited	128
*	number of procedures to remember	1	24
*	forms compilation consuming	none	time
*	field processing specifications progr.	progr. only	forms &
*	automatic type conversion	yes	no
*	runs on low cost terminal HP2621	yes	no
*	test input optionally from disc file	yes	no
*	mixing forms and standard writes	yes	yes
*	program development time	short	longer
*	automatic buffer declaration (COBOL,SPL)	yes	no

References

[1] J. Weizenbaum, Computer Power and Human Reason. From Judgement to Calculation, 1976, W.H. Freeman & Company

[2] ITA/3000, by SWS SoftWare Systems AG, Schoenauweg 20, CH-3007 Bern, Switzerland

[3] SCREEN MANAGER, by Avantech Informatique, 2020 University, Suite 1628, Montreal, Quebec H3A 2A5

[4] ESP/3000, by Intertec Diversified Systems Inc., 2625 Park Boulevard, Palo Alto, Cal. 94306

[5] R.M. Green, Optimizing On-line Programs, Technical Report, 1981, Robelle Consulting Ltd. (see p. 49)

[6] A.R. Morris, V/3000 failed at the CCSSRD, Newsletter HP Bonneville Regional Users Group, 1982

MIMER, A COMPUTER TYPE INDEPENDENT DBMS

Sven G Johansson
Uppsala University Data Center, Box 2103, S-750 02 Uppsala, Sweden

A general DataBase Management System, DBMS, could be characterized by the admittance of a wide independence between application programs and data. Many computerized information systems of today are dynamic enough to handle modifications in, or expansion of the database structure without affecting the existing application programs. Unlike the majority of existing DBMSs, MIMER admits, in addition, a wide independence between DBMS and different computer brands, as well as different general programming languages.

For different reasons, computer hardware and software used to be sold together by the same supplier, which tied up an application system to a specific brand of computer. In present time, it is obvious that hardware type independence brings only advantages.

To make the implementation of MIMER on any type of computer system is remarkably easy. MIMER software is split in two parts: One is dependent of computer type and/or operative system, one is independent. The latter part, written in standard Fortran, comprises around 98 percent of the total software.

The portability of the system is attained by making the integrated program modules very simple and well-structured, with their critical computer dependent parts well defined and separated. In this way, the computer dependent parts are easily and rapidly redesigned to an optimal fit of a specific computer type. The coding of the computer dependent routines for a new installation is done, by experience, in a month or two.

A computer application is likely to change over time. When you finally have finished a complex system analysis and accomplished the corresponding programming work to build up an application, you definitely want it to run and function as long as possible without doing any major modifications on it - even if you changed to a new type of computer. Our environment is dynamic and constantly changing. Therefore we need a simple way to define and redefine the database structure since parts of our environment are mapped into the database.

MIMER is general in the sense of application independence. MIMER is applicable in small projects, for instance research studies, as well as modest size projects. It is also applicable in large systems like overall information systems including management and administation functions.

MIMER is used for many types of applications. For example:

- economy, budget and planning systems
- pay-roll system
- information systems for governmental agencies
- medical information systems
- medical laboratory systems
- education tool in universities

MIMER is now installed on the following computer types:

BURROUGHS	IBM
CONTROL DATA	NORD
DEC 10/20	NOVA
ECLIPSE	PDP-11 serie
HONEYWELL-BULL	PRIME
HP	UNIVAC
ICL	VAX

The structure of MIMER - the relational model

Data to be handled by MIMER are organized in tables (relations) according to the relational data model. Every table has a specific table name and consists of a certain number of columns. These columns have names and type attributes. A table must have a defined primary key which is defined by one, two or several columns. A primary key value is unique in a table.

Example:

PRODUCT

PRODNO	PNAME	MANNO
P1	PNAME1	M4
P2	PNAME2	M2
P3	PNAME3	M4
P4	PNAME4	M2

MANUFACT

MANNO	MNAME
M2	MNAME2
M3	MNAME3
M4	MNAME4

PRODCOMP

PRODNO	SUBSTNO	PWEIGHT
P1	S2	5MG
P1	S4	4MG
P2	S1	
P2	S4	

SUBSTANC

SUBSTNO	SNAME
S1	SNAME1
S2	SNAME2
S3	SNAME3
S4	SNAME4

Data in a table refer to certain fixed phenomena, that is actual facts, in an activity. In the table PRODUCT we have data about

- identities of products (PRODNO)
- names of products (PNAME)
- manufacturer of products (MANNO)

All data items in this table describe in different ways PRODUCT. "Products" is the object class described in the table PRODUCT. The object class comprises many objects, that is many different products, all with one thing in common - they go as "products". Each object of the class is described in the same form in the table - that is with the same set of characteristics, PRODNO, PNAME and MANNO. In a similar manner, every other table in the database is a description of a certain object class. In this example we also have the following object classes:

- product composition (in the table PRODCOMP)
- substances (in the table SUBSTANC)
- manufacturers (in the table MANUFACT)

Data describe different attributes of the objects. In the relational database, we see the characteristics as columns in the table comprising a certain object class.

The table PRODUCT contains data describing product identities. These may be represented by current numbers (PRODNO). By the current number, each object in the PRODUCT table is identified unambiguously. PRODNO constitutes the primary key in the table PRODUCT. Every table in MIMER must have a primary key which unambiguously identifies the objects of the table. The primary key is represented by one characteristic (column) or made up by several characteristics (as for instance PRODNO and SUBSTNO in the PRODCOMP-table).

Thus, in a MIMER database, we have data about:

- objects, which are members of different object classes, where each object class is represented by a table, and where you find data about a certain object on a certain row in the table

- characteristics of the object, where every characteristic of a certain object class is represented by a column in the table of this object class

- identities of the object - primary keys.

Between the different tables there are semantic connections which means that one characteristic of one table also is member of another table. In this way, you find the characteristic MANNO, identity of a manufacturer in the MANUFACT-table. Between PRODUCT and MANUFACT there is a clear connection via MANNO. As MIMER uses the relational model, the connection is legitimate, because the information about MANNO in both tables, is received from the same domain. We could easily call the column MANNO something else in the PRODUCT table and yet maintain the semantic connection. What is important is that we store the same data in the two columns, not the identical naming of the two columns.

MIMER MODULES

A database handler with interface to general programming languages to create data independent application programs.

A query language to define and build databases, manipulate and extract data.

A report generator to structure reports, including statistical calculations.

A forms management system which makes data entry operations a simple and reliable matter.

A program generator which converts defined reports and queries to a conventional programming language, thus creating an application program.

Utility programs to execute frequent operations against the database.

The data base handler

The nucleus of the system. It handles the physical structure of the database, the mappings needed between logical and physical structures and the maintenance of data and metadata. The physical data organization is of no concern to other subsystems. That is why redefinition of the database is possible without making any modifications of existing application programs. You get a good data independence

For instance, if the data values of PRODNO in the previous ly defined database are stored as binary integers in two bytes is of no concern to the user. He only states that the program wants the PRODNO-data as a character string of length 10 (for instance). This automatic format conversion is handled by the database handler.

One very important task of the database handler is to maintain the database. To update the database, the user does not have to say how to do it, just what to do.

The primary physical organization is based upon the B-tree technique with an additional facility to define secondary indices.

The physical database is continuously reorganized by updating operations in order to optimize retrieval operations and by direct re-use of free space on the secondary memory. No overflow technique is used.

When several users at the same time perform operations on the database, the database handler subsystem prevents confrontations to happen, thus offering a reliable database.

MIMER supports a multi-user environment. If, one day, a system crash is a fact, MIMER uses its recovery facilities to restore the database in a valid state.

The programming language interface

The programming language interface gives you the possibilities to create your own application programs written in a host language, for instance Fortran, Cobol, Pascal, Lisp, using the MIMER system as a database handler.

The application programs are highly data independent. Just the names of the columns and the names of the tables are used for communications.

The programmer operates with defined routines which may be embedded in the programming language.

Two types of routines exist:
- Row-oriented operations
- Set-oriented operations

Row-oriented operations is used to operate on a record-by-record basis, where one row at the time in a table will be investigated.

Set-oriented routines allow more powerful operations.

UNION
INTERSECTION
DIVIDE
DIFFERENCE
JOIN

are examples of set-oriented functions.

The command language MIMER/QL

Very often, the user has demands on fast answers to his questions without any programming. A fairly simple method is using MIMERs built-in command language - MIMER/QL - which comprises some twenty different commands. This sub-system can be used by non-programmers or parametric users and is primarily working from a terminal, with a display screen or type-writer terminal.

Knowing the MIMER/QL syntax you

- define new databanks
- define new tables
- load tables with data
- retrieve stored data
- change or delete stored data
- add new data in the tables
- redefine old tables
- pre-define a number of commands for repetitive use and make use of these defined **command procedures**

To define the table PRODUCT in databank PRODUCTDB:

```
DEFINE TABLE PRODUCT (PRODNO IS I2: PNAME IS C40,
    MANNO IS I2) IN PRODUCTDB;
```

To load or unload data between the table PRODUCT and operative system files:

```
COPY PRODUCT FROM 'SEQ.INPUT';
COPY PRODUCT TO 'SEQ.OUTPUT';
```

To define the range of table identifiers:

```
ALIAS PRODUCT (P,Q);
ALIAS MANUFACT (M);
```

Now the identifiers P and Q are both alias to the table PRODUCT and M to the table MANUFACT.

To put a query on the database: Find the product name and manufacturer name manufactured by companies in UPPSALA:

```
GET P.PNAME, M.MNAME WHERE M.ADDRESS EQ
    UPPSALA AND P.MANNO EQ M.MANNO;
```

To store the answer in a new table instead of getting it on the terminal:

```
GET NEWTAB (P.PNAME, M.MNAME) WHERE M.ADDRESS
    EQ UPPSALA AND P.MANNO EQ M.MANNO;
```

The contents of the new table, NEWTAB, is displayed by GET NEWTAB.*.

To insert a new row in the table MANUFACT:

```
INSERT MANUFACT (MANNO='101', MNAME='UPPSALA
    DATACENTER', ADDRESS='UPPSALA');
```

To replace values of some columns for every row where some condition is fulfilled:

```
UPDATE P (PNAME='NEW-PRODUCT-NAME') WHERE
    P.PRODNO EQ 75;
```

To delete rows in a table fulfilling a specified condition:

```
DELETE P WHERE M.MNAME EQ 'CRASHED COMPANY'
    AND P.MANNO EQ M.MANNO;
```

To remove a complete table:

```
REMOVE TABLE NEWTAB;
```

MIMER/QL procedures

To make use of MIMER/QL even simpler, it is possible to predefine sequences of commands combined with prompting on the terminal. When a number of MIMER/QL-commands are regularly run it is practical to bundle them up in one procedure. By using one MIMER/QL-command and stating the procedures name the whole predefined sequence is then executed. You are simply turned from a command mode to a simpler prompting mode, where the user only answers ready-made questions.

You also have the possibility to direct the execution through a menue command, which offers at the terminal, different possible operations to be initiated by the user. The procedures are always executed in a predefined status with incomplete MIMER/QL-commands, to which the user answers, when prompted. During the procedure run, the MIMER/QL-commands get completed and executed.

In addition to the standard MIMER/QL-commands, you have within a procedure, special procedure commands with functions like:

- substitution
- assignment
- conditional go-to
- loop facilities
- menue facilities

For example, a procedure, say FIND-PRO can be built for answering the query:

"Find those products which have a specific substance."

The user will execute one (or several) procedure(s) by typing the command:

```
EXEC PROCLIB(procedure);
```

If the above mentioned procedure FIND-PRO has been defined and stored in a procedure library called PLIB the example could be:

```
? EXEC PLIB(FIND-PRO);

THIS PROCEDURE GIVES YOU THE NAMES OF THOSE
PRODUCTS AND THE CORRESPONDING MANUFACTURER
CONSISTING OF A SPECIFIC SUBSTANCE

GIVE SUBSTANCE NAME: S4

_______________________________________
PNAME               MNAME
_______________________________________
 PNAME1             MNAME4
 PNAME2             MNAME2

2 ROWS FOUND

GIVE SUBSTANCE NAME:
```

This example shows that the procedure contains one initialization phase for describing the function of the procedure. Then the user is asked what substance name he/she is interested in, whereafter the query is evaluated and the result is displayed. The user may also have this procedure repeated to give a new substance name. Leaving the procedure is done by giving (exclamation marks).

MIMER/PG - a data language handler for pilot and production applications

Using the query language alone, the output layout will be made in a standardized form. However, when this is not satisfying, a report generator is connected to the query language to specify a wanted output layout.

The report generator of MIMER/PG is used to define a wanted layout and to connect a specified report with a query during execution time. In MIMER/PG you are all the time working in a LISP-system in an interpretative mode, thus making it easy for you, to change the definition and execute it again.

The performance of this system will not be acceptable in routine work but is quite sufficient during the development and test phases.

When finally the wanted specifications run as they should, you may use the program generator module of MIMER/PG. This module will generate application programs out of the generated reports, store the programs in the user library for later executions.

```
MANLIST M.MNAME, P.PNAME WHERE P.MANNO EQ
    M.MANNO
```

A previously defined report identified by MANLIST is connected to the query language to make manufacturer name (MNAME) and product name (PNAME) to be transferred to the report. What will happen with these names is defined inside the MANLIST report. The report output may look like:

```
COMPILATION OF MANUFACTURERS AND THEIR PRODUCTS

MANUFACTURER          PRODUCT

MAN 1                 P1
                      P6
                      P3
NO OF PRODUCTS: 3

MAN 2                 P10
                      P7
                      P2
                      P4
NO OF PRODUCTS: 4

TOT NO OF PRODUCTS: 7
```

During the definition of the report you may directly test it in an interpretative mode. A user often wants to modify the report definition. This is a simple operation as well as the re-test of the report. When the defined report is satisfactory you let the program generator compile the definitions into application programs in Fortran or Cobol. These programs will be stored in a user program library for routine work.

You may, whenever you want, implement new functions in the report generator. The only condition is a know-how of using the LISP programming language.

The MIMER/PG system will decrease the conventional programming effort for simple listings, tabulations and statistical operations.

The myth behind the name

Long ago, it is said, in the heathen times, there was a giant, who lived far up in the North. He was master of a remarkable well, which was brimful of wisdom. When the giant drank out of this well he could instantly give the correct answer to any question. Everybody came to have their problems taken care of, and the giant had a prompt solution for them all. His name was MIMER.

It is also said that the giant lived where Uppsala is now. So, when we found both the geographical and the functional connection between this heathen giant and our database management system, we could not resist to call it MIMER.

HP 3000 IUG København 1982

Presentation by T.W. Andreassen/J. Drevdal

"THE USE OF IFPS USING HP3000 IN THE NORCEM GROUP

Siviløkonom Tor Wallin Andreassen, Department of Finance

1. Presentation of the Norcem group

2. Historical background for the EDP development in Norcem

   - technical
   - Norøk, Plancode

3. Why IFPS? Choosing and evaluating a DSS tool.

4. What is done so far

   - Financial reporting (models, reports, menudriven cmdfiles, database, registering, and extracting data from the base).
   - adhoc analysis
   - an overhead demonstration of some menudriven systems.

5. Our future goals

   - Financial reporting
   - adhoc analysis
   - organizing for a DSS group
   - DSS philosofy
   - IFPS user training
   - Norcem's IFPS hotline

One of the main reasons why we looked for a new modelling language was a recognition of the fact that EDP-personnel in the future would be a scarce commodity and thus represent a bottleneck in the development and spreading of EDP in the group.

If we could transfer the programming or building of models to where the needs was felt, we would make ourselves more independent of this bottleneck.

When looking for a modelling language which would cover our needs we scanned the market and ended up with two competiting alternatives. After having established a few key checkpoints we had the two products installed on our own HP/3000, the designing of a new reporting system started. Based on the large amount of data that would be handled in the system we decided to design everything around a large database where all the reported data would be stored.

Around this database we made several Cobol programs, and we integrated all functions (extracting and registering data, IFPS-interface etc.).

In IFPS we started to build several separate models and reports based on menu-driven commandfiles which we at a later stage integrated into a complete tree-structured system.

Through one (1) command the user will be introduced to the system and may go in any direction from there.

With regard to adhoc analysis we have so far made models for investment analysis, three years budgetting models, liquidity management, working capital management, one year budgetting models.

What we are working on today is trying to distribute the registering of data to where the datas originate (business unit level) and through machine networks communication transfer these data to the group's central database. We marketing this concept for all our business units worldwide.

Through a special IFPS commandfile system called KRIB (again integrated with the same database concept) we allow the business units to test the data which it is about to transfer to the group and may discover inconsistencies in the data <u>before</u> they are transferred to the group.

After an agreed deadline where all subsidiaries have transferred their data to the group's central database, the Department of Finance may start consolidating these data. The time used for consolidation is through this concept reduced by at least two days.

At the business unit level we are introducing a new interactive accounting software package which we plan to integrate with IFPS.

At the group's headoffice we are working for establishing a new Decision Support Group, inhouse IFPS user training and the setting up an IFPS-hotline.

<u>Siviløkonom Jarle Drevdal,</u> EDP-Department

1. The choosing and evaluation of IFPS.

2. What is done regarding EDP-technical aspects.

   - Data base system and related programs for the Financial Consolidation system and related programs for the Financial Consolidation system (NORRAPP). Use of the machine network system DS/3000.

   - User defined Fortran subroutines. Integration with other systems (query language, full screen editor, ledger system etc.).

Early spring 1981 we started looking for a software tool which could solve two main objectives:

First to be general DSS tool, second to perform financial consolidation, the evaluation was summerized on seven points:

1. How good is the system as a DSS tool?
2. How well does the system perform general consolidation?
3. How easy is data input from different sources?
4. How are reports generated?
5. How user friendly is the system for non DP personnel?
6. Vendor aspects.
7. Price.

IFPS is now installed on 6 of our HP3000 machines.

The general consolidation demand for several hundreds of sequential files each period (13 periods a year). To avoid this, we built a system based on one (IMAGE) database. Each plant or department deliver their data for consolidation directly into this database via a (COBOL) program providing a full screen facility (V/3000 based). On the other side of the database, data can be extracted on a large set of criteria, and a number of IFPS datafiles are generated. These files are mass purged after use. By using the machine network system DS/3000, data delivered on different machines all end up in the same database on the same (central) machine immediately.

The HP3000 system allow the operative system (MPE) commands to be performed via calls (intrinsics) in programming languages like COBOL, FORTRAN, BASIC etc. This makes it possible to execute other systems like databased inquiery (QUERY/3000), full screen editor (TDP/3000) graphics system (DSG/3000), ledger systems etc. From inside IFPS via user written FORTRAN subroutines. Such subroutines are useful for a lot of other purposes too. One example is to pick up specific data items from ledger or other databases directly.

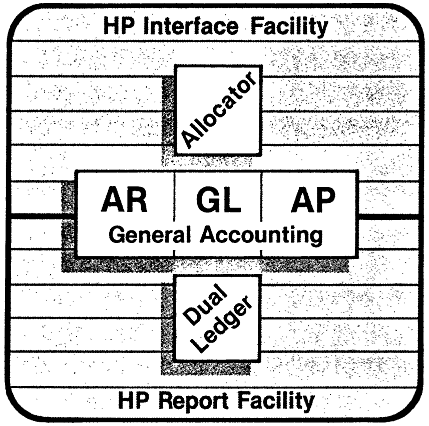
Introducing HP Financial Accounting

HP Financial Accounting is a series of eight software products for the HP3000 which offer you a range of solutions and can be tailored to fit your financial accounting needs. HP Financial Accounting applications are based on Hewlett-Packard's proven customization technology.

HP General Accounting is positioned for companies who have standard accounting and bookkeeping requirements. It is a low-cost, integrated application combining a comprehensive set of general ledger, accounts payable and accounts receivable functions.

HP General Ledger, HP Accounts Payable and HP Accounts Receivable can be installed separately and include additional screen and data base customization capability not offered in HP General Accounting.

Features in HP Dual Ledger and HP Allocator expand the functions of HP General Ledger to address the needs of organizations with more sophisticated ledger requirements. HP Allocator provides a simple automated way of performing sophisticated cost allocations. HP Dual Ledger provides a secondary set of books for automatically maintaining and reconciling management reporting requirements.



And, to help shape these products to your environment, Hewlett-Packard offers two additional customization tools. HP Report Facility and HP Interface Facility provide simple, cost effective solutions for all your financial accounting reporting and interfacing requirements.

Evaluating HP Financial Accounting

To help you evaluate HP Financial Accounting, general descriptions as well as information about Hewlett-Packard's support services for standard application products are available in this manual. Product Evaluation Guides are also available from your Sales Representative for a more detailed evaluation of functions, screens and reports.

You can best evaluate HP Financial Accounting when you have completely defined your financial accounting requirements . This includes an extensive analysis of your current system, organizational environment, plans for future expansion, deficiencies of your current accounting system, whether manual or automated, and a clear definition of additional features desired. Hewlett-Packard's Financial Specialists are available to help you define your requirements and match them to the features and benefits of HP Financial Accounting.

HP General Accounting

HP General Accounting is an integrated general accounting solution consisting of the comprehensive feature set and benefits available in HP General Ledger, HP Accounts Payable and HP Accounts Receivable. This application is designed to fit the needs of customers with standard general accounting requirements who may have unique reporting and interfacing requirements, but do not need customization of screens and data base structure.

HP General Accounting is a low-cost application providing ease of implementation and integration into your accounting environment. And, because it is offered by Hewlett-Packard, HP General Accounting can be easily upgraded to provide more sophisticated accounting capabilities when your needs change.

HP General Ledger

HP General Ledger automates the collection, organization, and summarization of your financial information. This is the heart of your financial system and must be flexible enough to support your current organization as well as any future organizational and policy changes.

HP General Ledger provides you with this flexibility to define charts of accounts, cost centers or other organizational units.

You decide how you want to see the management and financial information needed to run your business. You can review summary information with the detail backup available either on-line or in hard-copy reports. This kind of visibility helps you control expenses and keeps you informed about your financial position. Three separate budgets are maintained by account and/or cost center for measuring performance and for responsibility reporting.

Accounting entries generated within HP Financial Accounting are automatically posted to the general ledger. You can automate input of accounting entries from other systems using HP Interface Facility. Central validation assures that all data, whether entered on-line into HP General Ledger or from outside systems, is correct. The integrity of your financial information is protected.

HP General Ledger is an integrated application module within HP Financial Accounting. By itself, it provides the kind of financial control you need. Coupled with the advanced features available in HP Allocator and HP Dual Ledger, HP General Ledger solves the problems of more sophisticated, multi-national companies.

HP General Ledger Features

* On-line or batch data entry, validation and posting

* Multiple companies with unique charts of accounts, cost center organizations, security, fiscal year and parameters.

* Total flexibility in defining charts of accounts and cost centers

* Three separate budgets kept by account, cost center, or account within cost centers.

* Responsibility reporting and on-line analysis of expenses.

* Accruals provide automatic reversal bookings.

* Standard vouchers automatically posted each period.

* Ability to post to future or past periods.

* Automatic period and year-end closing procedures.

HP General Ledger Reports and Reviews

* Voucher Review

* Balance Sheet

* Subsididary Journal Review

* Profit and Loss Statement

* Account Summary Review

* Statement of Financial Position

* Account Detail Review

* Trial Balance Review

* Trial Balance

* Management Information

* Responsibility Reports

* Audit Reports

* Batch Status

* Maintenance Reports

HP Accounts Payable

HP Accounts Payable controls your liabilities and helps reduce the amount of operating capital needed to offset those liabilities. Cash requirement projections, coupled with flexible disbursement policies, allow you to simulate your cash position for optimal cash flow.

Automatic payment proposals are generated based on your specific policies regarding discounts, terms of payments, and due dates. These proposals can be made against multiple banks and are available for your review, modification, and approval before disbursements are actually made.

Automated bank reconciliations are designed so your cash accounts accurately reflect the bank balance. Payments which have not yet cleared the bank can be assigned to a reconciliation account for better cash management.

To safeguard working capital, extensive security and validation mechanisms are incorporated into HP Accounts Payable.

As a stand-alone product, HP Accounts Payable can be easily interfaced to your existing general ledger and/or purchase order management system. (For further interfacing information, please read about HP Interface Facility in this section).

As an integrated part of HP Financial Accounting, all accounting distributions will be posted into HP General Ledger. Invoices and cash disbursements may be recorded in any currency, and currency gains or losses will be automatically posted to your appropriate general ledger accounts.

In addition to regular check payment methods, HP Accounts Payable handles bank transfers.

HP Accounts Payable Features:

* On-line vendor review and analysis with quick access to vendors by a short name.

* Automatic voucher numbering.

* Corporate vendor turnover information for better visibility of total turnover with several vendors from same corporation.

* Effective discount analysis for better cash control.

* Automatic payment proposals with on-line maintenance, review, and approval of the proposals.

* Cash or accrual methods.

* Recurring payments automatically booked in each period.

* Automatic Use Tax, Value Added Tax, discount, and due date calculations.

* Flexible accounting distribution with system controlled balances.

* Multiple banks and multiple currencies with automatic reconciliations.

* Automatic booking of any currency gain or loss.

HP Accounts Payable Reviews and Reports:

* Corporate Review
* Summary Aged Trial Balance
* Vendor Review
* Detail Aged Trial Balance
* Single Open Item
* Open Item Register
* Open Items by Vendor
* Open Item Ranking
* Open Item Detail
* Trial Balance
* Reconciliation
* Cash Requirements Forecast
* Recurring Payments
* Discounts Taken/Lost
* Payment Proposal
* Audit Report
* Bank Reconciliation
* Vendor Listing
* Tax Reports
* Payment Register

HP Accounts Receivable

HP Accounts Receivable interactively records and controls the indebtedness of your customers. Increased cash inflow and better credit control directly affect the liquidity of your company. To better manage your cash resources, HP Accounts Receivable provides sophisticated aging methods, flexible cash application and close credit and collection control.

HP Accounts Receivable is designed with special capabilities for better cash management, easier data entry and better visibility into your business relationships. By maintaining customer turnover information by month for both the current and past year, you are able to analyze historical trends and project future business turnover.

To effect timely distribution of overdue invoices, special attention has been given to automatic printing of delinquency notices which can be written at up to five levels of severity. Exception reports are available to focus more effectively on the areas of particular concern, especially in the sensitive areas of credit and collection control. HP Accounts Receivable helps you maintain that sensitive balance between optimized cash inflow and good customer relationships.

Used stand-alone, HP Accounts Receivable can be easily interfaced with external ledgers, sales analysis, or sales order systems.

As an integrated part of HP Financial Accounting, all accounting distributions will be automatically posted into HP General Ledger. Invoices and cash receipts may be recorded in any currency, and currency gains or losses will be automatically posted to appropriate general ledger accounts.

Automated bank reconciliation capabilities ensure that your cash accounts accurately reflect the bank balance. Receipts which have not yet cleared the bank can be assigned to a reconciliation account for better cash visibility.

HP Accounts Receivable Features:

* On-line customer review and analysis with quick access to customers by a short name.

* Corporate turnover information for better control of accumulated credit limits with several customers from the same corporation.

* Automatic discount calculation and write-offs.

* Flexible aging by due date, invoice date, or estimated payment date, based on customer's actual payment history.

* Flexible automatic or manual cash application features.

* Recurring invoices booked in any future period.

* Delinquency notices automatically printed in any language with records of dates sent and severity level used.

* Direct debiting features.

* Handles multiple payment methods with automatic bank reconciliation.

* Multiple currencies with automatic booking of currency gain or loss.

HP Accounts Receivable Reviews and Reports:

* Corporate Review

* Summary Aged Trial Balance

* Customer Review

* Detailed Aged Trial Balance

* Open Items by Customer

* Open Item Register

* Open Item Review

* Open Item Ranking

* Open Item History

* Trial Balance

* Bank Reconciliation

* Credit Limit Listing

* Payment History Analysis

* Audit Report

* Bank Reconciliation

* Customer and Corporate Listing

* Recurring Entries

* Customer Statements

* Delinquency Notices

* Unearned Discounts

HP Dual Ledger

HP Dual Ledger is designed for companies that need to provide different sets of financial information for accounting and management purposes. This is especially useful if your company operates in more than one country or if you have complicated intra-corporate reporting requirements.

HP Dual Ledger is an expansion of HP General Ledger. Both sets of books may be kept in the same currency, or in two different currencies. If they are kept in different currencies, HP Dual Ledger provides an automatic revaluation of the accounts based on changes in currency rate. This revaluation is done by account, so that special inventory or depreciation accounts can be handled separately.

Your charts of accounts may be completely different in the two sets of books. Each accounting entry is made only once. If different currencies are used, the entries are automatically converted into the appropriate currency of your secondary books.

HP Dual Ledger Features

* Two charts of accounts per company.

* Entries may be automatically created in both books with one entry.

* Automatic reconcilation of both books.

* Automatic currency revaluation based on individual accounts.

* Simulation capability to analyze future financial position based on fluctuating currency rates.

* Automatic posting of currency gains or losses.

* Historical currency analysis.

* The same reports and on-line reviews which are available for HP General Ledger are provided for the secondary ledger.

HP Allocator

HP Allocator is an advanced cost allocation system for use with HP Financial Accounting. Sophisticated allocation criteria may be defined using simple data entry screens. Easy auditability assures control over allocations in companies with complicated organizational structures.

Allocation criteria which are not account based (such as number of employees, number of square feet, etc.) may be automatically entered from external systems or entered interactively.

You can allocate both budgeted and actual amounts from any group of accounts and/or cost centers and from either chart of accounts when HP Dual Ledger is installed. HP Allocator supports up to 99 levels of allocations, or you can use a sequential allocation process.

Allocation criteria are interactively defined and can be based on fixed amounts or percentages, variable amounts or relative percentages. Simulating allocations allows you to analyze the effect of your allocation criteria and methods before automatically booking the allocated amounts to your general ledger accounts.

HP Allocator Features:

* Allocates actual amounts and budgets.

* User defined allocation criteria, methods and bases.

* Allocates amounts from primary and/or secondary ledger

* Supports both multi-pass or sequential processes.

* Provides simulation capability for control and audit.

* Automatically books allocations to HP General Ledger.

HP Report Facility

HP Report Facility is a powerful report writer which you can use to either modify the standard reports provided as part of HP Financial Accounting or to define completely new reports.

Because HP Report Facility is designed for accountants, your accounting staff can easily get the information they need without any special programming or data base knowledge. Reports can be designed using a simple "check the box" approach.

The most typing involved is the report title. You can define simple, ad-hoc reports within minutes. More complex, production type reports can be easily defined by accessing additional definition screens.

HP Report Facility was used to design all standard reports provided with HP Financial Accounting. Therefore you can make changes to standard report formats easily. Pre-printed forms such as checks and statements are easy to define - no need to change the format of your business forms.

Consolidation reporting, responsibility reporting, financial and management reporting are all easily defined within HP Report Facility. And you can use the same report format for analyzing different sets of data.

HP Report Facility Features

* Designed for Accountants.
* Consolidated Statements.
* Exception Reporting.
* Responsibility Reporting.
* Ad-hoc and complex reporting capabilities.
* Pre-printed forms easily defined.
* On-line and hard-copy reports available.
* Simple page and line formatting.
* Report formats easily copied from other reports.
* Sample layouts printed.

HP Interface Facility

With HP Interface Facility your programming staff will not need to write those tedious but important interface programs required for a truly integrated system. HP Interface Facility uses simple on-line screen definitions of external files to automatically generate the necessary interface files.

HP Interface Facility is a customization tool designed to assist your systems personnel in interfacing HP Financial Accounting to your external systems. We know that accounting systems seldom stand alone, and that your EDP environment is often a mix of several hardware and software solutions.

With HP Interface Facility your analysts can easily modify the standard interfaces provided with HP Financial Accounting or add totally new ones to integrate HP Financial Accounting into your systems environment - without programming.

Data from other systems can be easily interfaced with HP Financial Accounting. Using simple data entry screens, your systems analyst can describe these external files to HP Interface Facility. They are then converted into the format needed by HP Financial Accounting or your external systems and processed according to your needs.

HP Interface Facility greatly increases the productivity of your programming staff and reduces the amount of time necessary for implementation. HP Interface Facility also increases your flexibility to add other application solutions later.

HP Interface Facility Features

* Interactive data specification for files going to or coming from HP Financial Accounting applications.

* Automatic restructuring of data within the files.

* Data conversion (ASCII to EBCDIC or vice versa)

* Sorting and summarization capabilities on data within the files for better audit control.

* Standard interface files pre-defined by HP.

* Capability to select individual records for processing.

Customization

Customization is a unique application technology developed by Hewlett-Packard to allow you to easily tailor our business applications to fit the requirements of your environment. Customization features are extensive and include the capabilities needed to tailor your screens, reports, interfaces, data bases and even processing specifications to your business environment.

Customizing Data

Data structures can be modified by:

* Adding data sets.

* Adding new data items.

* Deleting non-critical data items.

* Changing the length or type of data items.

Customizing Screens

Data entry and retrieval screens can be modified by:

* Changing existing screens.

* Designing new screens.

* Changing function key definitions.

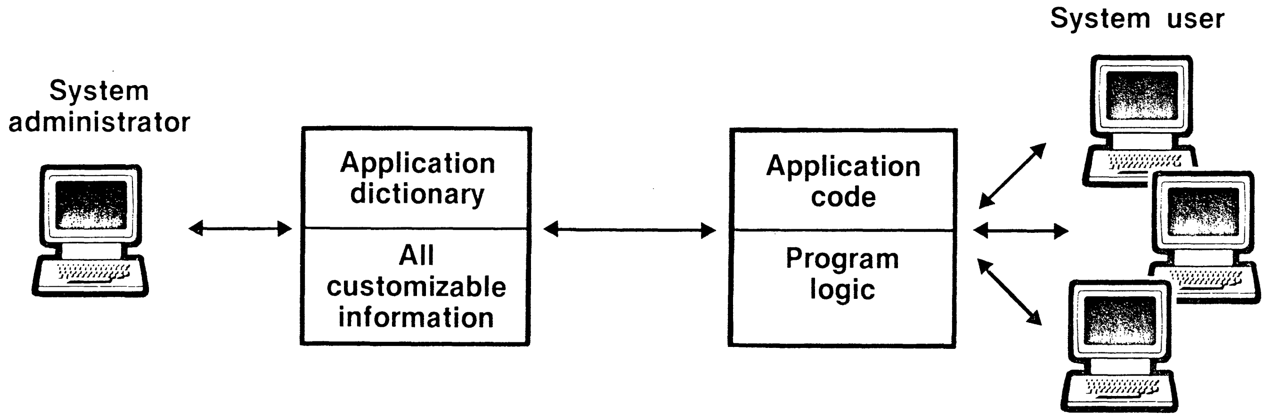
* Changing the sequence in which screens are displayed.

Customizing Reports and Interfaces

When HP Report Facility and HP Interface Facility are installed, you may alter standard reports and interfaces or create totally new reports and interfaces.

Expanded Customization

Expanded customization may be necessary for handling processing specific to your operation for such things as additional validation through external systems or updating external systems with financial information. This processing may be tailored to your business through special COBOL programs written by your analysts. These programs are accessed and executed at specified times within HP Financial Accounting.



IPB is a so called decision support system. This is a relatively new word for systems specially designed to support decision makers at all levels in the organization. DSS differs from management information systems in several ways. Where MIS focuses on information aimed at middle management, DSS focuses on decision aimed at managers. Where MIS is based on structured information flow, DSS emphasizes flexibility, adaptability, and quick response. Where MIS tries to integrate EDP jobs by business function, such as production MIS, marketing MIS etc., DSS is user initiated and controlled, and where MIS supplies inquiry and report generation usually with a data base, DSS supports the personal decision making style of individual managers.

In order to give an idea and understanding of what DSS stands for, let us look at some of the characteristic ways of using DSS.

DSS tends to be used at less well structured, unspecified problems that managers typically face, usually in a search learning process, where the decision maker works directly with the computer in a dialogue.

DSS attempts to combine the use of models or analytic techniques with traditional data access and retrieval functions.

DSS specifically focuses on features which make them easy to use by non computer people in an interactive mode, and

DSS emphasizes flexibility and adaptability to accomodate changes in the environment and the decision making approach of the user.

Another characteristic of DSS is how the work with DSS is organized in the organization. This can be described in the following figure:

```
   +-->      O    O     O     O    O        User
   !          .    .     .     .    .
   !           .    .    .    .    .        Intermediary
   !            .    .    .    .    .
   !            +---------------+
Adaptive        !               !           DSS generator
feedback        !               !
   !            +---------------+
   !            .    .    .    .    .
   !           .    .    .    .    .        Technical support
   !          .    .     .     .    .
   +-->      O    O     O     O     O       Toolsmith
```

The USER is the person facing the problem or decision - the one that must take action and be responsible for the consequences.

The INTERMEDIARY is the person who helps the user, perhaps merely as a clerical assistant to push the buttons of the terminal, or perhaps as a more important staff assistant to interact and make suggestions.

The TECHNICAL SUPPORTER is responsible for the hardware and that the

DSS generator is available and will run on the computer.

The TOOLSMITH develops new technology and new facilities for the DSS generator. He shall be ahead of the user's problems and have new tools available before the user recognizes the problem.

After this short introduction to DSS in general, I should like to go into more details with a specific decision support system, IPB, which stands for Interactive Planning and Budgeting. IPB is like most other DSS based on a table structure. The IPB table has 30 columns and up to 100,000 lines. The IPB language allows the user to do all kinds of manipulations with the data in IPB tables. There is no prior definition of calculation rules or meaning to the data in IPB tables. It is completely up to the user to define the rules of calculation and the meaning, which should be given to the data in a table. Each line in a table has a text part, where the user can describe what the line stands for followed by figures. It might be sales forecast per month for the next two years or it might be recipes, where each column shows the use of different compnonents to produce final products. It could be anything you might think of that can be given a numeric value. IPB allows the user to do all kinds of calculations by rows, by columns, and by elements and to create new rows or new columns from existing rows and columns or to create new tables from already existing tables.

The IPB command language is built up in a hierarchical way, which means that the higher you come up in the hiearachy the more powerful the commands become. At the lowest level you have the commands:

DATA
UPDATE
ALTERNATIVE
MODEL - BUILD

which all operate on lines and/or columns in data tables.

The command DATA is used to access or create tables. Under the command DATA you can add or delete lines to a table, modify existing lines, and move lines or columns from one place to another.

The command UPDATE is a very powerful and user friendly command for updating existing data tables. The command allows you to specify which columns and lines should be updated in an interactive mode. For instance, when you want to add some new periode budget figures to an existing budget, you will normally use UPDATE.

The command ALTERNATIVE is used to analyse the consequences of changes in the restraints. E.g. what happens to our liquidity and earning if we increase the price by 10 per cent, expect an increase in unit cost of Dkr. 280 per unit, and change the terms of credit from 45 days to 60 days. Problems of that type are known to any manager and will normally require tedious calculations and very seldom more than one alternative will be analysed. By IPB you can let the computer do all the calculations in a few seconds and the manager can consentrate on his real task, which is to generate suggestions and alternatives and decide which actions should be taken.

The command MODEL in IPB is used to access models, which are sets of calculation rules that should be applied to a data table. Under the command MODEL you can define all types of calculations including IF---ELSE constructions. Moreover, the command MODEL allows you to use the IPB function library, which has a number of financial functions available, such as internal rate of return, net present value and so forth. In IPB the data tables and models, e.g. set of calculation rules, are separated, which means that the same model can be applied on a number of data tables and a data table can be used as input to several models.

First when you use the command BUILD, you define which set of calculations should be carried out and which data table should be used as input. The result of a BUILD command is a new data table with the same structure as all other data tables, containing the results of the calculations as spe- cified in the model.

At the next command level you will find a number of commands which operate on complete data tables or on specified parts. Commands as ADD, SUBTRACT, MULTIPLY, and DIVIDE allow you to add, subtract, multiply, or divide whole data tables, element by element, in one command. Each of these four commands can handle up to 20 IPB data tables in one blow.

Another very powerful command at this level is the MATRIX command by means of which you can carry out a number of different operations on one or more data tables. For instance, if you have one table with sales forecasts for final products, another table with recipes for the products, the command MATRIX can give you a complete requirement plan, periode per periode, for all the components used to produce the final products.

The command REFERENCE is used to connect several lines in different data tables. It could be all lines depending on oil price. In case of a change in oil price all relevant lines in all tables can be updated in one command.

The REPORT and WRITE commands are closely related, almost in the same way as MODEL and BUILD. Under REPORT the user can specify in details what his report should look like. Again the report layout and actual data are separated, so that different data tables can be printed with the same layout specification, and a specific data table can be printed with different layouts. First when the command WRITE is used, a data table will be related to a report layout stored under the command REPORT.

The last command at this level is SCHEME, which from a data file can create a reporting scheme. The scheme will have the text part from all the specified lines, but instead of figures there will be fields in which new figures can be filled. The schemes can be distributed in the organization and for instance new budget figures can be filled in. The command UPDATE can then be used to update the relevant data table with the new figures.

Apart from the already mentioned commands, which all operate on data or data tables, IPB has a set of utility commands. The first two, MERGE and GET, can be used on any IPB file type, e.g. data tables, models, or report structures in order to create new files from those already existing. MERGE is used to merge two existing files to a new file, whereas GET is used to pick out parts from several existing files and put the pieces together to a new file.

COPY will make a copy of any IPB file. A useful command in connection with alternative generation, where the user does not want to destroy his base case.

PURGE will remove any IPB file from the system and finally STOP will stop an IPB session.

The highest command level in IPB is the strategy level. The command STRATEGY allows the user to set up a strategy file, holding all IPB commands and instructions. The contents of the strategy file can then any time be executed by the command AUTO followed by the name of a strategy file.

AN EXAMPLE

A shipowner has among other a 15-year-old bulk carrier. The vessel was built in "the good old days" before escalating oil prices.

It is possible by changing the stern (to a bulp stern) and by changes of the engine of the vessel to reduce cost for bunker oil. The changes of the engine will cause a minor increase of maintenance cost. More specific we have:

Fuel expence savings	US$ 1,900,000 p.y.
Increase maint. cost	US$ 200,000 p.y.

A shipyard will make the changes at a price of US$ 8,500,000. The shipowner's bank will finance this amount on the following conditions:

Loan	US$ 8,500,000
Interest rate (%)	12
Term of loan	8

Positive cash flow achieved by the investment can be invested at 15% interest per year. Financing of a negative cash flow will cost 17% yearly interest.

Uncertain factors are:
Inflation rate on

- oil prices
- maintenance cost

Development in interest rates

Analyse an 8-year-period as to pay back period and Internal Rate of Return.

Given this problem we can start by setting up a data table with all the assumptions and restraints. This is done by the command DATA INVDA, where INVDA is a user defined name for the data table and the response from IPB is that a new file is created. The user can now start to fill in his information. In this example the first 8 lines are used for comments. An asterisk as first character on a line will in all IPB files mark the line as a comment line. IPB will automatically assign line numbers to all lines, using the numbers 1.00, 2.00, 3.00, 4.00,---. This later allows the user to add up to 99 lines between two already existing lines, using the decimal part of the line number. The user can also at any time change the generation of line numbers by the instruction INTERVAL. This is done on line 9.00, where the next line number is defined to 10.00 and the step to 1.00 in order to have all the savings from line 10.00 and onwards. The two lines 10 and 11 define the savings. They consist of a text part describing the type of saving, an equal sign to separate the text from numerical information, and finally, the numeric value of the saving. The last figure will automatically be repeated for all columns. After the two saving lines the line number generation has again been changed to start on line 19.1 with steps of 0.1, and a new set of comments is given before the terms of loan are placed on line 20 and onwards. Finally, after some new comments, the internal parametres are placed on line 30 and onwards. To finish the building of a data table the instruction READY is used to close the table and store it on the disc.

We can now define the rules of calculation which should be used on the data table. As for the data table there is a number of comment lines in the model to explain what is going on in the different parts. Also the structure of the model follows the principles from the data table with different types of calculation placed at certain line intervals. All calculations concerning savings and repayment of loan are from line 20 and onwards. The calculation of cash flow starts on line 30 and continues to line 36, whereas the calculation of key figures to evaluate the investment has been placed from line 50 and onwards.

The first real model line is line 20, where total savings are calculated as the sum of data lines 10 to 19. In this way the model becomes more general and in case of other savings than the two already in the data table the new savings can just be placed in the data table from line 13 and onwards. They will automatically be taken into consideration without changing anything in the model. On line 21 the yearly repayment is calculated as total loan divided by number of years. This calculation should only be carried out for 8 years. This is controlled by C=1,8. Accumulated instalment is calculated on line 22 as last year's figure, L22(-1), plus last year's repayment L21(-1). Rest loan on line 23 is total loan, data line 22 or D22, minus accumulated instalment, L22, as long as the difference is positive, after that the rest is zero. The interest to be paid is rest loan, L23, multiplied by interest, D20, and divided by 100. Finally, total debt service can be calculated as instalment plus interest.

The annual cash flow from savings and repayment of loan is specified on line 30. Hereto comes interest on cash flow and on accumulated cash flow. Cash surplus can be reinvested at 15%, whereas deficit has to be financed at 17%. The calculation rule on line 31 therefore says that for positive cash flow the interest given on data line 30 is used, whereas data line 31 is used, if the cash flow is negative. On line 32 the interest on accumulated cash flow is calculated according to the same rule. The problem here is only that total accumulated cash flow is not yet defined. We only state that it will be calculated on line 34, and that we want to base our interest calculation on accumulated cash flow from last periode, L34(-1). IPB will then take care of the correct sequence of calculations. Total yearly interest can now be defined on line 33 as the sum of the two foregoing lines. Total accumulated cash flow is defined on line 34 as accumulated figure of last periode, L34(-1), plus this periode's contribution plus interest. Finally, the final annual cash flow is calculated on line 35 as annual cash flow plus interest.

On line 50 we ask for the internal rate of return based on a cash flow, where only savings and repaymnet of loan are taken into account. To get the internal rate of return we use the IPB function INTERNAL and simply refer to the model line by which the cash flow is defined, L30. On line 51 we calculate the internal rate of return, when reinvestment of positive cash flow and financing of negative cash flow have been taken into account. The next two model lines give net present value for the same two cash flows, using the interest stated in data line 30, column 1, D30(C1), for discounting.

The last executable model line is line 200, which says PLACE D10-D19. This instruction will simply transfer data line 10 to 19 directly from the input table to the output table. The purpose of this instruction will normally be that the user wants to show some of the basic assumptions in the final report. The last instruction, READY, will close the model and store the cal- culation rules on the disc.

In order to have the calculations carried out, the user must define a name for the output table and state which model and which input table should be used. In this example the command could be

```
BUILD INVRE FROM INVMO AND INVDA
```

and when the calculations are finished IPB returns with

```
READY FOR COMMAND
```

We might now look into the output table using the command DATA and just ask for a raw listing, but normally we want to design a report layout, which is done under the command REPORT. Under REPORT the user can define headings, choose the columns from the data table that he wants to see, and define subheadings that will be placed over each column. He can define the number of decimals that he wants, make underlinings, put in text lines, and pick out the lines from a data table in the sequence that he wants.

When the results are ready and the report layout has been defined, the command for writing a report is WRITE. In this example the full command would be

```
WRITE INVRE AFTER INVLA 51 SUPPRES
```

where 51 gives the number of lines available per page on the output unit and SUPPRES will replace all zeroes by blanks.

The result of the investment analysis should for internal rate of return and net present value be read in the following way: If the project stops in one of the years of 1982 to 1989, the internal rate of return and net present value are then shown for each of the years.

```
IPB ver 3.2, May 1982.
GIVE NUMBER OF CHARACTERS PER LINE
80
READY FOR COMMAND

  1.00 DATA INVDA
OLD FILE   29 LINES  LAST LINE  33.00   MON, SEP  6, 1982, 11:11 AM
COLUMNS  1,30

 34.00 LIST T
  1.00 ******************************
  2.00 *                            *
  3.00 * DATA FILE FOR INVESTMENT   *
  4.00 ******************************
  5.00 *                            *
  6.00 * SPECIFICATION OF SAVINGS   *
  7.00 ******************************
  8.00 *
 10.00 FUEL EXPENCE SAVINGS (1000$)  = 1900
 11.00 MAINTENANCE SAVINGS  (1000$)  = -200
 19.10 *
 19.20 ******************************
 19.30 *                            *
 19.40 * TERMS OF LOAN              *
 19.50 ******************************
 19.60 *
 20.00 INTEREST ON LOAN      (%)     = 12
 21.00 TERM OF LOAN          (NOS)   = 8
 22.00 LOAN                  (1000$) = 8500
 29.10 *
 29.20 ******************************
 29.30 *                            *
 29.40 * INTERNAL PARAMETERS        *
 29.50 ******************************
 29.60 *
 30.00 INTEREST ON POSITIVE CF (%)   = 15
 31.00 INTEREST ON NEGATIVE CF (%)   = 17
 32.00 *
 33.00 *

 34.00 READY
READY FOR COMMAND
```

```
  1.00 DATA INVMO
OLD FILE   41 LINES  LAST LINE 211.00   MON, SEP  6, 1982, 11:11 AM
COLUMNS  1,30

212.00 LIST
  1.00 ******************************
  2.00 *                            *
  3.00 * INVESTMENT MODEL           *
  4.00 ******************************
  5.00 *
 20.00 TOTAL SAVINGS                 = SUM(D10-D19)
 21.00 INSTALMENT ON LOAN            = D22/D21          C=1,8
 22.00 INSTALMENT ACC.               = L22(-1) + L21(-1)     C=2,10
 23.00 REST LOAN                     = D22 - L22 IF D22-L22 GE 0 ELSE 0
 24.00 INTEREST ON LOAN              = L23*D20/100
 25.00 TOTAL DEBT SERVICE            = L21 + L24
 29.10 *
 29.20 ******************************
 29.30 *                            *
 29.40 * CASH FLOW CALCULATION      *
 29.50 ******************************
 30.00 ANUAL CASH FLOW EXCL. INT.    = L20 - L25
 31.00 INTEREST ON ANNUAL CF         = L30*D30/100 IF L30 GT 0 ELSE L30*
 32.00 INTEREST ON ACC. CF           = L34(-1)*D30/100 IF L34(-1) GT 0 E
 33.00 INTEREST ON CASH FLOW         = L31 + L32
 34.00 ACCUMULATED CASH FLOW         = L34(-1) + L30 + L33
 35.00 ANNUAL CASH FLOW              = L30 +L33
 49.10 *
 49.20 ******************************
 49.30 *                            *
 49.40 * INVESTMENT ANALYSIS        *
 49.50 ******************************
 49.60 *
 50.00 IRR IF NO INTEREST ON CF      = INTERNAL L30
 51.00 IRR WITH INTEREST ON CF       = INTERNAL L35
 52.00 NPV IF NO INTEREST ON CF      = CAPITAL L30,D30(C1)
 53.00 NPV WITH INTEREST ON CF       = CAPITAL L35,D30(C1)
 54.00 *
199.10 ******************************
199.20 *                            *
199.30 * SPECIFICATIONS             *
199.40 ******************************
199.50 *
200.00 PLACE D10-D19
210.00 *
211.00 *

212.00 READY
READY FOR COMMAND
```

```
  1.00 BUILD INVRE FROM INVMO AND INVDA
READY FOR COMMAND

  1.00 REPORT INVLA
OLD FILE   24 LINES  LAST LINE  24.00   MON, SEP  6, 1982, 11:13 AM

 25.00 LIST
  1.00 HEADING I N V E S T M E N T - A N A L Y S I S
  2.00 HEADING INVESTMENT IN BULP STERN ON M/S MARY
  3.00 SEQUENCE TEXT(30) C1-C8
  4.00 SUBHEADING 1982,1983,1984,1985,1986,1987,1988,1989
  5.00 DECIMAL 0
  6.00 LINE -
  7.00 TEXT SAVINGS
  8.00 D200-D210
  9.00 LINE -
 10.00 D20
 11.00 LINE 2
 12.00 D21,D24
 13.00 LINE -
 14.00 D25
 15.00 LINE 2
 16.00 D30-D32
 17.00 LINE -
 18.00 D35
 19.00 LINE 2
 20.00 D34
 21.00 LINE 2
 22.00 D50-D53
 23.00 LINE
 24.00 LINE =

 25.00 READY
READY FOR COMMAND
```

```
  1.00 WRITE INVRE AFTER INVLA 51 SUPPRESS

                                        I N V E S T M E N T - A N A L Y S I S
                                        INVESTMENT IN BULP STERN ON M/S MARY

                                    1982  1983  1984  1985  1986  1987  1988

                                   -----------------------------------------
SAVINGS
FUEL EXPENCE SAVINGS (1000$)        1900  1900  1900  1900  1900  1900  1900
MAINTENANCE SAVINGS  (1000$)        -200  -200  -200  -200  -200  -200  -200
                                   -----------------------------------------
TOTAL SAVINGS                       1700  1700  1700  1700  1700  1700  1700


INSTALMENT ON LOAN                  1063  1063  1063  1063  1063  1063  1063
INTEREST ON LOAN                    1020   893   765   638   510   383   255
                                   -----------------------------------------
TOTAL DEBT SERVICE                  2083  1955  1828  1700  1573  1445  1318


ANUAL CASH FLOW EXCL. INT.          -383  -255  -128         128   255   383
INTEREST ON ANNUAL CF                -65   -43   -22          19    38    57
INTEREST ON ACC. CF                        -76  -140  -189  -221  -234  -223
                                   -----------------------------------------
ANNUAL CASH FLOW                    -448  -374  -289  -189   -74    60   216


ACCUMULATED CASH FLOW               -448  -822 -1111 -1300 -1374 -1314 -1098


IRR IF NO INTEREST ON CF           ***** ***** ***** ***** *****   -16
IRR WITH INTEREST ON CF            ***** ***** ***** ***** ***** ***** *****
NPV IF NO INTEREST ON CF            -333  -525  -609  -609  -546  -436  -292
NPV WITH INTEREST ON CF             -389  -672  -862  -970 -1007  -981  -900

                                   =========================================

READY FOR COMMAND

  1.00
```

1. INTRODUCTION

IPB (Interactive Planning and Budgeting) is a modelling language which originally was developed for budgeting and financial planning purposes on the HP3000. During the last 2 years an increasing interest for IPB has been registgered from a number of production companies.

The one thing these companies have in common, is that the production planning proces involves many changes and several iterations. At the same time the planners usually want the possibility of planning on a 'bottoms-up' basis, that is on a product-by-product basis.

Our recent development activitites within IPB has focused on these applications.

The following presentation will be separated in two parts

- first a brief discussion of our goals and a presentation of a small example and

- secondly a description of practical use in a major production company.

2. THE GOALS OF THE DEVELOPMENT.

The three most important goals of our recent development aimed at production planners were to:

1. Enable the planner to - in an easy way - model his production system on the HP3000. All the fundamental modelling facilities of course allready existed in IPB, but especially the handling of large recipes and list of parts were improved.

2. Facilitate the boring and tedious 'musts' of production planning, like updating of product calculations, combining sales forecast with recipes and rawmaterial requirements, etc.

3. Enable the planner to - THROUGH AN INTERACTIVE LEARNING PROCES WITH HIS HP3000 - improve his understanding of the production system and improve plans.

In addition it is an obvious advantage to be able to use the same system for different planning functions (production, purchase/stock, financial, budgetary etc.), i.e. eliminate the 'systems interface problem'.

IPB enables you to set up a 'man-machine-system' which supports the decision making through an interactive learning process. Or in other words, makes it possible for an intelligent person to use the HP3000 as a qualified sparring partner.

3. AN EXAMPLE

As illustration of some of the possibilities consider the following example, where complexity and number of data have been reduced due to presentation rather than systems limitations.

A candy producer makes four candy products which involves 7 different types of raw material as well as some production equipment and labour.
Recipes, sales forecast and buying prices are described in following three tables:

RECIPE:

RECIPES, LABOUR AND EQUIPMENT USE PER 100 KILOS FINISHED GOODS

	CANDY THINGS	CANDY GIZMOS	CANDY WIDGETS	CANDY MEN
* PRODUCTION REQUIREMENTS				
LIQUORICE (KILOS)..........	50	10	50	10
WHEAT FLOUR (KILOS)..........	10	40	10	10
SUGAR (KILOS)..........	10	10	5	5
GOLDEN SYRUP (KILOS)..........	10		10	5
SALT (KILOS)..........	10	10	10	5
SAL AMMONIAC (KILOS)..........		10		
GUM ARABIC (KILOS)..........	10	20	15	65
LABOUR (HOURS)..........	10	20	35	5
MIXER EQUIPM (HOURS)..........	10	5	10	30

SFCAST:

SALES FORECAST , CANDY PRODUCTS. PLANNING YEAR 1983.

	FIRST QUARTER 1983	SECOND QUARTER 1983	THIRD QUARTER 1983	FOURTH QUARTER 1983
* FIGURES IN KILOS				
CANDY THINGS..................	9700	10200	8950	9500
CANDY GIZMOS..................	4400	4800	5580	4700
CANDY WIDGETS..................	1200	1400	1700	2500
CANDY MEN..................	3400	3500	3700	3600

```
BUYPRIC:
                                PRICES.
                       RAW MATERIALS AND LABOUR.

                             ------------

 LIQUORICE     (KILO) ..........       9.85
 WHEAT FLOUR   (KILO) ..........       1.05
 SUGAR         (KILO) ..........       4.65
 GOLDEN SYRUP  (KILO) ..........       8.10
 SALT          (KILO) ..........       0.60
 SAL AMMONIAC  (KILO) ..........      13.25
 GUM ARABIC    (KILO) ..........       9.00

 LABOUR        (HOUR) ..........      52.75
```

Product calculations.

Based on the above assumptions a product self cost calculation and printing of result could be formulated like:

```
MULTIPLY RECIPE BUYPRICE TO COST
BUILD SLFCOST FROM MODEL1 AND COST
WRITE SLFCOST AFTER LAYOUT1
```

which leads to

```
                                          SELF COST. CANDY PRODUCTS.
                                        PER 100 KILOS FINISHED GOODS.

                                     CANDY     CANDY     CANDY     CANDY
                                    THINGS    GIZMOS   WIDGETS       MEN

                                  ---------------------------------------
 LIQUORICE....................      492.5      98.5     492.5      98.5
 WHEAT FLOUR..................       10.5      42.0      10.5      10.5
 SUGAR........................       46.5      46.5      23.3      23.3
 GOLDEN SYRUP.................       81.0                81.0      40.5
 SALT.........................        6.0       6.0       6.0       3.0
 SAL AMMONIAC.................                132.5
 GUM ARABIC...................       90.0     180.0     135.0     585.0
                                  ---------------------------------------
 TOTAL MATERIALS .............      726.5     505.5     748.2     760.7
 LABOUR.......................      527.5    1055.0    1846.2     263.7
                                  ---------------------------------------
 TOTAL COST...................     1254.0    1560.5    2594.5    1024.5
                                  =======================================
```

or on a product by product basis

```
BUILD CSTPRO1 FROM MODEL2 AND COST
WRITE CSTPRO1 AFTER LAYOUT3
```

```
                                   SELF COST. CANDY PRODUCTS.
                                  PER 100 KILOS FINISHED GOODS.

                                          CANDY         % OF
                                         THINGS        TOTAL

                                     ----------------------
LIQUORICE.....................            492.5         39.3
WHEAT FLOUR...................             10.5          0.8
SUGAR.........................             46.5          3.7
GOLDEN SYRUP..................             81.0          6.5
SALT..........................              6.0          0.5
GUM ARABIC....................             90.0          7.2
                                     ----------------------
TOTAL RAW MATERIALS ..........            726.5         57.9
LABOUR........................            527.5         42.1
                                     ----------------------
TOTAL ........................           1254.0        100.0
                                     ======================
```

Combining sales forecasts with recipes.

Consequences of sales forecasts as to use of raw materials, labour and production equipment can be analyzed on a product by product basis. For instance, what are the production input requirements based on the sales forecast for CANDY WIDGETS?

```
MATRIX RECIPE(C3) * SFCAST(L3) TO CONSEQ1
WRITE CONSEQ1 AFTER LAYOUT2
```

This leads to

```
                                  USAGE OF RAW MATERIALS, LABOUR AND EQUIPM.
                                  BASED ON SALES FORECAST FOR CANDY WIDGETS.

                                        FIRST     SECOND      THIRD     FOURTH
                                      QUARTER    QUARTER    QUARTER    QUARTER
                                         1983       1983       1983       1983

                                     --------------------------------------
* PRODUCTION REQUIREMENTS

LIQUORICE    (KILOS)..........            600        700        850       1250
WHEAT FLOUR  (KILOS)..........            120        140        170        250
SUGAR        (KILOS)..........             60         70         85        125
GOLDEN SYRUP (KILOS)..........            120        140        170        250
SALT         (KILOS)..........            120        140        170        250
GUM ARABIC   (KILOS)..........            180        210        255        375

LABOUR       (HOURS)..........            420        490        595        875

MIXER EQUIPM (HOURS)..........            120        140        170        250
                                     --------------------------------------
```

One way to continue with an analysis for all four products would be to add the consequences of each of the products to a total, build some keyfigures and finally print the result. Formulation:

```
ADD CONSEQ1 CONSEQ2 CONSEQ3 CONSEQ4 TO TOTAL
BUILD PRODREQ FROM MODEL3 AND TOTAL
WRITE PRODREQ AFTER LAYOUT4
```

```
                          USAGE OF RAW MATERIALS, LABOUR AND EQUIPM.
                          BASED ON SALES FORECAST FOR ALL CANDY PROD.

                             FIRST   SECOND    THIRD   FOURTH
                           QUARTER  QUARTER  QUARTER  QUARTER    TOTAL
                              1983     1983     1983     1983     1983

                          --------------------------------------------
* PRODUCTION REQUIREMENTS

LIQUORICE    (KILOS)......    6230     6630     6253     6830    25943
WHEAT FLOUR  (KILOS)......    3190     3430     3667     3440    13727
SUGAR        (KILOS)......    1640     1745     1723     1725     6833
GOLDEN SYRUP (KILOS)......    1260     1335     1250     1380     5225
SALT         (KILOS)......    1700     1815     1808     1850     7173
SAL AMMONIAC (KILOS)......     440      480      558      470     1948
GUM ARABIC   (KILOS)......    4240     4465     4671     4605    17981

LABOUR       (HOURS)......    2440     2645     2791     2945    10821
LABOUR HOURS AVAIL. ......    2600     2600     2600     2600    10400
                          --------------------------------------------
 LABOUR SURPLUS/DEFICIT        160      -45     -191     -345     -421

MIXER EQUIPM (HOURS)......    2330     2450     2454     2515     9749
MIXER HOURS AVAIL.........    2600     2600     2600     2600    10400
                          --------------------------------------------
 MIXER SURPLUS/DEFICIT....     270      150      146       85      651
                          --------------------------------------------
```

4. AN IMPLEMENTATION

4.1 PROBLEM

Canner Ltd. is a company which produces and sells 1200 different kinds of articles within canned products. After having been introduced to IPB the company chose to use 'modelling language' to the solution of a longstanding problem in the production planning.

The production is divided into a number of different production lines in 5 different factories. The pre-treatment and the packing of the products are manually operated, and the sealing of the packing is done by machinery.

The manning can to a certain degree be adjusted at 3 months' sight. The importance of the company in the regional community and retraining problems when new staff is employed imply however, that a fairly stable staff is desirable.

The problems are thus:

- Tactical staff planning at 3-12 months' sight
- Production planning, consequences from 1 to 3 months as a basis for the production planning at the operational level.

The basis of the production planning is a 12 months' forecast for the different articles. These forecasts are revised each month. They are transferred from the finance/forecast system to IPB through a special program. At the same time information about the stock at the beginning of the month are transferred after the end of each month.

The stocking policy is described as percentages of estimated sales the following months. Consequently the calculation of the production plan is mathematically simple but the quantity of data is too high for being manually operated.

The production plans are calculated for each factory and are converted into personnel units, by multiplying production plans in 1000's by personnel unit factors per article.

The available labour expressed as personnel units is calculated with a basis in the temporary manning plans, and the supply/need totally and for each factory are related as to provide the basis for a manual correction of manning plans, stocking policy or distribution between factories.

New iterations are carried out and are thus the principal element in the learning process until reasonable stocks and manning plans are established.

First of all the system is directed towards production planning but the further applications of the established structure are evident:

- Sales forecasts and unit prices can immediately produce sales budgets.
- Production plans and unit costs can immediately be combined to unit cost budgets.
- Production plans can be related to already established list of parts/recipes for purchase planning.
- Optimizing of stock for raw materials as well as manufactured goods can be developed when the historical data for forecast uncertainties and times of delivery are systemized.

**) PERSONNEL UNIT FACTOR: Coefficient, expressing use of labour per 1000 produced articles, specified per article.

4.2 SOLUTION

Presentation of system structure, relations to other planning functions and 'key reports' will be based on overhead slides.

6 SEP. 1982

I.C. SYSTEMS n.v.

gouverneur verwilghensingel 4 B-3500 hasselt

tel.: +32-11.227701

THE FINANCIAL MODEL PROCESSOR

paper presented at HP 3000 Users Meëting
Copenhagen, October 82

AUTHORS:

prof.Ir.georges schepens
Facultés Universitaires Notre Dame de la Paix
NAMUR , BELGIUM

J.luc beyers
Managing Director BEYERS & PARTNERS
BRASSCHAAT, BELGIUM

The ***FINANCIAL MODEL PROCESSOR F.M.P.***
AUTOMATIC PILOT
FILE PROCESSOR

are products from Beyers & Partners
Michielssendreef 26
B 2130 Brasschaat
BELGIUM
Tel. 03/651.91.14 - 03/234.11.08

who take care of
- *further development and maintenance*
- *documentation*
- *management courses and support*
- *licences to final users or software houses.*

1. GENERAL OVERVIEW

The F.M.P. and its complementary packages

The Financial Model Processor (F.M.P.) is an interactive system for financial modelling. It was conceived for managers and business students, who understand the internal logic of a financial problem, but lack the expertise and especially the time to convert it into a computer based simulation program.

The F.M.P. allows one to conceive his own financial models and to process them on a computer without any previous exposure to computer programming. This is achieved by a permanent dialogue guiding the user through all procedures and providing extra comment if desired, or corrective action if an erroneous answer is detected.

We are definitely talking about 'Computer assisted Financial Analysis', since

YOU are conceiving the model and the reports and defining the data that will be used in the analysis of alternative scenario's

and THE COMPUTER is doing all the computations, is providing storage and printing capacity, but also guides you step by step through the process.

The heart of F.M.P. is its very powerful and applications-oriented model definition language, exhibiting functions like:

TAXABLE
LIFO
FIFO
EXPONENTIATION
logical operations
....

The user has the possibility to build and adapt his own financial models with a limited number of simple but powerful instructions, also understandable to non-F.M.P.-users.
This means that the F.M.P. allows the manager to work by himself on the computer, without being dependent on classical programming languages.

Since every task is performed in dialogue with the computer, no programming effort is required to modify models, generate reports, enter data and update files.

The 'Financial Model Processor' includes preprogrammed modules for
- discounted cash flow calculations,
- funds flow analysis,
- backward computations,
- stochastic simulations.

The F.M.P. also allows one to store models, report definitions or data on disk. Consequently not every user has to conceive his own models. He can also rely on standard models, permanently stored on file (e.g. for the analysis of financial statements, budgetting,...).

The strength of such a package lies in the possibility to perform extensive sensitivity analysis, with hardly any extra effort, using simple procedures for model or data modifications.

F.M.P. also allows consolidation of different models, by making transfers of data from one model to another one simple.
Its 'File Processor' extension adds many data handling features, including selective data retrieval and update in files.

Its 'Automatic Pilot' extension, allows easy and fast processing of complex consolidations.

Its Implementation

The F.M.P. is written in a high-level language, close to the FORTRAN 77 standard.

The F.M.P. and its complementary packages are available in English, Dutch, German and French, on a wide range of computers, among whom the HP 1000 and HP 3000.

Modeldefinitions, originally written in another language, are automatically translated.

The F.M.P. works as an interpreter for user-written models. Therefore a model modification never requires recompilation. Considerable time savings can thus be realised.

The authors

The concepts of the 'FINANCIAL MODEL PROCESSOR (F.M.P.)' originated from a close co-operation between professor Georges SCHEPENS (Facultés Universitaires Notre-Dame de la Paix of Namur, F.U.N.D.P.) and Luc BEYERS (managing director of Beyers & Partners pvba) when both lectured at Prof. Vlerick's Management Center at the State University of Ghent (R.U.G.). G.Schepens lectured on Operations Research and Information Systems and L.Beyers on Business Planning & Control.

Scientific Support

The publication of papers on specific subjects (e.g. analysis of financial statements, investment analysis,...) and the organisation of training programmes is done in collaboration with Prof. E. DE LEMBRE (R.U.G.), Prof. M. GUILLAUME (F.U.N.D.P.), Prof. H. OOGHE (R.U.G.) and Prof. C. VAN WYMEERSCH (F.U.N.D.P.).

Applications

The use of F.M.P. is spreading fast in two distinct areas.

Business and Industry, where more than 80 user contracts were already granted. The main fields of applications are:

- analysis of financial statements
  - balance sheet
  - income statement
  - funds flow statement
  - ratio analysis

- long range planning
- investment analysis
- budgetary planning and control
- liquidity planning
- inventory simulation and control
- consolidation
- project evaluation
- lease or buy decisions
- sales analysis and projections
- market simulation
- breakeven analysis
- ...

Users range from small companies upto important banks and industrial groups.

Education, where several universities have a user licence, among whom the universities of Ghent, Leuven and Namur in Belgium, Erasmus in the Netherlands and Växjö in Sweden.

Key points of the F.M.P. in education are

- the very high degree of user friendliness of the F.M.P., which shortens the specific training of the user to a couple of hours.

- and the fact that the F.M.P. allows to focus a course on the definition of the financial logic and the conception of alternative scenarios (and away from the computational burden).

Licences

Licences to final users or software houses are exclusively granted by

BEYERS & PARTNERS pvba
Michielssendreef 26
B 2130 Brasschaat BELGIUM
Tel. 03/651.91.14 - 03/234.11.08

2.BRIEF DESCRIPTION OF THE F.M.P.

The data matrix

The input data and computational results we will be working on, are stored in a two-dimensional table. A typical F.M.P. matrix size on an HP 3000 is 150 lines and 24 columns. Special subsystems enable you to transfer information from one model to another, allowing a modular approach to complex problems.

The Model

A model is a series of instructions, defining lines and columns.

Each instruction consists of:

- line or column number
- line or column label
- status: every line or column can be defined as being
    - either an INPUTLINE: for this line we will provide external data
    - or a COMPUTATION LINE: this line will be calculated on the basis of one or more other lines.

```
Ex. L12=UNITS SOLD=INPUT          INPUT LINES
    L15=PRICE PER UNIT=INPUT

    L18=SALES=L12 * L15           COMPUTATION LINE

    C1=1981=INPUT                 INPUT COLUMNS
    C2=1982=INPUT

    C5=GROWTH=C2-C1               COMPUTATION  COLUMN
```

Since the F.M.P. was conceived and is being mainly used for financial applications, the columns will normally represent time periods (years,quarters,months,...).

The menu

The choice of the task to be executed is done by means of the so-called 'Menu', i.e. a list of all the tasks the F.M.P. is able to execute, as there are: input and modification of data, modification of the model, printing or creation of a file of the model, data or reports, printing out a plot, and a number of more specific tasks, that will be discussed further on. After the execution of a task, the F.M.P. returns to this Menu, enabling you to choose the next task. Once within a task, the F.M.P. will get all additional specifications by means of a question and answer system.

The data

Interactive data entry can be done:
- either on a line per line basis
- or on a column per column basis

In the former case several input modes are possible, such as:
- specific values in one or more periods
- constant values
- linear evolutions
- exponential evolutions
- linear, degressive, sum of the digits, or accelerated depreciation methods

The reports

To define a report one has to introduce a title and the desired column and line numbers. The F.M.P. extracts the corresponding data elements from the matrix and produces the report, either on screen or on paper. It is possible to insert subtitles, underline data, even to overwrite parts of the numerical data.

Input and Modifications

The input of models, report definitions and data is fully computer-controlled, for manual input as well as for input from files.

Specific subsystems allow the user to modify models, report definitions and input data very easily.

Specific subsystems

Four subsystems in the F.M.P. refer to specific application fields:

- DISCOUNTED CASH FLOW calculations. It includes the computation of the "internal rate of return", the "net present value" and the "equivalent period values".

- FUNDS FLOW analysis. By answering a questionnaire the user defines references to the balance sheet and the cash flow elements, enabling the F.M.P. to compute the corrected sources and uses of funds for a given time period.

- ITERATIVE MODULE. It offers the possibility to reach a target value (e.g. profit) by adapting a specified free variable (e.g. units sold).

- STOCHASTIC SIMULATION. The user can define statistical instead of deterministic information for several elements. He can also define which results have to be captured and displayed in the form of frequency distributions, after a number of simulations (Monte-Carlo).

3. FILE PROCESSOR - AUTOMATIC PILOT

When using the F.M.P. to handle large financial problems requiring a vast amount of data, one may come up with two suggestions.

1. It would be nice to go beyond the present transfert file system and to provide a data retrieval system where the simple fact of writing L12=PROFIT=INPUT and C4=1980, would retrieve the profit of 1980 from the corresponding file, and enter it in the required slot of the F.M.P. matrix... .

2. The conversational mode is an ideal way to guide the user step by step through the F.M.P., while taking corrective action whenever inappropriate input is encountered. But when it boils down to rerun for the n-th time a sequence of several models, with all the file handling this may require, it would be nice to be able to define this sequence of commands as a jobstream and launch it with a single command... .

These ideas have gone their way, and result into two complementary products to the F.M.P., with whom they are fully compatible.

The first idea led to the FILE PROCESSOR (F.P.).

It facilitates the exchange of data between the F.M.P. model and large files on disk by allowing selective data retrieval and update.

The file handling section of the FILE PROCESSOR takes care of file printing, total or partial duplication of files, concatenation, elimination, conversion to and from formatted files. This last feature is meant to ease the integration of the F.M.P. with other computer applications.

The second idea led to what we nicknamed the AUTOMATIC PILOT (A.P.).

It accepts a jobstream, this is a sequence of F.M.P. and F.P. commands that have been previously entered in a file. The execution of a jobstream is launched by a single command given in the F.M.P. menu.

Thus the AUTOMATIC PILOT may be seen as a "BATCH" extension of the F.M.P..

But there is more ...

The name automatic pilot originates from one of its commands, called 'FMP', that allows the user to interrupt the automatic control at a given point and switch back to conversational mode, where he recovers the full potential of F.M.P. and F.P. tasks. At any time he can switch back to automatic control and proceed within the jobstream where he left it.

When running the AUTOMATIC PILOT from a terminal, interactive control will also be returned to the user as soon as inappropriate commands are encountered. The user may then stop the session or take corrective action in conversational mode and subsequently return to automatic control.

FINANCIAL PLANNING FOR A WHOLESALE TRADE

As an example we will consider a wholesale trade and build a model for its financial planning. For the sake of simplicity we will limit the trade to a single product. This model includes a Profit & Loss Statement and a Cash Flow Statement.

Column definition

We wish to obtain the results for 6 months of activity and also the overall mid-year results. Since we will have to enter an initial situation for some of the data, we will use 8 columns. The first 7 columns are input columns, the 8th column is a computation column:

```
C1=INITIAL SIT.=INPUT
C2=JANUARY=INPUT
C3=FEBRUARY=INPUT
C4=MARCH=INPUT
C5=APRIL=INPUT
C6=MAY=INPUT
C7=JUNE=INPUT
C8=MID-YEAR=TOTAL OF L2 TO L7
```

Data

1. Sales:
- *units sold in each period*
- *sales price in each period*
- *terms of payment: net 30, end of month, corresponding to an average collection period of 45 days or 1.5 periods.*

2. Purchases:
- *considering that we have to include a supplement of 2% the purchased goods, in order to provide for losses caused by theft, accidents etc., the consumption of goods in each month will be 2% above sales. Assuming immediate re-ordering and a delivery time of two weeks, purchases are equal to consumption, but shifted over half a period. Purchases are paid with another delay of two weeks or half a period.*
- *purchase price in every period*
- *inventory-valuation method: Last-in, First-out*
- *initial inventory in units (price per unit is 'purchase price' in initial period)*

3. Fixed costs

4. Depreciation

5. *Cash balance in initial period*

6. *Losses carried forward from previous years*

7. *Results of accounts receivable and accounts payable from preceding periods: cash flow from past*

8. *Negative cash balances will be financed with a bank loan, at an interest rate of 1.5% per month, payable at the end of each quarter.*

9. *Taxes: Income tax rate of 48%, payable in every period.*

Part of these data can be considered as being structural, fixed for the company, such as inventory losses and payment terms. Interest rate, though being an external element, will also be considered as pre-determined for this problem, since we consider that it is not susceptible to negotiation, and thus cannot be changed.
These informations will be built in into the model. Other data (the above underlined ones) are external to the model, and will be adapted for the analysis of alternative situations. They are defined as being INPUT LINES:

```
L1=UNITS SOLD              =INPUT
L2=SALES PRICE             =INPUT
L3=PURCHASE PRICE          =INPUT
L4=INITIAL INVENTORY       =INPUT
L5=FIXED COSTS             =INPUT
L6=DEPRECIATIONS           =INPUT
L7=INITIAL CASH BALANCE    =INPUT
L8=LOSSES CARRIED FORWARD  =INPUT
L9=CASH FLOW FROM PAST     =INPUT
```

After having defined and entered the model, we will specify the data, for all lines and columns defined as being INPUT LINES and INPUT COLUMNS. As mentioned before, we can enter a value for every period, but also a linear or exponential evolution, where we specify the starting value and the growth rate per period. The data that are used for this example are shown on the next page.

DATA

	INITIAL	JANUARY	FEBRUARY	MARCH	APRIL	MAY	JUNE
1. UNITS SOLD	0.	1000.	1050.	1100.	1150.	1200.	1250.
2. SALES PRICE	0.	20000.	20000.	20000.	20000.	20000.	20000.
3. PURCHASE PRICE	12000.	12360.	12731.	13113.	13506.	13911.	14329.
4. INITIAL INVENTORY	800.	0.	0.	0.	0.	0.	0.
5. FIXED COSTS	0.	5500000.	5500000.	5500000.	5500000.	5500000.	5500000.
6. DEPRECIATIONS	0.	125000.	93750.	70313.	62500.	62500.	62500.
7. INITIAL CASH BALANCE	1000000.	0.	0.	0.	0.	0.	0.
8. LOSSES CARRIED FORWARD	-8800000.	0.	0.	0.	0.	0.	0.
9. CASH FLOW FROM PAST	0.	3000000.	4000000.	3500000.	250000.	0.	0.

The computations we wish to execute are scheduled below:

COMPUTATIONS	RESULT IN P & L STAT.	RESULT IN CASH FLOW
Sales=units sold * sales price	SALES	
shifted over 1.5 period		RECEIPTS
Purchases		
Consumption=units sold * 1.02		
Purchases=inventory replenishment=consumption shifted over 0.5 periods		
Purchase payment=purchases*purchase price shifted over 0.5 periods		PAYM.MATERIAL
Material costs=consumption*(LIFO-purchase price)	MATER. COSTS	
Fixed costs	FIXED COSTS	FIXED COSTS
Depreciations	DEPRECIATION	
Interests		
Bank loan=last month's cash balance if negative		
Interest=bank loan * 0.015	INTEREST	
Payment interest=sum of interests per 3 months		PAYM.INTEREST
Total costs=fixed costs+depreciation +material cost+interest	TOTAL COSTS	
Gross profit=sales - total costs	GROSS PROFIT	
Income taxes= 0.48 * gross profit if positive; if negative carried forward to next period, which also happens for losses from the preceding periods	INCOME TAXES	INCOME TAXES
Net profit=gross profit - income taxes	NET PROFIT	
Cash flow month=receipts - payments (incl.from past periods)		CASH FLOW MONTH
Cash balance=cash balance last month + cash flow of the month		CASH BALANCE
Net present value=NPV of Cash Flow in period 0, discount rate 1.5 % per month		NET PRES.VALUE

These computations are translated into the following COMPUTATION LINES in the FMP-Model:

```
L10=SALES                    =L1*L2
L11=RECEIPTS                 =SHIFT L10 OVER 1.5
L12=CONSUMPTION              =1.02*L1
L13=INVENTORY REPLENISHMENT  =SHIFT L12 OVER 0.5
L14=PURCHASE MATERIALS       =L13*L3
L15=PAYMENT MATERIAL         =SHIFT L14 OVER 0.5
L16=SUPPLY FOR LIFO          =L4+L13
L17=MATERIAL COSTS           =LIFO: SUPPLY L16,PRICE L3,CONSUMPTION L12
L18=BANK LOAN                =RE-ACCESS TO L31
L19=INTEREST                 =0.015*L18
L20=PAYMENT INTEREST         =SUM OF L19 FROM C2 TO C7 PER 3
L21=TOTAL COSTS              =L5+L6+L17+L19
L22=GROSS PROFIT             =L10-L21
L23=TAXABLE RESULT           =L22+L8
L24=TAXABLE BASIS            =TAXABLE OF L23
L25=INCOME TAXES             =0.48*L24
L26=NET PROFIT               =L22-L25
L27=CASH FLOW MONTH          =L11+L9-L15-L5-L25-L20
L28=CASH MUTATION            =L7+L27
L29=CASH BALANCE             =CUMULATIVE SUM OF L28
L30=BANK LOAN                =COMPARE L29 TO 0. AND GET (-L29,L0,L0)
L31=NET PRESENT VALUE        =NET PRESENT VALUE OF L27 WITH 1.5 % IN C1
```

*For the reports we wish to include subtitles, underlining, etc..
For this purpose we include some TEXT LINES in the model:*

```
T32=5,'CONSUMPTION'-
T33=5,'SUPPLY'-
T34=5,'MATERIAL COST'-
T35=43,7'  ----------'
T36=3,'EARNINGS'-
T37=3,'COSTS'-
T38=3,'PROFIT'-
T39=33,96'-'
T40=31,8'  **********'
T41=+45,'IN REFERENCE PERIOD 0, DISCOUNT RATE OF 1.5 % PER MONTH',100,32' '
T42=+31,'          - '
T43=+43,6'         - '
T44=+103,'         - '
```

SCENARIO : FINANCIAL PLANNING FOR WHOLESALE TRADE

REPORT 1 : INVENTORY

	INITIAL	JANUARY	FEBRUARY	MARCH	APRIL	MAY	JUNE
CONSUMPTION							
1.UNITS SOLD	-	1000.	1050.	1100.	1150.	1200.	1250.
12.CONSUMPTION	-	1020.	1071.	1122.	1173.	1224.	1275.
SUPPLY							
4.INITIAL INVENTORY	800.	-	-	-	-	-	-
13.INVENTORY REPLENISHMENT	-	510.	1045.	1096.	1147.	1198.	1249.
16.SUPPLY FOR LIFO	800.	510.	1045.	1096.	1147.	1198.	1249.
MATERIAL COST							
3.PURCHASE PRICE	12000.	12360.	12731.	13113.	13506.	13911.	14329.
17.MATERIAL COSTS	-	12423600.	13616051.	14684102.	15804256.	16978680.	18209620.

```
SCENARIO :              FINANCIAL PLANNING FOR WHOLESALE TRADE


REPORT  2 : PROFIT & LOSS STATEMENT
                                  JANUARY     FEBRUARY    MARCH       APRIL       MAY         JUNE        MID-YEAR
-----------------------------------------------------------------------------------------------------------------------
 EARNINGS
 --------
 1.UNITS SOLD                       1000.       1050.       1100.       1150.       1200.       1250.       6750.
 2.SALES PRICE                     20000.      20000.      20000.      20000.      20000.      20000.           -
10.SALES                        20000000.   21000000.   22000000.   23000000.   24000000.   25000000.  135000000.

 COSTS
 -----
17.MATERIAL COSTS               12423600.   13616051.   14684102.   15804256.   16978680.   18209620.   91716309.
 5.FIXED COSTS                   5500000.    5500000.    5500000.    5500000.    5500000.    5500000.   33000000.
 6.DEPRECIATIONS                  125000.      93750.      70313.      62500.      62500.      62500.     476563.
19.INTEREST                            0.      69777.      89379.      21928.       2251.          0.     183335.
                                --------------------------------------------------------------------------------------
21.TOTAL COSTS                  18048600.   19279578.   20343794.   21388684.   22543430.   23772120.  125376207.


 PROFIT
 ------
22.GROSS PROFIT                  1951400.    1720422.    1656206.    1611316.    1456570.    1227880.    9623793.
25.INCOME TAXES                        0.          0.          0.          0.          0.     395421.     395421.
26.NET PROFIT                    1951400.    1720422.    1656206.    1611316.    1456570.     832459.    9228372.
                                **********  **********  **********  **********  **********  **********  **********
```

SCENARIO : FINANCIAL PLANNING FOR WHOLESALE TRADE

REPORT 3 : CASH FORECAST (IN MIO. FR)

	INITIAL	JANUARY	FEBRUARY	MARCH	APRIL	MAY	JUNE
11.RECEIPTS	-	0.000	10.000	20.500	21.500	22.500	23.500
9.CASH FLOW FROM PAST	-	3.000	4.000	3.500	0.250	0.000	0.000
15.PAYMENT MATERIAL	-	3.152	9.807	13.844	14.938	16.085	17.288
5.FIXED COSTS	-	5.500	5.500	5.500	5.500	5.500	5.500
20.PAYMENT INTEREST	-	0.000	0.000	0.159	0.000	0.000	0.024
25.INCOME TAXES	-	0.000	0.000	0.000	0.000	0.000	0.395
27.CASH FLOW MONTH	-	-5.652	-1.307	4.497	1.312	0.915	0.292
29.CASH BALANCE	1.000	-4.652	-5.959	-1.462	-0.150	0.764	1.057
30.BANK LOAN	-	4.652	5.959	1.462	0.150	0.000	0.000

31.NET PRESENT VALUE -0.184 IN REFERENCE PERIOD 0, DISCOUNT RATE OF 1.5 % PER MONTH

USING CHARACTER MODE EFFICIENTLY

BY J. VAN DAMME SYDES N.V.
A. GOSSETLAAN 30A
1720 GROOT-BIJGAARDEN
BELGIUM

ABSTRACT:

The discussion on whether to use block mode or character mode terminals often stops on the difficulty to implement screen design in character mode.

This lecture presents a solution to this problem that is both easy to implement and computer-resource friendly.

This text is prepared and printed using FSEDIT ((C) SYDES N.V.)

COMPARING BLOCK MODE VERSUS CHARACTER MODE

Whether to choose block mode or character mode depends largely on the type of application. In general the following advantages and disadvantages of both systems should be considered:

- BLOCK MODE ADVANTAGES:

  . only one response time per screen transaction
  . all terminal features can be used (such as 'delete character')
  . the process, handling the terminal can be swapped out during input, while in character mode the process has to be present after entering each field.
  This may lead to excessif swapping on overloaded systems.

- CHARACTER MODE ADVANTAGES:

  . each field is immediately processed as it is entered. This improves the interactivity of the system.
  . each field has to be handled only once, while in block mode, the full screen is handled each time an error was entered on a field.
  . features, not available on the terminal can be emulated by software.
  . disc accesses to read data from disc can be made concurrently with terminal output.

THE CLASSICAL IMPLEMENTATION

In a classical character mode implementation, the screens used are not or almost not formatted. This is because formatting is to be done by the application program itself, using escape sequences and keeping track of the row and column positions of each field.

Each field is tested against tables and/or files as it is entered, but once a field is accepted by the program, it is not possible to correct that field.

Error messages are often intermingled with the input an output.

If a screen has to be slightly redesigned - say a line has to be inserted - then a lot of code must be modified in order to reflect the new positions of all the subsequent fields.

THE SYDAID IMPLEMENTATION

SYDAID uses a new approach towards implementation of interactif programs.

An application is seen as a set of screen formats. Each screen format is composed of one or more fields and each field has a set of specifications. These specifications indicate what has to be done when that field is entered.

The designer first creates a screen format by drawing it on the screen using the standard MPE EDITOR or any other text editor. Subsequently she adds a one-line specification to each line of the format indicating the start position of the fields and the screen enhancements

Then each field is further specified:

- comparing the field to other fields, values or tables
- looking up and modifying data-bases and files,
- modifying the sequence in which the fields are shown.
- manipulating data by computation or concatenation.
- issuing error and warning messages under certain conditions.
- modifying the sequence in which screens are handled.
- dynamicaly using the programmable function keys.

In most cases no further programming in COBOL or any other programming language is necessary, but still a powerfull and easy to use interface was designed to communicate with any host language.

A screen, designed with SYDAID, is totally independent of the external data-base. This means that the data-base may be restructured in several ways whithout modifying or even re-compiling the screen.
These structure changes include:

- changing a fields type or length.
- changing a master data-set into a detail data-set.

THE SYDAID IMPLEMENTATION

The terminal user of a screen, designed with SYDAID, has the best of character mode and block mode available:

- The user can move the cursor forward and backward over the fields.
- fields can be modified using special features including character deletion and insertion.
- Real interactivity, on a field by field basis.

15 IDEAS ON IMPROVING MPE SECURITY

NORMAN B. WRIGHT
U.S. OFFICE OF PERSONNEL MANAGEMENT

A few years ago, when the number of Hewlett Packard 3000 sites was somewhat less than one thousand, it used to be sufficient to put a few passwords and lockwords on key accounts and files. We could then take refuge from our worried management behind the mythical "technically knowledgeable user". "The system is secure," we would say, "except from the technically knowledgeable user who is intent upon breaking its security". For many installations, this provided a moderate degree of safety. We could be relatively certain who the few technically knowledgeable users were who would be capable of breaking security. We could also take steps to assure that these users were not maliciously intent on circumventing security. At worst, we could keep a very close eye on them.

No more! The user community at most Hewlett Packard 3000 sites has outgrown the ability of one system or security manager to be personally in touch with each member. Furthermore the sophistication and knowledge of even casual users has now grown to such a point that very few of us can take refuge in the myth of the "knowledgeable user". Most users can be assumed to have had previous exposure to computers, and to be in some degree acquainted with operating systems and utilities. The widespread use of microcomputers is proliferating this knowledge to a point where most of us have users who are not professional programmers, but who nonetheless know enough to attempt disk dumps, system crashes, and security breaches of considerable ingenuity. Since the movie TRON, every system can be said to be fair game for this sort of attempt.

The following ideas are offered, not as an exhaustive checklist of security measures, but as a list of workable ideas which you may wish to consider in setting up or improving the security of your HP3000 installation.

1. Establish control over the physical security of the computer itself. While the advent of the minicomputer brought a breath of fresh air to the large "closed shop" environment, the growth to "super" minis has brought us back full circle. We have met the enemy and he is us. Most HP3000 installations now deal with information which is far too valuable or sensitive to afford the luxury of the "open shop". At a minimum the computer, and its tape and disk library should be in a secured environment with only those persons absolutely required for its operation able to enter.

2. Appoint a security manager. Have this person spend a certain amount of time thinking about security each month, in proportion to the amount of potential loss at stake. One of the key points in your security program should be that it is always changing, and continually improving. The security manager should carry on a continuing risk analysis, pinpointing current vulnerabilities of the installation. He or she should be inventive enough to consider all potential motivations: financial gain, malicious sabotage, corporate embarassment, and mischievous fun. Your best source of what is vulnerable on your system will always be your own in-house technically knowlegeable users. Keep them thinking regularly about security problems on your system. There are always going to be holes and weak points. Concentrate on the ones that are the most obvious with the highest potential loss to the installation.

3. Make your personnel security conscious. Make certain that they understand the sensitivity of certain data and are following established procedures in dealing with it. The greatest security risks, of course, involve your own personnel who must have day-to-day contact with sensitive or valuable information. Fortunately, this is also your first line of defense. Make certain that the need for security is known and understood by all employees. Check frequently to see that established procedures are being followed, not being pushed aside in the crush of day-to-day business. Make sure that your employees feel free to report even accidental or casual security violations to the security manager.

4. Establish manual or automated cross-checking procecures for information which is particularly valuable or sensitive. As with money, it is usually better to have at least two people involved in the handling of sensitive data so that collusion between them would be necessary for fraud or theft to be perpetrated.

5. Pay particular attention to the movement of magnetic tape, disk, and other media. Regardless of how elegant and effective your online security techniques might be, they could always be rendered useless by the theft of a single system dump or backup tape from your installation. The only way to protect against this (short of data encryption) is to establish very tight controls on the removal of such media. Dump tapes in particular may need to be kept under lock and key and bulk erased after they have expired. If you have tape or disk media which are routinely shipped or taken from the site, you may want to establish a program of cross checking their contents. At any rate, insist upon accurate logs for all information on magnetic media which leaves your computer room, including a record of what was taken, who took it, where it went, and for what purpose.

6. Store sensitive data separately. Due to the storage and handling problems with dump tapes, you may wish to consider backing up and storing particularly sensitive or critical data seperately from your SYSDUMP procedures. Backup media for the sensitive data can then be subjected to additional cross checking, perhaps even placed in custody of someone who will take overall responsibility for its security. Since it is on independent media, it can be placed under seperate lock and key, and purged from the system prior to all SYSDUMP procedures. If you wish to make doubly sure the data is destroyed from the disk, overwrite it instead of purging it. The program "BLATFILE" in the User's Group Library performs this function.

7. Lock up the key capabilities of the system and check them frequently. It is well known on the Hewlett Packard 3000 system that users with privileged mode capability (PM) or with system manager (SM) can easily break almost all security mechanisms. Reserve the use of these capabilities to a few users and make certain that extra precautions are excercised over them. If your system account structure is highly volatile, you may wish to set up auditing procedures to check, at periodic intervals, to make certain that these capabilities have not "leaked" out to other users. Privileged mode is notorious for doing this since it is also, by some quirk of MPE, required for restoring data bases. The CS capability for using the distributed systems lines, is another one which you should consider restricting if your installation uses this facility.

8. Use the MPE password system or a good alternative. MPE password protection at both the account and user levels has some excellent advantages, if used correctly. Making your passwords randomly generated strings of letters and numbers affords a measure of increased security which is highly recommended. You should plan on changing passwords periodically, at irregular intervals, perhaps to coincide with the departure of key personnel such as programmers or operators. Remember that these personnel frequently gain privity to passwords other than those authorized to them. Using MPE's double password system allows you to change the global account passwords and leave the user passwords the same. Changing passwords has the twofold advantage of requiring the security manager to keep up to-date-records of the user population and requiring the user population to keep close touch with the security manager. Users who no longer have current need for access to the system, but who have failed to notify the security manager, will be automatically excluded by these periodic changes.

9. Get the passwords out of your streams. In order to change passwords easily and painlessly, you must develop methods for removing them from job stream files used as a regular part of development and production. There are a variety of packaged programs and utilities available to help with accomplishing this. They include the extended stream facility of Vesoft's MPEX, and at least two programs available free in the Users Group Library -- JES and STREAMER. All of these programs depend upon programmatic insertion of the passwords into the job stream file before it is streamed. The password is usually obtained from a password file or from the system itself at run time. If one of these packages does not have enough flexibility for your installation, it will pay you to write a simple one yourself. The requirement for passwords in the job stream file is a major security problem in MPE which will also make changing passwords regularly a forbiddingly burdensome task.

10. Use the LOGON,NOBREAK UDC to control users. This is particularly applicable for users who are dedicated to only a few different functions on the system, such as payroll or inventory clerks. A properly constructed UDC can tightly restrict what such a user could do on the system, allowing him or her to access only those functions which are authorized. The UDC can be set to automatically log off the user upon completion of the specific function. Setting the UDC on an account-wide basis (;ACCOUNT) will alleviate the time-consuming task of having to log onto each newly created user id. For those few users whom you wish to allow privileged access on the account, you can set UDC's with an overriding command at the user level which will deactivate the LOGON,NOBREAK facility.

11. Consider writing your own security screening program. This program could be used in conjunction with a system or account-wide UDC to check for a variety of user defined security violations. Many installations may wish to restrict certain users to specific terminals (logical devices), or to specific time periods during the day or week. In designing such a program you may find the seldom-used LOCATTR attribute of the user id useful for futher screening and restricting user capabilities. A user-written security screening program can also do additional password or protection prompting, and logging for installations where several users are using the same user id. An interesting example of this type of program is found in the KMGR program in the Users Group Library which provides extra logon security for privileged accounts. A security screening program, coupled with the LOGON UDC provides good capability for customized sign-on protection.

12. Consider disabling the :LISTF command entirely. MPE's :LISTF command has been faulted frequently because it gives all users the capability of listing the file directory contents of the entire system. Thus, for example, when the user sees the program DISKED2.PUB.SYS, the temptation to experiment can prove almost overpowering. Particularly on college campuses, where some of the most severe security problems of this kind exist, locking up the :LISTF capability and a variety of other capabilities based on it (LISTDIR2, PURGEFILE, etc) seems to be a good precaution. Alternative commands for listing the user's group and account fileset can, of course, be provided. Almost any use of the UDC in a security program, it should be added, will mean that the :SETCATALOG command itself will also have to be disabled at the user level. Otherwise the user will easily learn to circumvent the UDC commands with an overriding UDC. A good program of system and account UDC's can frequently alleviate the need for numerous user UDC's anyway. You may wish to abandon the many problems of MPE's UDC facility entirely, in favor of a programmatic capability designed as an integral part of the security screening program.

13. Consider using private volumes to enhance your security. Much has been written about the use of the private volume capability in terms of increased data handling flexibility and backup. However, private volumes also provide one of the best methods for tightly restricted access to key data on the system. Data such as a payroll account or other critical information can be physically removed from the system when not in use on line. When the information is required on line, it is protected from unauthorized tampering by the necessity of UV (use private volumes) capability which can be granted only to the authorized users.

14. Program security into your applications. Regardless of what measures you take to restrict access to the system, you are also going to have to protect against the inside job -- the authorized user who uses the data in unauthorized ways. As we mentioned earlier, your best line of defense will always be other personnel and system cross checks designed to prevent this. However, most users find they must also look at the applications programs or packages themselves to further identify restrictive mechanism which can be implemented. A key feature to all sensitive applications should be some form of logging facility to track what transactions were made and by whom. The logging capability could be a built-in one, such as IMAGE's transaction logging, or it might be one which is designed into the application. Logging capabilities frequently serve a variety of useful functions in addition to the security function.

15. Establish a vigorous random auditing program. Your entire security edifice will collapse without constant monitoring to determine if and when security breaches are being attempted. The security manager should bring into play everything that he knows about the system to periodically monitor activity. Use the log files and programs which manipulate them (LISTLOG2, READLOG, CENSOR, etc.); use online monitors (OPT3000, SOOIV); use programmatic or manual checks on other logs such as DBAUDIT for IMAGE data base logging, or monitoring your own set of logs from security or transaction screening programs. The auditing program should check for application-defined "unusual", "excessive", or "special" conditions. Try to make the programmatic definitions of these terms parameterized so that they can be varied. One crucial element of the auditing program is that it must be constantly changing and improving. As quickly as one check or audit is permanently installed and performed on a regular basis it can be assumed that another way will be found to circumvent the existing checks. Only by constantly changing and improving the security system faster than the sophistication of your average user -- a sophistication which is itself constantly increasing -- can you hope to offer any assurance of a secure system.

```
 ################ h  #################
#############    h         ############
##########      h            ##########
#########      hhh    ppp     #########
#########     h   h  p   p    #########
#########    h   h  p   p     #########
#########   h   h  p ppp      #########
##########        p          ##########
############     p         ############
 ############## p ####################
```

COMPUTER SYSTEMS DIVISION
Accounting Systems Group
19447 Pruneridge Ave.
Cupertino, Ca. 95014
(408) 725-8111 x 4348

Mike Pechulis, Senior Programmer Analyst
****

SYSTEM SECURITY IN AN OPERATORLESS ENVIRONMENT

A B S T R A C T

The MIS group supports the accounting function within the Computer Systems Division of Hewlett-Packard. We have had an operatorless, multi-machine, networking environment for three years. This paper will discuss various techniques we have used to enhance our system's security and integrity. Topics presented will include:

MONITOR
Complete system security, application control and friendly user interface in a single online program.

PRIVATE VOLUMES
How private volumes can be used to insure data integrity.

DATA BASE AUDIT
An online program for analysis of Image log files.

GETBASE
A program which provides a secure interface between applications and data bases in a networking system.

OMNI/3000
A program from the contributed library which has been modified to regenerate your account structure and list the attributes of specified accounts, groups, users, and files.

B A C K G R O U N D

The MIS group handles all accounting data processing within the Computer Systems Division (CSY) of Hewlett-Packard. Our role within the accounting department is two-fold:

1) we support and develop computerized accounting systems.
2) we test many of HP's new hardware and software products. This includes pre-release testing for reliability and functionality.

Our philosophy centers around the use of HP software and hardware in a "distributed" data processing environment; an environment in which the computing power is where the people and the problems are located. This includes addressing the problems of operatorless computers and system security.

Currently, our local network of three HP3000 systems fits into a larger CSY network of seven HP3000 systems in three different locations. The three local systems (a Series 64, 44, and a 40SX) are linked together by the X.25 Protocol of DS3000. All three machines are running on the MPE IV Operating System. The Series 64 handles our cost accounting and program development, the Series 44 handles our general accounting, and the Series 40 is dedicated to the GA/3000 Accounts Payable module. The Series 40, along with its accompanying disc drives and terminals, is located directly in the user's area. Not only do the users handle their own software applications, but they also handle many of the operator's tasks such as backup.

We have seventeen disk drives spread across the network for a total of 2200 Mb of disc storage. On each machine, there is only 100 Mb of system domain. The remaining space consists of private volumes. These include four HP7935H 404 Mb drives currently being alpha-tested. One 2619A does the printing for three machines by using the DS/DSN X.25 Protocol to copy spoolfiles from the Series 44 and 40. Our systems group of ten professionals supports an accounting department of 40 people.

M O N I T O R

With today's emphasis on distributed processing, computers and their associated peripherals are being placed in open user environments. Here there are no locked doors for security or experienced operators to help monitor the system. In this environment, the system itself must be used to secure its own data and Accounting Structure.

Our approach was to develop a secure, network interface between the computer user and the MPE operating system. We called this interface "Monitor" because it controls what a user can access. Each Monitor controls a single account and provides access to the groups within that account. When a user logs on to a group, he is automatically captured by Monitor and is presented with a menu screen of options. Each option runs an online application program or streams a particular batch job. The menu of options available to the user depends on the group or user/group log-on. After successful completion of an option, control is returned to Monitor and the user is prompted for another input.

Applications are run through Monitor so that no user, except systems personnel, ever needs to get to the Operating System (see figure 1.0). To stream a batch job or run a program online no longer requires a user to know all the necessary file equations and MPE parameters. The local network Monitor data base automatically provides this information to the system.

Depending on the user log-on, Monitor can automatically issue additional file equations, remote log-ons, or other environment set-ups required before a certain application can run. Each Monitor option can also request a sequence of multiple commands. This is helpful when an online application needs several different file equations in order to execute successfully. A help facility which explains the Monitor options is available for each group log-on.

Monitor provides the system with some additional security features:

Log-ons can be selectively disallowed by account, group, user, or combination.

Besides being able to place passwords on each group, the user can also customize his own password for each Monitor option.

All passwords kept in the Monitor data base are encrypted for added security.

The NOBREAK option and the Job Control Words (JCW) within the User Defined Command (UDC) prevent unauthorized termination of Monitor by logging the session off.

There is no need to use Query to access and update the Monitor data base. Maintenance and reporting is done through an online, View-driven interface program. For audit purposes, the program generates a report listing all the capabilities, UDCs, commands and access flags associated with each group and user.

Monitor is independent of any logical device and can be run in non-block mode from any terminal. This system can be installed on an HP3000 running stand-alone or one linked to other HP3000 computers in a network.

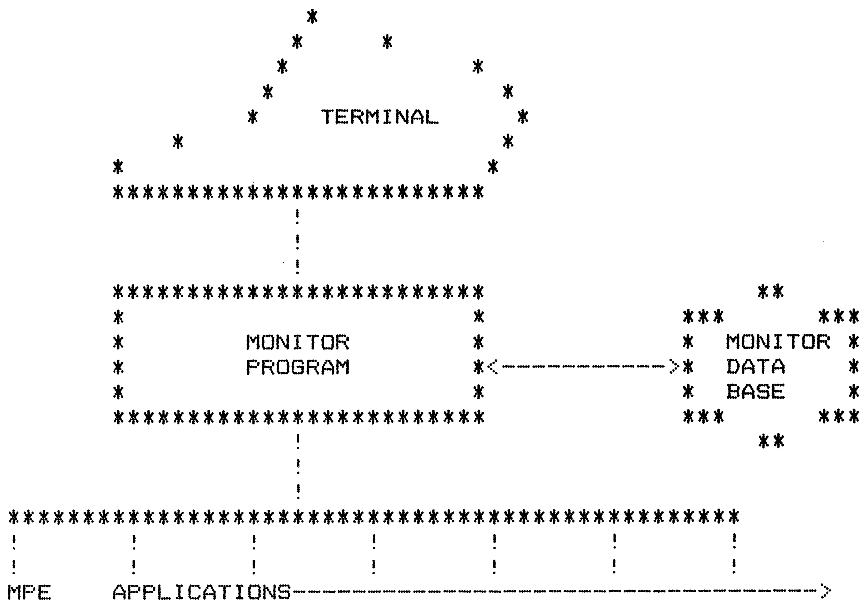


Figure 1.0 Monitor Overview

P R I V A T E   V O L U M E S

A private volume is a removable disc pack whose data does not reside in the system domain. A serial disc is a type of private volume which, like a tape, is a serial device. In moving to a multi-machine, networking environment, private volumes have proven their worth in several key areas.

Since our group functions as both an experimental test center and an accounting production center, we protect our data from system failures (hardware or software) by keeping all of our accounting application files on private volumes. The system domain consists only of the SYS account, spooler space, the system directories, and the operating system. During our regular backup, we create a Cold Load tape which contains only MPE and the files in the SYS account. Since major system failures almost never affect information on private volumes, a Reload in our environment simply means restoring the latest Cold Load tape. We have reduced our Reload from an average of nine tapes to only one tape and our Reload time to 30 minutes. In addition, we never lose any user-inputted data. When we were alpha-testing the Series 64, we "tested" this concept between 30 and 40 times over a five month period. Not once did we lose any accounting data.

Partial system dumps that used to be done to tape are now done to private volumes (serial discs). Each HP7925 disc pack can hold about three 1600 bpi tapes worth of information and an HP7935H disc pack can hold more than ten. By using serial discs, we have reduced our backup time by a third. Not only are multiple tape mounts eliminated, but also private volumes do not suffer from missing load points on HPIB tape drives or from parity errors that frequently plague older tapes.

Processing that used to go to tape now goes to private volumes. We gain the advantage of fast disc access in combination with the same ease of removability and storage of tapes. By having data files on a private volume, the production jobs control the file access. It is no longer possible for an operator to mount the "wrong" tape and thereby write over important data. If the wrong private volume is mounted, the program will not be able to access the file. No important data will have been over-written and lost. With multiple private volume disc packs, we also gain a great deal of flexibility. In our environment, we run online applications during the morning and afternoon, a system backup at lunch, and batch processing at night. Over the course of one day, one disc drive may have had several different disc packs. During down-time on one system, private volumes can be moved to another machine with a similar accounting structure. This enables us to move our development and testing files off of the downed system and onto another machine in our network.

D A T A B A S E   A U D I T O R

The need to maintain both the security and integrity of data bases is crucial in any production environment. Data security centers on the ability to determine who accessed a data base, when this access took place, what was done, and what program was used to permit the access. Data integrity requires the ability to completely and accurately recover a corrupted database.

In terms of data security, the current DBRECOV utility has only limited auditing capability with no flexibility for specifying data parameters such as who, when, and what program. The user should be allowed to set specific search values on these parameters. With these capabilities, full auditing control is maintained over the databases.

To insure data integrity during a normal recovery, the DBRECOV utility must go to the beginning of the log file and work its way through to the end. There are only limited allowances for partial recovery. Partial recovery should give the user the option of specifying multiple start and stop times (ie. time windows). In the present system, partial recovery is enabled only to the extent of starting at the first record of the log file and then stopping at a given time stamp. A system that would allow partial recovery plus the examination of IMAGE commands in the log file would allow a more complete and accurate recovery of corrupted data bases.

Our Database Audit system gives a user increased flexibility in both auditing and partial recovery. A user can now selectively examine database log files and then do a partial recovery with multiple time windows. This examination looks at part, or all, of the characteristics of the file (who accessed it, what they did, when they did it, and what program was used). The parameters that the program uses are input in the form of commands. In batch mode, these commands can also come from an editor file.

The output of the system will depend on the application chosen. In the audit mode, a report will be generated with the information requested by the user. The user will also have some flexibility as to the format of the report. The standard output device for this report will be the system line printer. In the partial recovery mode, the output will be a disc file that contains log records (see figure 2.0). These records will be exact copies of the records from the master log file, chosen according to the criteria given by the user. This file will be used for recovery in place of the master log file.

```
                      *
                     *     *
                    *           *
                   *              *
                  *    TERMINAL    *
             *                    *
         *                       *
         ************************
                     !
                     !
                     !
         ************************          *****************
         *                      *         *                *
         *       DATA BASE      *<-------*    MASTER     *
         *     AUDIT PROGRAM    *         *   LOG FILE    *
         *                      *          *               *
         ************************          *****************
                     !
                     !
                     !
          *****************************************
          !                                      !
          !                                      !
         \!/                                    \!/
*********************                       ******************
*                  *                       *                *
*   AUDIT         *                       *    NEW         *
*   REPORTS      *                         *   LOG FILE     *
*           *                               *                *
*********                                   *******************
```

Figure 2.0   Audit Program Overview

G E T B A S E

With network-oriented systems, data bases can be spread across various groups and accounts on several different machines. Because of the dynamic nature of a network, it is not desirable for each application to have hard-coded data base access routines. This would considerably decrease the flexibility of the application. Such an approach requires hard-coding the fully-qualified base-name, the data base password, and the access mechanism for remote machines. Application programs should only have to supply the data base name and then let a network program determine the location of the database (group, account, and machine), set-up the access path, and provide the necessary password to open the data base.

To accomplish this, we developed a subroutine called Getbase which acts as an interface between an application program and a network. Each machine in the network has a local data base which contains all the necessary information for opening data bases on all machines (see figure 3.0). A call to Getbase by an application program does the following:

determines the location (name, group, account, and machine) of the target data base.

opens up any needed dslines.

logs-on to the remote computer containing the target data base.

provides the file equations necessary to point the application program to the correct data base.

returns to the calling program the correct data base name and password.

Maintenance for all the local Getbase data bases will be provided by an online block-mode program. This program allows you to add, delete, or update information concerning your production data bases. The user can change the location of a data base (group, account, and machine), the dslines to each machine, and the passwords. Data base passwords are all encrypted for added security. The maintenance program runs on the host computer and can be used to update all of the remote data bases simultaneously.

By centralizing our passwords in one database, we have gained both flexibility and security in our data base network.

```
*************************
*                       *
*      APPLICATION      *
*        PROGRAM        *
*                       *
*************************
          /!
           !
           !
           !
          \!/
************************              **
*                      *           ***     ***
*        GETBASE       *           *  GETBASE *
*        PROGRAM       *<---------->*  DATA     *
*                      *           *  BASE     *
************************           ***     ***
                                      **
```

Figure 3.0   Getbase Overview

O M N I / 3 0 0 0

One of the primary responsibilities of a System Manager is to monitor the attributes of his account structure (MPE capabilities, access, private volume name, passwords, etc...). This insures that no unauthorized changes have been made to the system. In addition, the system directories must be maintained by adding or deleting certain groups/accounts off of one Private Volume and onto another. To gather the necessary information to accomplish these tasks, a System Manager must go through the LISTDIR2 utility and individually check each group, user, and account. When dealing with even a few entries, the tedium of the job makes it easy to bypass a security risk. Also, when information has to be taken off of one private volume and put on another, the System Manager must manually recreate the account structure for the new disc.

Our approach was to develop OMNI/3000, an automated way of helping System Managers maintain and monitor their account structure. It can be run in either batch or online mode and will generate reports for either the terminal or line printer. The program has three main functions: one function recreates the account structure of either a private volume or an entire system, the second function reports on the attributes of specified accounts, groups, users and files, and the third function compares the account structure found on the system with a master copy found in a database and makes any appropriate corrections. (see figure 4.0).

When recreating the account structure for a private volume, OMNI/3000 reads in the old account structure and creates a job file containing the appropriate ALTACCT, ALTGROUP, and ALTUSER commands. The user need only input the new private volume name and then stream the newly created job file. For use on a "null" system, OMNI/3000 can also be used to generate the entire account structure of a machine.

OMNI/3000 allows the user to scan an account structure for certain values and then print out the results in a clear, concise format. In addition to the search criteria provided in the LISTDIR2 utility, a System Manager can also ask for the following information:

> Find all accounts, group, users, with System Manager, Account Manager, or Privilaged Mode capability.
>
> Find all accounts and groups on a private volume, even if the private volume is not mounted.
>
> Find all files with a percentage utilization of less than "X" percent.
>
> Find all files greater than "X" sectors.
>
> Find all groups and accounts greater than "X" sectors, or "X" cpu seconds, or "X" elapsed time.
>
> Find all groups and accounts near their limit of disc space, CPU seconds, or elapsed time.

Instead of LISTDIR2's block format, OMNI/3000 uses a print format similar to MPE's ":REPORT" command. The groups, users, and accounts are printed down the page and the attributes (ie. capabilities, access, passwords, etc...) are printed across the line. This allows the user to quickly scan the data for any discrepancies.

The OMNI/3000 data base holds all the attributes of each user, group, and account on the system. The program compares the attributes in the data base with the attributes on the actual system. The program uses the database as a "master" copy of the account structure. When there is a difference, the program generates a listing of the error and then automatically corrects the problem. This data base is maintained by an online, block-mode program.

```
                      *
                     *     *
                    *            *
                   *               *
                  *    TERMINAL     *
             *                     *
         *                        *
         ************************
                    /!
                     !
                     !
                     !
                    \!/
         ************************                 **
         *                      *            ***       ***
         *      OMNI3000        *            * OMNI3000 *
         *      PROGRAM         *<---------->*   DATA   *
         *                      *            *   BASE   *
         ************************            ***       ***
                     !                            **
                     !
                     !
          ****************************************
          !                                      !
          !                                      !
         \!/                                    \!/
*********************                       ******************
*                  *                      *                  *
*   MAINTENANCE   *                      *   ACCOUNT        *
*   REPORTS      *                        *  STRUCTURE       *
*           *                              * JOB FILE        *
*********                                   *******************
```

Figure 4.0 OMNI3000 Overview

SECURITY/3000: A NEW APPROACH TO LOGON SECURITY

by Eugene Volokh,
VESOFT Consultants
506 N. Plymouth Blvd.
Los Angeles, CA 90004 USA
(213) 464-7523

ABSTRACT

With the advent of computers, a new age in information processing dawned. And, together with it, a new threat to information security was born -- the threat of computer crime. This paper will tell you why you should buy SECURITY/3000, VESOFT Consultants' answer to at least one part of the security problem -- logon security.

THE PROBLEM

Security is the art of restricting access to certain entities. One of the fundamental aspects of all security systems is accessor identification, i.e. determining who the person who wants to access a given entity really is. In computer security, this form of security is called logon security. Once this level of security is passed, the accessor is no longer Joe Smith, but rather JOE.PAYROLL (with the underlying assumption that JOE.PAYROLL really is Joe Smith). If John Doe can sign on as JOE.PAYROLL, the computer will still think that he is Joe Smith, and will let him do anything that Joe can do, regardless of any further levels of security that exist on the system. Thus, if your logon security system is inadequate, no additional layers of security can help -- your system is wide open to any would-be computer criminals.

Thus, a good logon security system must ensure user id integrity, and, if a violation is detected, it must take proper measures to inform appropriate people (e.g. the system manager and the console operator) of the violation. Furthermore, it must protect against internal tampering with the security system by maintaining a clear audit trail of all security modifications. And, it may also be of benefit for it to prevent user access at times and from places in which it is easiest to penetrate system security (e.g. on weekends, after hours, over telephone lines, etc.).

Unfortunately (for you, but fortunately for us vendors), HP's logon security system does not meet the above requirements. No time of day, day of week, or terminal number security is provided; no audit trail of password removals, additions, or modifications is kept; only the console operator is informed of any violations (via a message that is virtually indistinguishable from a number of other messages sent to the system console); and, as we will demonstrate below, user id integrity is easily compromised.

In order to ensure user id integrity, HP's logon security system uses passwords. However, these passwords provide only an illusion of security, because:

* There is only one password for each level (user, account, or group) of security. Thus, knowing this one password guarantees that you can penetrate that level of security.

* Many people treat passwords as a dispensable nuisance, and thus readily reveal their passwords to unauthorized people. To a person who does not perceive the security value of a password, it is much easier to tell someone the password rather than to question whether that someone should really know it. Similarly, people have no qualms about writing passwords down if they have trouble remembering them (in one case, the user actually wrote the password down and stuck it to her terminal!).

* Passwords are stored in the system in clear text. Thus, they may be readily found in job streams, discarded LISTDIR2 listings, on :SYSDUMP tapes, etc.

Thus, if you are relying on HP's logon security system, you and your information can easily fall prey to computer crime.

THE SOLUTION

USER ID INTEGRITY

Instead of conventional passwords, SECURITY/3000 user personal profile passwords -- answers to personal questions such as "WHAT IS YOUR MOTHER'S MAIDEN NAME?", "WHAT CITY WAS YOUR GRANDFATHER BORN IN?", etc. (If you don't like our questions, you can configure your own.) Instead of asking the same question all the time, SECURITY/3000 asks a random one out of a number of questions (up to 30, user-configurable). And, instead of keeping the answers stored in clear text, it encrypts them using a special one-way encryption system, through which the answers can not be decrypted by anybody.

Thus, passwords are automatically imbued with a psychological security significance; knowledge of all passwords is required to be sure of being able to access the system, even though the user is asked only one at logon time; and, passwords are made impossible to determine. Thus, SECURITY/3000 avoids the disadvantages of HP's logon security system.

VIOLATION REPORTING

Unlike HP's logon security system, which reports security violations only to the system console, SECURITY/3000 reports them to the system console (in inverse video, to distinguish them from ordinary console messages), prints a user-definable memo to the system line printer, and logs them to its own log file for future reference (thus providing a permanent record for further interrogation). This "three-alarm system" makes sure that attempted security violations are acted upon, not ignored.

AUDIT TRAIL

Although an account manager should be able to add, change, or remove user security within his own account, there must be some means provided to keep track of his actions. Under HP's logon security system, an account manager can create a fictitious user id, log on to the system under it, and do something that he shouldn't be doing without being afraid of getting caught. With SECURITY/3000, all user additions, changes, and deletions are logged to the SECURITY log file, thus allowing an auditor or system manager to determine who created, altered, or removed a given user id.

TIME OF DAY, DAY OF WEEK, AND TERMINAL NUMBER SECURITY

Most security violations will not occur on Tuesday at 2:30 in the middle of your payroll department; they will most likely be done in the dead of night on a weekend across a telephone line. If your payroll clerks work only on weekdays from 9 to 5 on terminals 31, 32, 33, and 35, any attempt to access the payroll account at any other time from any other place is inherently a security violation. Does HP's logon security system protect you against this? No. Does SECURITY/3000? Yes.

In short, SECURITY/3000 gives you what HP's logon security system doesn't give -- security.

CONCLUSION

If you are an average HP shop, you have tens of millions of dollars flowing through your computer. If you want to keep those tens of millions of dollars, what you need is SECURITY/3000.

Practicality in the Design of Software Testing Tools

Gary E. Marcos
Information Networks Division
Hewlett-Packard
Cupertino, California

Most software testing tools do not recover their initial costs. They are built during the worst part of the product lifecycle, the testing phase, where expectations are high and pressure to release the product is great. This causes two significant problems: (1) An attitude that software testing tools are a poor investment because they are poorly designed, undocumented, unmaintainable, unextensible and short-lived. And (2), that the choice of testing tool is a difficult decision to make, with its benefits being unclear. These problems can be overcome by designing software testing tools early in a product's lifecycle. By doing so the investment will be recaptured many times over.

This paper addresses the successful design of software testing tools. The design takes place in three steps. The first step, planning testing objectives early, identifies resources to be allocated to the testing effort. It also sets the objectives of the testing effort. The second step, determination of the feature set, identifies the features that the software testing tool must have in order to recapture its investment. Features such as ease of use, maintainability, and extensibility are discussed. The third step, determination of services of the software testing tool, identifies the area of testing that will be enhanced through the tool's use. Typically, software testing tools provide services in those areas of testing that can be automated, such as functional testing, reliability testing and performance testing. Two case studies are provided when the discussion is complete.

TESTING and SOFTWARE TESTING TOOLS

Software testing provides many crucial services designed to reduce costs and ensure end-user satisfaction. For example, costs are reduced by identifying software failures within the lab rather than in the field, where failures are significantly more expensive to identify and resolve. To find and fix a software bug in the field can be fifteen times more expensive than to find and fix it in the lab [Daly77]. End-user satisfaction is increased by delivering a product that meets the design objectives and is of high quality. This can account for an increase in sales by establishing the reliability, quality and overall image of the product [Miller81]. Software testing tools are mechanisms of applying the advantages of software on itself. Thus the benefits of automation, cost reduction, are applied to the testing process. Manual labor is reduced, the time in the testing cycle is reduced, test cases become easily repeatable and testing is more thorough. The final results are increased engineer productivity, lower costs and a higher degree of quality. The need for automating the mechanisms of testing stem from the fact that much of the testing effort is mechanical. There is the repetitive cycle of running tests, determining test success or test failure and generation of test cases. Manual testing is error prone, hard to duplicate and time consuming. Through automation, software testing tools can make significant contributions to the mechanics of testing.

THE PROBLEMS

Unfortunately, software testing tools have acquired a reputation as a poor investment. This has held back the progress of producing quality software due to the lack of such tools. The primary reason that software testing tools are viewed as a poor investment by managers and engineers alike stems from the fact that most software testing tools are built and not designed. At some point during the coding phase, debug phase or testing phase the software engineer (whose primary responsibility is to get the product released) determined the need for a testing tool and has built it. The tool may aid in the the product's release but the investment in the tool itself may be lost. For example, expendiblity is often equated with software testing tools; the tool is useful in a particular environment, the environment changes slightly and the tool is no longer useful. Or, the few people (or person) who knew how to use the tool are no longer available and the new engineers avoid the tool due to it's complexity or lack of documentation. These are experiences which do not have much visibility during product development and product release. However, when resources are requested, these experiences are recalled and give rise to the philosophy that investing in software testing tools is a poor investment.

A well recognized problem in the software community is our present inability to quantify software quality. It somehow manages to elude our best efforts. This problem is also a major obstacle in the acceptance of software testing tools. Managers are forced to weigh the cost of the software tool against the intangible benefits derived from the tool's use. This problem does not prevent us from recognizing the value of software tools which enable us to build even more software, for example, compilers, editors, cross-referencers, cross-assemblers and symbolic debuggers. It should not be a problem in recognizing the value of software testing tools. Just as automation is being applied to the production of software through the use of compilers and editors the same process of automation should be applicable to the testing process.

THE SOLUTION: DESIGN

Testing takes place in the phases of a product's lifecycle that account for seventy-five percent of the product's cost. Twenty-five percent occurs during the testing phase prior to product release and fifty percent occurs during the maintenance phase after product release [Myers76]. In order to recover the costs of building software testing tools the tools must be designed to survive passage from the testing phase into the maintenance phase, where we have the most to gain by reducing manual labor and improving the quality of the testing effort. Myers states in his book Software Reliability Principles and Practices, "The best way to dramatically reduce software costs is to reduce the maintenance and testing costs." [Myers76]. In order to survive the passage, the software testing tool must be designed to meet the needs of not only the testing phase but also the needs of the maintenance phase. This is actually quite an easy task, as there are many similarities between the two phases. They are related through the fact that adding enhancements to the product or fixing bugs both require that the product be tested again. Preserving the usefulness of the testing tool over the lifespan of the product might appear to be obvious. However, it is a perspective that is usually lost in the pressure to release the product. Typically, the scope of software testings tools has been to aid in the certification and release of the product. The effect of this approach is to build a tool very narrow in scope and one whose usefulness after product release is minimal. The key to designing successful software testing tools is not to forget that 50 percent of the costs associated with the product will be spent after the product is released.

PLAN TESTING OBJECTIVES EARLY

Successful software testing tools need to be designed to survive passage from the testing phase into a useful life in the maintenance phase of a product. There are three steps to be considered in the design. The first step is to PLAN TESTING OBJECTIVES EARLY. This step is the most important as it will clearly define the resources available in which the software testing tool is to be built. Most important, in this step, is the time allocated to design the software testing tool. Before an engineer can design, the engineer needs to be allocated time in which to accomplish that task. Software testing tools which fail are usually those tools which have been designed and built in the testing phase by the engineers responsible for releasing the product. This is not surprising, as the testing phase is the last phase before product release, usually the most pressured and therefore the worst time to make design decisions. Not all of the time available to the testing phase can be spent on software testing tools. Some forms of testing such as ease of use, documentation, portability and adherence to standards do not lend themselves to automated means of testing. They require qualitative decisions. By planning the testing objectives some decision will be reached on the resources necessary to accomplish the task. From that available pool of resources will come the valuable time to design your successful and practical software testing tool or tools.

DETERMINE THE FEATURE SET

The second factor in designing a successful software testing tool is to DETERMINE THE FEATURE SET of the tool. The proper feature set will preserve the initial investment in the tool once the product enters the maintenance phase. Choosing the correct feature set depends heavily on the product being produced and the environment in which it is being produced. In all cases careful evaluation must be given to the features desired in the testing phase and then in the maintenance phase. Certain features are more important than others. As an example, the following scenario is presented. The engineers who have written software for the product (and may have written the software testing tool itself) most likely will move on to other projects. Thus, it is common in the maintenance phase to have new engineers supporting the product now released. Important features to be included in the feature set of the software testing tool in this case are "ease of use" and "adequate documentation". Because most software projects follow the above trend and because some tools may not be easy to use "adequate documentation" is the most important feature of any software testing tool. If the product under test will be subject to major enhancements in its lifecycle then extensibility, "the degree to which it is possible to extend the original design so as to accommodate enhancement, new features and new options" [QMT81], may be a major feature in the feature set of the software testing tool. Features to be considered are those of any well designed product: documentation, ease of use, maintainability, enhanceability, on time delivery, reliability, performance, portability and functionality.

The environment within which the software testing tool is developed usually has little to do with the product under test or the testing methodology. The environment is a function of the resources that are required to execute the software testing tool. For example, in the testing of IMF/3000 the author was responsible for writing a program which could be transported to more than one IBM host. This tool was designed to exercise the functionality of IMF/3000. One of the main objectives of this tool was portability, i.e., the ability to move the program from one host to another with a minimum of code modifications. Regardless of what proportions of the feature set are determined by the testing methodology or the environment, the key to designing a successful software testing tool rides on the preservation of the features of the tool from the testing phase into the maintenance phase where the costs of the tool can be recovered.

DECIDE ON SERVICES

The third step in designing a successful software testing tool is to DECIDE ON THE SERVICES that the software testing tool will provide. Most software testing tools aid in functional testing, reliability testing, and performance testing, reducing the amount of manual labor involved. This step is perhaps the easiest step of all because the engineers who will be designing, implementing and testing the product already recognize from experience that many aspects of the testing process are laborious, unrewarding and most likely will have to be repeated several times (most likely late at night and on weekends) before and after the product has been released. Thus, the incentive already exists to identify those areas of testing that can be automated in order to reduce the inefficiencies associated with manual testing. Some tools, considerably closer to being state of the art in the field of software testing, examine the internal characteristics of the software product under test (white box testing) and report on the degree of testing thoroughness. They give a simple measurement of the branches in the code that were exercised by a test case. More importantly, they report on the branches that were not executed, informing the engineer that there are untested portions of code. Within Information Networks Division such a tool already exists under the name of Path Flow Analyzer and is being used to improve the thoroughness of the testing effort.

One should choose the services of a software testing tool with consideration that the services must be useful in both the testing phase and the maintenance phase of the project. For those who wish to know more on the subject there are many excellent text books and tutorials on testing and software testing tools (some of which are mentioned in the bibliography) which explain the many different types of testing, their area of application, benefits and drawbacks, and even case histories.

SUMMARY

Software testing tools are going to be built to aid in the testing phase and maintenance phase of a product's lifecycle. Certainly there will be some risks in investing in software testing tools from the view of recapturing their investment. For example, resources will have to allocated earlier in the product's lifecycle to design the testing tool. Surely, this is a better approach than not planning for software testing tools whatsoever, and losing the investment in the few tools that are built. By approaching the subject of software testing tools as specified in this document you can not only recapture the investment in the tool but you may significantly reduce the costs associated with maintaining software once it has been released to the marketplace.

CASE STUDY 1

In this case study we will examine a software testing tool that embodies many of the features discussed in this paper. The testing tool is called IMLTEST. It is used to exercise the IMF/3000 intrinsics (Interactive Mainframe Facility/3000). IMLTEST due to its feature set, fills the needs of both the testing phase and the maintenance phase. It does so by providing flexibility in designing test cases during the testing phase and by providing automation of test cases during the maintenance phase. IMLTEST also demonstrates design techniques that can be used as the basis for any software testing tool which is responsible for exercising a set of intrinsics or user callable procedures. The design concepts behind IMLTEST are presently being applied to future testing tools which will exercise products whose user interface is a set of intrinsics.

IMLTEST has the following feature set: ease of use, a wide range of functional capabilities, ease in automating test cases and excellent documentation. From this feature set the following benefits have been realized:

* IMLTEST greatly reduces the effort expended in calling a set of intrinsics.

* IMLTEST enables progressing from the testing phase into the maintenance phase easily by providing the means to automate functional tests. Testing for compatibility and ensuring that regressions have not taken place due to bug fixes or enhancements becomes a significantly less labor intensive task.

* IMLTEST enables the rapid transition of continued testing and maintenance from one engineer to another through its ease of use.

IMLTEST was designed to aid in the areas of functional and reliability testing. Functional testing ensures the existence of a set of features as described in the design documents. Reliability testing ensures the correct execution of functions under a variety of conditions such as a heavily loaded system or mean time between failure. IMLTEST was designed, coded and debugged in six months. IMLTEST provides the following services:

* Interactively call all the IMF intrinsics

* Develop test cases very fast by freeing the user from having to write programs, thus eliminating the overhead of declaring variables, debugging syntax, checking return codes, etc.

* Read the commands from a file instead of a terminal in order to automate the process of testing.

* Easily determine how the product is to be used and become almost immediately productive developing a test case.

* Supply legal and illegal parameter values to any intrinsic

* Ensure the consistency of output regarding test success or failure.

During the testing phase, flexibility, that is, the ability of the testing tool to exercise a wide range of functions of the product under test was determined as a major component of the test tool. Some of the product specific commands in IMLTEST which address functional testing are:

OPEN #NUM =CONFIGURATION FILE, WAIT
: Opens a particular device using the configuration file specified. If WAIT is specified all I/O is waitio. Calls IMF intrinsic OPEN3270.

CLOSE
: Closes a particular device. Default device is the one last used. Calls IMF intrinsic CLOSE3270.

READ (#FIELD,#MAXIMUM)
: Reads and prints the contents of the field specified. Calls IMF intrinsic READFIELD.

WRITE #FIELD
: Specifies that data after the WRITE command is to be written to the field specified. Calls the IMF intrinsic WRITEFIELD.

RECV
: Informs IMF to await the arrival of data from the IBM host. Calls the IMF intrinsic RECV3270.

TRANS KEY
: Calls IMF TRAN3270 intrinsic with key specified. KEY may be anyone of ENTER, PAX, PFXX, SYSREQ, etc.

To demonstrate the ease in which a test case can be written a sample test case which logs on to an IBM host and logs off again is provided.

EXAMPLE A

```
@*******************************************************
@  Sample test case of logging on to a host application
@  and logging off again.
@ NOTE: statements beginning with
@      the "@" sign are comments.
@*******************************************************

@ Open the IBM 3270 device emulating 3277 device 1.
OPEN #1 =CONFIG31,W

@ Receive the IMF banner
RECV

@ Receive the logon screen from the host
RECV                            (line 1 )

@ Put logon message in field 5          (line 2 )
WRITE #5                        (line 3 )
logon applid( dcp3271 )         (line 4 )
```

```
@ Send message to host              (line 5 )
TRANS ENTER                         (line 6 )

@ Receive screen from host          (line 7 )
RECV                                (line 8 )

@ Logoff from host                  (line 9 )
WRITE #1                            (line 10 )
logoff                              (line 11 )

@ Transmit logoff message to host   (line 12 )
TRANS ENTER                         (line 13 )

@ Close device 1
CLOSE

@****************************
@ END OF EXAMPLE
@****************************
```

Automation plays the most important role during the maintenance phase. It is in this phase that most cost savings associated with testing can be recovered. IMLTEST provides many commands to aid in the automation of test cases. Its also provides commands which aid in the management of test cases. A sample of commands is provided in the following example along with a demonstration of the ease in which a test case, once developed, can be automated for reliability testing.

FILE = FILENAME
: IMLTEST opens the file specified and reads commands from that file until end of file is encountered. The file is then closed. This command enables automation of testcases and the generation of reliability scripts...see FLOOP command.

FLOOP #START #END
: This command initializes IMLTEST for looping from START to END for the file specified in the FILE command. The file is opened, the commands read and executed until end of file is detected. The file is then closed. This process continues END - START + 1 times. Using this mechanism a series of commands can be executed any number of times. Having many devices using the FLOOP command is a means of generating reliability and stress test cases.

CMPFILE =FILE
: Failures during the testing effort are not always catastrophic and easily detectable. This command will compare one or more received screen images to a file where screen images of known correctness are stored. This almost eliminates the manual effort of verifying that incoming screens have not been incorrectly processed. The only manual labor involved is to build the file of screen images that are known to be correct.

NAME =TESTNAME
: This command sends a message to $STDLIST or the console to indicate the start of a new test case. It is part of the test management logging facility.

Its purpose is to clearly show what test case is executing, error conditions that arise, and upon test case completion, a message indicating success or failure of the test.

XPECT #ERROR NUMBER
This commands informs IMLTEST that the next intrinsic call should return the specified error number. This is the means of automating error cases.

The following example uses the test case developed in EXAMPLE A and creates a reliability script. The reliability script causes IMLTEST to log on and off an IBM host 20 times. While this is only an example of a single device logging on and off, IMLTEST has the capability of spawning up to 32 devices, each with a different device number, each logging on and off 20 times and all of them sharing a single script file very similar to the one below.

EXAMPLE B

```
@***************************************
@  Example of FLOOPing
@  Log on and off the host twenty times
@***************************************

@ Identify the name of this test case
NAME =testcase 1 example

@ Open the IBM 3270 control unit device 1.
OPEN #1 =CONFIG31,W

@ Receive the IMF banner.
RECV

@ Loop twenty times on file "LOOP20".
@ LOOP20 is a file which contains lines
@ 1 through 13 from EXAMPLE A.
@
@ Input file to read commands from.
FILE =LOOP20

@ Now loop 20 times
FLOOP #1 , #20

@ IMLTEST has finished looping. Close the device.
CLOSE

@****************************************
@ END OF EXAMPLE B
@****************************************
```

ALTERNATIVES

The most obvious alternative to building a test tool was to write individual test programs calling the intrinsics to be tested. This approach had several drawbacks.

Overhead -- Each program would have to duplicate variable names, data structures and intrinsic declarations all associated with just trying to make the intrinsic call.

Complexity -- The intent of the test case, even if documented, would be lost in the program structure and elements necessary to support the intrinsic calls.

Management -- A proliferation of programs whose individual testing objectives are unclear, whose dependencies and side affects are unknown when running together. Programmers usually design their own output messages on test success or failure. Over time, with test cases being designed by many different programmers, there would arise a multitude of output test messages, inconsistent and often confusing.

Leverage -- Future test cases could leverage very little off previous work. Reduction of manual labor in the development of new tests would be minimal.

Another approach is to design a product that uses all the intrinsics. Pass-Thru is an example of such a tool, using almost all the IMF/3000 intrinsics. Pass-Thru is that portion of IMF/3000 which maps IBM display formats into HP display formats enabling users to use HP terminals and printers to emulate IBM terminals and printers. This approach does not provide an adequate design. It does not provide for flexibility during the testing phase, nor does it provide for automating test cases during the maintenance phase. It does provide instant access to the large number of IBM 3270 screens that are available. It also provides instant feedback if a particular screen should cause an error within the IMF/3000 intrinsics.

SUMMARY

IMLTEST has been in use for approximately 3 years. It is considered an extremely valuable testing tool by those involved in the maintenance of IMF. It is also invaluable in the testing of major enhancements to IMF such as IMF/SDLC. The design of IMLTEST has greatly reduced the time it takes to call the IMF intrinsics. It also has made significant cost savings in the automation of test cases. Software testing tools being developed in the Datacommunications lab today are leveraging from the contributions of IMLTEST.

CASE STUDY 2

In this case study we will examine a test management tool that focuses only on the maintenance phase. The purpose of presenting this tool is to contrast how tools can be effectively designed to meet the needs of the testing phase, maintenance phase or both phases.

The software tool to be discussed, called Test Logging Facility, solves many of the problems associated with test management. The Test Logging Facility's contribution is the automated means in which it standardizes the output of all test cases regardless of whether the product under test is an operating system, compiler, or data base management system. It aids in answering the two most important questions in managing a set of tests. First, has a test or series of tests run to completion? Secondly, did the test output indicate test success or test failure?

The test logging facility aids in the reduction of labor and the encouragement of automation of test cases by handling all output related to test commencement, test progress, error conditions and test termination. The information is passed to the logging facility through intrinsic calls or, alternatively, depending on the product under test, by running programs that call the intrinsics. There are five basic message types in the standard. They are (1) test, (2) test unit, (3) test subunit, (4) comment, and (5) error. Each one of the message types is associated with a standard method of reporting.

In order to standardize the output of tests, definitions had to be given to the terms used. The following is the essence of the definitions. A test is a collection of test subunits designed to exercise a product. There should be only one test per product. A test subunit is a single program or one or more commands (such as file equations if file equations were being tested). Therefore, in order to test a compiler, one might have several separate programs which exercise the various functions of the compiler. The collection of programs is considered a test. A subunit was defined to provide a means of checkpointing large programs which tested many functions of a product all within a single program. A comment is any information the user wishes to post in the output file. It may not contain information on error conditions. The error message contains information regarding any failures detected.

Test monitoring is the last phase of the testing effort. It consists of application of tests, recording test output results, determination of success or failure of test cases, and all supporting documentation. Management of the testing effort, especially during the monitoring phase, can become extremely complex when test cases become automated (as they should) in the maintenance phase. The problems arise from a lack of standards governing the output of test cases. A lack of standards in a large testing effort (such as the scope of a quality assurance department or any large project) results in a major difficulty in determining the success or failure of a test or series of test cases. Some tests are considered successful if they end with no output messages. Some tests output information as they execute, as a normal part of their operation. In either case, to anyone but the author of the test cases, the success or failure of the tests can be shrouded in mystery. The lack of standards has two significant side effects. Firstly, it locks the engineer who wrote the tests into running the tests, as the engineer is the only one who can accurately decipher their output. Secondly, it prevents the smooth transition of bringing in a new engineer as the test output must be explained in detail. The test output may vary dramatically from product to product, but also, even within the same product.

The benefits derived from the test logging facility are:

* Creation of standard test execution history

* Reducing test engineers monitoring efforts. Efforts can then be channeled into test specification and development.

* Highlighting features of test, such as commencement, checkpoints, errors, and termination.

The following intrinsics are provided by the test logging facility. They may be called from any language. By following the examples provided in the appendix the reader can match the intrinsic call to the message it creates. Two examples are provided. The first contains examples of all output messages. The second is the output of a typical test.

LOGSTART(TESTMSG)
This must be the first intrinsic called. It creates the output file and the first message labeled "START". Time stamps are posted and the testmsg is printed.

LOGBEGIN(LOGINFO, UNIT'NAME, UNIT'TITLE)
This message marks the beginning of a test unit. The user passes a unique integer as a log identifier, a test unit name and a test unit title. The message associated with this intrinsic call begins with "BEGIN".

LOGERR(LOGINFO, UNIT'NAME, ERROR'MSG)
This intrinsic logs the occurrence of an error condition. It posts two messages. Both begin with the word "ERROR". They are designed to be highly visible. A short description of the error is printed from the parameter ERROR'MSG.

LOGERR2(LOGINFO, UNIT'NAME, ERROR'DESCRIPTION)
This intrinsic enables the user to post any other relevant information about the error detected. It is provided as a means of supplementing error information but is not intended to be used for dumps, long listings, etc. The message associated with this intrinsic call begins with "ERR2". LOGERR must be called before LOGERR2.

LOGCKPT(LOGINFO, UNIT'NAME, MESSAGE)
This intrinsic prints a checkpoint message identifying the progress of the test unit. The message associated with this intrinsic call begins with "CKPT".

LOGCOMM(LOGINFO, UNIT'NAME, COMMENT)
This intrinsic prints a comment to aid in the identification of a test or test cases. It is not used to log error information. The message associated with this intrinsic call begins with "COMM".

LOGEND(LOGINFO, UNIT'NAME, SELFCHECK)
This intrinsic will log the end of a test unit. If the test unit produces errors the number of errors detected is printed if the selfcheck parameter is set to true. The message associated with this intrinsic call begins with "END".

LOGSTOP
This intrinsic is called at the completion of the test. It posts the message beginning with "STOP". It prints several items which summarize the result of the test.

SUMMARY

The test logging facility has been in use within the Quality Assurance department since 1979. New test cases are designed to use the intrinsics while older test cases have been retrofitted to include them. This effort has significantly reduced the labor involved in test monitoring. It has also contributed to the distribution of the test cases outside the Quality Assurance department as the output of test cases are standarized and easily understood, regardless of the product under test.

```
START   TEST LOG  TUE, AUG  3, 1982,  9:48 AM     Pascal test for logging

BEGIN     Test 1          1  09:48:27.0   Start of first test
 CKPT     Test 1          1  09:48:27.1   Checkpoint message

 ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR

 ERROR    Test 1          1  09:48:27.4   This is an error message
 ERR2     Test 1          1  09:48:27.4   This is an err2 message
 COMM     Test 1          1  09:48:27.5   This is a comment message
END       Test 1          1  09:48:27.6   ET: 000:00:00.6                                                    ERRORS=    1

STOP    TEST LOG  TUE, AUG 03, 1982, 09:48 AM     ET:0:0       TOTAL TEST UNITS   1  COMPLETED   1       TOTAL DETECTED ERRORS=    1
```

APPENDIX

(example of every Test Logging Facility message)

```
START   TEST LOG  TUE, AUG  3, 1982,  9:44 AM     Testing summary log

BEGIN     FILECOMPARE     1  09:44:02.2   Comparison of Files ztest2 and ztest

 ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR

 ERROR    FILECOMPARE     1  09:44:03.0   Records fail to match at record        3, byte  7
 ERR2     FILECOMPARE     1  09:44:03.0      Test =
 ERR2     FILECOMPARE     1  09:44:03.2     X1,X3: BOOLEAN;
 ERR2     FILECOMPARE     1  09:44:03.4      Master =  (record 3)
 ERR2     FILECOMPARE     1  09:44:03.5     X1,X2: BOOLEAN;

 ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR

 ERROR    FILECOMPARE     1  09:44:04.5   Extra record in test file at record    4
 ERR2     FILECOMPARE     1  09:44:04.8      Test =
 ERR2     FILECOMPARE     1  09:44:05.0     {This is an extra line}

 ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR

 ERROR    FILECOMPARE     1  09:44:06.9   Records fail to match at record        11, byte  4
 ERR2     FILECOMPARE     1  09:44:07.7      Test =
 ERR2     FILECOMPARE     1  09:44:08.4     X3  := FALSE;
 ERR2     FILECOMPARE     1  09:44:08.7      Master =  (record 13)
 ERR2     FILECOMPARE     1  09:44:09.0     X2  := FALSE;

 ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR**ERROR

 ERROR    FILECOMPARE     1  09:44:09.5   Missing record in test file at record  12
 ERR2     FILECOMPARE     1  09:44:15.3      Master =  (record 14)
 ERR2     FILECOMPARE     1  09:44:15.5     WRITELN (OUTPUT,X1,X2);
END       FILECOMPARE     1  09:44:15.7   ET: 000:00:13.5                                                    ERRORS=    4

STOP    TEST LOG  TUE, AUG 03, 1982, 09:44 AM     ET:0:0       TOTAL TEST UNITS   1  COMPLETED   1      TOTAL DETECTED ERRORS=    4
```

APPENDIX

(example of Test Logging Facility in live example)

Bibliography

[Daley77] Daly, Edmund B., "Management of Software Development" IEEE Transactions on Software Engineering, May 1977

[Miller81] Miller, Edward. , Howden, William E. "Tutorial: Software Testing & Validation Techniques", Second Edition, 1981. Published by IEEE Computer Society Press.

[Myers76] Myers, Glenford J., "Software Reliablility Principles & Practices", John Wiley & Sons, New York, 1976.

[QMT81] Quality Management Team. "Software Quality (working definition)", Revision 2.0. Information Networks Division, 1981.

IMAGE INTRODUCES AN END TO THE BROKEN CHAIN

JORGE GUERRERO
HEWLETT-PACKARD COMPANY
INFORMATION NETWORKS DIVISION

IMAGE INTRODUCES AN END TO THE BROKEN CHAIN

An IMAGE product enhancement by the name of Intrinsic Level Recovery, or ILR for short, is being released with IMAGE version B.04.00 This feature depends on the Q-mit (C.01.00 or D.01.00) or later versions of MPE and will enable IMAGE to patch-up automatically any broken data chains that might have been caused by a system failure.

A broken chain might occur in the synonym chain of a master data set or in any data path (chain) in detail data sets.

Since this feature adds some overhead (more about that later) it has been implemented in such a way that it can be enabled or disabled at any time. When this feature is disabled (which is the normal state) IMAGE operates as before with no added overhead.
When this feature is ENABLED with DBUTIL (there is another flag added to the root file) IMAGE will copy all of the buffers that will be modified to a special log file. This privilege mode file is created by DBUTIL at the time the data base is enabled for ILR. Its name is derived from the root file name with two zeroes added at the end. For instance, a data base named STORE will have the ILR log file name STORE00. If the data base was enabled for ILR and is later disabled, the ILR log file is automatically purged by DBUTIL. Also, since the ERASE function of DBUTIL is to leave the data base in a state similar to a 'just-created' state, it will purge the ILR log file and reset all flags.
Once the data base is enabled for ILR, only the intrinsics that modify pointers are logged onto the ILR log file. These intrinsics are DBPUT and DBDELETE. Remember that DBUPDATE only modifies the data portion of an entry, not the key/search or sort items. In order to log these changes, IMAGE creates an additional control block at DBOPEN time only if the data base is enabled for ILR.

This control block, named the ILCB, is nothing more than an extra data segment that is utilized as an intermediate staging area for the buffers that will be modified. Its operation is as follows: when an intrinsic (DBPUT or DBDELETE) starts its execution, IMAGE will detect all of the buffers that will be modified by the intrinsic. These buffers get posted to the ILCB (before modification) and then modified in the DBCB. When all of the buffers are modified in the DBCB, and before posting to the actual data sets, the information in the ILCB is posted to the ILR log file. After posting the modified buffers to the data sets, the intrinsic-in-progress flag is turned off in the ILR log file.

Pictorially, it looks like this:

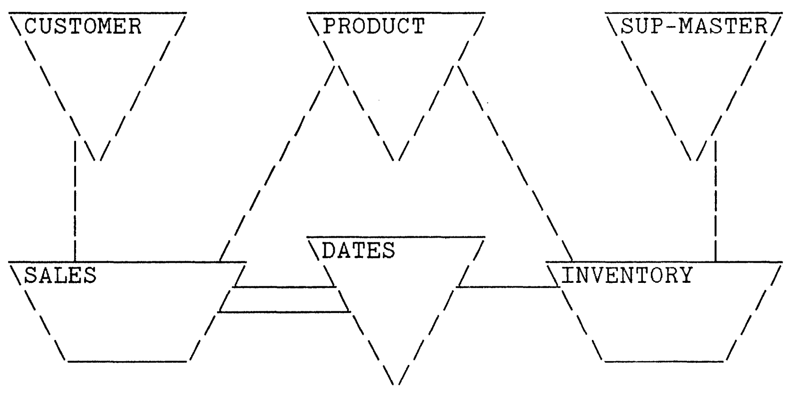
```
                   MEMORY                                        DISC
                                            |
                                                             _____     _____
     ULCB       DBCB        ILCB                             \   /     \   /
   |      |   |      |    |header|}         |                 \ /       \ /
   |      |   |      |    |.    .|}                            \/  _____ \/
   |      |   |      |    |.    .|}         |          _____       \   /
   |______|   |      |->  |.    .|}------------->   |.ILR .|       \__/
              |      |    |______|}         |       |.Log .|
              |      |                              |.File.|
              |      |                      |       |.    .|
              |______|                              |______|
                                            |
```

The ILR log file gets overlayed with the latest intrinsic information, so the information found there is only for the intrinsic in progress and not for all of the intrinsics or transactions that have occurred in the data base. If the intrinsic never ends its execution due to a system failure, the next time the data base is DBOPENed, the ILR log file is checked. If an intrinsic was in progress, the before-modification buffers get posted to the data sets and the data base structural integrity is restored.

The overhead incurred to perform this function consists of the following:

one shared priv mode file (ILR log file)
one shared extra data segment (ILCB)
disc I/O to post info from ILCB to ILR file

To analyze this overhead, the STORE data base, documented in sections II and III of the IMAGE manual, is utilized here as an example.



Schematic Diagram of the STORE Data Base

The ILR log file is composed of three parts wich are:

- ILR log file header
- data sets user labels
- data sets buffers

The header area is 50 words long and is utilized to keep track of the intrinsic in progres flag, intrinisc type, data set number, etc.

The data sets user labels area follows immediately after the header area and occupies 6 words per data set.

The data sets buffers area starts on a sector boundary after the two previously mentioned areas, and it should hold as many buffers as can be modified by one intrinsic.

The number of buffers that can be modified depends on the structure of the data base. For a master data set, whether manual or automatic, the maximum number of buffers that can be modified is 4. On a detail data set is equal to 4 times the number of paths from an automatic master pointing to it, plus 3 times the number of paths from a manual master pointing to it, plus one or:

Max'mod'buffs = (4 * Autos) + (3 * Manuals) + 1

The maximum number of buffers that can be modified is greater for a detail than for a master data set. The exception would be for detail data sets with no paths which would be only one, but if it does not have any paths then it does not have any pointers and no broken chains can be generated.

The maximum number of buffers is only utilized for the case that <u>all</u> of the entries that need to be modified reside in <u>different blocks</u>. This number includes the migrating secondaries case in automatic masters and sorted data chains in the associated detail data set. However IMAGE will try to accomodate synonyms in the same block in master data sets and, periodic chained DBUNLOADs and DBLOADs should position detail data entries within the same blocks. For this reasons the maximum number of modified buffers might not ever be reached.

On the STORE data base, the maximum number of buffers that can be modified by a DBPUT or DBDELETE intrinsic is 15, since the SALES detail data set has 2 paths from an automatic master and 2 paths from manual masters. The maximum size of the buffers is governed by the DBSCHEMA command, BLOCKMAX, which has a default value of 512 words. The actual size is calculated by DBSCHEMA and reported on the TABLE portion of the output list generated by DBSCHEMA. For the STORE data base the calculated buffer length is shown in Figure 3-5 (Section III) of the IMAGE manual and its value is reported as 511 words. Since each buffer has a 10 word overhead, for a total of 521 words per buffer, each will occupy 5 sectors in the ILR logfile. The ILR log file size calculations for the STORE data base are:

A) Record size: 10 + buffer len

```
(10 + 511) / 128 = 5 sectors/rec
                       or
                   640 words/rec
```

B) ILR log file header and ulabels: 50 + (6 * sets)

```
50 + (6 * 6) = 86 words
                  or
              86/640 = 1 record
```

C) Total records and size:

(15 + 1) * 5 sectors/rec = 80 sectors

Add 5 sectors for the file label.

Since this file only holds the buffers for the last intrinisc it is not necessary to make it any larger. The space occupied by this file is negligible compared to the spaced occupied by the data base itself.

The ILCB size depends on the structure of the DBCB. It should hold as many buffers as can be modified in the DBCB before posting to the data sets. The DBCB should hold at least as many buffers as can be modified (by a put or delete operation) to minimize buffer 'roll-over', thus minimizing disc I/O to the ILR log file.
For the STORE data base, the number of buffers in the DBCB should be set to at least 15 buffers. That would be the maximum number of buffers that can be modified as shown in the previous example. More buffers can be set in the DBCB to increase IMAGE's performance but 15 will increase ILR's performance. The number of buffers in the DBCB can be set with DBUTIL's SET command to maximize performance for this enhancement.

As far as the number of disc I/Os from the ILCB to the ILR log file is concerned, the overhead is as follows. At the beginning of the intrinsic, one disc I/O is done to record date, time, data set number and other flags, including an 'intrinsic-in-progress-flag' which will be turned off at the end of the intrinsic. The additional disc I/O will depend on the number of buffers that the DBCB can hold. For instance, if the DBCB can hold only 8 buffers and 15 buffers have to be modified, some of the buffers will have to be posted before they are overlayed. On the other hand, if the DBCB can hold 20 buffers then we can wait until the end of the intrinsic to start posting the modified buffers. Once these modified buffers get posted to the data sets, an additional I/O has to be performed on the ILR log file to turn off the 'intrinsic-in-progress-flag'.
So the minimum number of additional disc I/Os is 3 and, the maximum will depend on the maximum number buffers that have to be modified <u>and</u> the number of times that the DBCB buffer area has to be 'rolled-over'. The following table shows the number of disc I/Os from the ILCB to the ILR log file for number of buffers in the DBCB vs maximum number of modifiable buffers. The number of disc I/Os shown in the table do not include the disc I/O to turn 'on' and 'off' the intrinsic-in-progress-flag. An additional value of 2 should be added to the values found in the table.

DISC I/Os (FROM ILCB TO THE ILR LOG FILE)
BUFFERS IN DBCB VS MAXIMUM NUMBER OF MODIFIABLE BUFFERS

MAXIMUM NUMBER OF MODIFIABLE BUFFERS (worst case)

Max'mod'buffs = (4 * Autos) + (3 * Manuals) + 1

		5	8	11	14	17	20	23	26	29	32	35	38	41	44	47	50	53	56	59	62	65
	4	2	2	3	4	5	5	6	7	8	8	9	10	11	11	12	13	14	14	15	16	17
B	6	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	9	9	10	10	11	11
U	8	1	1	2	2	3	3	3	4	4	4	5	5	6	6	6	7	7	7	8	8	9
F	10	1	1	2	2	2	2	3	3	3	4	4	4	5	5	5	5	6	6	6	7	7
F	12	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5	6	6
E	14	1	1	1	1	2	2	2	2	3	3	3	3	3	4	4	4	4	4	5	5	5
R	16	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4	5
S	18	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3	4	4	4	4
	20	1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3	3	4	4
	22	1	1	1	1	1	1	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3
I	24	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	3	3	3	3	3	3
N	26	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	3	3	3	3	3
	28	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3	3
	30	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3
D	32	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3
B	34	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2
C	36	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2
B	38	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
	40	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
	42	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
	44	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2
	46	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2
	48	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	50	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2
	52	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2
	54	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2
	56	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2
	58	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2
	60	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2

NOTE: Add 2 to the values show in the body of the table.
The values shown here are for the worst case situation.
The minimum number of modifiable buffers is given by

Min'mod'buffs = (Autos) + (Manuals) + 1

It is of paramount importance to set the number of buffers in the DBCB to a value that will minimize buffer rollover and hence, disc I/O. Don't forget to utilize the previously mentioned formula to calculate the maximum number of buffers that can be modified in a put or delete operation and set them in your data base.

Other miscellaneous information about this enhancement:

- DBSTORE/DBRESTOR has been modified to store/restore the ILR log file.

- DBINFO has an added mode to find out if the data base is enabled for ILR and was recovered automatically upon DBOPEN.

- Since this enhancement forces the posting of buffers, it is not compatible with output deferred. If the data base is enabled for ILR, an attempt to turn on output deferred (DBCONTROL with mode=1) will fail. So it is necessary to turn the ILR flag off (disabled) if you need to execute in output deferred mode.

- The creator of the ILR log file is always the creator of the data base, regardless of the creator id that enables the data base.

- If the data base is released before enabling for ILR, the log file is released automatically at creation time.

This enhancement will greatly improve your IMAGE data base integrity and will save you a lot of time in unloading/loading to recover from a system failure.

KSAM DESIGN GUIDELINES FOR OPTIMIZATION

JORGE GUERRERO
HEWLETT-PACKARD COMPANY
INFORMATION NETWORKS DIVISION

KSAM DESIGN GUIDELINES FOR OPTIMIZATION

Objective.

The objective of this presentation is to familiarize you with the different ways to optimize the design of a KSAM file set. This will be achieved by analyzing the overhead utilized by KSAM when a file set is accessed.

At the end of this presentation you should be able to optimize the design of a KSAM file set.

Introduction to topic: KSAM file set design

In order to understand how to optimize the design of a KSAM file set it is necessary to analyze what it is that needs to be optimized.
First of all, we want to optimize (minimize) the resources used by KSAM when it is active.
These resources are:

1. Key and data file space.
2. One private extra data segment per file open.
3. Disc I/O to access data.

The data file disc space utilized can be reduced by choosing a blocking factor that makes the data block size as close as possible to a sector boundary. This minimizes wasted space at the end of each data block. Sometimes there will be more than one blocking factor that gives the same utilization and the task of choosing a value becomes uncertain. For instance, a record size of 80 bytes with a blocking factor of 16 yields the same space utilization than a blocking factor of 32. For every situation the blocking factor chosen should be large enough to hold all of the records that are introduced in a logical transaction. If the logical transaction includes writing 19 records then the larger blocking factor of 32 should be chosen to minimize buffer roll-over.

The key file size is optimized by KSAM when the file is built. The key blocks are created with their own key blocking factor to optimize disc space utilization.

The other resources that need to be optimized are, the size of the extra data segment and the number of disc I/Os to access the data.

The extra data segment (XDS) size should be optimized for systems with low memory configurations since there is one XDS created for every file open, even if the same process opens the file more than once.

If the file is opened many times, the resources that are utilized could exceed the available memory and a problem of excesive swapping would be created. The problem of memory shortage in a system can be simply resolved by acquiring more memory. However, if this is not an appropriate solution, the XDS size should be optimized to minimize memory resources.

To fully understand how to optimize the XDS size it is necessary to understand the different components that come into play when it is created. These components are:

1. data file record size
2. data file block size
3. key file block size
4. number of levels in the B-tree

The data file record size is difficult to optimize since the data record would have to be compressed, or some fields within it suppressed in order to reduce it.

The data file block size can be reduced by choosing a smaller blocking factor. However, that value has already been chosen to optimize disc space utilization and disc I/O per transaction. Reducing it here would hinder that effort.

The next item to analyze is the key file block size. The size of the block can be reduced by choosing a smaller key blocking factor; however, that could increase the number of levels in the B-tree, which is the 4th point to analyze. Since the XDS will try to have one block for each level, reducing the block size might cause more levels to be created. If that is the case, the target of reducing the XDS size is not reached. Also, more levels generate more I/O. Remember that the number of levels indicates the maximum number of I/Os to get to the key entry. Once the key entry is obtained, an additional I/O get the data record.

If it is assumed that the same space in the XDS is retained regardless of the key blocking factor, then what has to be optimized is the number of levels in the B-tree, so that less I/O is generated.
To understand how to generate fewer levels, a brief analysis of the B-tree is presented here.

A B-tree will hold as many keys as there are active records in the data file. Whenever a data record gets added to or deleted from the data file, the corresponding keys are also added or deleted to/from the B-tree. When a key gets added or deleted the B-tree might expand or contract its number of levels accordingly.

KSAM-4

When a KSAM file set is being loaded with data records in either sequential or random fashion, it will start creating different levels in the B-tree of the key file. Thus there is more disc I/O to get to the key entry. This means that to optimize the design, from the I/O side, it is important to know when a level will be created in the B-tree. In other words, how many key entries (active records) can be held at each level? The answer to this question is complex since a level can hold a range of entries before it creates another level. So, there is a minimum and a maximum number of entries in a level.

The minimum number of key entries (MinK) in any level is calculated with the formula:

$$\text{Mink} = 2 * (F/2 + 1)**(L-1) - 1$$

where F is the key blocking factor and
L is the number of levels.

The following figure illustrates how to find the minimum number of keys in a level. The figure is a 2 level (L) B-tree with a key blocking factor (F) of 4.

```
root->    [*]300[*]\\\[*]\\\[*]\\\[*]
           |     |
           |    [*]400[*]500[*]\\\[*]\\\[*]          <-leaf
           |
          [*]100[*]200[*]\\\[*]\\\[*]                <-leaf
```

Notice that since all of the leaves are half empty, the deletion of any key will cause the B-tree to contract one level.

The maximum number of key entries (MaxK) in any level is calculated by the formula:

$$\text{MaxK} = (F + 1)**(L) - 1$$

where F and L are the same as before

To illustrate the maximum number of keys in a level, see the following figure.

```
root->   [*]  5[*] 10[*] 15[*] 20[*]
          |     |     |     |     |
          |     |     |     |    [*] 21[*] 22[*] 23[*] 24[*]   <-leaf
          |     |     |     |
          |     |     |    [*] 16[*] 17[*] 18[*] 19[*]         <-leaf
          |     |     |
          |     |    [*] 11[*] 12[*] 13[*] 14[*]               <-leaf
          |     |
          |    [*]  6[*]  7[*]  8[*]  9[*]                     <-leaf
          |
         [*]  1[*]  2[*]  3[*]  4[*]                           <-leaf
```

Notice that since all of the blocks are full, adding a key creates another level.

However, a given level does not have to be full before creating another level. There is a critical point in the B-tree, before it reaches the maximum value, where it might split and create another level. So there is another concept to explore and that is the critical number of keys that can be held at any level before creating another level.

The critical number of keys (CritK) is calculated with the formula:

```
         CritK = 2 * (F/2 + 1)**(L) - 2

         where  F  and  L  are the same as before
```

This formula is obtained by subtracting 1 from the minimum value of the next higher level. And, once again, to illustrate the critical number of keys in a B-tree see the following figure.

```
root->   [*] 30[*] 60[*] 90[*]120[*]
          |     |     |     |     |
          |     |     |     |    [*]130[*]140[*]150[*]160[*]   <-leaf a
          |     |     |     |
          |     |     |    [*]100[*]110[*]\\\[*]\\\[*]       <-leaf b
          |     |     |
          |     |    [*] 70[*] 80[*]\\\[*]\\\[*]           <-leaf c
          |     |
          |    [*] 40[*] 50[*]\\\[*]\\\[*]               <-leaf d
          |
         [*] 10[*] 20[*]\\\[*]\\\[*]                   <-leaf e
```

Notice that if a key is added to leaf 'a', it splits and its middle value migrates to the root block. This will cause the root block to split and generate another level. However, no spliting occurs if a key is added to the other leaves.

Since the B-tree does not split until it reaches the critical value, it can be loaded with keys up to the critical value without creating another level. Once the B-tree has reached the critical number of keys, it can split any time a key is added up to the maximum number of keys. Since the number of keys for any level depends on the key blocking factor and what has to be optimized (minimized) is the number of levels, the appropriate blocking factor must be chosen to give the least number of levels.
To facilitate the task of chosing a blocking factor, the tables in appendix A should be of help.
Table A-1 shows the allowable blocking factors for different key lengths vs key block sizes. The key lengths shown range from 2 to 50 bytes long and are of even sizes only. It is not necessary to show values for odd key lengths since all of the key entries are allocated on word boundaries. For instance, a key length of 3 bytes will occupy 4 bytes in the key file, and the calculations for block sizes should be done with 4 bytes or 2 words. The default blocking factors are also shown. The default values are those calculated by KSAM when the files are built and no blocking factors are specified.

The allowable blocking factors can be read across on table A-1. As an example for a key length of 10 bytes, the allowable blocking factors are shown on the same row and the default blocking factor is 112 which produces a key block of 8 sectors. Note that if a blocking factor of 120 is given to KSAM when the file is built, a value of 126 will be chosen, since 126 will give better space utilization. So, only the allowable blocking factors are shown in this table.

The next item to analyze is the number of levels created by such a key (10 bytes). First, it is necessary to know how many active entries or records are going to be in the stored in the data file. For this example, a value of 10,000 is chosen.

Table A-2 shows the minimum, critical, and maximum number of records that can be held at different levels for a given blocking factor. So, for a blocking factor of 112 (which is the default for a key length of 10 bytes), more than 12,000 records can be held at 2 levels. However the B-tree might split any time a key is added beyond the 6,496th key, since this is the critical value as shown in the table. The next step is to choose a blocking factor that will produce fewer levels. The next highest blocking factor taken from table A-1 is 126. Upon examining the tables it is noticed that this blocking factor also produces 3 levels, so the next value should be used, which is 140. Finally, a blocking factor that will produce 2 levels only is found. As the table shows, up to 10,080 records can be held in two levels.

Choosing a key blocking factor larger than 140 would seem appropriate, however, there would be no reduction in the number of levels, as shown in the table. Instead, an increase in the XDS size would be obtained and that is one of the resources that needs to be reduced. Hence the smallest key blocking factor that produces the minimum number of levels should be chosen to minimize I/O and XDS size.

C. SUMMARY.

To optimize the design and resources that a KSAM file set utilizes the appropriate blocking factors for both the data file and the key file should be chosen.
For the data file chose a blocking factor that minimizes disc space and disc I/O per transaction.
For the key file, a key blocking factor that minimizes the number of levels in the B-tree should be chosen. This key blocking factor can be obtained by using tables A-1 and A-2.

```
      A                                                            A
     A A                                                          A A
    A   A                                                        A   A
   A     A                                                      A     A
   A     A                                                      A     A
   AAAAAAA                                                      AAAAAAA
   A     A                                                      A     A
   A     APPPPPPPP                                              A     A
   A     A P      P                                             A     A
           P      P
           P      P
           PPPPPPP
           P
           P
           P     PPPPPPPP
          ppp     P      P                *****************************
                  P      P                *                           *
                  P      P                * MISCELLANEOUS KSAM TABLES *
                  PPPPPPP                 *                           *
                  P                       *****************************
                  P
                  P      EEEEEEEE
                 PPP     E      E
                         E
                         E    E
                         EEEEEE
                         E    E
                         E
                         E      ENN      NNN
                         EEEEEEEE NN      N
                                  N N     N
                                  N  N    N
                                  N   N   N
                                  N    N  N
                                  N     N N
                                  N      NN DDDDDDD
                                 NNN      N  D     D
                                             D      D
*****************************                D      D
*                           *                D      D
* MISCELLANEOUS KSAM TABLES *                D      D
*                           *                D      D
*****************************                D     D IIIIIII
                                            DDDDDDD     I
                                                        I
                                                        I
                                                        I
                                                        I
                                                        I
   A                                                    I   XX     XX
  A A                                                IIIIIII X     X
 A   A                                                        X   X
A     A                                                        X X
A     A                                                         X
AAAAAAA                                                        X X
A     A                                                       X   X
A     A                                                      X     X
A     A                                                     XX     XX
```

K S A M Table A-1

<u>KSAM BLOCKING FACTOR FOR KEY LENGTH VS BLOCK SIZE</u>

	BF \	BLOCK SIZE IN SECTORS															
									1K								2K
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	2	24	50	74	100	126	152	178	202	228	254	280	306	330	356	382	408
K	4	20	40	62	84	104	126	148	168	190	212	232	254	276	296	318	340
E	6	16	34	54	72	90	108	126	144	162	182	200	218	236	254	272	290
Y	8	14	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254
	10	12	26	42	56	70	84	98	112	126	140	154	170	184	198	212	226
L	12	12	24	36	50	62	76	88	100	114	126	140	152	164	178	190	204
E	14	10	22	34	46	56	68	80	92	104	114	126	138	150	162	174	184
N	16	10	20	30	42	52	62	74	84	94	106	116	126	138	148	158	170
G	18	8	18	28	38	48	58	68	78	88	98	106	116	126	136	146	156
T	20	8	16	26	36	44	54	62	72	80	90	100	108	118	126	136	144
H	22	8	16	24	32	42	50	58	66	76	84	92	102	110	118	126	136
	24	6	14	22	30	38	46	54	62	70	78	86	94	102	110	118	126
I	26	6	14	22	28	36	44	52	58	66	74	82	90	96	104	112	120
N	28	6	12	20	28	34	42	48	56	62	70	76	84	92	98	106	112
	30	6	12	18	26	32	40	46	52	60	66	72	80	86	94	100	106
B	32	6	12	18	24	30	38	44	50	56	62	70	76	82	88	94	102
Y	34	4	10	18	24	30	36	42	48	54	60	66	72	78	84	90	96
T	36	4	10	16	22	28	34	40	46	52	56	62	68	74	80	86	92
E	38	4	10	16	22	26	32	38	44	48	54	60	66	72	76	82	88
S	40	4	10	14	20	26	30	36	42	46	52	58	62	68	74	78	84
	42	4	10	14	20	24	30	34	40	44	50	56	60	66	70	76	80
	44	4	8	14	18	24	28	34	38	44	48	52	58	62	68	72	78
	46	4	8	14	18	22	28	32	36	42	46	50	56	60	66	70	74
	48	4	8	12	18	22	26	30	36	40	44	50	54	58	62	68	72
	50	4	8	12	16	20	26	30	34	38	42	48	52	56	60	66	70

^
default value

NOTE:
The key length is only shown in even number of bytes since the key entries are allocated on word boundaries.

K S A M TABLE A-2

MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max		B-tree level				
		1	2	3	4	5
		1	5	17	53	161
K	4	4	16	52	160	484
		4	24	124	624	3,124
E		1	7	31	127	511
	6	6	30	126	510	2,046
Y		6	48	342	2,400	16,806
		1	9	49	249	1,249
	8	8	48	248	1,248	6,248
		8	80	728	6,560	59,048
B		1	11	71	431	2,591
	10	10	70	430	2,590	15,550
L		10	120	1,330	14,640	161,050
		1	13	97	685	4,801
O	12	12	96	684	4,800	33,612
		12	168	2,196	28,560	371,292
C		1	15	127	1,023	8,191
	14	14	126	1,022	8,190	65,534
K		14	224	3,374	50,624	759,374
		1	17	161	1,457	13,121
	16	16	160	1,456	13,120	118,096
		16	288	4,912	83,520	1,419,856
F		1	19	199	1,999	19,999
	18	18	198	1,998	19,998	199,998
A		18	360	6,858	130,320	2,476,098
		1	21	241	2,661	29,281
C	20	20	240	2,660	29,280	322,100
		20	440	9,260	194,480	4,084,100
T		1	23	287	3,455	41,471
	22	22	286	3,454	41,470	497,662
O		22	528	12,166	279,840	6,436,342
		1	25	337	4,393	57,121
R	24	24	336	4,392	57,120	742,584
		24	624	15,624	390,624	9,765,624
		1	27	391	5,487	76,831
	26	26	390	5,486	76,830	1,075,646
		26	728	19,682	531,440	14,348,906
		1	29	449	6,749	101,249
	28	28	448	6,748	101,248	1,518,748
		28	840	24,388	707,280	20,511,148
		1	31	511	8,191	131,071
	30	30	510	8,190	131,070	2,097,150
		30	960	29,790	923,520	28,629,150
		1	33	577	9,825	167,041
	32	32	576	9,824	167,040	2,839,712
		32	1,088	35,936	1,185,920	39,135,392
		1	35	647	11,663	209,951
	34	34	646	11,662	209,950	3,779,134
		34	1,224	42,874	1,500,624	52,521,874

K S A M TABLE A-2

MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

	Min Crit Max	B-tree level				
		1	2	3	4	5
		1	37	721	13,717	260,641
K	36	36	720	13,716	260,640	4,952,196
		36	1,368	50,652	1,874,160	69,343,956
E		1	39	799	15,999	319,999
	38	38	798	15,998	319,998	6,399,998
Y		38	1,520	59,318	2,313,440	90,224,198
		1	41	881	18,521	388,961
	40	40	880	18,520	388,960	8,168,200
		40	1,680	68,920	2,825,760	115,856,200
B		1	43	967	21,295	468,511
	42	42	966	21,294	468,510	10,307,262
L		42	1,848	79,506	3,418,800	147,008,442
		1	45	1,057	24,333	559,681
O	44	44	1,056	24,332	559,680	12,872,684
		44	2,024	91,124	4,100,624	184,528,124
C		1	47	1,151	27,647	663,551
	46	46	1,150	27,646	663,550	15,925,246
K		46	2,208	103,822	4,879,680	229,345,006
		1	49	1,249	31,249	781,249
	48	48	1,248	31,248	781,248	19,531,248
		48	2,400	117,648	5,764,800	282,475,248
F		1	51	1,351	35,151	913,951
	50	50	1,350	35,150	913,950	23,762,750
A		50	2,600	132,650	6,765,200	345,025,250
		1	53	1,457	39,365	1,062,881
C	52	52	1,456	39,364	1,062,880	28,697,812
		52	2,808	148,876	7,890,480	418,195,492
T		1	55	1,567	43,903	1,229,311
	54	54	1,566	43,902	1,229,310	34,420,734
O		54	3,024	166,374	9,150,624	503,284,374
		1	57	1,681	48,777	1,414,561
R	56	56	1,680	48,776	1,414,560	41,022,296
		56	3,248	185,192	10,556,000	601,692,056
		1	59	1,799	53,999	1,619,999
	58	58	1,798	53,998	1,619,998	48,599,998
		58	3,480	205,378	12,117,360	714,924,298
		1	61	1,921	59,581	1,847,041
	60	60	1,920	59,580	1,847,040	57,258,300
		60	3,720	226,980	13,845,840	844,596,300
		1	63	2,047	65,535	2,097,151
	62	62	2,046	65,534	2,097,150	67,108,862
		62	3,968	250,046	15,752,960	992,436,542
		1	67	2,311	78,607	2,672,671
	66	66	2,310	78,606	2,672,670	90,870,846
		66	4,488	300,762	20,151,120	1,350,125,106
		1	69	2,449	85,749	3,001,249
	68	68	2,448	85,748	3,001,248	105,043,748
		68	4,760	328,508	22,667,120	1,564,031,348

K S A M TABLE A-2

MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max		B-tree level				
		1	2	3	4	5
		1	71	2,591	93,311	3,359,231
K	70	70	2,590	93,310	3,359,230	120,932,350
		70	5,040	357,910	25,411,680	1,804,229,350
E		1	73	2,737	101,305	3,748,321
	72	72	2,736	101,304	3,748,320	138,687,912
Y		72	5,328	389,016	28,398,240	2,073,071,592
		1	75	2,887	109,743	4,170,271
	74	74	2,886	109,742	4,170,270	158,470,334
		74	5,624	421,874	31,640,624	##############
B		1	77	3,041	118,637	4,626,881
	76	76	3,040	118,636	4,626,880	180,448,396
L		76	5,928	456,532	35,153,040	##############
		1	79	3,199	127,999	5,119,999
O	78	78	3,198	127,998	5,119,998	204,799,998
		78	6,240	493,038	38,950,080	##############
C		1	81	3,361	137,841	5,651,521
	80	80	3,360	137,840	5,651,520	231,712,400
K		80	6,560	531,440	43,046,720	##############
		1	83	3,527	148,175	6,223,391
	82	82	3,526	148,174	6,223,390	261,382,462
		82	6,888	571,786	47,458,320	##############
F		1	85	3,697	159,013	6,837,601
	84	84	3,696	159,012	6,837,600	294,016,884
A		84	7,224	614,124	52,200,624	##############
		1	87	3,871	170,367	7,496,191
C	86	86	3,870	170,366	7,496,190	329,832,446
		86	7,568	658,502	57,289,760	##############
T		1	89	4,049	182,249	8,201,249
	88	88	4,048	182,248	8,201,248	369,056,248
O		88	7,920	704,968	62,742,240	##############
		1	91	4,231	194,671	8,954,911
R	90	90	4,230	194,670	8,954,910	411,925,950
		90	8,280	753,570	68,574,960	##############
		1	93	4,417	207,645	9,759,361
	92	92	4,416	207,644	9,759,360	458,690,012
		92	8,648	804,356	74,805,200	##############
		1	95	4,607	221,183	10,616,831
	94	94	4,606	221,182	10,616,830	509,607,934
		94	9,024	857,374	81,450,624	##############
		1	97	4,801	235,297	11,529,601
	96	96	4,800	235,296	11,529,600	564,950,496
		96	9,408	912,672	88,529,280	##############
		1	99	4,999	249,999	12,499,999
	98	98	4,998	249,998	12,499,998	624,999,998
		98	9,800	970,298	96,059,600	##############
		1	101	5,201	265,301	13,530,401
	100	100	5,200	265,300	13,530,400	690,050,500
		100	10,200	1,030,300	104,060,400	##############

K S A M TABLE A-2

MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

	Min Crit Max	B-tree level				
		1	2	3	4	5
		1	103	5,407	281,215	14,623,231
K	102	102	5,406	281,214	14,623,230	760,408,062
		102	10,608	1,092,726	112,550,880	##############
E		1	105	5,617	297,753	15,780,961
	104	104	5,616	297,752	15,780,960	836,390,984
Y		104	11,024	1,157,624	121,550,624	##############
		1	107	5,831	314,927	17,006,111
	106	106	5,830	314,926	17,006,110	918,330,046
		106	11,448	1,225,042	131,079,600	##############
B		1	109	6,049	332,749	18,301,249
	108	108	6,048	332,748	18,301,248	1,006,568,748
L		108	11,880	1,295,028	141,158,160	##############
		1	111	6,271	351,231	19,668,991
O	110	110	6,270	351,230	19,668,990	1,101,463,550
		110	12,320	1,367,630	151,807,040	##############
C		1	113	6,497	370,385	21,112,001
	112	112	6,496	370,384	21,112,000	1,203,384,112
K		112	12,768	1,442,896	163,047,360	##############
		1	115	6,727	390,223	22,632,991
	114	114	6,726	390,222	22,632,990	1,312,713,534
		114	13,224	1,520,874	174,900,624	##############
F		1	117	6,961	410,757	24,234,721
	116	116	6,960	410,756	24,234,720	1,429,848,596
A		116	13,688	1,601,612	187,388,720	##############
		1	119	7,199	431,999	25,919,999
C	118	118	7,198	431,998	25,919,998	1,555,199,998
		118	14,160	1,685,158	200,533,920	##############
T		1	121	7,441	453,961	27,691,681
	120	120	7,440	453,960	27,691,680	1,689,192,600
O		120	14,640	1,771,560	214,358,880	##############
		1	127	8,191	524,287	33,554,431
R	126	126	8,190	524,286	33,554,430	2,147,483,646
		126	16,128	2,048,382	260,144,640	##############
		1	137	9,521	657,017	45,334,241
	136	136	9,520	657,016	45,334,240	##############
		136	18,768	2,571,352	352,275,360	##############
		1	139	9,799	685,999	48,019,999
	138	138	9,798	685,998	48,019,998	##############
		138	19,320	2,685,618	373,301,040	##############
		1	141	10,081	715,821	50,823,361
	140	140	10,080	715,820	50,823,360	##############
		140	19,880	2,803,220	395,254,160	##############
		1	143	10,367	746,495	53,747,711
	142	142	10,366	746,494	53,747,710	##############
		142	20,448	2,924,206	418,161,600	##############
		1	145	10,657	778,033	56,796,481
	144	144	10,656	778,032	56,796,480	##############
		144	21,024	3,048,624	442,050,624	##############

K S A M TABLE A-2

MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max		B-tree level				
		1	2	3	4	5
		1	147	10,951	810,447	59,973,151
K	146	146	10,950	810,446	59,973,150	##############
		146	21,608	3,176,522	466,948,880	##############
E		1	149	11,249	843,749	63,281,249
	148	148	11,248	843,748	63,281,248	##############
Y		148	22,200	3,307,948	492,884,400	##############
		1	151	11,551	877,951	66,724,351
	150	150	11,550	877,950	66,724,350	##############
		150	22,800	3,442,950	519,885,600	##############
B		1	153	11,857	913,065	70,306,081
	152	152	11,856	913,064	70,306,080	##############
L		152	23,408	3,581,576	547,981,280	##############
		1	155	12,167	949,103	74,030,111
O	154	154	12,166	949,102	74,030,110	##############
		154	24,024	3,723,874	577,200,624	##############
C		1	157	12,481	986,077	77,900,161
	156	156	12,480	986,076	77,900,160	##############
K		156	24,648	3,869,892	607,573,200	##############
		1	159	12,799	1,023,999	81,919,999
	158	158	12,798	1,023,998	81,919,998	##############
		158	25,280	4,019,678	639,128,960	##############
F		1	163	13,447	1,102,735	90,424,351
	162	162	13,446	1,102,734	90,424,350	##############
A		162	26,568	4,330,746	705,911,760	##############
		1	165	13,777	1,143,573	94,916,641
C	164	164	13,776	1,143,572	94,916,640	##############
		164	27,224	4,492,124	741,200,624	##############
T		1	169	14,449	1,228,249	104,401,249
	168	168	14,448	1,228,248	104,401,248	##############
O		168	28,560	4,826,808	815,730,720	##############
		1	171	14,791	1,272,111	109,401,631
R	170	170	14,790	1,272,110	109,401,630	##############
		170	29,240	5,000,210	855,036,080	##############
		1	175	15,487	1,362,943	119,939,071
	174	174	15,486	1,362,942	119,939,070	##############
		174	30,624	5,359,374	937,890,624	##############
		1	179	16,199	1,457,999	131,219,999
	178	178	16,198	1,457,998	131,219,998	##############
		178	32,040	5,735,338	1,026,625,680	##############
		1	183	16,927	1,557,375	143,278,591
	182	182	16,926	1,557,374	143,278,590	##############
		182	33,488	6,128,486	1,121,513,120	##############
		1	185	17,297	1,608,713	149,610,401
	184	184	17,296	1,608,712	149,610,400	##############
		184	34,224	6,331,624	1,171,350,624	##############
		1	191	18,431	1,769,471	169,869,311
	190	190	18,430	1,769,470	169,869,310	##############
		190	36,480	6,967,870	1,330,863,360	##############

K S A M TABLE A-2

MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS
FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max	B-tree level				
	1	2	3	4	5
	1	199	19,999	1,999,999	199,999,999
K 198	198	19,998	1,999,998	199,999,998	##############
	198	39,600	7,880,598	1,568,239,200	##############
E	1	201	20,401	2,060,601	208,120,801
200	200	20,400	2,060,600	208,120,800	##############
Y	200	40,400	8,120,600	1,632,240,800	##############
	1	203	20,807	2,122,415	216,486,431
202	202	20,806	2,122,414	216,486,430	##############
	202	41,208	8,365,426	1,698,181,680	##############
B	1	205	21,217	2,185,453	225,101,761
204	204	21,216	2,185,452	225,101,760	##############
L	204	42,024	8,615,124	1,766,100,624	##############
	1	207	21,631	2,249,727	233,971,711
O 206	206	21,630	2,249,726	233,971,710	##############
	206	42,848	8,869,742	1,836,036,800	##############
C	1	213	22,897	2,450,085	262,159,201
212	212	22,896	2,450,084	262,159,200	##############
K	212	45,368	9,663,596	2,058,346,160	##############
	1	219	24,199	2,661,999	292,819,999
218	218	24,198	2,661,998	292,819,998	##############
	218	47,960	10,503,458	#############	##############
F	1	223	25,087	2,809,855	314,703,871
222	222	25,086	2,809,854	314,703,870	##############
A	222	49,728	11,089,566	#############	##############
	1	227	25,991	2,963,087	337,792,031
C 226	226	25,990	2,963,086	337,792,030	##############
	226	51,528	11,697,082	#############	##############
T	1	229	26,449	3,041,749	349,801,249
228	228	26,448	3,041,748	349,801,248	##############
O	228	52,440	12,008,988	#############	##############
	1	233	27,377	3,203,225	374,777,441
R 232	232	27,376	3,203,224	374,777,440	##############
	232	54,288	12,649,336	#############	##############
	1	237	28,321	3,370,317	401,067,841
236	236	28,320	3,370,316	401,067,840	##############
	236	56,168	13,312,052	#############	##############
	1	239	28,799	3,455,999	414,719,999
238	238	28,798	3,455,998	414,719,998	##############
	238	57,120	13,651,918	#############	##############
	1	255	32,767	4,194,303	536,870,911
254	254	32,766	4,194,302	536,870,910	##############
	254	65,024	16,581,374	#############	##############
	1	273	37,537	5,142,705	704,550,721
272	272	37,536	5,142,704	704,550,720	##############
	272	74,528	20,346,416	#############	##############
	1	277	38,641	5,371,237	746,602,081
276	276	38,640	5,371,236	746,602,080	##############
	276	76,728	21,253,932	#############	##############

K S A M TABLE A-2

MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max		B-tree level				
		1	2	3	4	5
		1	281	39,761	5,606,441	790,508,321
K	280	280	39,760	5,606,440	790,508,320	##############
		280	78,960	22,188,040	#############	##############
E		1	291	42,631	6,224,271	908,743,711
	290	290	42,630	6,224,270	908,743,710	##############
Y		290	84,680	24,642,170	#############	##############
		1	297	44,401	6,615,897	985,768,801
	296	296	44,400	6,615,896	985,768,800	##############
B		296	88,208	26,198,072	#############	##############
		1	307	47,431	7,304,527	1,124,897,311
L	306	306	47,430	7,304,526	1,124,897,310	##############
		306	94,248	28,934,442	#############	##############
O		1	319	51,199	8,191,999	1,310,719,999
	318	318	51,198	8,191,998	1,310,719,998	##############
C		318	101,760	32,461,758	#############	##############
		1	331	55,111	9,148,591	1,518,666,271
K	330	330	55,110	9,148,590	1,518,666,270	##############
		330	109,560	36,264,690	#############	##############
		1	341	58,481	10,000,421	1,710,072,161
F	340	340	58,480	10,000,420	1,710,072,160	##############
		340	116,280	39,651,820	#############	##############
A		1	357	64,081	11,470,677	2,053,251,361
	356	356	64,080	11,470,676	2,053,251,360	##############
C		356	127,448	45,499,292	#############	##############
		1	383	73,727	14,155,775	
T	82	382	73,726	14,155,774	##############	
		382	146,688	56,181,886	##############	
O		1	409	84,049	17,230,249	
	408	408	84,048	17,230,248	#############	
R		408	167,280	68,417,928	#############	

The minimum value is the absolute minimum number of keys that a B-tree can hold for a given level, if a key is deleted then the B-tree will contract one level. An example is shown in the following figure of a 2 level (L) B-tree with a key blocking factor (F) of 4 and with minimum number of keys.

```
[*]300[*]\\\[*]\\\[*]\\\[*]
 |     |
 |    [*]400[*]500[*]\\\[*]\\\[*]
 |
[*]100[*]200[*]\\\[*]\\\[*]
```

The minimum number of keys (MinK) is given by the formula:

```
Mink = 2 * (F/2 + 1)**(L-1) - 1
```

The 'critical' value is the minimum number of keys that the B-tree can hold on a given level before splitting and creating another level. This value is given by subtracting 1 from the minimum value of the next higher level.

```
[*] 30[*] 60[*] 90[*]120[*]
 |     |     |     |     |
 |     |     |     |    [*]130[*]140[*]150[*]160[*]
 |     |     |     |
 |     |     |    [*]100[*]110[*]\\\[*]\\\[*]
 |     |     |
 |     |    [*] 70[*] 80[*]\\\[*]\\\[*]
 |     |
 |    [*] 40[*] 50[*]\\\[*]\\\[*]
 |
[*] 10[*] 20[*]\\\[*]\\\[*]
```

The critical number of keys (CritK) is given by the formula:

```
CritK = 2 * (F/2 + 1)**(L) - 2
```

The maximum value is the maximum number of keys that the B-tree could hold if it would be possible to load it with no 'holes' whatsoever. An illustration of this concept is shown below on a 2 level B-tree with a key blocking factor of 4 and maximum number of keys.

```
[*]  5[*] 10[*] 15[*] 20[*]
 |     |     |     |     |
 |     |     |     |    [*] 21[*] 22[*] 23[*] 24[*]
 |     |     |     |
 |     |     |    [*] 16[*] 17[*] 18[*] 19[*]
 |     |     |
 |     |    [*] 11[*] 12[*] 13[*] 14[*]
 |     |
 |    [*]  6[*]  7[*]  8[*]  9[*]
 |
[*]  1[*]  2[*]  3[*]  4[*]
```

The maximum number of keys (MaxK) is given by the formula:

```
MaxK = (F + 1)**(L) - 1
```

Using Interprocess Communication

Gregory A. Grimm

HP Computer Systems Division
MTS Development Engineer
19447 Pruneridge Ave.
Cupertino California 95014

With the advent of the file system Inter-process Communication (IPC) facility, Users of the HP3000 computer system under MPE have more tools available to them than ever before for the development of applications. This paper will examine IPC as a whole; therefore both message files and software interrupts will be included in the discussion.

The examination of IPC is focused in three main areas: firstly, the possible reasoning behind the use of IPC is dicussed including system management and actual process to process communication. Second, the current tools (Intrinsics) available to the user is discussed in enough detail so that they will become comfortable to use. Lastly, an example of the use of IPC in an application environment is discussed leaving the user with a better understanding of file system IPC and its use in solving application problems.

File system IPC uses a special file called a message file. A message file can be thought of as a FIFO queue. When one reads the first record from the file (queue) it is logically deleted from the file. Writing a record to the message file logically adds the record to the end of the file (queue). Since IPC is part of the file system, any operations on the message file use file system intrinsics.

File system IPC is a tool that is used by application programmers to overcome previous constraints found on the HP 3000 of the following general types. The first type of problem is that of system management. The HP 3000 running under MPE IV can cause restrictions such that the applications programmer may not be able to get the most out of his machine. The first and probabally the largest example of this is the problem of the 32K stack.

The HP3000 has a maximum stack size of 32K minus the amount that is used by the operating system. This can severely hamper the programmer so that he/she cannot fit the applications program into the machine at one time. There are two solutions to this problem. The first is for the programmer to use extra data segments along with his stack. The programmer then has to learn a new set of intrinsics to manage the extra data segments. These intrinsics, GETDSEG, DMOVEIN, DMOVEOUT, etc. are different than any file system intrinsics that have been used so far. In addition these intrinsics cause high instruction overhead.

A second solution to the problem is the use of the IPC facility in the MPE file system. There are three main advantages to using the IPC facility. The first is that to use IPC one only needs to know how to use the file system instrinsics. The familiar FOPEN, FREAD, FWRITE, FCONTROL and FCLOSE are used instead of a new set of intrinsics. The second advantage is that file system IPC is less complex than using extra data segments. The last advantage is that most application programs that are written using a top-down structured approach should be able to use the IPC facility with few problems. The top-down modular approach requires that each module be a self contained part of the program with a clearly defined interface between each modules. Figure 1 shows the top 2 levels in the hierarchy chart of an application program that is ideal to use IPC.

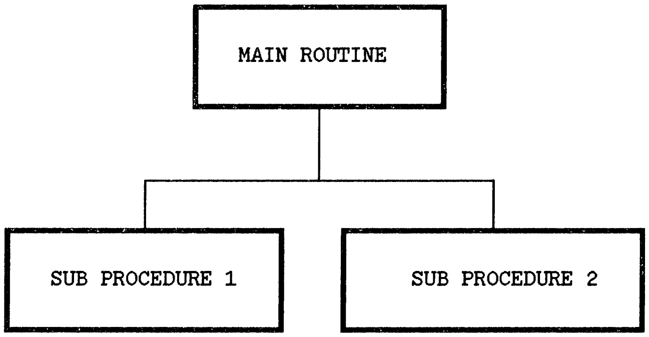


Figure 1

File system IPC uses a special kind of file called a message file in order to communicate between processes. Each *module* in our hierarchy chart in figure 1 would become a process on their own. Of course each process (or module) would have its own stack thus solving the stack size problem. The clearly defined interface for each module would become message files. Each message file would be uni-directional thus insuring that no other program would distroy the parameters to each module. This is shown in figure 2.

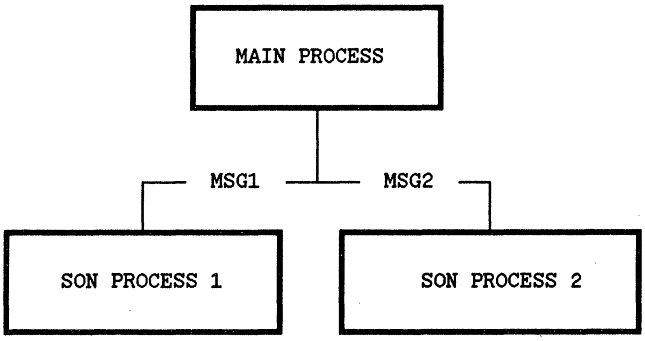


Figure 2

For each procedure interface two message files would be used. The first would contain all the values that are being passed to the procedure. The second message file would contain all the values that are being passed back to the caller. Since a module could be called from more than one place there must be mechanisim to determine where the procedure was called from. This mechanisim is provided through IPC message files via writer IDs.

In addition to stack size limitations some performance problems in application programs can also be addressed such as input devices that require constant attention. With the use of IPC message files and separate processes each of the input devices can have its own process constantly reading from the device and convey the need for some processing by a separate process via message files. This allows the process that is attending the input device to do only that and not get slowed down by doing other processing. This process stops only if it can not write to the message file (extended wait). This can be prevented by using nowait I/O with message files and by making the message files larger than is likely to be needed. In this way the input device process never needs to be waited.

Another performance problem can be solved by having the different modules (processes) on different machines. Each node of a data communications network can do one small task of the whole job. In this way each process can compete for resources on his own machine. An additional advantage to this set-up is the fact that if one machine goes down the whole application does not. In fact one could program redundancy into the application so that the application could survive up to one half of the machines to crash.

Another common problem that modules in a computer program have is the tendency for them to accidentally write over each other's important data. This problem can be partially solved by the use of IPC. Since each module has his own stack, another module (process) cannot write on his own stack. This allows important data to be protected by the module that owns the data. Of course this will not protect the data if there is a bug in the module itself; however each module is small enough to verify its correctness.

IPC is very useful during the development phase of an application program for program testing. There are generally two types of testing methods: top-down and bottom-up testing. With top-down testing the top level modules are tested first, the bottom level are tested last. When the bottom level is tested the application program is finished. Bottom-up testing is a strategy in which the bottom levels of a program are tested first. This is done by writing driver programs to test each module or set of modules. This approach is often mixed with top-down testing to get what is sometimes called middle-out testing. Using middle-out testing allows the use of top-down testing for most of the modules but uses bottom-up testing for critical modules near the bottom of the hierarchy chart.

File system IPC facilitates bottom-up testing. Since the calling sequence is done through message files, it is an easy matter of producing test cases for the module by using simple editor files. To check the results of the module all one need do is verify that the contents of the outgoing message file is correct. Figure 3 shows this relationship.

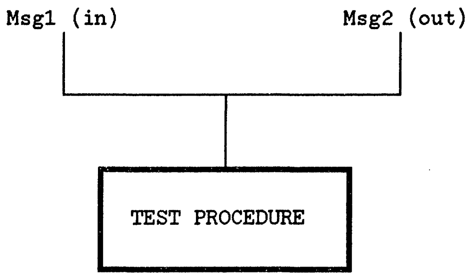


Figure 3

The only difficulty to this method of testing involves a module that does some type of updating of a disc resident data structure. To insure correctness of the module, the data structure must be verified to be correct after the test run of the module.

Another very useful way of applying file system IPC is using soft interrupts. When an I/O is started (no wait) to a message file, an interrupt handler is set up and when the I/O completes, MPE interrupts the process that started the I/O.

The final category of problems solved by file system IPC is actual process to process communication. For applications that can be partially overlapped in time, the applications programmer needs a method to synchronize the processes. The easiest way to do this is to use IPC message files. In order to check point two processes that run concurrently all that needs to be done is to have two message files. Each process opens up one file as a reader and one file as a writer (the file opened as a reader would be opened as a writer by the other process). When a process is ready to check point, it writes a record to the message file it has open as a writer. It then issues a read on the file it has opened as a reader (with wait option). When the other process is ready to check point he writes a record to the file which is now read by the other process. At this point each process is off and running. If additional information had needed to be exchanged between the two processes it could have been done during these operations.

To use file system IPC the application programmer must master file system intrinsics. The following is a summary of what each intrinsic will do (more information can be gained from your Intrinsic Manual or your File System Reference Manual):

FOPEN: Establishes the connection to the message file.

A process will be identified as a reader (one who reads from the message file) or as a writer (one who writes to the message file). Care must be taken to correctly set the AOPTIONs and FOPTIONs of the file. In addition the condition code should always be checked (as in all intrinsic calls).

FREAD: Logically deletes one record from the message file and returns it to the user.

If the process tries to read from an empty file which has at least one writer open, the process will be blocked until there is a record written to the file or all writers have closed the file. If no writers had the file open when the reader reads from the empty file, CCG will be returned unless it is the first read after opening the file or extended wait is in effect.

FWRITE Logically adds one record to the end of the message file.

If the process tries to write to a full file which has at least one reader open, the process will be blocked until there is one record read. If no readers have the file open when the writer wrote to the full file, CCG will be returned unless it is the first write after opening the file or extended wait is in effect.

FCONTROL Sets control functions to a message file.

- Disable/Enable extended wait (see FREAD/FWRITE)
- Disable/Enable reading the writer ID (each writer is uniquely identified and when a reader can read the writer ID he can determine who wrote the current record)
- Disable/Enable Non distructive read (a reader can read the next record without logically deleting it from the file)
- Arm/Disarm soft interrupts (way to pass the address of the trap handler for soft interrupts).

FINTSTAT Enables soft interrupts for the process.

FINTEXIT Exits from a soft interrupt trap handler.

FFILEINFO Returns status information.

- Number of readers
- Number of writers
- Plabel of the soft interrupt trap handler

FCLOSE Breaks the connection to the message file.

More detail of each file system intrinsic can be found in the Intrinsics Manual or the File System Reference Manual.

Finally, an example of using IPC in an application is presented below.

> A customer has a set of *n* I/O devices that will cause some data to be updated. These I/O devices need constant attention. If we do not clean up an I/O before another comes, we will lose the first I/O. We can assume that there is enough time between I/Os on the I/O devices to complete up to two FWRITEs to a message file. We cannot count on any more time than that. The data consists of an IMAGE data base on one HP3000 with a set of MPE files that also need be updated on another HP3000. The customer has a separate HP3000 that will moniter the I/O devices.

This application has a solution using IPC. Let us discuss the solution in terms of data flow. Each I/O device requires a process devoted to reading data from it. When the process (and the device) is started up it opens up a message file on the computer that is monitoring the devices. To avoid data over-flow the message file must be created large enough. Once the message file is opened the process reads (with wait) from the device and writes the result to a message file. The data that has been taken from the device is now cannot be lost due to over-writing. This message file is the raw data file.

On one of the other two machines there is a process that reads from all the raw data files (one for each I/O device) and determines what kind of transaction needs to be done. It then writes the data into a transaction message file. Each type of transaction has a transaction file.

Each transaction has a process on the MPE file machine which reads the data from the transaction message file and update the MPE files. It then writes the data to a IMAGE message file. In this way a process on the IMAGE machine can read the data from the IMAGE message file and update the IMAGE data base. Figure 4 shows how this all fits together.

In this way we could solve the customer's problem by using file system IPC. Each of the processes discussed here would be fairly simple to write, thus increasing our chances of few bugs.

File system Interprocess Communication is a powerful tool that can be used to solve many application problems on the HP3000. In addition it has the following advantages: most of the machine.

* usable by high level languages
* uses file system security
* debugging ease
* remote file access

These advantages make IPC helpful in many applications.

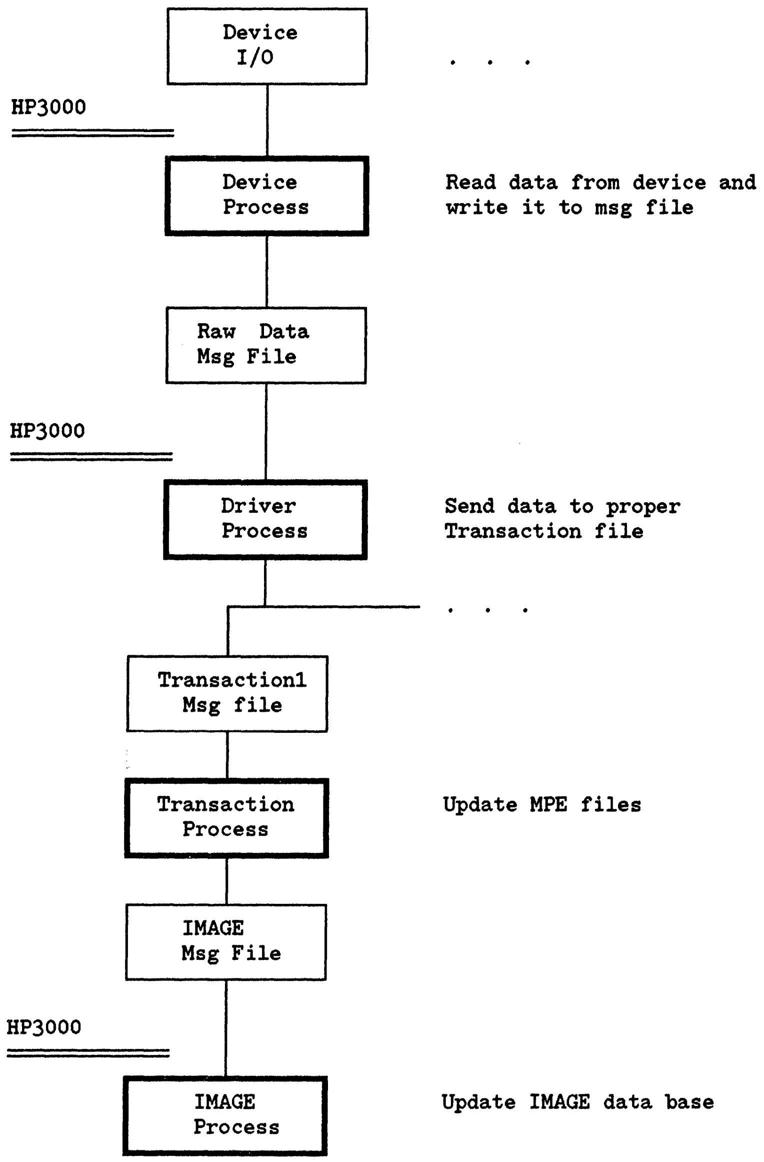

Device I/O
. . .
HP3000
Device Process
Read data from device and write it to msg file
Raw Data Msg File
HP3000
Driver Process
Send data to proper Transaction file
. . .
Transaction1 Msg file
Transaction Process
Update MPE files
IMAGE Msg File
HP3000
IMAGE Process
Update IMAGE data base

Figure 4

THE STRUCTURE OF APPLICATION PROGRAM SAMPLER (APS)/3000

Abbas Rafii

Hewlett-Packard
Computer Systems Division
19447 Pruneridge Avenune
Cupertino, Ca. 95014
(408)-725-8111

ABSTRACT:

Application Program SAMPLER (APS/3000) is a friendly interactive performance measurement software product for tuning application programs on the HP 3000. This program can be used to produce program CPU and wait time profiles at several levels of detail in terms of logical structures of the source program such as segments, procedures and address regions. A unique feature of the product permits the programmer to see both the time spent in the user code and the time spent executing system (an SL) code on behalf of calls from user code. This tool can be used with most major languages. In this paper the design, internal structure and application of APS/3000 are presented. A status sampling technique is discussed. Direct and indirect cost components are defined. Certain interesting aspects of data analysis and on-line data presentation techniques are described. Finally, a case study is presented.

1. *Introduction*

Performance evaluation tools are increasingly being used to determine the dynamic behavior of various software systems, ranging from application programs to the components of complex operating systems. The measurement results are normally used to locate the system bottle-necks for tuning purposes. Additionally, performance profiles can highlight possible logical errors in a program when the actual dynamic behavior of the program does not conform to the expected results.

Software performance tools report their results in terms of either number of events or percentages of resource utilization [1,2,3]. Examples of events are number of reads, writes, process launches, interrupts, etc. Utilization figures are normally given for CPU, IO channels, network channels, etc.

APS/3000 provides CPU utilization profiles of software systems under normal load without the need for special program instrumentation [4,5,6]. In a typical application of the tool, the user runs the program under study through the SAMPLER interface. After the measurement is completed, the user is first presented with an overview of the behavior of his program, and then more detailed profiles of various sections of the program can be obtained interactively. With a little assistance from compilers and the segmenter, precise performance profiles can be produced in terms of programmer defined entities such as code segments, procedure names, procedure relative source line numbers and procedure relative absolute addresses. Graphic on-line summary reports can also be produced as the measurement progresses.

In the following sections of this paper, we give a brief overview of the architecture and program structure of the HP/3000 for a better understanding of the measurement mechanism and performance profiles. The measurement mechanism is then described. The concept of direct and indirect CPU utilization, which provides a very informative view of program execution cost, is given. Data reduction techniques and the requirements imposed by the interactive nature of the user interface are discussed. We conclude with a few remarks on the implementation and by presenting a case study using the tool. In the following text, the APS/3000 commands which relate to the topics being discussed are shown within {} marks [Ref. 10].

2. Machine and Program Organization

The HP/3000 is a 16-bit stack operation machine [7,8]. Process address space consists of separate variable size code and data segments. Each process executes a program file. Processes can share up to 192 global code segments. Part of the global code segments are permanently allocated to operating system (MPE) [9] functions (e.g. file system, memory manager, scheduler, network handler, etc.). The remaining segments can be dynamically allocated to frequently shared code such as compiler libraries, utilities and data base services. Each program file has up to 63 local (non-sharable) code segments. Therefore, the total code space of a program consists of 63 local code segments (containing the code a user writes) and 192 global segments. In addition to code segments, each process has its own set of data segments (including at least one stack segment).

A process is defined in the system by a Process Identification Number (PIN) and the program file it is executing. Since code and data are separate, processes can easily share the same program file.

The state of a program is kept in a four word stack marker (Figure 1). Stack markers are chained to reflect the history of procedure invocations up to the present. A stack marker defines the procedure return address which consists of a code segment number and displacement within the segment.

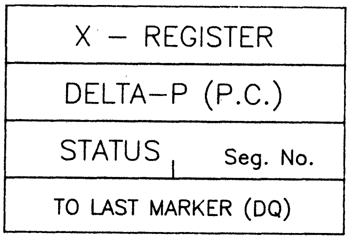


Figure 1 A stack marker

3. Measurement Mechanism

As the name implies, APS/3000 uses a sampling technique to collect its data. It is driven by a programmable software clock. The clock can be instructed to interrupt the system at fixed intervals (ranging from 5 to 1000 milliseconds). A clock interrupt is a high priority event. Therefore, when the external interrupts are not disabled, it can pre-empt any programs, and record (sample) the status of the system just prior to the interrupt. The status information provides us with the segment number and address of the interrupted instruction. In a multi-programming system, however, there is no guarantee that every time the clock ticks it interrupts the program we wish to monitor. Therefore, a screening of the samples must take place either at sampling time or later at data reduction time. For this reason, we need to indentify the name of the interrupted program and the process number (PIN) of the correspoding interrupted process. If we screen the samples based on the program identification {AM}, we are actually studying the shared execution of a program by all processes running that program. This is useful for programs such as editors, compilers, etc. If we screen the samples based on the process identification {RM}, the resulting data can be used to provide the behavior of a program with a single user for a specific set of input data. The latter option is useful for tuning application programs. APS/3000 is fairly flexible in the screening of its data. Screening can be done at measurement time and/or later at data reduction time. Any combination of process number and/or program name can be used as screening parameters. It is therefore possible to measure the execution of several programs at once. For the casual user, the choice of screening parameters is done automatically.

In Figure 2, the organization of APS/3000 is shown. It consists of four internal modules, each independent but communicating processes, called SAMPLER, LOGGER, DISPLAY and ANALYZER. SAMPLER module sets up the measurement and operates the sampling mechanism. The clock interrupt receiver contains the logic necessary to identify the processes or programs under study. It compares the status of the interrupted system with the data stored in a known data segment. If any component of the system under study is found to have been active just prior to the interrupt, the system status is sampled. Each sample contains sufficient information to allow the reconstruction of the system's activity later at data reduction time. Among other things, a sample contains a process number, code segment number, interrupted instruction address within the segment and some control information. Samples are collected in buffers in a data segment. When a buffer is filled, the LOGGER module is activated to store the buffer on disk or tape via the file system. Because of the real time constraints of the measurement, the file system must be fast enough to complete its operation before the other buffer (in a double buffering scheme) is filled. The timing can be controlled by the degree of buffering and sampling interval. These parameters are pre-tuned for the user.

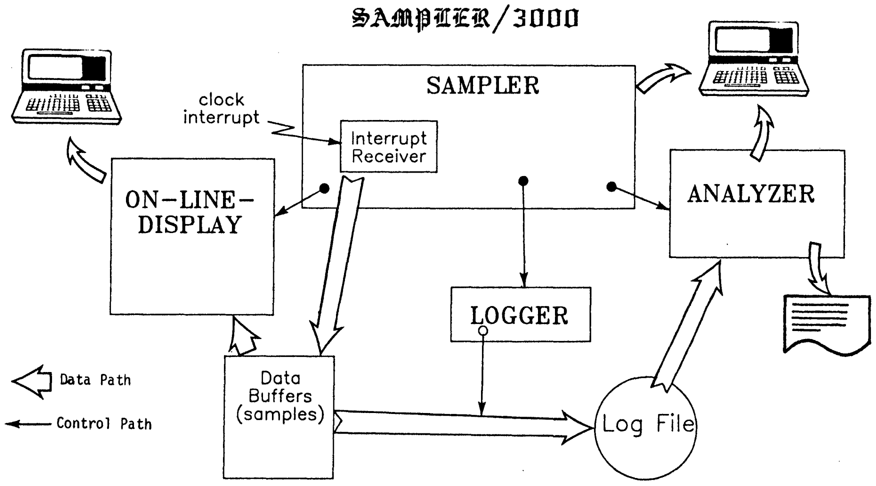


Figure 2 Organization of Application Program SAMPLER/3000

In addition to storing raw data, the clock interrupt receiver prepares summary statistics of the ongoing measurement. This is done by incrementing a set of counters in a designated data segment. There are sets of counters corresponding to all program code segments, process numbers, program file numbers, etc. These counters can be displayed for the user by the DISPLAY module {OD} while the measurement is in progress. In fact, for some applications, these summary reports are sufficient to give an overview of the CPU utilization of the system under study. When the measurement is completed (or stopped), the ANALYZER module {RE} is invoked to prepare a variety of program profiles under user control.

4. Direct and Indirect CPU Utilization

An application program frequently calls for the services of the operating system during its execution. Usual system calls are I/O requests, library accesses, general resource requests, etc. The true CPU cost of running a program, therefore, consists of the time the CPU directly executes user code plus the time it exercises the operating system routines invoked as the result of calls originating from the user code. We refer to these two program cost components as direct and indirect CPU times, respectively. Before discussing the many ways direct and indirect CPU times can be presented to give a realistic performance profile of a program, we describe the measurement technique to determine indirect costs.

In the HP/3000 execution environment under MPE operating system, system services are provided either by explicit calls to pre-defined procedures (called intrinsics) or by implicit system procedure calls generated by compilers to simulate high level language statements (e.g. PASCAL file operations).

As mentioned earlier, the global MPE segments and othe SLs occupy a known separate area in the address space of a program, namely the lower 192 code segments. When the sampling mechanism inspects the status of the system at a clock interrupt, it can determine if the interrupted code segment is a system or user segment. Focusing only on the samples for a given process number, one can obtain the distribution of program time over user segments and system segments executing on behalf of the user program. (This gives a direct CPU time profile).

The direct CPU time profiles do not, however, show what percentage of system code execution can be associated with the requests (calls) originating from user code. In other words, we would like to determine the actual processing cost of calls to the operating system functions embedded in the user source statements. We refer to this as the indirect cost (or CPU utilization) of program statements.

In order to obtain indirect cost samples we need to produce a new set of measurements, in addition to our usual samples of interrupted program status. The procedure stack markers help us to backtrack and inspect the procedure invocation history of the interrupted program [Figure 3]. The algorithm to get both direct and indirect samples is as follows: At each clock interrupt determine if the program under study is interrupted. If so, record the status of the interrupted program: <seg><displacement>. This is the direct cost sample. Then determine if the <seg> is a system or user code segment. If it is a user code segment the indirect cost does not apply. If it is a system code segment, search the procedure invocation chain of the interrupted program starting from the most recent marker. Stop searching either when the chain is exhausted or when a user segment is found along the way. In the latter case, sample the the status(i.e. point of call) in the user segment: <seg'><displacement'>.

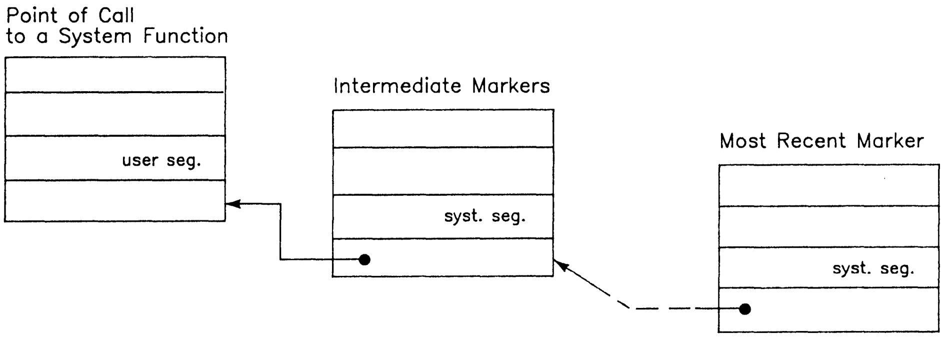


Figure 3 Procedure invocation chain leading to a user segment

The string of samples obtained in this way consist of two-tuples (<direct>, <indirect>) measurements. The <indirect> sample only applies when the <direct> sample is a system segment. In the following string of samples a, b, c, d's are user segments and x, y, z's are system segments:

```
    Time: t1 t2 t3 t4 t5 t6 t7 t8 ...
  Direct:  a  x  b  c  y  z  d  x ...
Indirect:  -  a  -  -  d  c  -  d ...
```

At time t5, for instance, system segment y is active and it is invoked by a sequence of one or more calls originating from the user segment d.

The string of samples which is obtained in this way could be processed to give a variety of information about the execution behavior of a program and its demand on the system services. The direct samples can be used to produce the usual direct CPU utilization profiles of various areas of a program. The combination of direct and indirect samples can give profiles where the total cost of operating system services requested from an area in the user code is super-imposed on the direct cost of executing the user code in the same area. These profiles give a more realistic representation of a programs's cost. Additionally, the indirect cost originating from a given area in the user code can be expanded in terms of the elements in the system code involved in servicing the user calls. The latter technique can be used in the following way. A part of the user code (e.g. a user segment) can be selected as a window into the internals of the operating systems. Those parts of the operating systems which are exercised to service the requests from the window can be determined by presenting their utilization profiles. {IS}. This type of profile is easily obtained by fixing on the direct samples of a section of the user code and including into the analysis all the corresponding indirect samples.

The following example should clarify these concepts. Consider a program section preparing and doing a WRITE operation via the file system:

```
        ..  ................

        040  MOVE BUF := " DISPLAY MESSAGE ";
        063  FWRITE (TOFILE, BUF, LEN, CONTROL);
        070  IF ERROR THEN EXIT (WITH_ERROR_NUM);
        ..  ................
        ..  . . . . . . . . .
```

Let us assume that the direct CPU measurement shows that 5% of the total program time was spent directly executing the code generated by these statements {D}. This is actually only a part of the cost of these program lines. The direct cost of FWRITE is negligible. It consists only of the code to set up the parameters and a procedure call to the system function FWRITE. The indirect cost analysis shows that 15% of total program time was spent servicing the call {I}. Therefore, the realistic cost of these statements is 20% of total program time. Furthermore, one might be interested in using the FWRITE statement as a window into the file system and obtain the distribution of the 15% indirect cost over the file system code regions which were exercised in order to service this particular FWRITE request.

Wait times can be estimated inexpensively by counting the number of clock interrupts between two successive samples. This count is kept as part of each sample. ANALYZER has commands {W, WR} which produce direct+ indirect+wait times of a program. These profiles essentially provide the response time of program statements at the time of the measurement.

5. Measurement Accuracy and Sampling Overhead

The choice of the sampling rate is a tradeoff between the desired measurement accuracy and the sampling overhead. Our experiments show that collecting a sample for every 10000 program instructions provides a balance between the two factors.

The sampling overhead is the amount of CPU time used by the sampling mechanism at every clock interrupt to collect the status of the interrupted program. In addition to CPU usage, each time a data buffer is filled, a process wake-up and a file system WRITE are generated to log the samples. The latter events are relatively infrequent (once every 210 sample).

The average CPU time overhead is a function of the type of the sample (direct or direct+indirect), the sampling rate, the fraction of time an interrupt produces a sample and some other minor factors related to the operating systems detail.

In typical applications, the sampling apparatus should not cause any perturbation on the probability of finding the program counter in a certain location. As seen by the program which runs under SAMPLER, the processor is merely slowed down by a certain percentage. The only time bias may be introduced in the measurement is when the external interrupts are kept disabled for relatively long periods of time. If a clock time out occurs while the interrupts are disabled, SAMPLER tends to sample the code at the point where interrupts are re-enabled. The effect is not considered critical in general system and application programs which do not heavily deal with interrupts.

In Tables 1 and 2, the sampling overhead are presented for different members of HP/3000 family of computers. Table 1 gives the actual CPU time from the point the clock interrupt handler is entered until the exit from this routine with a sample. An interrupt which does not produce a sample has a very low overhead (about 1/10th. of time to take a sample). Table 2 gives the average CPU usage of the sampling mechanism where at every clock interrupt a direct sample is taken under no load condition (dispatcher is being sampled constantly). In order to maintain uniform levels of accuracy and overhead accross different CPU's, SAMPLER adjusts its default sampling rate based on the speed of the underlying processor.

Type	HP3000 Series33	HP3000 III	HP3000 44	HP3000 64
Direct Sample	2 msec	1	0.6	0.3
Dir+Indir Sample	3	1.5	0.9	0.4

Table 1: Approximate time to take one sample in milliseconds

Sampling Interval	Series 33	Series III	Series 44	Series 64
5 msec	25%	13%	8%	4%
10	14%	7%	5%	2.5%
20	7%	4%	2%	1%
25	6%	3%	1.8%	0.9
50	2.5%	1.3%	0.7%	0.3
100	1%	0.5%	0.3%	0.1

Table 2: Average sampling overhead in percent of real time (one direct sample per clock interrupt and no load on system)

6. User Interface and External Dependencies

The user interface is designed with the objective of providing a systematic and simple dialogue for solving typical user problems (while being flexible enough to meet the requirements of advanced users). It consists of a hierarchy of command environments. At each level a menu of valid commands guides the user to set up and perform the needed measurement. Although internally the measurement, data logging, on-line display and data analysis are independent modules, externally the user is given an integrated and uniform view of all these functions.

When a measurement completes and data analysis (data reduction) is invoked, an overview of the measurement is presented first. Then a menu of commands is displayed which guides the user to obtain the performance profile of his program at several levels of detail. The direct or direct/indirect performance profiles can be obtained for program segments, procedures within a segment, and procedure (or segment) relative address regions down to the level of every machine instruction in the program. [Internally, a multi level bucket (counter) structure is set up for each user request and the buckets are searched and filled as samples are read from the log file. Finally, the counters are plotted (using the character enhancement capabilities of HP terminals) together with appropriate labels].

There are certain services that Segmenter and compilers can provide for a performance measurement tool of this type. Since the source level entities such as procedure names, block names, line numbers, etc., loose their identity after the translation and transformation phases, it would be impossible for the measurement tool to relate run time data (i.e. absolute addresses) to the source level entities without their aid. A performance profile which does not readily relate to the source program is difficult to interpret and requires tedious re_mapping calculations by the reader. APS/3000 uses the PMAP information which is provided by the segmenter to produce profiles in terms of procedure names and procedure relative addresses. The new FPMAP option of PREP command appends the PMAP data to the prepared program file. After a measurement completes, this data is extracted and recorded in the log file. During the analysis, the PMAP data is used to determine the procedure names and boundaries in each segment.

A different class of program profilers exists which can produce the execution frequencies of the source statements. This is accomplished by instrumenting the code. There are several techniques to instrument a program: a) extra code is generated at compile time for each statement or branch, b) a preprocessor adds source level counters before each statement[[6] for Fortran] and then the resulting code is compiled, c) the object code is modified, etc. The profiles which are produced from such tools are useful for path flow analysis and coverage testing. They have a few drawbacks, however, when used for performance evaluation: i) the relative execution cost of each statement cannot be determined (the cost of executing an assignment statement N times, for instance, is much less than executing a complex expression the same number of times), ii) the indirect cost and wait time cannot be obtained, iii) they cannot be used on production (already compiled) programs, ii) they are usually language dependent.

7. A Case Study

In this example we consider a Pascal program called INDEX. This program generates a sorted list of all unique words (tokens) in a text file. It first opens an Ascii input file, reads the text line by line, uses a scanner routine to break down each line into words and enters the words into a table using a hashing technique. Finally, when the input text is exhausted, it compresses the table, sorts the entries and prints the sorted list. We use this example to demonstrate a number of APS/3000 profiles.

Figure 4 gives an overview of the code segment utilization of INDEX program. The symbolic segment names appear on the left column. Segment SEG' (the bottom line) is the only user segment of this program and shows a 55.7% utilization. The remaining time is used by the Pascal library and operating system (e.g. file system) segments which were called from this program. For instance, 30.8% of time was spent in the run time library PASCAL'LIBRARY3. When we trace the library calls to our program we note frequent calls to the string handling procedures of Pascal.

Next we study the procedures within segment SEG'. Figure 5 shows the direct and indirect execution profile of the procedures in this segment. We quickly observe that procedure BUBBLESORT accounts for 74% of the total segment time (since there is only one user segment this is equivalent to 74% of the total program time). The indirect part of each bar shows the fraction of time a library or operating system segment was exercised from calls originating from that procedure. Procedure SCAN is next in line with 14.3% utilization.

As the first step to optimize this program we replace the BubbleSort Algorithm with the more efficient QuickSort Algorithm. The other procedures remain unchanged. We use the same input data and obtain the procedure profile of the modified INDEX program. Figure 6 gives the direct and indirect execution profile after BubbleSort is replaced by QuickSort. Our first observation is that the program now runs three times faster with the given data set. This improvement factor can be derived by comparing the total number of samples before and after modifying the program (assuming identical sampling interval). The bottleneck of the program is no longer the sorting step. The procedure SCAN now has the highest relative utilization. The next obvious optimization step is to try to make the latter procedure more efficient.

Figure 7 is the source listing of procedure SCAN. The compiler generates a table of code offsets at the end of the procedure listing. Figure 8 is the procedure relative address profile of procedure SCAN. At offset range %207-%217 we note a peak of mostly indirect cost components (I's). Using the code offset table, we can map this range to statements 3 to 5 of the listing which includes a READ statement. The other highly visible bar corresponds to offset range %330-%340. (Figure 8). The mapping table leads us to statement 16 of the procedure which contains a call to the Pascal library procedure STRMOVE. From this point on, the optimization effort can continue in many directions and may involve re-writing parts of the code, and reducing calls to the expensive library calls.

8. Conclusion

In this paper we presented the general structure, a number of features and application of a performance evaluation tool. We defined the direct and indirect execution cost components of a program. We discussed the measurement mechanism, overhead and data analysis functions. The services of Segmenter and compilers in assisting this tool was described. We discussed a systematic approach to optimizing an application program, and demonstrated the power of APS/3000 to pinpoint the CPU execution cost of all parts of a software system.

REFERENCES:

1. HP3000 Computer Systems: "*On-line Performance Tool/3000 Reference Manual*", Part No. 32238-90001

2. IBM, MF/1 System Activity Measurement Facility of MVS

3. Holtwick, G.M. "*Designing a Commercial Performance Measurement System*", Proc. of ACM-SIGOPS Workshop on System Performance Evaluation, Harvard University, April 1971

4. Bussel, B. and Koster, R.A. "*Instrumenting Computer Systems and Their Programs*", AFIPS Conf. Proc. FJCC 1970, Vol. 17

5. Huang,J.C."*Instrumenting Programs for Symbolic-Trace Generation*", Computer Magazine, December 1980

6. Stucki, L.G. and Walker, H.D. "*Concepts and Prototypes of ARGUS*", (DYNA Dynamic Analyzer, page 67), Software Engineering Environments, Edited by Horst Hunke, North Holland Publishing, 1980

7. Blake, R.P. "*Exploring Stack Architecture*", Computer Magazine, May 1977

8. HP3000 Computer Systems: "*System Reference Manual*", Part No. 30000-90020

9. HP3000 Computer Systems: "*MPE Commands Reference Manual*", Part No. 30000-90009

10. HP3000 Computer Systems: "*Application Program SAMPLER/3000 Reference Manual and User Guide*", Part No. 32180-90001

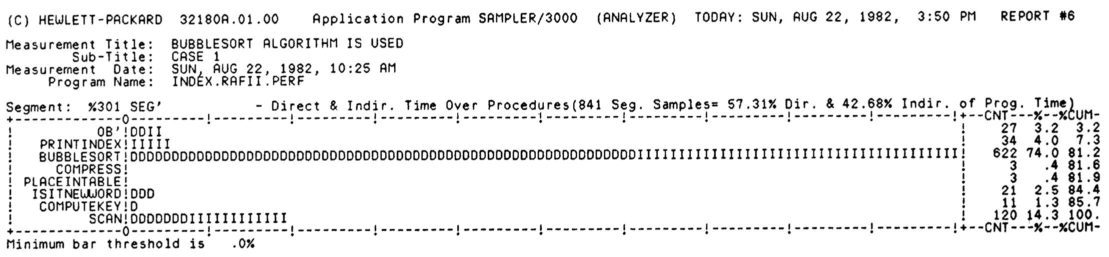
ACKNOWLEDGEMENTS:

Cliff Jager has given his administrative and technical support to the project. Danny Ku wrote the programmable clock interface.

```
(C) HEWLETT-PACKARD   32180A.01.00        Application Program SAMPLER/3000                    TODAY: SUN, AUG 22, 1982,  3:27 PM

     Measurement Title:  BUBBLESORT ALGORITHM IS USED
         Begin: SUN, AUG 22, 1982, 10:25 AM
         End:   SUN, AUG 22, 1982, 10:25 AM
     Analysis Subtitle:  CASE 1;
     Program(s):         INDEX.RAFII.PERF
     Log File:           SAMPLOG1     Version:  A.01.00
     Measurement Option: RUN & MONITOR A PROGRAM (RM Command)
     Measurement Type:   DIRECT AND INDIRECT CPU TIME
     Sampling Interval:  5 MILLISECONDS
     Number of Samples:  865
         OnICS Samples:  21
     Machine:            SERIES 64
     Memory Size:        1024 Kilo Words
     MPE Version:        MPE: D.00.20  BASE: D.00.20


GLOSSARY

     Octal Numbers .... All base 8 numbers start with the symbol "%"
     User Segments .... Segment numbers %301 through %377
     System Segments .. Segment numbers %1 through %277 (Note that this range includes the programs's SL segments too)
     CPU Time ......... = CPU Utilization = CPU Cost = CPU Load
     Direct CPU Time .. Amount of time CPU directly executes a section of a program
     System Call ...... A call from a user segment to a procedure in a system SL segment or program's SL segment
     Indirect CPU Time: Amount of time CPU executes system segments on behalf of the system calls originating from a
                        a section of the user code
     Wait Time ........ Real (Clock) time between the interruption and resumption of a program section  (Note that waits incurred
                        during the execution of system segments are attributed to the last sampled user code section)
     Program Time ..... Total CPU (direct+indirect) execution time of a program
     Turnaround Time... Real time from the beginning to the termination of a program
     OnICS Samples .... An operating system code was running on Interrupt Control Stack under PIN of the program under study
```

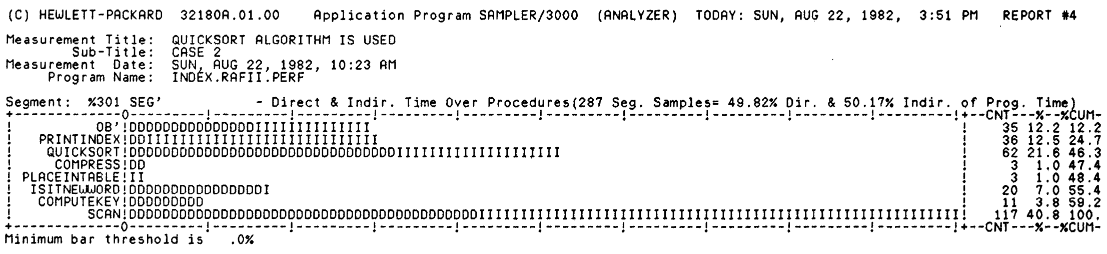
```
(C) HEWLETT-PACKARD  32180A.01.00    Application Program SAMPLER/3000  (ANALYZER)  TODAY: SUN, AUG 22, 1982,   3:28 PM   REPORT #1

Measurement Title:  BUBBLESORT ALGORITHM IS USED
        Sub-Title:  CASE 1;
Measurement  Date:  SUN, AUG 22, 1982, 10:25 AM

  Program Name(s):   INDEX.RAFII.PERF

Distribution of Direct CPU Utilization Over All Segments (865 Samples - Including OnICS samples)
+------------0---------!---------!---------!---------!---------!---------!---------!---------!---------!---------!+--CNT---%--%CUM-
!   FILESYS1'5!                                                                                                   !    1   .1   .1
!   FILESYS2'7!                                                                                                   !    1   .1   .2
!  FILESYS4'6A!                                                                                                   !    3   .3   .6
!   FILESYS3'6!DD                                                                                                 !   11  1.3  1.8
!   UTILITY1'2!                                                                                                   !    1   .1  2.0
!   DIRC'CHECK!                                                                                                   !    1   .1  2.1
!   KC'MISCSEG!DDDD                                                                                               !   24  2.8  4.9
! MOR'RIN'ABDP!                                                                                                   !    1   .1  5.0
!    FILESYS1A!DDDDD                                                                                              !   27  3.1  8.1
!      HARDRES!DDDD                                                                                               !   20  2.3 10.4
!     PDMANAGR!                                                                                                   !    3   .3 10.8
!      TERMMON!D                                                                                                  !    9  1.0 11.8
!      CLIB'03!                                                                                                   !    1   .1 11.9
!     HIOMDSC1!                                                                                                   !    1   .1 12.0
PASCAL'LIBRARY3DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD                                             !  266 30.8 42.8
PASCAL'LIBRARY2DD                                                                                                 !   12  1.4 44.2
PASCAL'LIBRARY1                                                                                                   !    1   .1 44.3
!         SEG'!DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD!  482 55.7 100.
+------------0---------!---------!---------!---------!---------!---------!---------!---------!---------!---------!+--CNT---%--%CUM-
Minimum bar threshold is   .0%
```

Figure 4   Segment utilization profile



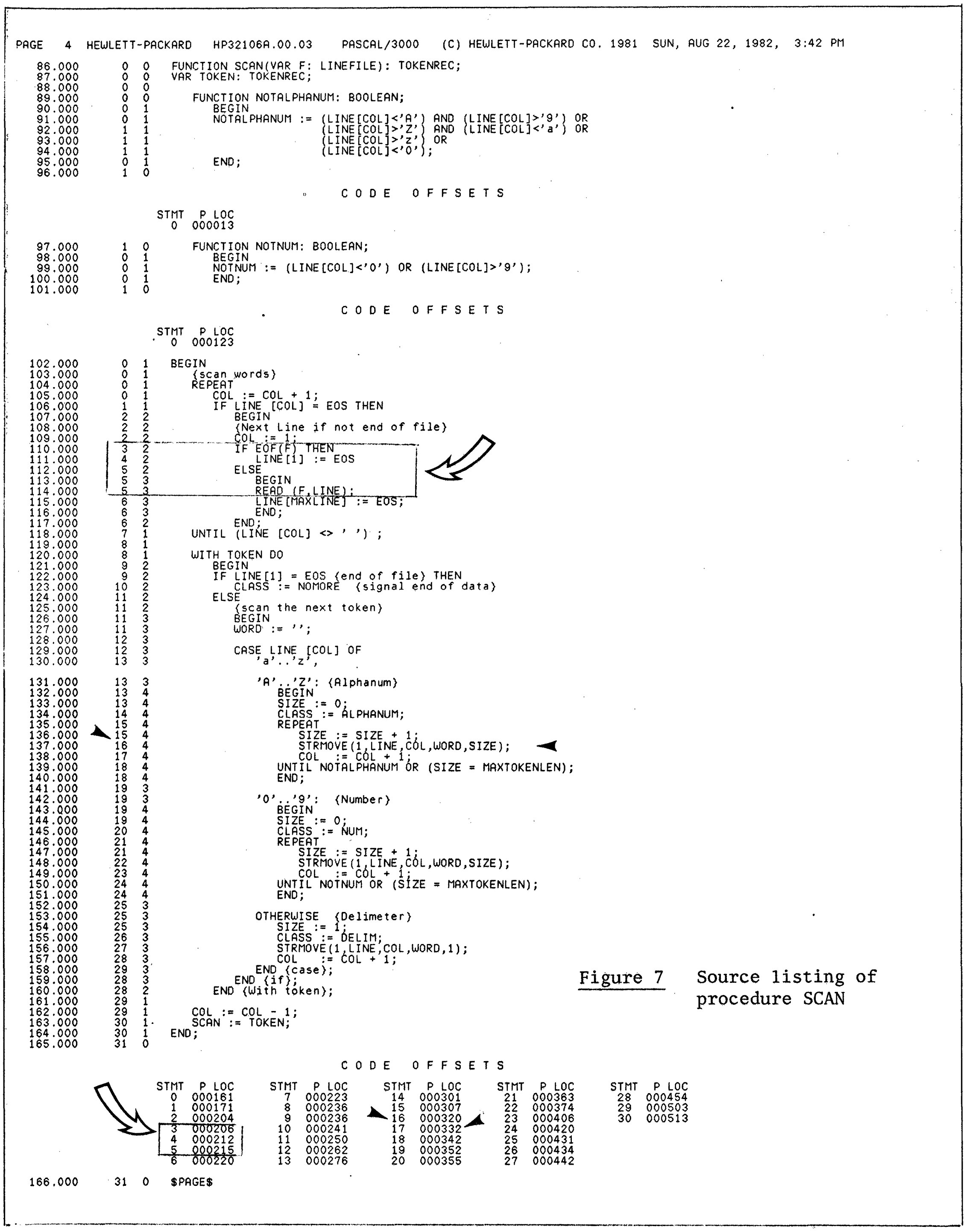
```
(C) HEWLETT-PACKARD  32180A.01.00   Application Program SAMPLER/3000  (ANALYZER)  TODAY: SUN, AUG 22, 1982,  3:50 PM   REPORT #6

Measurement Title:  BUBBLESORT ALGORITHM IS USED
        Sub-Title:  CASE 1
Measurement  Date:  SUN, AUG 22, 1982, 10:25 AM
     Program Name:  INDEX.RAFII.PERF

Segment:  %301 SEG'          - Direct & Indir. Time Over Procedures(841 Seg. Samples= 57.31% Dir. & 42.68% Indir. of Prog. Time)
+-------------0---------!---------!---------!---------!---------!---------!---------!---------!---------!---------!+--CNT---%--%CUM-
!          OB'!DDII                                                                                                !  27  3.2  3.2
!   PRINTINDEX!IIIII                                                                                               !  34  4.0  7.3
!   BUBBLESORT!DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII! 622 74.0 81.2
!     COMPRESS!                                                                                                    !   3   .4 81.6
! PLACEINTABLE!                                                                                                    !   3   .4 81.9
!  ISITNEWWORD!DDD                                                                                                 !  21  2.5 84.4
!   COMPUTEKEY!D                                                                                                   !  11  1.3 85.7
!         SCAN!DDDDDDDIIIIIIIIIIII                                                                                 ! 120 14.3 100.
+-------------0---------!---------!---------!---------!---------!---------!---------!---------!---------!---------!+--CNT---%--%CUM-
Minimum bar threshold is   .0%
```

Figure 5   Procedure profile of segment SEG' (BubbleSort case)



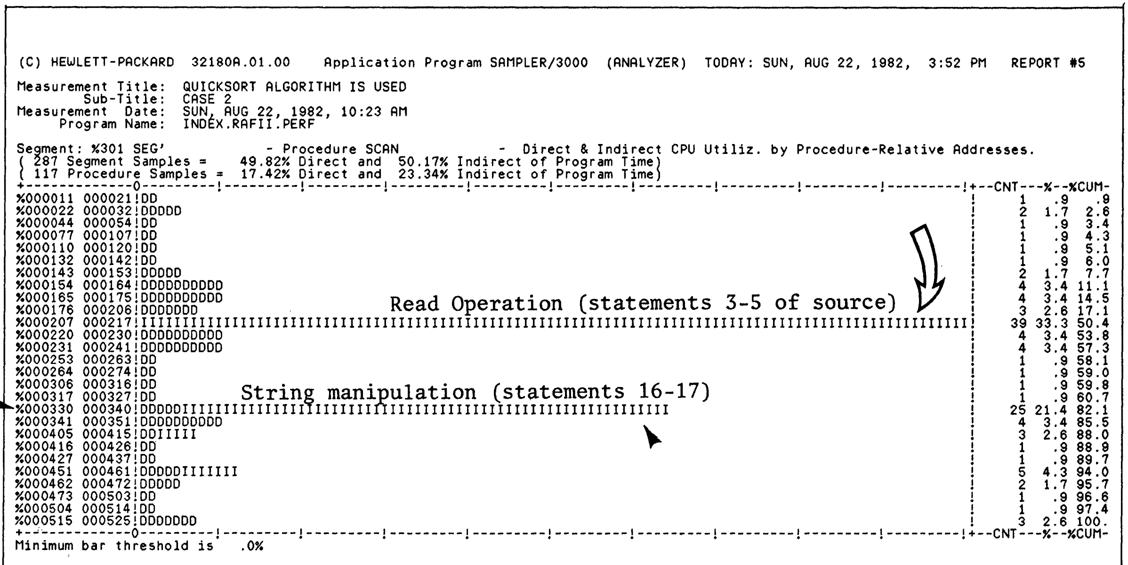
```
(C) HEWLETT-PACKARD  32180A.01.00   Application Program SAMPLER/3000  (ANALYZER)  TODAY: SUN, AUG 22, 1982,  3:51 PM   REPORT #4

Measurement Title:  QUICKSORT ALGORITHM IS USED
        Sub-Title:  CASE 2
Measurement  Date:  SUN, AUG 22, 1982, 10:23 AM
     Program Name:  INDEX.RAFII.PERF

Segment:  %301 SEG'          - Direct & Indir. Time Over Procedures(287 Seg. Samples= 49.82% Dir. & 50.17% Indir. of Prog. Time)
+-------------0---------!---------!---------!---------!---------!---------!---------!---------!---------!---------!+--CNT---%--%CUM-
!          OB'!DDDDDDDDDDDDDDDDIIIIIIIIIIIIII                                                                      !  35 12.2 12.2
!   PRINTINDEX!DDIIIIIIIIIIIIIIIIIIIIIIIIIIIII                                                                     !  36 12.5 24.7
!    QUICKSORT!DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDIIIIIIIIIIIIIIIIIIII                                                !  62 21.6 46.3
!     COMPRESS!DD                                                                                                  !   3  1.0 47.4
! PLACEINTABLE!II                                                                                                  !   3  1.0 48.4
!  ISITNEWWORD!DDDDDDDDDDDDDDDDI                                                                                   !  20  7.0 55.4
!   COMPUTEKEY!DDDDDDDDD                                                                                           !  11  3.8 59.2
!         SCAN!DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII! 117 40.8 100.
+-------------0---------!---------!---------!---------!---------!---------!---------!---------!---------!---------!+--CNT---%--%CUM-
Minimum bar threshold is   .0%
```

Figure 6   Procedure profile of segment SEG' (BubbleSort is replaced by more efficient QuickSort)



```
PAGE   4  HEWLETT-PACKARD   HP32106A.00.03    PASCAL/3000   (C) HEWLETT-PACKARD CO. 1981  SUN, AUG 22, 1982,  3:42 PM
    86.000       0  0    FUNCTION SCAN(VAR F: LINEFILE): TOKENREC;
    87.000       0  0    VAR TOKEN: TOKENREC;
    88.000       0  0
    89.000       0  0       FUNCTION NOTALPHANUM: BOOLEAN;
    90.000       0  1          BEGIN
    91.000       0  1          NOTALPHANUM := (LINE[COL]<'A') AND (LINE[COL]>'9') OR
    92.000       1  1                         (LINE[COL]>'Z') AND (LINE[COL]<'a') OR
    93.000       1  1                         (LINE[COL]>'z') OR
    94.000       1  1                         (LINE[COL]<'0');
    95.000       0  1          END;
    96.000       1  0

                                                  C O D E   O F F S E T S

                       STMT  P LOC
                         0  000013

    97.000       1  0       FUNCTION NOTNUM: BOOLEAN;
    98.000       0  1          BEGIN
    99.000       0  1          NOTNUM := (LINE[COL]<'0') OR (LINE[COL]>'9');
   100.000       0  1          END;
   101.000       1  0

                                                  C O D E   O F F S E T S

                       STMT  P LOC
                         0  000123

   102.000       0  1    BEGIN
   103.000       0  1       (scan words)
   104.000       0  1       REPEAT
   105.000       0  1          COL := COL + 1;
   106.000       1  1          IF LINE [COL] = EOS THEN
   107.000       2  2             BEGIN
   108.000       2  2             (Next Line if not end of file)
   109.000       2  2             COL := 1;
   110.000       3  2             IF EOF(F) THEN
   111.000       4  2                LINE[1] := EOS
   112.000       5  2             ELSE
   113.000       5  3                BEGIN
   114.000       5  3                READ (F,LINE);
   115.000       6  3                LINE[MAXLINE] := EOS;
   116.000       6  3                END;
   117.000       6  2             END;
   118.000       7  1       UNTIL (LINE [COL] <> ' ') ;
   119.000       8  1
   120.000       8  1       WITH TOKEN DO
   121.000       9  2          BEGIN
   122.000       9  2          IF LINE[1] = EOS (end of file) THEN
   123.000      10  2             CLASS := NOMORE  (signal end of data)
   124.000      11  2          ELSE
   125.000      11  2             (scan the next token)
   126.000      11  3             BEGIN
   127.000      11  3             WORD := '';
   128.000      12  3
   129.000      12  3             CASE LINE [COL] OF
   130.000      13  3                'a'..'z',

   131.000      13  3                'A'..'Z': (Alphanum)
   132.000      13  4                   BEGIN
   133.000      13  4                   SIZE := 0;
   134.000      14  4                   CLASS := ALPHANUM;
   135.000      15  4                   REPEAT
   136.000      15  4                      SIZE := SIZE + 1;
   137.000      16  4                      STRMOVE(1,LINE,COL,WORD,SIZE);
   138.000      17  4                      COL   := COL + 1;
   139.000      18  4                   UNTIL NOTALPHANUM OR (SIZE = MAXTOKENLEN);
   140.000      18  4                   END;
   141.000      19  3
   142.000      19  3                '0'..'9':  (Number)
   143.000      19  4                   BEGIN
   144.000      19  4                   SIZE := 0;
   145.000      20  4                   CLASS := NUM;
   146.000      21  4                   REPEAT
   147.000      21  4                      SIZE := SIZE + 1;
   148.000      22  4                      STRMOVE(1,LINE,COL,WORD,SIZE);
   149.000      23  4                      COL   := COL + 1;
   150.000      24  4                   UNTIL NOTNUM OR (SIZE = MAXTOKENLEN);
   151.000      24  4                   END;
   152.000      25  3
   153.000      25  3                OTHERWISE  (Delimeter)
   154.000      25  3                   SIZE := 1;
   155.000      26  3                   CLASS := DELIM;
   156.000      27  3                   STRMOVE(1,LINE,COL,WORD,1);
   157.000      28  3                   COL    := COL + 1;
   158.000      29  3                END (case);
   159.000      28  3             END (if);
   160.000      28  2          END (With token);
   161.000      29  1
   162.000      29  1       COL := COL - 1;
   163.000      30  1       SCAN := TOKEN;
   164.000      30  1    END;
   165.000      31  0

                                                  C O D E   O F F S E T S

                       STMT  P LOC      STMT  P LOC      STMT  P LOC      STMT  P LOC      STMT  P LOC
                         0  000161        7  000223       14  000301       21  000363       28  000454
                         1  000171        8  000236       15  000307       22  000374       29  000503
                         2  000204        9  000236       16  000320       23  000406       30  000513
                         3  000206       10  000241       17  000332       24  000420
                         4  000212       11  000250       18  000342       25  000431
                         5  000215       12  000262       19  000352       26  000434
                         6  000220       13  000276       20  000355       27  000442

   166.000       31  0    $PAGE$
```

Figure 7   Source listing of procedure SCAN



```
(C) HEWLETT-PACKARD  32180A.01.00    Application Program SAMPLER/3000  (ANALYZER)  TODAY: SUN, AUG 22, 1982,  3:52 PM   REPORT #5

Measurement Title:  QUICKSORT ALGORITHM IS USED
        Sub-Title:  CASE 2
Measurement  Date:  SUN, AUG 22, 1982, 10:23 AM
     Program Name:  INDEX.RAFII.PERF

Segment: %301 SEG'           - Procedure SCAN            -  Direct & Indirect CPU Utiliz. by Procedure-Relative Addresses.
( 287 Segment Samples =    49.82% Direct and  50.17% Indirect of Program Time)
( 117 Procedure Samples =  17.42% Direct and  23.34% Indirect of Program Time)
+-------------0---------!---------!---------!---------!---------!---------!---------!---------!---------!---------!+--CNT---%--%CUM-
%000011 000021!DD                                                                                                  !    1   .9    .9
%000022 000032!DDDDD                                                                                               !    2  1.7   2.6
%000044 000054!DD                                                                                                  !    1   .9   3.4
%000077 000107!DD                                                                                                  !    1   .9   4.3
%000110 000120!DD                                                                                                  !    1   .9   5.1
%000132 000142!DD                                                                                                  !    1   .9   6.0
%000143 000153!DDDDD                                                                                               !    2  1.7   7.7
%000154 000164!DDDDDDDDDD                                                                                          !    4  3.4  11.1
%000165 000175!DDDDDDDDDD                                                                                          !    4  3.4  14.5
%000176 000206!DDDDDDD                       Read Operation (statements 3-5 of source)                             !    3  2.6  17.1
%000207 000217!IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII!   39 33.3  50.4
%000220 000230!DDDDDDDDDD                                                                                          !    4  3.4  53.8
%000231 000241!DDDDDDDDDD                                                                                          !    4  3.4  57.3
%000253 000263!DD                                                                                                  !    1   .9  58.1
%000264 000274!DD                                                                                                  !    1   .9  59.0
%000306 000316!DD                                                                                                  !    1   .9  59.8
%000317 000327!DD          String manipulation (statements 16-17)                                                  !    1   .9  60.7
%000330 000340!DDDDDIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII                                    !   25 21.4  82.1
%000341 000351!DDDDDDDDDD                                                                                          !    4  3.4  85.5
%000405 000415!DDIIIII                                                                                             !    3  2.6  88.0
%000416 000426!DD                                                                                                  !    1   .9  88.9
%000427 000437!DD                                                                                                  !    1   .9  89.7
%000451 000461!DDDDDIIIIIII                                                                                        !    5  4.3  94.0
%000462 000472!DDDDD                                                                                               !    2  1.7  95.7
%000473 000503!DD                                                                                                  !    1   .9  96.6
%000504 000514!DD                                                                                                  !    1   .9  97.4
%000515 000525!DDDDDDD                                                                                             !    3  2.6  100.
+-------------0---------!---------!---------!---------!---------!---------!---------!---------!---------!---------!+--CNT---%--%CUM-
Minimum bar threshold is   .0%
```

Figure 8   Procedure relative address profile of procedure SCAN

FUTURE DIRECTIONS OF THE DATA DICTIONARY

Alan Pare', Hewlett-Packard
19420 Homestead Road
Cupertino, CA 95014

The data dictionary functions as a centralized controller over information about data within its domain. Since its emergence, it has provided the data processing community a vehicle for delivering increased productivity and localizing the standardization, documentation, and definition of its resources. Hewlett-Packard recently introduced the Rapid/3000 data dictionary which services its companion products. This paper will differentiate between active, passive, and in-line dictionaries and serve as an introduction to what the presentation will focus on.

The terminology used to address the subject matter of data dictionaries/directories has been done so under a variety of terms ranging from data dictionary, information manager, data directory, to resource manager. The concept of a data directory refers to the function of documenting information about where data is located and how it can be accessed. The term data dictionary deals with the definition of what data exists in the system and describing its corresponding attributes. For the purpose of this paper and the presentation, the data dictionary and data directory capability together will be referred to as simply the data dictionary system or DDS.

The benefits to be derived from the DDS are becoming more evident as its utilization becomes more widespread. Just as the data base technology flourished in the mid 1960's and 70's, the 1980's appear to be the decade for the DDS to approach maturity as an organizations primary solution to managing its data resources.

The data dictionary is a data base which contains the set of all attributes that describe entities such as elements, groups, files, jobs, processes, and transactions within its domain. For example, the set of attributes for a data element would include its length, type, value range, edit mask, and security level. The DDS also holds the definition of relationships between these entities such as file to file, file to element, group to element. Figure 1 illustrates the entity relationships for the Rapid/3000 data dictionary. The DDS therefore serves as a centralized repository for information about descriptions and relationships of system data resources and facilitates the better understanding of applications and data usage. The intrinsic value of this data about data resources, commonly referred to as metadata, is that it can provide an organization with the ability to manage its data resources more effectively by planning, assigning, maintaining, utilizing, and integrating it in a more controlled fashion. The utility of the DDS is therefore directly proportional to the number of system components interfacing into or out of it. The degree to which this takes place can be seen by looking at the three roles the DDS can take on.

In its passive role, the DDS is not cognizant of other systems and processes which operate within its domain, i.e. it does not interact with any of the language compilers, the DBMS, or the operating system. The active DDS role on the other hand is very much involved with other system activities at compilation time. It assures an organization that the data and system definitions employed by applications and the DBMS are consistent with those contained in the data dictionary. This is accomplished by providing software interfaces to the data dictionary for the language compilers, the DBMS schema processor, and the other relevant subsystems.

In its third role, the in-line DDS, the data dictionary system is is integrated with the execution time processes of the application program in calling the data management services of the DBMS and file system. The in-line DDS draws upon its data dictionary content to generate system calls, validate data, enforce security rules, edit data, and various other tasks including run-time compiler data definition assignments for the application program. Its total regulation of application execution guarantees the integrity of data and application invariance to system hardware or software changes. The performance degradation is a critical issue in the implementation of this DDS role. Figures 2-4 depict the various configurations of a data dictionary system. The processor/monitor illustrated in these figures serves as the system environment for accepting and reporting about data resources within its domain, executing the data dictionary maintenance utilities, and interfacing the DDS with the operating system. Additionally, for the in-line DDS it carries out the execution of the application program. The DDS intrinsics pictured in figures 2-4 function as the unique access path to the data dictionary thereby establishing system independence from its data base structure.

If one thinks of a DDS in an application context rather than a system one, the Rapid/3000 data dictionary is a hybrid between the active and in-line data dictionary. Figure 5 illustrates the data dictionary environment within the Rapid/3000 framework. In reality, this data dictionary is a passive one since it is not integrated with the system language compilers, DBMS schema processor, or other subsystems outside its realm.

The topic of the presentation will focus on the attributes, concepts, and objectives of a hybrid type system data dictionary and the benefits it would provide.

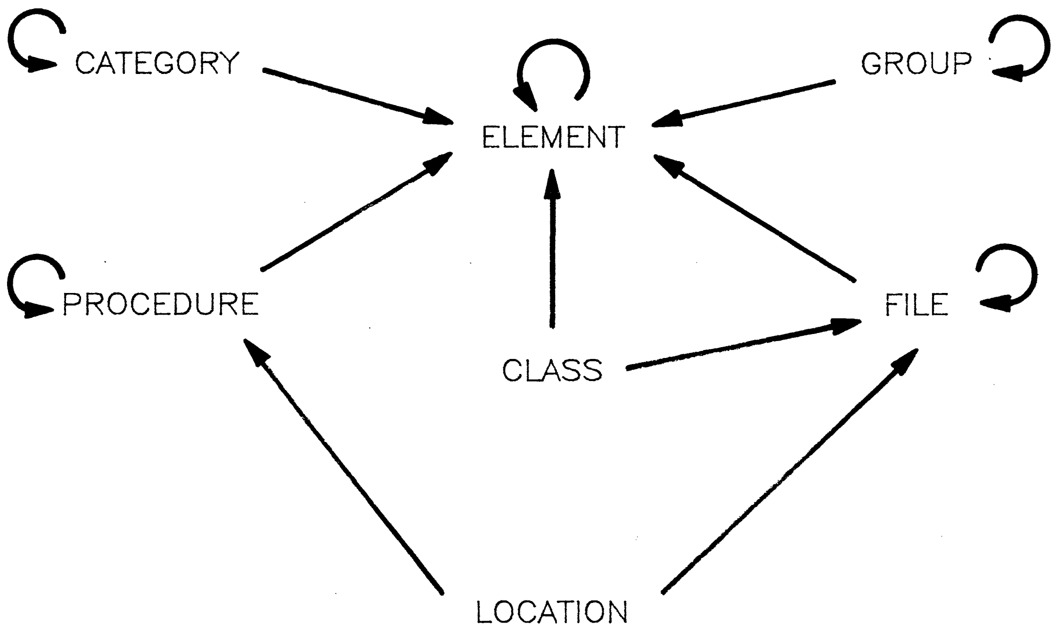


Figure 1. Rapid/3000 Data Dictionary Entity Relationships

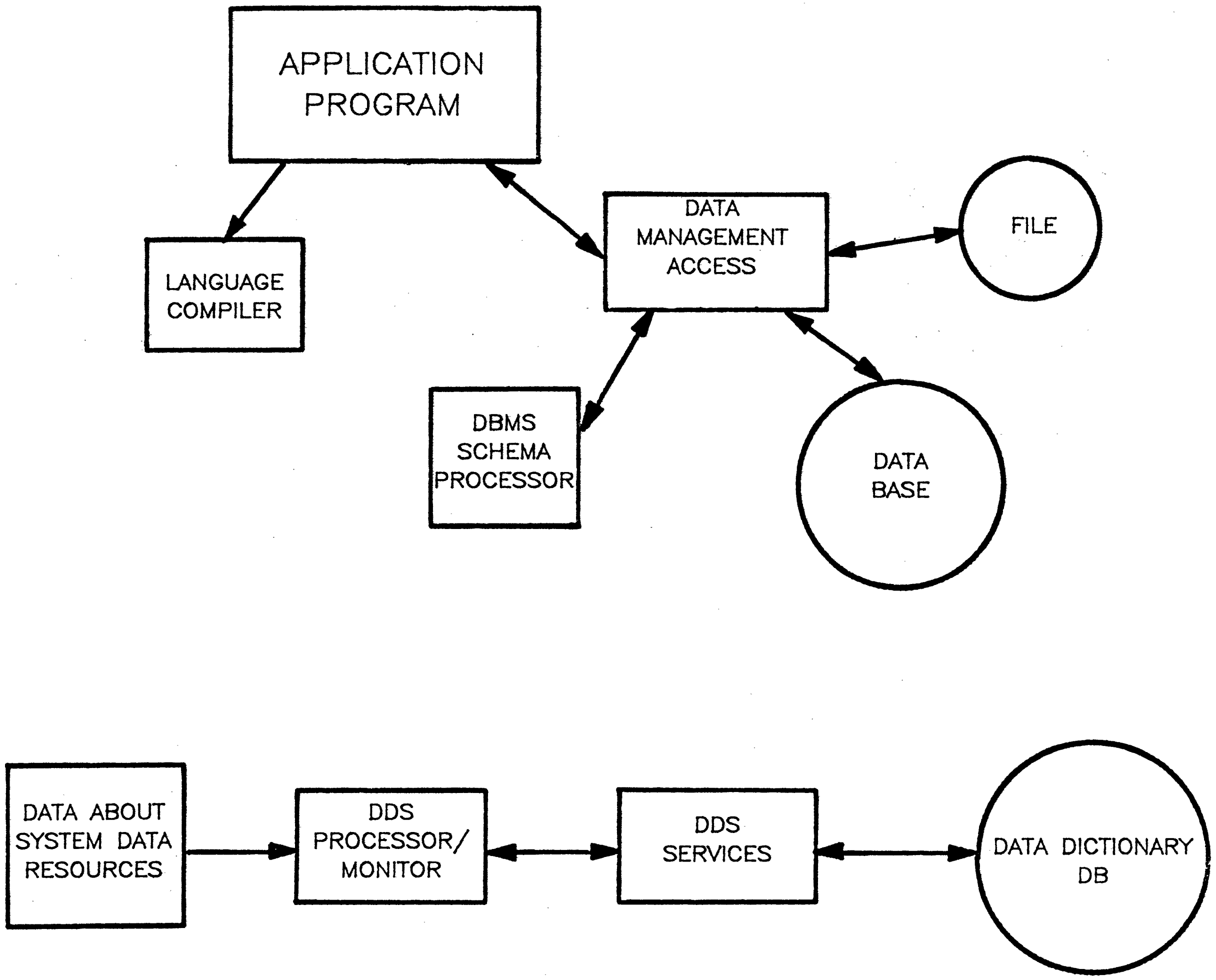


Figure 2. Passive DDS Environment

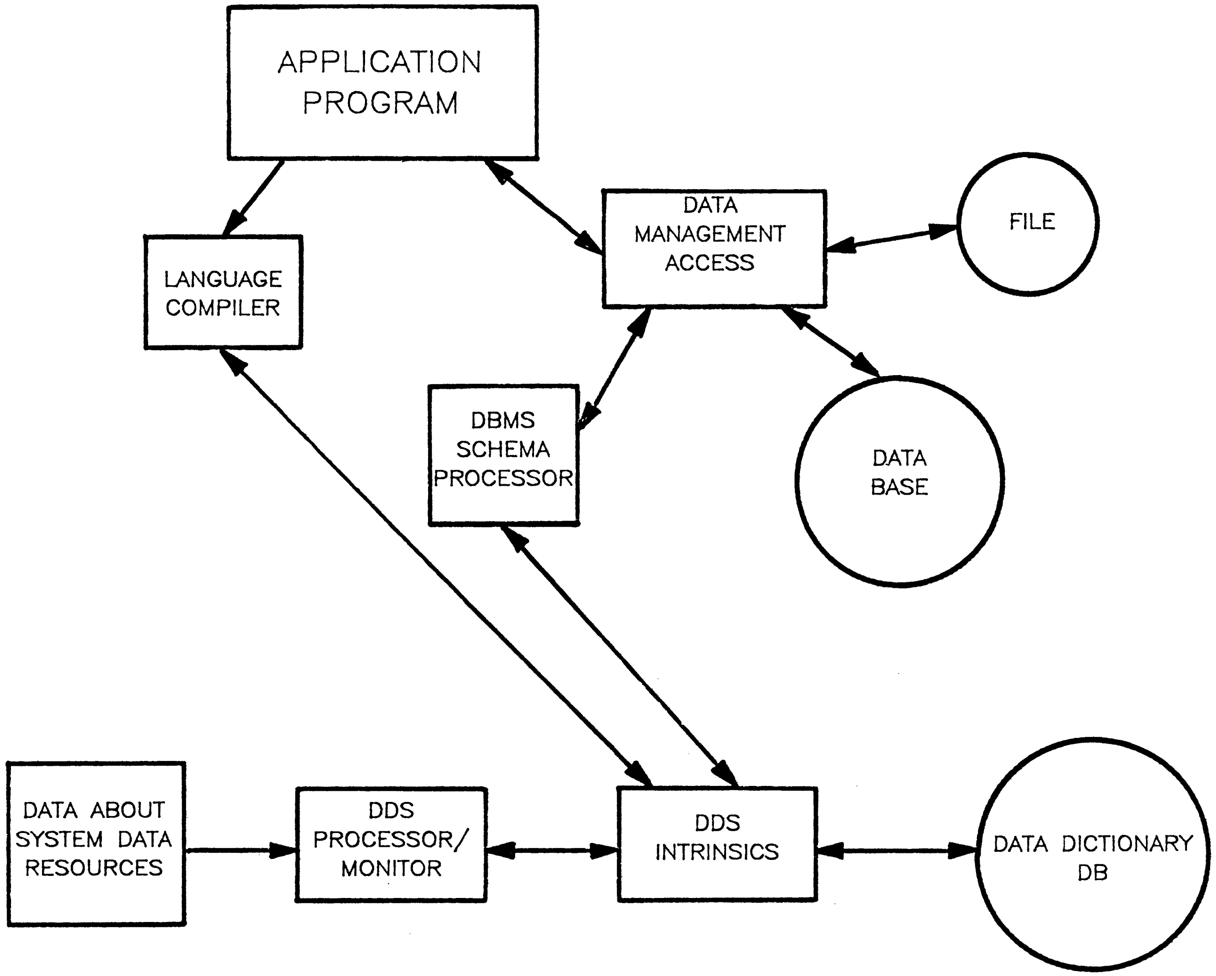


Figure 3. Active DDS Environment

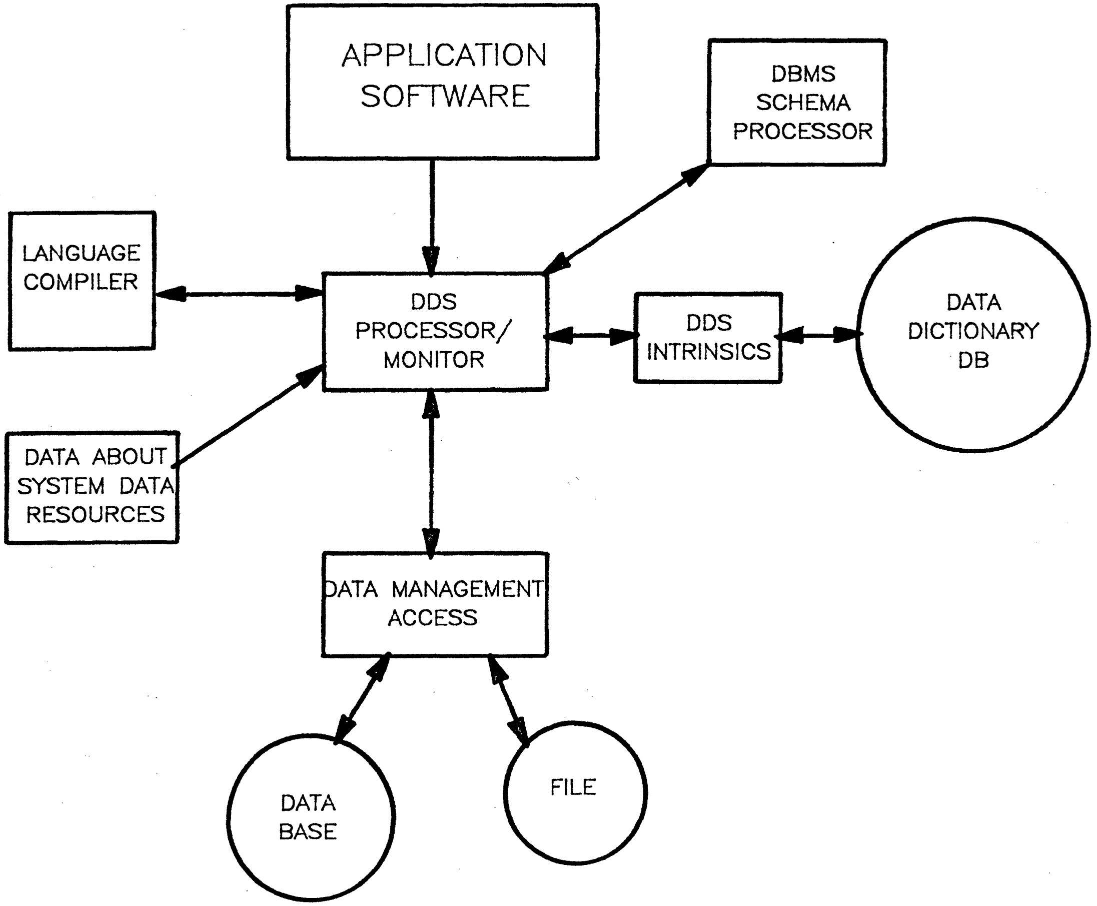


Figure 4. In–Line DDS Environment

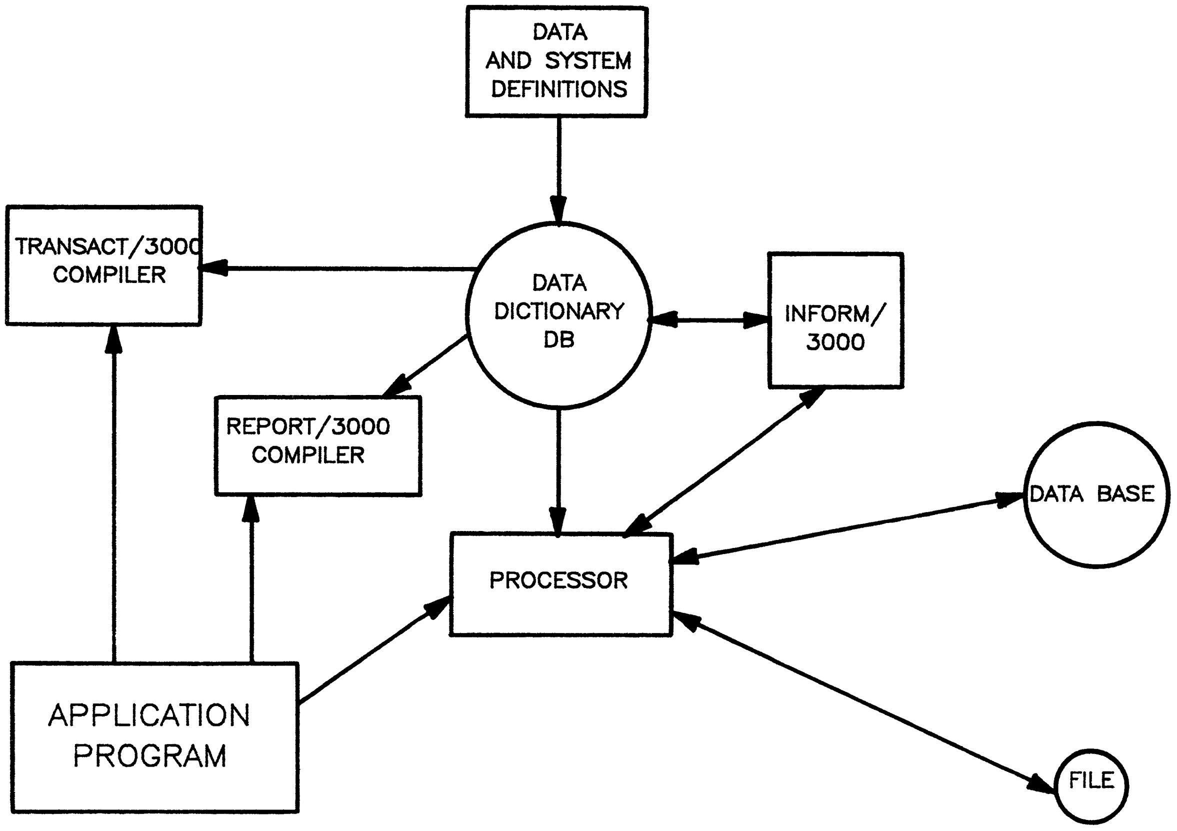


Figure 5. Rapid/3000 Data Dictionary Environment

A VERY SCHOLARLY DISCUSSION OF THE EFFECT OF SYNONYMS ON PERFORMANCE

by Eugene Volokh,
VESOFT Consultants
506 N. Plymouth Blvd.
Los Angeles, CA 90004 USA
(213) 464-7523

Presented at the Fall 1982 HPIUG Conference at Copenhagen

ABSTRACT

This paper will discuss the effect of synonyms on IMAGE/3000 performance, in particular the average number of I/Os necessary to perform a DBGET mode 7 (also known as a hashed, calculated, or keyed read).

A DISCUSSION OF THE EFFECT OF SYNONYMS ON PERFORMANCE

THE PROBLEM

As is well known, IMAGE/3000 master datasets use an access method called HASHING to implement keyed I/O against them. The essence of this method is that when a DBPUT is performed against a master dataset, the key is translated through a special HASHING FUNCTION (which is explained in Robelle Consulting's SMUG II micro-proceedings) to a record number within the dataset. Then the entry to be added is inserted into the dataset at the resulting record number. Now, when a DBGET mode 7 is performed with the same key, the key is translated through the same hashing function to the same record number; then, the entry we are looking for is just the record with that number. Thus, we should ideally be able to add or retrieve any entry in just one I/O (as opposed to considerably more I/Os needed for other access methods, such as KSAM).

However, a problem arises: what if two keys map to the same record number (this condition is known as a COLLISION)? We can't very well put both entries into the same record number, so we will have to put one of these entries (known as the SECONDARY) somewhere else, and put a pointer to it into the other entry (known as the PRIMARY). Now, if we want to retrieve the secondary, we will have to perform two I/Os: one to retrieve the primary, and then when we see that the primary is not the entry we want, a second to retrieve the secondary. A similar situation arises when a key maps to a record number that already has a primary and a secondary attached to it; then the entry is added to the so-called SYNONYM CHAIN for that record number (the primary and all its secondaries), and we now need three I/Os to get to it. Similar things occur when a key maps to a record number that has two synonyms, three synonyms, etc.

In short, access to primaries requires only one I/O; however, access to secondaries requires two or more I/Os (unless the blocking factor is greater than one).

It is also well known that the more full a dataset is, the more secondaries there are in that dataset; and, the more secondaries there are, the more I/Os are needed (on the average) to retrieve an entry. However, how many more I/Os are needed, nobody knows.

This brings up an interesting question: what is the relationship of the average number of I/Os needed to retrieve an entry to how full the dataset is (THE FILL FACTOR), measured from 0=empty, to 1=full.

THE SOLUTION

In this paper, I will attempt to solve the above question theoretically, i.e. derive a general formula that, given the fill factor, will return the average number of I/Os. In order to do this, however, some simplifying assumptions must be made:

First of all, I will assume that our hashing function is "good", i.e. regardless of the data fed to it, it will return each record number with roughly equal frequency. Thus, for instance, a hashing function that returns record number 17 half of the time, or returns odd record numbers more often than even record numbers is not "good". Fortunately, IMAGE's hashing function is a relatively good one. (Note: If the dataset's key is numeric, e.g. I2, a not-so-good hashing function is used by IMAGE; thus, these results may not be correct for datasets with numeric keys).

Secondly, I will assume that the master dataset in question has blocking factor 1. This will permit me to ignore the possibility of a synonym being in the same block as the primary, thus requiring no additional I/Os when it is retrieved. I must confess that this may be a quite fallacious assumption in that synonyms very often are in the same blocks as their primaries; however, although this fact may be decrease the average number of I/Os for a given fill factor, it should not influence the overall nature of the function we are seeking.

Now, we can get down to business. Let us define

  N = number of entries in the dataset,
  C = capacity of the dataset,
  F = fill factor = N/C,
  An X-ARY = an entry that is the Xth in a synonym chain, and thus takes X I/Os to retrieve. Thus, a 1-ARY is a primary, and 2-ARYs, 3-ARYs, 4-ARYs, etc. are all secondaries,
  E (X, I) = the expected number of X-aries in a dataset after I entries have been added.

Let us thus ask the following question:

  How many primaries can we expect in the dataset after I entries have been added to it, i.e. what is E(1,I)?

Well, we can expect as many primaries as are expected after I-1 entries have been added (i.e. E(1,I-1)) plus either 0 or 1 more primaries (depending on whether the Ith entry becomes a primary or not), i.e.

$$E(1,I) = E(1,I-1) + 1*(\text{probability that the Ith entry becomes a primary}). \qquad [1]$$

What is the probability that the Ith entry becomes a primary? An entry becomes a primary if it hashes to any slot in the dataset that is NOT OCCUPIED BY ANOTHER PRIMARY, i.e. that is empty or is occupied by a secondary! The probability that the Ith entry becomes a primary is thus equal to

$$\frac{\text{(the number of slots that are not occupied by another primary)}}{\text{(the total number of slots that it could possibly hash to)}}$$

The total number of slots that it could possibly hash to is, of course, C. The number of slots that are not occupied by another primary is C - (the number of slots that are occupied by a primary). When the Ith entry is being added, the expected number of slots that are occupied by a primary is by definition E(1,I-1).

Thus, the probability that the Ith entry will become a primary is

```
  (C-E(1,I-1))/C = 1-E(1,I-1)/C                                    [2]
```

Therefore, substituting [2] into [1],

```
  E(1,I) = E(1,I-1)+1-E(1,I-1)/C =                                 [3]
         = E(1,I-1)*(1-1/C)+1
```

Moreover, we know that E(1,0) = expected number of primaries when the dataset has 0 entries, i.e. when the dataset is empty = 0. Thus, we have a recursive formula for E(1,I). From it we can, for instance, determine that E(1,1) = E(1,0)*(1-1/C)+1=0*(1-1/C)+1=1, that E(1,2) = E(1,1)*(1-1/C)+1=1*(1-1/C)+1=2-1/C, etc.

But, we want a non-recursive, straightforward formula. We can obtain it by further substituting [3] into itself:

```
  E(1,I) = E(1,I-1)*(1-1/C)+1 =                                    [4]
         = (E(1,I-2)*(1-1/C)+1)*(1-1/C)+1 =
         = E(1,I-2)*(1-1/C)^2+(1-1/C)+1 =
           ...
         = (1-1/C)^(I-1)+(1-1/C)^(I-2)+ ... +(1-1/C)^2+(1-1/C)+1
```

The above is what is known in mathematical circles as a geometric progression, i.e. a sum of terms each one of which is (1-1/C) times less than the previous one. It is also known that the sum of the above progression is

$$\frac{(1-1/C)\hat{}I-1}{(1-1/C)-1} = \frac{(1-1/C)\hat{}I-1}{-1/C} = -C*((1-1/C)\hat{}I-1) = C*(1-(1-1/C)\hat{}I) \qquad [5]$$

Thus, E(1,I) = C*(1-(1-1/C)^I). If what we are looking for is the percentage of the dataset's entries that are primaries, we should consider E(1,I)/I = (C/I)*(1-(1-1/C)^I).

Now, let us take a closer look at (1-1/C)^I. Clearly,

(1-1/C)^I = (1-1/C)^(C*I/C) = ((1-1/C)^C)^(I/C) [6]

Also, we know that when C is sufficiently large (say, in excess of 100), (1-1/C)^C is close to (and in fact gets closer as C increases) the constant 1/e = 1/2.71828... = 0.367879....

Thus, for large C, we have (substituting [6] into [5])

E(1,I)/I = (C/I)*(1-(1-1/C)^I) = (C/I)*(1-(1/e)^(I/C)) =
= (C/I)*(1-e^(-I/C)) = (1-e^(-I/C))/(I/C). [7]

So, the expected fraction of primaries in our dataset, which contains N entries, is E(1,N)/N = (1-e^(-N/C))/(N/C). But, N/C = F! Therefore,

$$\frac{E\ (1,N)}{N} = \frac{1-e^{-F}}{F} \qquad [8]$$

Ugh.

At last, we have a solid result -- we have determined the expected number of primaries in a dataset given its fill factor. An interesting corollary to the above is that when the dataset is full, i.e. N=C or F=1, E(1,N)/N = (1-e^(-1))/1 = .632121.... Thus, even when a dataset is full, 63% of all entries in it should be primaries.

However, the battle is far from over yet. What we are trying to do is to determine the average number of I/Os needed to retrieve a given entry. Before doing the actual derivations, let me explain the method that we are going to use:

The average number of I/Os needed to retrieve a given entry is equal to the total number of I/Os needed to retrieve all entries in the dataset divided by the total number of entries.

The total number of I/Os needed to retrieve all entries in the dataset is the total number of I/Os needed to retrieve all primaries + the total number of I/Os needed to retrieve all 2-aries + ... But, the total number of I/Os needed to retrieve all X-aries = the expected number of X-aries * the number of I/Os needed to retrieve an X-ary. Clearly, to retrieve an X-ary we would need X I/Os; thus, the total number of I/Os needed to retrive all X-aries = X*E(X,N).

Therefore, the total number of I/Os needed to retrieve all entries in the dataset is 1*E(1,N)+2*E(2,N)+3*E(3,N)+.... And, the average number of I/Os needed to retrieve a given entry = the number of I/Os needed to retrieve all entries / number of entries =

$$\frac{1*E(1,N)+2*E(2,N)+3*E(3,N)+\ldots}{N} \qquad [9]$$

We have already determined E(1,N). Now, if we can only determine E(2,N), E(3,N), ..., we will be almost done. So, this is what we will attempt to do below.

How are we going to determine E(X,I) for X>1?

Well, reasoning similar to the one we used for E(1,I) indicates that

$$E(X,I) = E(X,I-1)+ \text{(probability that the Ith entry becomes an X-ary)} \qquad [10]$$

Now, what is the probability that the Ith entry becomes an X-ary? To answer this, we must ask: under what conditions would an entry become an X-ary? The answer to this is: if and only if it hashes to a slot that is currently occupied by a primary which has a synonym chain of length exactly X-1. How many entries like this are there? Well, the distinguishing feature of this kind of entry is that it has in its synonym chain an (X-1)-ary but no X-aries. Thus, the number of such primary entries = number of (X-1)-aries - number of X-aries = E(X-1,I-1) - E(X,I-1). Thus, the probability that the Ith entry becomes an X-ary is

$$\frac{E(X-1,I-1)-E(X,I-1)}{C} \qquad [11]$$

and thus

$$E(X,I) = E(X,I-1)+(E(X-1),I-1)-E(X,I-1))/C = \qquad [12]$$
$$= E(X,I-1)*(1-1/C)+E(X-1,I-1)/C$$

To obtain a non-recursive formula, we expand the above into

$$E(X,I) = \sum_{J=1}^{I} \frac{E(X-1,J-1)}{C} * (1-1/C)^{\wedge}(I-J) \qquad [13]$$

In particular, let us try to determine $E(2,I)$ = the expected number of 2-aries after I entries have been added.

$$\begin{aligned} E(2,I) &= \mathrm{sum}(J=1:I \mid E(1,J-1)/C * (1-1/C)^{(I-J)}) \approx \\ &\approx \mathrm{sum}(J=1:I \mid E(1,J)/C * (1-1/C)^{(I-J)}) = \\ &= \mathrm{sum}(J=1:I \mid C*(1-e^{(-J/C)})/C * (1-1/C)^{(I-J)}) \approx \\ &= \mathrm{sum}(J=1:I \mid (1-1/C)^{(I-J)}) - \\ &\quad \mathrm{sum}(J=1:I \mid e^{(-J/C)} * (1-1/C)^{(I-J)}) = \\ &= \mathrm{sum}(J=0:I-1 \mid (1-1/C)^{J}) - \\ &\quad \mathrm{sum}(J=1:I \mid e^{(-J/C)} * (1-1/C)^{(I-J)}) \end{aligned} \qquad [14]$$

Now, we can observe that $\mathrm{sum}(J=0:I-1 \mid (1-1/C)^J)$ is just $E(1,I)$; moreover, $(1-1/C)^{(I-J)} = ((1-1/C)^C)^{((I-J)/C)} \approx e^{(-(I-J)/C)}$. Thus,

$$\begin{aligned} E(2,I) &= E(1,I) - \mathrm{sum}(J=1:I \mid e^{(-J/C)} * e^{(-(I-J)/C)}) = \\ &= E(1,I) - \mathrm{sum}(J=1:I \mid e^{(-I/C)}) = \\ &= E(1,I) - I*e^{(-I/C)}. \end{aligned} \qquad [15]$$

So, $E(2,N) = E(1,N)-N*e^{(-N/C)}$; furthermore,

$$\frac{E(2,N)}{N} = \frac{E(1,N)}{N} - e^{-N/C} = \frac{E(1,N)}{N} - e^{-F} \qquad [16]$$

Now, we can proceed to the general question: what is $E(X,I)$ (or, what is $E(X,I)/I$)?

I claim that for $X>=2$,

$$\frac{E(X,I)}{I} = \frac{E(X-1,I)}{I} - \frac{(I/C)^{(X-2)}}{(X-1)!} * e^{-I/C} \qquad [17]$$

where $X! = X$ factorial $= 1*2*3*...*X$.

I will prove this claim via a device known as mathematical induction; i.e. I will demonstrate that the claim is valid for $X=2$, and if it is valid for $X=A$, then it is also valid for $X=A+1$. Clearly, this will demonstrate that the claim is valid for $X=2$, and therefore for $X=2+1=3$, and therefore also for $X=3+1=4$, etc.

First of all, I must prove that the claim is valid for $X=2$. In case of $X=2$, the claim says that

$$\frac{E(2,I)}{I} = \frac{E(1,I)}{I} - \frac{(I/C)^0}{0!} * e^{-I/C} = \frac{E(1,I)}{I} - e^{-I/C} \qquad [18]$$

But, this was proven above in the derivation of $E(2,I)$. Thus, we know that our claim is valid for $X=2$.

Now let us assume that the claim is valid for X=A, i.e.

```
  E (X,A)    E (X-1,A)    (A/C)^(X-2)     -A/C
  ------- = ---------- - ------------ * e                        [19]
     A           A             X!
```

We must show that it is also valid for X=A+1.

```
  E(A+1,I) = SUM(J=1:I | E(A,J-1)/C * (1-1/C)^(I-J)) =              [20]
           = SUM(J=1:I | (E(A-1,J-1)-J*(J/C)^(A-2)/(A-1)!*e^(-J/C))*
                         (1-1/C)^(I-J)/C) =
           = SUM(J=1:I | E(A-1,J-1)*(1-1/C)^(I-J)/C) -
             SUM(J=1:I | (J/C)^(A-1)/(A-1)!*e^(-J/C)*
                         (1-1/C)^(I-J)/C) =
           = E(A,I) -
             SUM(J=1:I | (J/C)^(A-1)/(A-1)!*e^(-J/C)*e^(-(I-J)/C)) =
           = E(A,I) -
             SUM(J=1:I | (J/C)^(A-1)/(A-1)!*e^(-I/C)) =
           = E(A,I) - e^(-I/C)/(A-1)!*SUM(J=1:I | J^(A-1))/C^(A-1) =
```

We know that SUM(J=1:I | J^(A-1)) = I^A/A + smaller powers of I, and is thus approximately equal to I^A/A; so,

```
  E(A+1,I) = E(A,I) - e^(-I/C)/(A-1)!*                               [21]
             SUM(J=1:I | J^(A-1))/C^(A-1) =
           = E(A,I) - e^(-I/C)/(A-1)!*(I^A/A)/C^(A-1) =
           = E(A,I) - e^(-I/C)/A!*I*(I/C)^(A-1)
```

Thus,

```
  E (A+1,I)    E (A,I)    (I/C)^(A-1)     -I/C
  --------- = -------- - ------------ * e                        [22]
      I           I            X!
```

This is indeed is the result predicted by our claim for X=A+1. Thus, we know that if the claim is valid for X=A, it is valid for X=A+1.

Thus, we are justified in saying that our claim is true. Finally, we have obtained a general (albeit recursive) formula for E (X,I).

Now comes the final phase of our proof -- the derivation of the average number of I/Os needed to retrieve a record. For the remainder of this discussion, let us for convenience use the fill factor F = N/C.

The average number of I/Os needed to retrieve a record is equal to the total number of I/Os needed to retrieve all records divided by the number of records.

The total number of I/Os needed to retrieve all records is equal to (the expected number of 1-ARYs)*(I/Os needed to retrieve a 1-ARY)+ (the expected number of 2-ARYs)*(I/Os needed to retrieve a 2-ARY)+...

But, we know that the expected number of X-ARYs is simply E(X,N); the number of I/Os needed to retrieve a X-ARY is simply X; and, the number of records is N.

Thus, the average number of I/Os needed to retrieve a record is

$$\frac{1*E(1,N)+2*E(2,N)+\dots}{N} = 1*\frac{E(1,N)}{N} + 2*\frac{E(2,N)}{N} + \dots \qquad [23]$$

We have found that

$$\frac{E(X,N)}{N} = \frac{E(X-1,N)}{N} - \frac{F^{(X-2)}}{(X-1)!} * e^{-F} \qquad [24]$$

Thus,

$$\frac{E(X,N)}{N} = \frac{E(1,N)}{N} - \frac{F^0}{1!}*e^{-F} - \frac{F^1}{2!}*e^{-F} - \dots - \frac{F^{(X-2)}}{(X-1)!}*e^{-F} \qquad [25]$$

Now,

$$\begin{aligned} E(1,N)/N &= (1-e^{(-F)})/F = \qquad [26] \\ &= e^{(-F)}*(e^F-1)/F = \\ &= e^{(-F)}*(1+(F^1)/1!+(F^2)/2!+(F^3)/3!+\dots-1)/F = \\ &= e^{(-F)}*((F^1)/1!+(F^2)/2!+(F^3)/3!+\dots)/F = \\ &= e^{(-F)}*((F^0)/1!+(F^1)/2!+(F^2)/3!+\dots) \end{aligned}$$

So,

$$\begin{aligned} E(X,N)/N &= E(1,N)/N - \qquad [27] \\ &\quad ((F^0)/1!+(F^1)/2!+\dots+(F^{(X-2)})/(X-1)!)*e^{(-F)} = \\ &= ((F^0)/1!+(F^1)/2!+\dots)*e^{(-F)} - \\ &\quad ((F^0)/1!+(F^1)/2!+\dots+(F^{(X-2)})/(X-1)!)*e^{(-F)} = \\ &= (F^{(X-1)}/X!+F^X/(X+1)!+\dots)*e^{(-F)} = \\ &= \mathrm{sum}(Y=X-1:\mathrm{infinity} \mid F^Y/(Y+1)!)*e^{(-F)} \end{aligned}$$

We are trying to calculate

$$1*\frac{E(1,N)}{N} + 2*\frac{E(2,N)}{N} + 3*\frac{E(3,N)}{N} + \dots = \qquad [28]$$

$$\begin{aligned} &= 1*(F^0/1!+F^1/2!+F^2/3!+\dots)*e^{(-F)} + \\ &\quad 2*\qquad (F^1/2!+F^2/3!+\dots)*e^{(-F)} + \\ &\quad 3*\qquad\qquad (F^2/3!+\dots)*e^{(-F)} + \dots \end{aligned}$$

How many times will F^(Y-1)/Y! be included in the above sum for any given Y? Well, it will be included once in the first row, with a factor of 1; once in the second row, with a factor of 2; and so on until the Yth row, in which it will occur with a factor of Y. In the (Y+1)st row, it will no longer occur, because the (Y+1)st row contains only terms starting with (F^Y)/(Y+1)! Thus, in total, F^(Y-1)/Y! will occur in the resulting sum with a factor of (1+2+3+...+Y) = (Y+1)*Y/2. Thus, the average number of I/Os will be

```
  = sum(Y=1:infinity | F^(Y-1)/Y!*(Y+1)*Y/2) * e^(-F) =          [29]
  = sum(Y=1:infinity | F^(Y-1)/(Y-1)!*(Y+1)/2) * e^(-F) =
  = sum(Y=1:infinity | F^(Y-1)/(Y-1)!*(Y-1)/2+
                       F^(Y-1)/(Y-1)!*2/2) * e^(-F) =
  = sum(Y=0:infinity | F^Y/Y!*Y/2) * e^(-F) +
    sum(Y=0:infinity | F^Y/Y!) * e^(-F) =
  = sum(Y=1:infinity | F^Y/(Y-1)!)/2 * e^(-F) +
    sum(Y=0:infinity | F^Y/Y!) * e^(-F) =
  = (F/2+1) * sum(Y=0:infinity | F^Y/Y!) * e^(-F)
```

Here we can again observe that sum(Y=0:infinity | F^Y/Y!) = e^F. Thus, the average number of I/Os is

```
  =  (F/2+1) * sum(Y=0:infinity | F^Y/Y!) * e^(-F) =               [30]
  = (F/2+1) * e^F * e^(-F) = = 1+F/2
```

Eureka! So,

THE AVERAGE NUMBER OF I/OS REQUIRED FOR A DBGET MODE 7 AGAINST AN IMAGE/3000 MASTER DATASET WITH BLOCKING FACTOR 1 AND FILL FACTOR F IS 1+F/2.

At last.

THE APPLICATIONS

Now that all of you have been dazzled by the above display of Incredibly Incomprehensible Mathematical Formulae, an important question comes up:

WHY BOTHER?

As an amateur mathematician (i.e. someone who is crazy enough to actually LIKE the above mumbo-jumbo), I can answer the above with "Why not?" or "Because it's there" or "But aren't you just overwhelmed by the incredible beauty of the result?".

However, as a computer programmer who works on what is essentially a business computer, I feel that I ought to show some applications of the above mess. Well, here they come.

For one, note that the formula generated depends only on how full the dataset as a percentage, not on the actual number of entries in the datasets. Thus, if we compare the efficiency of IMAGE mode 7 DBGETs and KSAM FREADBYKEYs, we find that IMAGE should theoretically require only 1 to 1.5 I/Os, whereas KSAM requires on the order of log N I/Os.

Another point that this proves is that increasing the capacity of a dataset will not necessarily greatly improve the efficiency of master dataset access (as is commonly believed). For instance, if the capacity of a dataset that is 80% full is doubled, the average number of I/Os required for a DBGET mode 7 against this dataset will decrease from 1.4 to 1.2, which may not be significant enough to offset the two-fold increase in disc space usage.

On the other hand, if you see that the fraction of secondaries is much larger than 1-(1-e^(-F))/F and/or the average number of I/Os is much greater than 1+F/2, you may have come across a case in which IMAGE's hashing algorithm behaves badly. In this case, changing the capacity very slightly may cut the number of secondaries and the average number of I/Os dramatically. In one case which I ran across, changing the capacity by 1 cut the average number of I/Os in half! The actual number of secondaries and the actual average number of I/Os may be obtained from various database utilities such as DBLOADNG, DBSTAT2, or Robelle Consulting's HowMessy.

CONCLUSION

Thus, the two major results that were derived in this paper are:

If a dataset has fill factor (number of entries/capacity) F,

the expected fraction of primaries in that dataset is:

$$P(F)=\frac{1-e^{-F}}{F} \qquad [8]$$

and the expected average number of I/Os needed to retrieve an entry (if the dataset has blocking factor 1) is:

$$A(F)=1+F/2 \qquad [30]$$

A DISCUSSION OF THE EFFECT OF SYNONYMS ON PERFORMANCE

EXPERIMENTAL EVIDENCE

The following is a table of the results that were expected and the results that were obtained for several test datasets:

			Actual		Expected	
--C--	--N--	--F--	%Primaries	Avg num IOs	%Primaries	Avg num IOs
650	525	80.8%	67.2%	1.411	68.6%	1.404
331	193	58.3%	75.1%	1.337	75.8%	1.291
331	193	58.3%	75.6%	1.275	75.8%	1.291
15000	10582	70.5%	71.0%	1.387	71.8%	1.352
331	118	35.6%	80.5%	1.203	84.1%	1.178
250	206	82.4%	66.0%	1.456	68.1%	1.412

Of the following, some datasets (the one marked +) have primary percentages that are significantly less than expected and average numbers of I/Os that are significantly more than expected. The datasets marked - are exact copies of those datasets with slightly changed capacities -- their primary percentages and average numbers of I/Os are quite close to those expected:

			Actual		Expected	
--C--	--N--	--F--	%Primaries	Avg num IOs	%Primaries	Avg num IOs
+ 128	123	96.1%	7.3%	13.008	64.3%	1.480
+ 127	123	96.8%	52.0%	1.667	64.0%	1.484
- 126	123	97.6%	70.7%	1.407	63.9%	1.488
+ 320	284	88.9%	37.3%	2.810	66.3%	1.444
- 319	284	89.0%	65.5%	1.444	66.2%	1.445
+ 400	158	39.5%	69.6%	1.456	82.6%	1.198
- 399	158	39.6%	86.1%	1.158	82.6%	1.198

These datasets are good examples of instances in which slightly altering the dataset capacity (even decreasing it!) can significantly improve performance.

INTRODUCTION TO STEP BY STEP

By Michel Kohon, 20 Oct 81

8, route des Moines
95420 MAUDETOUR-EN-VEXIN
FRANCE

BACKGROUND

Since we introduced the Step by Step idea, we have found some discrepancies between what it means and what people think it means. The objectives of this document are:

- to explain Step by Step for end users,
- to specify the Step by Step method for DP professionals,
- to give Step by Step's main advantages and its possible adverse consequences.

TOWARD STEP BY STEP

Description of Data Processing:

To understand Step by Step, it is essential to analyze the DP function. Like all other human beings, DP personnel have to deal with two concepts:

- the reality,
- the users' dreams (also called requirements).

The reality is a difficult thing to change, to transform or simply to understand. The users' requirements are difficult to change, to transform or simply to understand. From the reality, the system analyst or programmer will try to develop a system which fulfills the users' dreams. Users can see the impossibility of their trying to accomplish this themselves.

Packaging:

The traditional DP approach to such a problem has been:

- to obtain the users' requirements,
- to formalize the requirements, and
- to design a system which would deal with all aspects of the requirements, and define programs covering all requirements;
- to program and implement all these programs together or, at least, to try.

This is called "packaging". Let's examine why it is likely to fail.

Getting the requirements:

- users do not know "specifically" what they want;
- users do not think enough about exceptions or specific circumstances;
- skilled users sometimes do not exist.

One way to reduce these problems is to educate the users and to improve the skills of the DP staff in questioning users. But whatever is done to improve the situation, the facts obtained from the users are only a partial statement of their requirements.

The second way, extensively used in the "packaging" method, is to have endless interviews and studies. This investigative period is sometimes so long that the company changes its product lines in the meantime, creating a high degree of confusion in the requirements. The dreams tend to be a mirage.

Formalizing the requirements:

- From a written document, users can only visualize the general lines of a system, not the details or the consequences.

- Specifications tend to be too detailed for parts of the system, while lacking specificity for other parts.

- System design is likely to generate a huge amount of paperwork consisting of the translation of users' requirements into program specifications, data base structure and paths of information in and out of the computer.

- Program design will include all possibilities requested by users, from the most useful to the least, from the most complicated to the least.

- Detailed analysis is likely to discover that new functions are necessary, which were not seen before, meaning new programs, an increase in program complexity or design changes; in any case, the first delay in the project.

Programming:

- From the system analyst's specifications, the programmer will create a program, probably to meet the technical objectives rather than the users' objectives.

- For interactive programs, he will have to think like a user, which is impossible because DP people are computer professionals.

- The programmer will find some errors in the analysis, which will delay the whole project because an essential program will not work correctly on time.

- Like the designer or the system analyst, the programmer would like to have a perfect system, so he will add beautiful but useless features. This is called the "galloping elegance" syndrome.

STEP BY STEP

As we said earlier, the users' requirements are likely to change. Why is this so?

The users' requirements are determined by the type of business they are involved with. The business can be production, sales, marketing, service, etc. Business can vary depending on season, crop, day or other factors. The consequences for data processing from a change in business are new and different information needs. One way of taking these changes into account is to make an exhaustive list of the possibilities and program all of them.

We also said that it is difficult for a user to visualize the end result of a program, and even more difficult when it is an interactive one. Why is this so?

A program is not static. The actions it performs vary dynamically, depending on the information that is entered. It is a moving body and is unlikely to be adequately described without using jargon. The same applies to mathematics or astronomy, or films. How can we visualize a film from a script? This is why the sooner you show the program to the user, the better it will be for his understanding.

The following two concepts are the foundation of Step By Step:

- to discover the users' actual requirements and to program all of them;
- to give the programs to the users as soon as possible.

Nothing really new, but keep these two concepts carefully in mind. You will be surprised by the methods of achieving them.

Getting the requirements:

What are we really interested in? To know what a user wants. If you ask a user "What drug do you want?", he will answer, "Vitamin C". In fact, the question is too specific. What we should ask the user is: "What are your problems?" He will perhaps tell you: "I can't recover my payments quickly enough. I have to pay too much in financing costs". If he knows why, you can start the second part of the study; otherwise, you will have to go into a problem-solving routine. Problem-solving is also a service; we advise you to use some of the KEPNER & TREGOE methodology to help you. In any case, you will have to find out why the payments are not cashed quickly enough.

We seem to be far from data processing. Not so. We are only processing non-structured information (the users' explanation) without a computer. But you know that the computer is not the most vital element for data processing.

Anyway, going back to the example (a real situation, by the way), we still find that "the invoices are not processed quickly enough". It appears that our user needs an invoicing system. Mind you, if you had asked, "What do you want?", he would probably have answered, "I want an order entry system", because this was his overall, long-term objective, and because it will solve his cash problem.

The confusion arises partly from mixing long-term objectives with short-term problems. If you could provide, overnight, an order entry system, including the invoicing package, there would be little or no problem. You would have solved the user's immediate problem. But, of course, you cannot implement it overnight, not even in a fortnight. You will conduct a long study and come back with phase one of the project ready, i.e., the order input. You will not solve the problem which has created the need of your expensive intervention (the cash is still not arriving faster).

Both you and the user will be very frustrated.

Breaking the Problem into Steps:

Obtaining the user requirements is a two-fold process:

- to identify the problems which have created the need of DP assistance;
- to set long-term objectives as well as short-term ones.

Identifying the long-term and short-term objectives will permit you, with the users, to draw a line of actions within an overall strategy. You will move from point A to point Z through points B, C, D,..., with each point being an objective. But how to order these points?

To provide a solution to the top problem means that you will give the maximum result in a minimum of time, and you will repeat this with each successive point. Order the objectives from the maximum payoff to the minimum. These will be your steps. Now you can make your design and organize the system in a structured way. Do not go into details. Remain flexible. The document you are preparing is the final report. It consists of:

- long-term objectives,
- objectives for the first and all following steps,
- sub-systems or programs related to each step, and
- priorities.

When this document is approved, you must write the programs - those of the first step, of course!

Programming a Step

A few remarks beforehand:

- The final aim is the program, not the analysis. So, until the program or its results are in the hands of the user, nothing is completed.

- There is a general law governing the whole world or any part of it: the offer-demand law. It applies to DP, as it applies to apples. We will see how.

A step is usually a move of two to three programs, but it could be more. To write the programs, you will have programmers, but never enough. Why is this so? Because the demand (user's request) automatically adjusts itself to the offer (programmer resources). We won't prove this here, but you can believe it.

The strange thing is that the more programmers you get, the more complicated the resulting system will be. This is the very well-known law stating that "adding people to a project will delay it in direct proportion to the number of added men". The explanation is easy to state: by increasing the offer, you increase the demand.

The only logical way to escape this dilemma is to limit the offer. How can we do that?

One way is to limit resolutely the number of programmers working on a project.

This approach is already frequently used, but for different reasons: to shorten communications and avoid misunderstandings, and to add flexibility and improve responsibilities. This is a method we still recommend using.

A second way is to limit explicitly the amount of time allocated to a program or system.

Let's imagine for a moment that we've said we have two weeks to program our step with the existing manpower. No more than two weeks. How can we best solve the problem in the amount of time given? The natural way will be to put on paper what the MUSTS and the WANTS are. If both can be produced in two weeks, we will program both, but that is unlikely. Therefore, how do we determine which steps will be MUSTS and which will be WANTS?

It really depends on their nature. If they are, or some of them are, necessary before we can program the next step, they have to be part of the NEXT STEP MUSTS. If they are not necessary before step X, insert them into the MUSTS of step X. The most important objective is to find the absolute MUSTS which can be produced with the CURRENT staff in a limited period of two weeks.

Well! That's it! There is nothing more to Step By Step. However, we will come back to two important questions.

How to Establish a Time Limit:

In our examples, we took two weeks, and this was not arbitrary. We think that two weeks is a manageable period. Less is difficult and dangerous, as stepping too quickly will permit neither the proper education of users nor the corrections to the programs. Programs must still be of outstanding quality, and, of course, you need some tools and structured design to step correctly.

More than two weeks is too much. Generally, users do not wait for you. They have other activities, very often repeated on a monthly basis. Therefore, to give them the program(s), you will need another week, which means that the program(s) will go into operation only after three weeks.

Experience also shows that two weeks is a "manageable" piece of time, possible and realistic.

How to Establish the Discipline:

This is the most difficult part of Step by Step, since it deals with human willingness - both the users' and the DP team's willingness. Here are a few tips which might help, but success depends more on the company and personalities than on anything else.

It is good to introduce Step by Step to the users. The more of a selling job you do before starting a project, the easier it will be. If the users do not want it, don't undertake the project at all, as users without commitment will never help you.

Use the MUSTS and WANTS method with the users. Look at all the problems - theirs and yours. Some MUSTS are yours. Explain why. Again, try to solve their problems. It is amazing how often you will find that a program can be easily replaced by a simple manual procedure.

Challenge the users. Everything is a MUST for them at the beginning. Ask always and again - WHY? Quickly, some of these MUSTS will turn out to be WANTS.

Never go back on the two weeks allowed. It MUST be done in two weeks. Try to imagine that in two weeks' time, it will be the End of the World. Users will laugh, but they will, as well, appreciate your concern.

STEP BY STEP PROGRAMMING

Step by Step can also be applied to implementing a particular program. The methodology known as "step-wise refinement" is not very far from Step by Step.

Like step-wise refinement, Step by Step needs a structured programming approach. Programmers can use almost any language to program in a structured way, with or without GO TO statements. It will be very dangerous to program Step by Step in an unstructured way, as the code, which will constantly change, will look lke a war theatre.

With Step by Step, unlike step-wise refinement, the analyst has to bargain with the programmer to obtain a workable piece of code in two or three days, then a second piece of code in a similar amount of time, and so on till the end of the step. This is a kind of sub-stepping, where the analyst has to play the user. It requires teamwork and discipline to keep the programmer from testing the exception module when the general one is not yet working.

STEP BY STEP: PROS AND CONS

One of the major bonuses for the users is that they take control of the product as soon as possible. This means that users can see the program reacting to their input. They can visualize how it will work day after day. In general, users ask for small modifications. It is a MUST, at this time, to avoid asking for and, therefore, to avoid making modifications which are lengthy, or which should be included in a further step.

A request may look like a MUST; the analyst and the users should determine whether it is. Both should remember that a MUST means that the program will not work when put into production. In other words, a modification needed a few days after the scheduled implementation is not a MUST.

It is the analyst's responsibility to plan this MUST for a further step. The users should remember that when they start using the program, they may discover that the request was irrelevant or inadequate. Irrelevant, because they might find that what they have proposed will make the program very cumbersome to use.

One of the major pitfalls with Step by Step is the possibility of stepping forward without clear final objectives. There is nothing more expensive than programming a step, then being obliged to re-program it during a further step. This will happen if the original objectives are absent or forgotten, and the result will look like a drunken man's walk.

A second problem can come from "stepping" too quickly. If two weeks is the agreed pace, stand firm on that. Rushing will decrease the quality of the programs, in terms of reliability and documentation; it will also kill DP team communication, as few people will be aware of a step's content.

A third problem is a switch in priorities. It is in the nature of Step by Step that steps may be swapped, due to reshuffling of priorities. This is not dangerous, so long as the DP team is aware of it, and organized for it.

CONCLUSION

We like to make analogies between data processing and other technologies or sciences. Analogies are very important, as you often create a new technique in your field which is very old in another field. Knowing this ancient technique may save you a lot of time.

When the first rockets were launched toward the planets, they were fully programmed. Everything was automatic and followed an unchangeable program. It all seemed logical, as all the parameters needed to find the target were known: the distance between planets, their relative speed and weight, the rocket's acceleration, weight, ... But the project failed. The objectives were not reached.

The next rocket generation included a new feature. Technicians were able to change the rocket's direction slightly during the flight to counteract the non-programmed consequences of the environment's reactions. This time the objectives were reached.

Implementing an on-line system in a commercial environment is like sending a rocket toward a moving object. You need to adjust and control the direction. This is Step by Step.

October 20, 1981
Michel Kohon

Transaction Processing Using VPLUS by Michael A. Casteel Vice President Computing Capabilities Corporation Mountain View, California, USA

Introduction

In the several years since the introduction of VPLUS, then called VIEW/3000, not much has been published regarding its application or techniques for using it. At the same time, thousands of HP3000 sites have begun to use it, either for their own purposes or as the 'front end' to an application package such as MM/3000. With such a population of users, much more deserves to be said about VPLUS than has been.

This paper is about transaction processing using VPLUS. In general, this means programmatic use of VPLUS, as opposed to data entry applications using ENTRY and REFORMAT. The following sections describe the use of the VPLUS intrinsics in on-line transaction processing; where they are used and what they do. There is also a discussion of some principles for transaction processing using VPLUS. New and prospective users can't be blamed for puzzling over the 30-plus intrinsics, 8 FORMSPEC menus and umpteen feature covered in the VPLUS manual. What could they all be for? When do you use them? It is hoped that this paper will provide a few answers.

What is Transaction Processing?

Most HP3000 users probably have a good feeling for Transaction Processing, but not necessarily in the context of VPLUS. Let's begin with some very basic definitions of 'Transaction' in order to develop a common ground for this presentation. My favorite definition of Transaction is from the user's point of view,

Transaction: A meaningful unit of work for a computer user, e.g. 'Add a New Customer' or 'Open a Work Order'.

This definition is most valuable as a reminder that transactions are not just programming exercises. The point is not how elegant, fast or powerful a transaction is; the point is that it get the job done.

It's very easy to get distracted by technicalities, such as the relative merits of Data Base Management systems, or Block mode versus Character mode operation. As we get deeper into the subject of Transaction Processing using VPLUS, we may even find ourselves distracted. When one is designing or developing an on-line system, however, it is essential to keep in mind that the real goal of the system is to help people get their jobs done. This is implicit in the currently popular phrase 'User Friendly' although I prefer to use more descriptive words such as Functionality, Flexibility, Consistency and Predictability. We'll get into more detail later; in fact, the user's point of view ought to be considered in every significant aspect of system design and development.

As we consider the topic of Transaction Processing using VPLUS a slightly more rigorous definition could help the discussion. A useful definition for Transaction when using VPLUS is,

Transaction: The processing of a sequence of VPLUS forms in order to accomplish a task, e.g. 'Add a New Customer ' or 'Open a Work Order'.

Many transactions will involve only one form. Some (hopefully few) might need dozens. 'Add a New Customer', for example, would probably only need one form in which to enter information about a new customer, such as name, address and telephone number. The processing of this form includes editing or validating the information (did the user remember to enter the customer's name?), building a record and writing it to the data base.

A typical system will include quite a few such simple, one-form transactions used to maintain a number of important files, particularly the Table or Master files common to most applications.

Files of Parts, Customers, Employees, Vendors, Codes, etc. usually require a facility to add, display, update and possibly delete individual records. These 'file maintenance' activities can generally be accomplished with simple transactions involving one VPLUS form. Usually, a single form will accommodate all the maintenance activities for each file. In this context, we may have transactions such as:

What is Transaction Processing? (continued)

Add Part   Change Part Display Part   Delete Part
Add Customer   Change Customer Display Customer   Delete
Customer

etc.

Other transactions may involve processing a number of different forms, usually associated with a number of related records in different files. An Order Entry transaction, for example, may begin with the processing of an Order Header form for entry of the usual order information: order number, order date, customer number, etc. Once the Order Header has been completed, the transaction advances to an Order Line form for entry of Quantity, Price, Part Number, and so on. Typically the Order Line form will be processed one or more times, as an order may consist of any number of lines. Such a transaction consists of the processing of an indefinite number of forms, i.e.

Order Header Form Order Line   Form Order Line   Form Order Line
Form

. . .

Although this transaction enjoys a certain element of simplicity, in that it only involves two different forms, there are highly complex transactions requiring dozens of forms. Such transactions are fairly rare, which is just as well, since few users could figure out how to operate such a complex transaction.

Processing

The processing of each form in a VPLUS transaction naturally divides between Form Processing and Data Processing. Form Processing includes all the functions implemented by VPLUS: Forms management, terminal interaction, data transmission and field processing. It can be accomplished simply by calling the appropriate sequence of VPLUS intrinsics.

Data Processing is a function of the application program itself. This includes performing advanced, application-dependent edits, building or changing data base records, and otherwise implementing the logic of the application. Transaction processing using VPLUS is accomplished by a suitable combination of Form and Data Processing functions.

Form Processing

There is implicit in VPLUS a natural sequence of Form Processing operations which apply to each form used in a transaction. The steps, and the intrinsics which implement them, are:

What is Transaction Processing? (continued)

1) Obtain the form. Call VGETNEXTFORM. 2) Initialize the form. Call VINITFORM. 3) Display the form. Call VSHOWFORM. 4) Accept input. Call VREADFIELDS. 5) Edit the input. Call VFIELDEDITS. 6) Finish processing. Call VFINISHFORM.

This relatively simple procedure accomplishes a lot of the work of transaction processing. The balance, the Data Processing functions, fit very neatly into the same procedure.

Data Processing

Transaction Processing combines Data Processing functions with the Form Processing procedure outlined above. An expanded outline showing the integration of Data Processing with Form Processing is as follows:

1) Obtain the form. This is accomplished with the VGETNEXTFORM intrinsic. At the start of a transaction, it is usually desirable to clear the terminal screen and display a particular form. The steps to accomplish this are:

   a) Move the form name to NFNAME in the Communications Area b) Clear (move zero to) the FREEZAPP and REPEATAPP options in the Communications Area, and to the status word (CSTATUS) c) Call VGETNEXTFORM d) Check the status word. If not zero, there's trouble.

In the course of the transaction, when advancing from one form to the next, it is normally not necessary for the application program to control NFNAME or the options in the Communications Area. These can easily be controlled by VPLUS Form Processing using the CHANGE verb. Thus, the usual steps are:

   a) Clear the status word b) Call VGETNEXTFORM c) Check the status word.

2) Once the next form has been obtained, it must be initialized. Failure to do so will cause the data from the previous form to be displayed in the new one. Initialization is accomplished by:

   a) Call VINITFORM b) Check the status word.

At this point there are often application-specific Data Processing functions to perform. There may be special initialization required such as moving the application program name and version number to the form. Or, there may be some transaction processing to do. For an inquiry or update transaction a record must be retrieved and moved into the form. Only then is the next step appropriate:

What is Transaction Processing? (continued)

3) Display the form. This is accomplished by a simple call to VSHOWFORM. With the exception of VOPENTERM and VCLOSETERM, this is the only intrinsic which actually writes to your terminal screen.

For certain transactions this may be all the processing required for the form; consider a form in the middle of an appending sequence in a transaction which displays a series of records. Once this form has been displayed, we might just as well go on to the next (returning to step 1 above), repeating until the screen is full.

Most transactions require further processing of the form. Often, the next step is to await input from the terminal user. The user may need to fill in the form with data for a new record; type changes to a record displayed in the form; or press a Special Function Key (f1-f8) to invoke some other operation. In such a case, the next step is:

4) Accept input from the user's terminal. This is done by a call to VREADFIELDS. Note: the preceding call to VSHOWFORM is needed in order to unlock the keyboard. If the user presses f1-f8, the corresponding number (1-8) will be returned in the Communications Area word LASTKEY. ENTER returns the value zero, transfers the contents of the form from the terminal to the VPLUS buffer in the program's data stack, and clears all the error flags for this form. Be sure to check the status word for errors.

If the user presses a Special Function key, the next step depends on the meaning assigned to the key. Often, the program will stop (or at least interrupt) the processing of the form; for example, when a user presses the key which means CANCEL or QUIT. Sometimes, the program will read the screen anyway (Autoread) and proceed with processing. Assuming the user made some entry which calls for continued processing, the next step is:

5) Edit the input. Data in the form may be edited quite extensively by calling the VFIELDEDITS intrinsic. If errors occur (NUMERRS not zero), a suitable error message can be obtrained by calling VERRMSG to retrieve the message text and VPUTWINDOW to move the message to the VPLUS window line.

Additional edits might be implemented in the program, such as file look-ups verifying numbers and codes, or complex application-specific edits. Errors resulting from these edits should be reported to the user, with an appropriate message for each, using the VSETERROR intrinsic. VPLUS will display the message associated with the first field in the form which is in error, whether the error was found by the program or in VPLUS processing.

Successful completion of the edits is usually a pererequisite to continuing form processing. When errors occur, the program should normally return to step 3, redisplaying the form with the results of edit processing. This can actually change some of the data on the

What is Transaction Processing? (continued)

screen, if the field contents were changed by the program or VFIELDEDITS. Also, VPLUS will highlight each field in error with the Error Enhancement specified on the FORMSPEC Globals Menu, position the cursor at the start of the first such field, and display the error message for that field (if you followed the above procedure). The user will then have the chance to correct the input and try again, or, if that's not the problem, to quit processing by pressing the assigned Special Function key.

6) Finish form processing. Call VFINISHFORM. Just as with the field edit processing in step 5, errors may occur. If so, you may wish to treat these as extended edit errors, using the procedure from step 5 and returning to step 3.

Often, the bulk of the Data Processing functions take place at this point, before going on to the next form. In an Add New Customer transaction, for example, now is the time to collect the edited input from VPLUS and build a new customer record.

This concludes the normal Form Processing of a single form. The next form in the transaction will repeat steps 1-6, and so on for each form. To summarize the combination of Form and Data Processing operations involved in processing one VPLUS form, I will restate the procedure. Steps marked with an asterisk (*) are Data Processing.

(*) Next form logic. 1) Obtain the form. Call VGETNEXTFORM. 2) Initialize the form. Call VINITFORM. (*) Initialization logic. 3) Display the form. Call VSHOWFORM. 4) Accept input. Call VREADFIELDS. (*) Special Function key logic. 5) Edit the input. Call VFIELDEDITS. (*) Field Edit logic. 6) Finish processing. Call VFINISHFORM. (*) Final processing logic.

Transaction Processing

You can see that the sequence of Form and Data Processing operations just described is appropriate for all types of transactions: Add, Delete, Display, etc. We are now considerably closer to understanding transaction processing using VPLUS. To process a transaction, we process the sequence of forms used in that transaction, one after the other. To process each form, simply follow steps 1-6 in the previous section. If you look closely at these steps, you'll notice that VPLUS intrinsic calls are doing most of the work. Except for the Data Processing requirements, you could write one program to follow those steps and use it for all the transactions you care to implement. For any transaction, use FORMSPEC to create the sequence of forms and define the field processing logic. To execute the transaction, run the 'universal program' and start it with the first form in the transaction.

Such an idea can only work in reality if we can also solve the Data Processing problem. What sorts of Data Processing functions are there, and how do they tie in with this "universal VPLUS processing cycle"?

Initialization Processing

After VPLUS form initialization processing is complete, the application program has a chance to move data to the form before displaying it. In an Add transaction, the program might preset some fields to default values not known to VPLUS. These values would automatically be placed in the next record unless changed by the user. In a transaction which involves retrieving records, the program might retrieve a record and move it to the form before display. See the section on Retrieval Processing.

Field Edit Processing

In most cases, VPLUS can perform most of the needed edit processing, but there are often a few edits which the program must perform. Quite often, one or more of the fields entered need to be looked up in a file or table, either to check that the value exists or to see that it doesn't. Occasionally, there are more complex edits which require programming beyond the capabilities of VPLUS, usually involving file access.

Add Processing

This is perhaps the simplest type of transaction we could consider. Once a form has been processed through Finish, we have a form full of data edited and processed to our specifications. What remains is to make a new record (or records) from the data in the form and write it to the data base or file. Typically, the form is just a picture of the record, with each data item in the record appearing as a field in the form.

Retrieval Processing

This is more interesting. Before we can retrieve a record, we must determine how to locate it. The usual way to locate a specific record (or set of records) is to match a data item value; for example, Customer Number = 24. So, we begin by processing a form in which the user enters such values to be used for record selection. The result is a set of values edited and processed by VPLUS. Using these values, locate a matching record, move it into the form and display it. This can be done during initialization of the next form in sequence, moving the data to the form before display (step 3). This uses two forms. Alternately, we might choose to vary the standard procedure slightly, move the data into the same form and return to step 3, which will re-display the form with data from the record.

What is Transaction Processing? (continued)

Both techniques are useful depending on the situation. Displaying the record in the same form is simpler, avoids the trouble of getting another form, and eliminates the cost of displaying another form on the terminal. On the other hand, the edits to be applied when processing the record (say, in an Update transaction) may be quite different from the edits appropriate for entering selection values. Using two forms allows two different sets of edits to be applied in VFIELDEDITS. Even in this case, much of the cost of using two forms can be eliminated by making the two forms members of the same forms family. This eliminates the cost of displaying a new screen, and the program only needs to deal with a single field layout.

Update Processing

The first step in updating a record is to retrieve it. The preceding section described the general procedure for retrieval. Once the record is displayed, the user has the opportunity to make changes and press ENTER, or to request some other function such as CANCEL or retrieve the next record. On reaching the end of the processing cycle for the form, we again have a set of values edited and processed, with the user's changes included. Move them back into the record and update it.

Delete Processing

As in update, begin by retrieving the record. When the user requests the function, delete the record from the data base.

Transaction Request/Menu Processing

A Menu is just a list of transactions provided for the convenience of the user, who will usually select one of the transactions to perform next. When a Menu form has been completely processed, the edited and processed input identifies the desired transaction. The next step is to proceed to processing the first form in that transaction. Note: Field Edit processing can be provided to verify the user's authority to perform the requested transaction, e.g. password checking.

A Programming Approach

The 'universal VPLUS processing cycle' makes a good basis for an on-line programming standard. It is possible to program all the Form Processing logic in a single routine to be used in almost all on-line transactions. This one routine will perform all the form processing, including user interaction. The points where Data Processing operations are appropriate are quite well defined, so the standard routine could PERFORM the (transaction-specific) logic at each such point.

Such an approach offers a number of advantages:

What is Transaction Processing? (continued)

1) Standardization	- Basic program structure is similar even when different programmers work. As important, the user interactions will tend to be consistent as a result of using the same algorithm in each transaction.
2) Economy	- The basic Form Processing logic only needs to be debugged once. This should be one less problem for all the programmers. In many cases, the sequence of forms for a transaction can be processed and reviewed before any Data Processing logic is coded.
3) Modularity	- The suggested partitioning of functions provides a ready definition of modules for programming. With the Form Processing already operating, many Data Processing modules can be developed and tested independently, e.g. Field Edit processing.
4) Maintainability	- Modifications to Form Processing logic will be greatly aided. For example, support for new VPLUS features can be added to the standard Form Processing logic and readily spread among all transactions. This is in addition to the extra ease of maintenance resulting from all programs having the same basic structure.

The Ultimate Program

It seems that the universal program is almost within our grasp. In order to process a transaction, this program would need to know:

1) The name of the first form.

2) The TYPE of Data Processing operations to perform: Add, Retrieve, Update, Delete, Menu.

3) The file or data set with which each form is associated.

4) The data item in the record with which each field is associated.

5) File lookup edits needed for each field.

That takes care of most transactions. Most, but not all. In the general case, there must still be a way to insert transaction-specific Data Processing logic, even if only for a few of the transactions.

This approach works. It is the basis for INSIGHT II, the first VPLUS-based Transaction Processor for the HP3000. Transaction Processing Using VPLUS

What is Transaction Processing? (continued)

Guidelines for Better Systems

Regardless of the application, there are a few principles which can help produce better transaction processing systems using VPLUS. For the most part, these principles deal with various aspects of system design rather than implementation. Experience has shown time and again that there is no substitute for good, carefully thought out systems design. A well-designed system is likely to survive even a poor implementation, while all the programming tricks in the book may not be enough to rescue a flawed design.

I wish to present here four principles which I try to apply when developing transaction processing systems using VPLUS:

1) Remember the poor user

2) Limit the number of screens in a transaction

3) Take advantage of block mode

4) Take advantage of terminal features.

Remember the Poor User

There are a few key words to keep in mind which bear directly on the user's perception of the system, affecting his degree of success using the system and thereby the success of the system itself. Briefly, these are:

Functionality	- It is important that the system provide the necessary functions to do the job. Make sure you understand the problem before trying to provide a solution, or your system is likely to the problem.
Consistency	- Develop and adhere to standards for all your VPLUS systems. It is essential to standardize the operation of all the transactions in an application. The "universal VPLUS processing cycle" presented earlier is a step in this direction. Take the next step and standardize each of your Transaction Processing cycles (Add, Display, etc.). It is desirable to standardize forms design as well. If you display the current transaction code, date, etc. on the screen, always put them in the same place so the user will know where to find them. In general, use common sense to avoid unnecessary differences between transactions. The user has enough problems as it is.
Flexibility	- I particularly recommend that you try to avoid over editing, and give the user the greatest possible flexibility consistent with correct system

Guidelines for Better Systems (continued)

operation. The editing features of VPLUS are a great help in maintaining the integrity of data in the data base, but it's easy to go too far. Think about what editing is important and what is inessential. The world is guaranteed to change faster than than the system. Today's extra edit may prevent next week's work from being processed.

Predictability - This keyword is important in making the user comfortable with the system. Too many surprises will unsettle anyone. A consistent systems design with a good set of standards should provide a good level of predictability.

Almost every user will come to value two particular functions which I believe should be included in every VPLUS system standard. I call them EXIT and REFRESH, and assign each function Function Key.

EXIT is the user's escape hatch, and serves to CANCEL whatever is going on. While in the midst of a transaction, the user should be able to terminate or abort the transaction by pressing EXIT. There may be some transactions (or parts of transactions) where you feel that EXIT should not be allowed. That's OK, but before disabling this function, please take another, careful look at the design of that transaction.

By REFRESH I mean a function which simply performs the VPLUS $REFRESH operation. Any number of accidents or errors can cause damage to the forms displayed on the terminal. If there's no predictable way to straighten things out, the user will be at a loss to continue working. With a REFRESH key, a problem such as a power failure can usually be rectified by resetting the terminal and pressing REFRESH.

Limit the Number of Screens in a Transaction

Everyone who has used VPLUS at anything less than 9600 baud is aware of the overhead involved in displaying a new form on the terminal. Strictly from a performance point of view, it makes sense to repaint the screen as infrequently as possible. A transaction which uses one form to input a record key and another to display the record will suffer if the screen continually needs to be redrawn. Either a high data rate or a forms cache in the terminal can improve the performance of this transaction, but even so, the alternation of displays is likely to prove tiresome to the user. If it can be done without cluttering the display, combining the two into a single form has an appeal from both the performance and esthetic aspects. In this particular example, this can even be done with a relatively minor impact on the program by replacing the two forms with two members of a forms family sharing the combined screen layout.

Take Advantage of Block Mode

Guidelines for Better Systems (continued)

Nothing puzzles me so much as the occasional VPLUS application which requires the user to struggle through the transaction, typing in one field at a time and pressing ENTER. Using a series of forms containing one field each, the system designer has used an excellent block mode tool to create what is effectively a character mode system. While this is surely a rare practice, I feel that it is worth pointing out that the true VPLUS block mode approach has its own advantages, including the ability to closely simulate character mode processing.

To me, the biggest advantage of block mode is that it gives the user instant, local access to all the fields in the form. If the computer flags an error in the 20th field of the form, the user can correct it or the first field (if that's where the problem really lies) with equal ease.

This does not require that the user fill in all the fields in the form before pressing ENTER to request edit processing. The user can press ENTER at any time and VPLUS (and the application program) will edit whatever has been typed. When the screen is updated with the results, all remaining fields which require input will be highlighted, and the cursor will be positioned to the first one. Thus, a user who prefers the field by field editing characteristic of character mode processing can have it, while those who prefer can fill in the whole form without waiting for a response from the computer. The user is in control.

To realize the full potential of VPLUS block mode, your program edits (such as file look-ups) should be field-oriented as VPLUS edits are. This means that the program should perform edit processing even if the form contains VPLUS-detected errors. For this reason, I recommend that edit routines should obtain screen data with VPLUS field access intrinsics such as VGETFIELD or VGETINT. If VGETFIELD returns an error, then VPLUS has already detected an error in the field, so no further editing is needed. Go on to the next field. If VGETFIELD returns successfully, then the program edits should be applied and any error reported using VSETERROR. This distinction of fields containing VPLUS-detected errors cannot be made if VGETBUFFER is used to obtain the contents of all fields at once.

The advantage of applying the program edits at the field level is that the user receives the full benefit from each press of the ENTER key. Otherwise, the user will no sooner clear all the VPLUS edit errors when the program edit errors will appear.

Take Advantage of Terminal Features

The most widely available and useful terminal feature is Screen Labels, available on all 262x terminals as well as the 2382. Virtually every meaningful VPLUS application has defined functions for the labeled keys, such as:

Delete record	Display Next record

Guidelines for Better Systems (continued)

Exit/Cancel/Abort	Display Previous record Print screen
Refresh screen	

Descriptive key labels on the screen, where they won't get lost, have got to be a great boon to the user. Define labels in FORMSPEC, and enable them when your program calls VOPENFORMF.

Aside from screen labels, the really useful terminal features are mostly limited to one terminal: the HP2624B. In addition to providing screen labels, this terminal supports:

Security Video	- blanks out sensitive fields
Local Edits	- the terminal itself will test for required fields, data types, and even justify and fill
Modified Field	- the terminal only transmits the contents of fields which were changed, not the whole screen
Forms Cache	- several forms can be stored in the terminal's cache and thereafter can be displayed at high speed since they need not be sent from the computer.

Although the 2626 also has Security Video, and a limited Forms Cache facility, the 2624B is clearly the terminal of choice for VPLUS applications.

HPTOOLSET - An Inside View

Lynn Smith
Hewlett Packard - IND
19420 Homestead Rd
Cupertino, CA 95014

HPTOOLSET is an integrated collection of software tools or subsystems, aimed at increasing programmer productivity on the HP3000. Each phase of program development: creation, modification, translation and execution, is simplified. HPTOOLSET minimizes the distinction between the various tools required to do program development, providing a friendly, uniform interface.

When HPTOOLSET was first conceived as a program development environment, the primary objective was to provide a powerful, yet simplified method for implementation of application programs. Beyond this were several more goals. We needed to provide an integrated package; the tools should blend together and have a common style of interface. The product should be extensible; we did not want to preclude the later addition of other tools or languages. It should be friendly and easy to use. The interface between the user and the system should be versatile to allow for customization, and modular to allow for localization. Finally, we did not strive to be compatible with existing HP products such as EDIT/3000, or MPE DEBUG. Nor was any attempt made to simulate products provided by outside vendors.

Throughout this paper I will show how the various portions of HPTOOLSET contribute to programmer productivity. It will become apparent how our original objectives have influenced the character of the different subsystems.

USER INTERFACE

The user's perception of HPTOOLSET hinges heavily on his interaction with the product. HPTOOLSET isolates this interface into a subsystem known as the USER INTERFACE (UI). This partitioning yields several benefits. All of the intelligence required to do terminal I/O is centralized in a few routines. If terminal requirements change, the modifications are needed in only one place, not throughout the product. The UI handles the programmable function keys and menus. Each subsystem need not be cognizant of the particular function key definition or menu layout at any given time. The function key and menu definitions are stored in files. This provides great flexibility for changing them, as well as aiding in localization.

One way of making HPTOOLSET convenient to use is to provide screen labelled function keys to execute commands. Generally, pressing a function key accomplishes the same purpose as typing the corresponding command, and the two can be used interchangeably. The available terminals provide only eight programmable function keys; too few for our purposes. So the function key labels are defined such that there are always two sets of commands available. One set, known as the permanent set, always remains the same. It contains functions that are relevant throughout HPTOOLSET. The temporary set is sensitive to the current activity. One label in every set is reserved for ALTSET,which is used to switch back and forth between the sets.

The function key labels for at least one set of commands are always displayed on the terminal screen. On the 262X series of terminals, the labels for the active set are displayed using the predefined template at the bottom of the screen. Both the permanent and temporary function key labels are displayed on the 264X terminals. They appear side by side at the top of the screen with arrows indicating which set is active. The labels are displayed in a highlighted template, which is protected by memory lock.

The UI is divided into three basic modes of interaction: command, menu, and visual modes. Each of these is characterized by the particular form of user interaction. Command mode is probably the most familiar to the user. Instructions are conveyed by typing them in response to the double arrow prompt ">>", or by pressing equivalent HPTOOLSET defined function keys.

Menu mode occurs when HPTOOLSET has a variety of items to present, from which the user can select one or more; or there are default characteristics which the user can modify. The user indicates his selection by typing a character to the left of the desired item(s), and pressing the function key corresponding to the operation to be performed on it. On menus that are used to define characteristics, HPTOOLSET highlights the options that the user can change. To modify them, the user simply types over the default with his own choice and presses the SETOK function key.

Visual mode is available when the user is reading or editing a file. A page is displayed on the screen, and modifications are made using the edit and cursor control keys on the terminal. HPTOOLSET uses the concept of "marking" in visual mode. By positioning the cursor and pressing the MARK function key, the user can indicate a character, word, line, or lines on which future operations will be performed. Function keys are used to operate on "marked" lines such as in MOVE or COPY. The function keys are also used to do global commands such as FIND or CHANGE.

A fundamental design decision for the UI was to determine which terminal(s) HPTOOLSET should support. We could design a special purpose terminal, perhaps allowing for code to be downloaded into terminal memory. This was appealing since it would enable customization of the product with the terminal it would be used on. However, this approach had various drawbacks. The product would be tied to specific, and potentially expensive, hardware. This precluded the large existing base of 2645 and 262X terminal users from using the product. Likewise, we considered supporting only the 2626 terminal, taking advantage of windowing and other special capabilities it provides.

The most viable alternative was to base HPTOOLSET on the 2645 terminal, and support all terminals that are 2645 compatible. This includes the 2622, 2623, 2624, 2626, 2647 and 2648 terminals. HPTOOLSET depends on many of the features of a 2645 such as block mode and user definable function keys. In order to accomodate the users of terminals that are not 2645 compatible, it was decided to provide a rudimentary subset of HPTOOLSET that will work on any HP terminal.

HELP

A major step towards making HPTOOLSET friendly and easy to use was the inclusion of an extensive online Help facility. The user can obtain information on any of the commands or subsystems at any time. Help is context sensitive, meaning the user gets different default explanations based on what he is doing at the time he accesses Help.

There are three methods to enter the Help subsystem. The user can type the command "HELP" when in command mode, press the permanent Help function key, or append a question mark at the end of a command. If Help is accessed after a syntax error, or the question mark is used, the correct syntax for the command is given. If Help is requested after a non-syntax error, additional information and possible corrective actions are given. When there was not any error, an overview of the current subsystem is displayed.

Help is organized in a tree structure. At the root is an overview of HPTOOLSET, with instructions on how to obtain more information. At the next level is an overview of each subsystem. Within each subsystem is a description of each of the commands explaining the syntax, parameters, operation, and sample usage. Function keys are used to traverse the tree. Like the function key definitions and menu forms, the Help screens are stored in a file. This makes Help easy to modify, expand or localize.

WORKSPACE

When developing a program, the user has to keep track of a set of files. One way that HPTOOLSET has simplified the development process is to associate the needed files together in a directory called a "Workspace". Through this mechanism, the user need only define the USL and Program file when the workspace is created. Subsequently, the appropriate file will automatically be invoked for Compile, :PREP, or :RUN. Additionally, the workspace keeps track of information that is pertinent for each source file. This includes the source language, tabs, a user defined label, and version information.

A fundamental program development feature is source version management. As a program evolves, it may change significantly. Through version management, the user can arbitrarily "freeze" a version of his source file. This version is protected from the changes that are made later, yet can be referenced at any time. This capability is provided with the SETVERSION command. The user can designate up to thirty two active versions of any source file.

Often several people are working on the same project. This usually requires the sharing of files. The workspace provides a mechanism to accomodate file sharing. The user can import a file via the USE command, or designate a specific version for others to access with the SETREFERENCE command. In this way, a stable version of the source file can be set aside for other users to reference, while the owner can continue to make modifications.

TSAM

In order to implement version management, HPTOOLSET needed the ability to store multiple versions of the same file within one physical file. Since neither the MPE file system, or other HP3000 tools provide this capability, we designed our own access method, TOOLSET Access Method (TSAM).

The requirements for TSAM are straight forward. It accomodates version information, provides reasonable performance within HPTOOLSET, and maintains the integrity of the user's file in the event of a system failure. Rather than actually copying the file each time a version is desired, the different versions are stored as changes to the previous version. This results in better disc utilization, since redundant information is not being stored. Another technique that TSAM uses to conserve disc space is blank compression. Trailing blanks are stripped from each line, saving as much as half the required file space.

EDITOR

Another feature of HPTOOLSET is the full screen editor. It displays a portion of the edit file on the screen for the user to modify. Changes are made by simply positioning the cursor and typing over the existing text, or by using the terminal edit keys: insert and delete line, and insert and delete character. As modifications are made, the file is automatically changed whenever carriage return or a function key is pressed. The need for time consuming "text" and "keep" operations is alleviated since the file is constantly being updated.

Function keys are used in the editor to move from one location to another in the edit file. They are also used to facilitate editing functions that cannot be done with the terminal editing keys, such as FIND, MOVE and COPY.

HPTOOLSET has integrated the READ and LISTING functions with the full screen editor. Their appearance is the same, only the user is restricted from making modifications. He is able to move through the file, mark lines, or do finds in the same fashion in each. All utilize visual mode, and therefore have a common interface.

As with many other subsystems, we had several alternative ways to define the editor. We could have made it line oriented such as EDIT/3000. If was felt that this method of editing was too cumbersome for most purposes; although we have provided commands that operate on the edit file in line mode for those occasions which it is appropriate. We considered doing a block mode editor, but felt it had the potential to create an undue load on the system.

PROGRAM KEY

The subsystem that is responsible for the translation phase of program development is known as the Program Key (PK). Its purpose is to interface with the COBOLII compiler and the MPE Segmenter to ready the user's program for execution. Currently, program translation is performed through three discrete steps: compile, prep, and run. HPTOOLSET has combined these together into a single function referred to as GO. GO takes the indicated source file(s) and compiles them into the default USL file. It then prepares the same USL file into the default program file. When this is complete, it begins execution of the program. By merging these steps into a single function, the user is relieved of the necessity of learning the specific details of each step. Nor need he be aware of the auxillary files, such as the USL and program file, that are required.

HPTOOLSET has implemented the time consuming compilation phase as a background process. This means that the user is free to do other work while the compile is executing. The compile executes in the same system queue as batch jobs, so it is not competing with the online sessions for system resources.

If the file being compiled is owned by the current workspace, the COBOLII listing is saved online in a file. This way the latest compile output is always readily available, without the need to juggle printed listings. When the listing is displayed, the user can toggle between it and the source file with the touch of a function key. This is invaluable for correcting compilation errors. Likewise, the user can use the listing when his program is executing to set breakpoints or display data items.

SYMBOLIC DEBUG

An area of program development that was in need of simplification was program debugging. In the past, there were many complex steps required to debug a COBOLII program. For the most part, it was the responsibility of the user to learn the details of these steps, as well as the intricacies of the system architecture. To determine the value of a data item, the user was required to perform the translation between variable allocation and actual memory addresses, based on information from the compiler's symbol table. He also needed to translate the Segmenter mapping and compiler code offsets to determine the location of a statement to set a breakpoint.

HPTOOLSET has internalized all of these translations. The user can reference his data items and program locations symbolically. He sets breakpoints using the Paragraph or Section name from his source, or the compiler generated statement numbers. Data items are displayed and modified simply by referencing them as they appear in the program. When a data item is displayed, by default it is output in the format defined in the COBOLII PICTURE clause. In this way the user does not need to convert it from its internal binary or octal representation. Data items are modified using the rules of the COBOLII MOVE statement.

HPTOOLSET provides additional capabilities to improve program debugging. The user can monitor the flow of his program using the TRACE command. It will display the name of each paragraph and section as it is executed. Likewise, he can display the previous paragraphs that were executed before reaching the current location, using the RETRACE command. He can monitor a specific data item with the DATATRACE command; which displays the value of the data item whenever its value changes.

There is quite a contrast between the old style of program debugging and HPTOOLSET's symbolic debug. HPTOOLSET provides the features to make debugging simple and easy to understand. The user does not need to learn the machine architecture, or be intimidated by the complexity of the required conversions. He can think of his program in the terms which it was written, using paragraph, section and data item names. This is a tremendous boost to productivity.

HPTOOLSET significantly improves programmer productivity. It provides the tools necessary for a programmer to develop and debug COBOLII programs. Online Help makes it easy to learn. The bookkeeping required to track all the necessary files is simplified through the workspace. At the user's discretion, he can create versions of his source file. Editing is quick and easy. Program translation is unified into a single operation, most of which is performed in the background, allowing the user to continue with other work. Most of all, debugging is convenient and powerful. It is easy to set breakpoints, display and modify data items, and trace program execution. In addition to providing a powerful set of capabilities, HPTOOLSET integrates them together with a common interface.

Using V/PLUS from Pascal

Technical Report, August 1982, by

DAVID J. GREER

Robelle Consulting Ltd.
27597-32B Avenue
Aldergrove, B.C.
Canada VOX 1AO
(604) 856-3838
Telex 04-352848

Introduction

This report describes the use of V/PLUS on the HP 3000 from the Pascal language, emphasizing the achievement of "good" programming practices through full use of Pascal's best features. Examples are drawn from the V/PLUS ENTRY program, which the author has rewritten in Pascal. No previous knowledge of Pascal is assumed, but a beginner's understanding of V/PLUS will be helpful to the reader.

V/PLUS is the terminal interface system supplied by Hewlett-Packard for their HP 3000 computer, and Pascal is a programming language that is just coming into use on the HP 3000. Can you access V/PLUS from Pascal? If so, what problems will you face and how can they best be overcome? Will you find any of the special features of Pascal helpful in simplifying V/PLUS programming?

These are the questions addressed by this report. In the process of answering them, we will give you a good look at the programming language Pascal and of "good" programming practices.

The structure of this report will follow the process of developing the ENTRY program in Pascal. The ENTRY program is described in the V/PLUS manual (HP Part #32209-90001), but the reader is discouraged from reading the sample ENTRY programs given in the manual. It is assumed that the reader has used FORMSPEC to define forms and that he has experimented with the ENTRY program.

Pascal/Robelle includes the complete source file for the ENTRY program. If you have Pascal/Robelle Version 3.1, obtain a listing of the ENTRY program and refer to it as you read this report.

Using V/PLUS from Pascal

Getting Started

Before accessing V/PLUS from a program, two things must happen: 1) a formfile must be opened and 2) the terminal must be set into block mode. The following Pascal program opens the file FORMFILE and the terminal. Unusual bits of code (i.e., non-standard or poor programming practices) are indicated by a comment (e.g., Note 1) and explained below.

```
program vplus (INPUT,OUTPUT);
type
  int = -32768 .. 32767;
var
  vcomarea = array[1..60] of int;  (* Note 1 *)
  vfilename= packed array[1..36] of CHAR;
procedure vopenformf;  INTRINSIC;     (* Note 2 *)
procedure vopenterm;   INTRINSIC;
procedure vcloseterm;  INTRINSIC;
procedure vcloseformf; INTRINSIC;
begin
   vcomarea[2] := 5;    (* Note 3 *)
   vcomarea[3] := 60;
   vfilename := 'FORMFILE ';
   vopenformf(vcomarea,vfilename);
   if vcomarea[1] <> 0 then
      WRITELN('Unable to open formfile. Error: ',vcomarea[1]:1)
   else
   begin   (* Note 4 *)
      vfilename := 'TERMINAL ';
      vopenterm(vcomarea,vfilename);
      if vcomarea[1] <> 0 then
         WRITELN('Unable to open terminal. Error: ',vcomarea[1]:1)
      else
      begin
         vcloseterm(vcomarea);
         vcloseformf(vcomarea);
      end;
   end;
end (*vplus*).
```

The type declarations declare names for forms of data structure that will be used in allocating several variables, or used as parameters to procedures. We define int as a type because we will need to allocate many single-integer variables for V/PLUS, and the format for the standard type INTEGER on the HP 3000 is a double word (32 bits).

The var declarations allocate the variables of the program (type declares only the form of potential variables). Please note that each variable is defined with a very precise type or form (e.g., vfilename : packed array[1..36] of CHAR), but none is initialized.

A few mechanical details: comments are enclosed within special symbols (i.e., (* Note 1 *)), external procedures must be declared, the mainline is enclosed by a begin-end pair, statements are separated by semicolons, colon-equals (:=) is the assignment operator, string literals are delimited by single quotes, WRITELN prints messages on the file OUTPUT, the program starts with a program heading defining the name of the program and the files it will access, and the program ends with a period (i.e., end.).

The notational conventions used in these samples are: reserved words in underlined lower-case, pre-defined words in CAPITALS, and user identifiers in lower-case.

(* Note 1 *)

The type of the vcomarea (array[1..60] of int) should be declared as a separate type. As the ENTRY program grows larger, it will be necessary to pass the vcomarea as a parameter. This CANNOT be done unless we have a type that describes the vcomarea. There is an additional benefit in declaring the type of the vcomarea as a separate type: we can change the layout and form of the vcomarea without seriously affecting the rest of the program.

(* Note 2 *)

The V/PLUS intrinsics must be declared for Pascal as external procedures. In HP Pascal, this is done by giving the procedure name, followed by the magic word INTRINSIC. Notice that the parameters are not declared; HP Pascal picks up those definitions from the system intrinsic file (SPLINTR.PUB.SYS), just like the SPL or FORTRAN compiler. Pascal/Robelle uses a slightly different convention for external procedures. In both cases, calling the V/PLUS intrinsics is strictly NON-STANDARD, because the V/PLUS parameters can have a variable format. Later, we will show how to protect yourself when using V/PLUS.

(* Note 3 *)

Pascal variables (the var part of the program) cannot be initialized at declaration time. Therefore, you must have explicit statements to assign them initial values (e.g., vcomarea[1] := 5). HP Pascal has an extension for initial values, but it is non-standard, and cumbersome to use.

(* Note 4 *)

We will use indentation to show the control-flow of the program. Levels that are controlled by a higher-level structure will be indented three spaces. A consistent indentation style is important for understanding programs.

The VCOMAREA

The V/PLUS communication area is used with every V/PLUS intrinsic. The current declaration of the vcomarea provides no definition information. For example, what is the meaning of vcomarea[2]? We have also seen that it would be useful to have a standard type which describes the vcomarea. The following type declaration improves on

Using V/PLUS from Pascal

our early declaration for the vcomarea:

```
   const
      vcomlen   = 60;      (* Note 1 *)
      vfnamelen = 16;
   type
      int = -32768 .. 32767;
      dbl = INTEGER;       (* Note 2 *)
      vlangtype = (cobol,basic,fortran,spl,rpg,Pascal);  (* Note 3 *)
      vmodetype = (collecting,browsing);
      vkeytype  = (vf0,vf1,vf2,vf3,vf4,vf5,vf6,vf7,vf8);
      vfnametype = packed array[1..vfnamelen] of CHAR;
      vrepeatopt = (vrepeatnorm,vrepeat,vrepeatappend);
      vfreezeopt = (vclear,vfreeze,vfreezeappend);
      vcomareatype =
         record        (* Note 4 *)
            status:     int;
            language:   vlangtype;
            length:     vcomlen .. vcomlen;
            extsize:    0 .. 0;
            mode:       vmodetype;
            lastkey:    vkeytype;
            numerrs:    int;
            filler1:    packed array[1..4] of CHAR; (* Note 5 *)
            labelopt:   int;
            cfname:     vfnametype;
            nfname:     vfnametype;
            repeatopt:  vrepeatopt;
            freezeopt:  vfreezeopt;
            filler2:    int;
            dbuflen:    int;
            filler3:    packed array[1..4] of CHAR;
            deleteflag: BOOLEAN;
            showcontrol:int;
            filler4:    packed array[1..16] of CHAR;
            numrecs:    dbl;
            recnum:     dbl;
            filler5:    packed array[1..4] of CHAR;
            termfilenum:int;
            filler6:    packed array[1..22] of CHAR;
         end;

   procedure initvarea(var varea : vcomareatype);
   begin
      with varea do     (* Note 6 *)
      begin
         status         := 0;
         language       := Pascal;
         length         := vcomlen;
         extsize        := 0;
         mode           := browsing;
         lastkey        := vf0;
         numerrs        := 0;
         filler1        := '    ';
         labelopt       := 1;
         cfname          := ' ';
         nfname          := ' ';
         repeatopt       := vrepeatnorm;
         freezeopt       := vclear;
         filler2         := 0;
         dbuflen         := 0;
         filler3         := '    ';
         deleteflag      := FALSE;
         showcontrol     := 0;
         filler4         := '          ';
         numrecs         := 0;
         recnum          := 0;
         filler5         := '    ';
         termfilenum     := 0;
         filler6         := '                    ';
      end;
   end (*initvarea*);
```

We now have a communications area which can be passed to procedures and is much easier to understand. The V/PLUS communication area is the most important data type used in the ENTRY program.

We introduce a new concept: procedures. A procedure is a separate entity which can be invoked by using its name (e.g., initvarea(vcomarea);). Parameters can be declared (e.g., var varea:vcomarea;), and the use of var in the parameter list indicates that changes to the parameter (i.e., varea) change the passed variable (i.e., vcomarea). Contrast this with a value parameter declared below.

(* Note 1 *)

The const declarations define names for literal constant values (e.g., vcomlen, instead of 60). Using names instead of literal values improves readability and makes it easier to change a program.

(* Note 2 *)

As mentioned in the first sample program, the standard type INTEGER is defined as 32 bits in HP Pascal (and Pascal/Robelle). To make it very clear to the reader whether you want a single integer or a double integer, we recommend creating two user types (int and dbl) in every program and using them in subsequent declarations, rather than INTEGER.

(* Note 3 *)

We declare a special SCALAR type for several fields in the communications area (e.g., mode : vmodetype). A SCALAR type is one consisting of explicitly named values (e.g., vmodetype = (collecting,browsing);) and no others. These constants can be assigned to variables and passed as constants to procedures (e.g., mode := collecting;).

Note that using scalar types is better than using constants. In general, we are not concerned with the numeric values associated with the scalar type; but SCALAR types are always assigned a value starting with zero, and incremented by one for each SCALAR type (e.g., collecting = 0 and browsing = 1). Another benefit is that the variable 'language' can ONLY take on one of the values in 'vlangtype' and no others (e.g., language := spl; is o.k., but language := 0; is not). This reduces the complexity of our data structures and adds redundancy, making our programs easier to read and debug.

(* Note 4 *)

The vcomareatype is declared as a record structure; this is equivalent to the 01-05-10 structure in COBOL, in that records may be nested within records. The type vcomareatype is used when we want to declare the actual V/PLUS communication area (e.g., var vcomarea : vcomareatype), and when the communication area is defined as a parameter (see the initvarea procedure above).

(* Note 5 *)

The word "filler" has no special meaning in Pascal. We use "filler" followed by a sequentially assigned number to represent areas of the communication area which should never be changed.

(* Note 6 *)

The initvarea procedure initializes the passed V/PLUS communication area to its initial values. This procedure should be called once at the start of the ENTRY program. The with statement allows us to refer to the individual fields of the varea. If we didn't have a with statement, we could do each assignment by fully qualifying the name. For example,

```
   varea.status := 0;
```

would be used instead of

```
   with varea do
      status := 0;
```

Improved Error Messages

Remember our first sample Pascal program? It opened a form placed the terminal in block mode. The initial version printed an error number only if it was unable to open the form or the terminal, and it didn't print any error message if it couldn't close the terminal or form. We would like to print an English message if this happens, and there is a V/PLUS procedure which will provide the message.

When developing a large program like ENTRY, we want to build up the Pascal environment to support our application. We declare standard utility procedures that enhance the Pascal environment.

```
   (*  The following procedure is called when an error occurs while
       opening or closing a V/PLUS file.  This procedure assumes that
```

```
     the user is NOT in block mode; therefore, the error message
     is printed on the standard output device.
*)

procedure writemessage(varea : vcomareatype);
var
   vmessbuf      : vbuftype;
   vmessbuflen   : vfixlen;
   vmesslen      : vvarlen;
   i             : 1 .. vbuflen;
begin
   vmessbuf := ' ';
   vmessbuflen := vbuflen;
   verrmsg(varea,vmessbuf,vmessbuflen,vmesslen);
   for i := 1 to vmesslen do
      WRITE(OUTPUT,vmessbuf[i]);
   WRITELN(OUTPUT);
end (*writemessage*);
```

We have used some types that haven't been introduced precisely. These are standard types required by many V/PLUS procedures, and we will declare them below.

Like the initvarea procedure, the writemessage procedure has one parameter: varea; but the writemessage parameter is missing the var part. This means that the call

```
     writemessage(vcomarea);
```

makes a copy of the vcomarea, and this copy is passed to writemessage. Any changes to the varea parameter will have NO effect on the passed vcomarea variable.

The following example is still the first program, but rewritten to be as clear and accurate as possible. We assume that the type declarations of the V/PLUS communication area are in a file called VCOMAREA.VIEW, and we will include this file in our source code.

```
program vplusbetter (INPUT,OUTPUT);
const
   vcomlen      = 60;
   vfnamelen    = 16;
   vbuflen      = 128;
type
   int = -32768 .. 32767;
   dbl = INTEGER;
$include vcomarea.view
   vbuftype = packed array[1..vbuflen] of CHAR;
   vfixlen      = vbuflen .. vbuflen;
   vvarlen      = 0 .. vbuflen;
var
   vcomarea : vcomareatype;
   vfilename: vfnametype;
procedure vopenformf(var varea : vcomareatype;
                     var vfname: vfnametype
```

```
                        );  INTRINSIC;

   procedure vopenterm(var varea : vcomareatype;
                       var vfname: vfnametype
                      );  INTRINSIC;
   procedure vcloseterm(var varea : vcomareatype);  INTRINSIC;
   procedure vcloseformf(var varea: vcomareatype);  INTRINSIC;
   procedure verrmsg(var varea : vcomareatype;
                     var buf    : vbuftype;
                     var buflen: vfixlen;
                     var actlen: vvarlen
                    ); INTRINSIC;
   $include initview.view
   $include wrtmess.view
   begin
      initvarea(vcomarea);
      vfilename := 'FORMFILE ';
      vopenformf(vcomarea,vfilename);
      if vcomarea.status <> 0 then
         writemessage(vcomarea)
      else
      begin
         vfilename := 'TERMINAL ';
         vopenterm(vcomarea,vfilename);
         if vcomarea.status <> 0 then
            writemessage(vcomarea)
         else
         begin
            vcloseterm(vcomarea);
            if vcomarea.status <> 0 then
               writemessage(vcomarea);
            vcloseformf(vcomarea);
            if vcomarea.status <> 0 then
               writemessage(vcomarea);
         end
      end
   end (* vplusbetter*).
```

This time, when declaring the V/PLUS intrinsics, we include the parameters. HP Pascal will check at compile time to make sure that the parameters passed to the V/PLUS procedures match exactly the ones declared above. This provides the necessary security within the Pascal program, and it will help prevent parameter-passing errors (where the wrong variable is passed to an INTRINSIC procedure).

We will assume that all of the procedure headings are declared correctly in a file: PROCS.VIEW. This file will be included in all subsequent examples, just like the standard VIEW types.

We make liberal use of subranges in declaring the vvarlen and the vfixlen types. By using a subrange (e.g., 0 .. vbuflen) instead of a generic type (such as int), we are telling both the compiler and the reader that any variables of type vvarlen can ONLY have values in the range of 0 and vbuflen. An assignment such as

```
   vmesslen := -1;
```

will be flagged as illegal at compile time, greatly reducing the amount of time needed to debug the program.

Structure of the ENTRY Program

Before starting the actual code for the mainline of the ENTRY program, we should specify exactly what we want the ENTRY program to do. First, the ENTRY program must open a user-specified form file. Next, the program must open a user-specified batch file. The terminal must be initialized, and the program must start collecting data from the forms in the formfile. Finally, the files must be closed and the terminal must be reset.

At this point in the development cycle, these are all the details we need to worry about in terms of processing. Users of the ENTRY program know that there is another collection mode: browsing, but we won't discuss the browse mode until we get to the discussion of the collect procedure.

The Mainline

We decompose the program steps described above into separate parts and we will use procedures to implement each piece. Because the opening of files is a necessary initialization of the program, we will leave all of that to one procedure. A possible mainline (the one we will use) could be:

```
   begin   (* entry *)
      if initentry then
      begin
         collect;
         completeentry;
         WRITELN(OUTPUT,'End of Entry/Robelle');
      end;
   end (* entry *).
```

The mainline is simple and easy to understand, but it contains all of the processing indicated by the program structure.

Initialization

We will continue our development of the ENTRY program by writing the initialization procedure. As we did with the mainline of the program, we will start with the mainline of the initialization procedure:

```
   begin  (* initentry *)
      initvarea(vcomarea);    (* Note 1 *)
      WRITELN(OUTPUT,'ENTRY/Robelle Consulting Ltd.(C) 1982');
      WRITELN(OUTPUT,version);

      initentry := FALSE;

      if openform then
```

```
        if openbatch then
           if openterm then
              initentry := TRUE
           else  (* openterm failed *)
           begin
              vclosebatch(vcomarea);
              if vcomarea.status <> 0 then
                 writemessage(vcomarea);
              vcloseformf(vcomarea);
              if vcomarea.status <> 0 then
                 writemessage(vcomarea);
           end
        else   (* openbatch failed *)
        begin
           vcloseformf(vcomarea);
           if vcomarea.status <> 0 then
              writemessage(vcomarea);
        end

     end (* initentry *);
```

The mainline of initentry reflects the structure that we desire. We must take care to close any open files when things go wrong. Remember that the mainline does not call the completeentry procedure (which would normally close the files) if the initentry procedure fails.

(* Note 1 *)

The vcomarea is a global variable. With the call to initvarea, we pass this global variable as a parameter. The initvarea procedure does not know the name of the global variable, so it uses the variable (varea) that was declared as a parameter (procedure initvarea (var varea:vcomareatype);).

We won't show the code for the entire initentry procedure, but we will show the code for the openform procedure. We will assume that there is another utility procedure which will read an input line from the terminal. This procedure always returns the number of characters read from the terminal.

```
   (* The following global constants and types are needed by the
      openform procedure.  We will show them here.
   *)

   const
      mpefnamemax = 36;
      inbufmax    = 80;
   type
      mpefnametype = packed array[1..mpefnamemax] of CHAR;
      inlentype    = 0 .. inbufmax;
   function readline : inlentype;  EXTERNAL;


   (* This procedure prompts the user for a forms file.  If
      the user does not enter a line, FALSE is returned;
      otherwise, TRUE is returned when a file has been
      successfully opened.
   *)

   function openform : BOOLEAN;
   var
      i        : 1 .. mpefnamemax;
      filename : mpefnametype;
      finished : BOOLEAN;
      inlen    : inlentype;
   begin
      repeat
      begin
         finished := FALSE;
         WRITE(OUTPUT,'Enter Forms File Name and press RETURN: ');
         PROMPT(OUTPUT);  (* Note 1 *)
         inlen := readline;
         if inlen = 0 then
         begin
            openform := FALSE;
            finished := TRUE;
         end
         else
            if inlen > mpefnamemax then
               WRITELN(OUTPUT,'Error:  Filename too long')
            else
            begin
               filename := ' ';   (* Note 2 *)
               for i := 1 to inlen do
                  filename[i] := inbuf[i];
               vcomarea.status := 0;
               vopenformf(vcomarea,filename);
               if vcomarea.status <> 0 then
                  writemessage(vcomarea)
               else
               begin
                  openform := TRUE;
                  finished := TRUE;
               end
            end
      end
      until finished;
   end (* openform *);
```

This procedure seems to cover most of the possibilties that could happen with user input. Does it handle all cases? I think it does; but there still could be cases that are not accounted for.

Notice how the procedure handles a null input line: it does not try to open the formfile; instead, it returns with openterm = FALSE. This provides some user friendliness to the ENTRY program. If the user forgets the formfile name, or runs the ENTRY program by mistake, he can easily "get out", or terminate, the program.

(* Note 1 *)

The PROMPT procedure is a special procedure that is available in HP Pascal and Pascal/Robelle. It prints out the contents of the current WRITE, with no carriage return or line feed. This one area of standard Pascal is very vague; individual implementors of Pascal have chosen different schemes for implementing carriage control. The PROMPT procedure is the convention used for carriage control on the HP 3000.

(* Note 2 *)

String literals (e.g., 'TERMINAL') can be assigned only to variables of one precise type: packed array [1..n] of CHAR. The ability to assign the filename without trailing blanks is non-standard. The packed attribute is important here; without it, you must assign each character of the string individually. Unlike SPL and FORTRAN, Pascal delimits string literals with single quotes, not double quotes. There is no standard way for taking substrings. This is why characters are assigned one at a time from inbuf to filename.

Completeentry

Now that the initialization routine is done, we should write its opposite: completeentry. This procedure is quite simple by comparison:

```
   (* Complete the ENTRY program by closing all open files.
   *)

   procedure completeentry;
   begin
      vcloseterm(vcomarea);
      if vcomarea.status <> 0 then
         writemessage(vcomarea);
      vclosebatch(vcomarea);
      if vcomarea.status <> 0 then
         writemessage(vcomarea);
      vcloseformf(vcomarea);
      if vcomarea.status <> 0 then
         writemessage(vcomarea);
   end (* completeentry *);
```

When these routines were originally written, they were tested before any other part of the ENTRY program was finished. The ENTRY program in its initial form just opened the files and closed them; but this early debugging found several coding errors.

Strings

Before starting the collect procedure, we need to build more of the enhanced Pascal environment to help our development of the ENTRY program. The first problem we solved was strings. Standard Pascal has no easy way of handling variable-length strings. HP Pascal includes some extensions in this area, but we are trying to stick as close to Standard Pascal as possible.

Using V/PLUS from Pascal

The ENTRY program has one type of buffer: VBUFTYPE. We would like to be able to insert short strings anywhere within a V/PLUS buffer, including string literals. In SPL, we would use the statement:

```
     MOVE VBUF(20) := "Record#";
```

In COBOL, we could use the statement:

```
     STRING SPACE-20, "Record#" DELIMITED BY SIZE,
              INTO VBUF.
```

We want to develop a Pascal procedure which could be called as

```
   insertstr('Record#',vbuf,20);
```

to do the same as the SPL and COBOL statements. We define a type, string, which represents the smaller string type (i.e., a string of up to twenty characters). Then we write the insertstr procedure as follows:

```
   const
      stringmax = 20;
   type
      string = packed array[1..stringmax] of CHAR;

   (*  Standard Pascal does not provide any way to insert
       substrings into bigger strings.  This procedure is used to
       insert a smaller string into a bigger buffer starting at a
       specific position.  If an error occurs, the buf is overridden
       with an error message.
   *)

   procedure insertstr(str:string; var buf:vbuftype; pos:int);
   var
      len : 0 .. stringmax;
      i   : 1 .. vbuflen;
   begin
      len := stringmax;
      while (len > 1) and (str[len] = ' ') do
         len := len - 1;
      if (pos > vbuflen) or (pos+len > vbuflen) then
         buf := 'Error:  Invalid position or length in insertstr'
      else
         for i := pos to pos+len do
            buf[i] := str[i-pos-i];
   end (*insertstr*);
```

We use defensive programming practices to ensure reasonable results from insertstr, even if unreasonable parameters are passed. We make sure that the position and length parameters do not result in a position that is out of the bounds of vbuftype. When this routine was first tested, the error message was printed several times.

More Utilities

Soon, we will get to the heart of the ENTRY program: the collect procedure. Before we do, we need some more utility procedures: first, a procedure like writemessage, which calls the verrmsg intrinsic to obtain an error message, then displays this message in the window area of the form. We will use this procedure whenever we have an error returned by a V/PLUS intrinsic.

```
(*  Call VERRMESG to obtain the current view error.  Print this
    error in the view window.
*)

procedure verror(varea:vcomareatype);
var
   vmessbuf    : vbuftype;
   vmessbuflen : vfixlen;
   vmesslen    : vvarlen;
begin
   vmessbuf    := ' ';
   vmessbuflen := vbuflen;
   verrmsg(varea,vmessbuf,vmessbuflen,vmesslen);
   varea.status := 0;
   vputwindow(varea,vmessbuf,vmesslen);
   error := TRUE;    (* Note 1 *)
end (*verror*);
```

This procedure is similar to our writemessage procedure. We use our V/PLUS type definitions to make life as easy as possible. Notice that we only needed to declare the necessary variables to communicate with the verrmsg and vputwindow intrinsics.

(* Note 1 *)

The variable error is defined globally. Whenever a user error (i.e., ENTRY error) or a V/PLUS-error is placed in the window, the error flag is set to TRUE. This prevents us from trying to write two error messages to the window, since there is only room for one.

Whenever we deal with a record from the batch file, we would like the user to be able to see which record is being added or modified. To do this, we declare a procedure that takes a passed record number and formats it into a position in the standard V/PLUS buffer. This procedure demonstrates how to call the dascii intrinsic.

```
(* The passed record number is formatted into the buffer at the
   indicated position.
*)
procedure formatrecnum(num:dbl; var buf:vbuftype; pos:int);
var
   tempbuf : string;
   len     : int;
function dascii(num:dbl;var buf:string) : int; INTRINSIC;
begin
   tempbuf := ' ';
```

```
      len := dascii(num,10,tempbuf);
      insertstr(tempbuf,buf,pos);
   end (*formatrecnum*);
```

This may seem like a small amount of code to place into one routine. We do it this way so that the DASCII function was declared only once. It hides some ugly details of the ENTRY program in one place, where it will be easy to modify if we develop a better procedure than dascii.

Now that we can format the record number in the window, we would like to format a standard status display for when there are no errors. The following procedure can be called from either the collect or browse procedures to format the status line.

```
   (* Format the status line in the window display.  The current
      mode and the current record form the status line.
   *)

   procedure formatstatus(mode:vmodetype; recnum : dbl);
   var
      vmessbuf    : vbuftype;
      vmessbuflen : vvarlen;
   begin
      vmessbuf := ' ';
      if mode = browsing then
         vmessbuf := 'ENTRY/Robelle                       Mode: Browse'
      else
         vmessbuf := 'ENTRY/Robelle                       Mode: Collect';
      insertstr(version,vmessbuf,15);
      insertstr('Record#',vmessbuf,60);
      formatrecnum(vcomarea,recnum+1,vmessbuf,68);
      vmessbuflen := 80;
      vputwindow(vcomarea,vmessbuf,vmessbuflen);
   end (*formatstatus*);
```

Some of our old friends are used again: insertstr and vputwindow. We check the vmodetype to determine which "phase" of the ENTRY program (i.e., browsing or collecting) to print in the status window.

We have only one more utility procedure to write! What do we do with logical errors in the ENTRY program itself? We need to provide for errors such as requesting the NEXT form when there is none. We define a global type:

```
   type
      errtype = (errnonext
                ,errnoprev
                ,errnodelete
                ,errnotrepeating
                ,errnobatchrecs
                ,errbadprev
                );
```

Next, we define a procedure that prints the appropriate error mesage in the window area. Remember our error variable? We check to make sure that an error has not already occurred, before showing our error message (we assume that V/PLUS error messages take precedence).

```
(* This procedure is called to print a customized entryprogram
   error message in the form window.
*)

procedure entryerror(errnum : errortype; recnum : dbl);
var
   vmessbuf    : vbuftype;
   vmesslen    : vvarlen;
begin
   if not error then
   begin
      case errornum of
      errnonext : vmessbuf := ' There are no more records';
      errnoprev : vmessbuf := ' There are no PREV records';
      errnodelete :
         vmessbuf := ' The DELETE key only works in Browse mode';
      errnotrepeating:
         vmessbuf := ' NEXT not defined for a non-repeating form';
      errnobatchrecs  :
         vmessbuf := ' The batch file is empty';
      errbadprev      :
         vmessbuf := ' PREV only defined for Browse mode';
      end;
      insertstr('Record#',vmessbuf,60);
      formatrecnum(recnum+1,vmessbuf,68);
      vmesslen := vbuflen;
      vputwindow(vcomarea,vmessbuf,vmesslen);
      error := TRUE;
   end;
end (*entryerror*);
```

Collect

Finally, we arrive at the central portion of the ENTRY program. We have a well-developed superstructure to make life easier. To review, we have:

verror	- used for V/PLUS errors.
entryerror	- used for logical ENTRY errors.
insertstr	- used for inserting strings.
formatstatus	- used to show the current status.

We will use all of these procedures in our development of the collect procedure. Before writing the code for collect, we will write down the pseudo-code that represents the mainline of the collect procedure:

```
initialize collect
repeat
   get the next form (vgetnextform)
   initialize the form (vinitform)
   format the status (formatstatus)
   show, read, and edit the form (showreadandedit)
   until exitprogram or end of the forms.
```

The exitprogram variable is a global variable, initially set to FALSE. Whenever the f8 key is pressed in either the collect or browse procedures, we will set the exitprogram variable to TRUE.

The vgetnextform and vinitform procedures are part of the V/PLUS intrinsics, and formatstatus is our utility procedure: showreadandedit is the only procedure we have to write. The initialization of collect is simple, so the resulting Pascal code, including error checking, is as follows:

```
   begin   (* collect *)
      vcomarea.mode       := collecting;
      vcomarea.deleteflag := FALSE;
      repeat
         vgetnextform(vcomarea);
         if vcomarea.status <> 0 then
            verror(vcomarea)
         else
         begin
            vinitform(vcomarea);
            if (vcomarea.status <> 0) or (vcomarea.numerrs <> 0) then
               verror(vcomarea)
            else
            begin
               if not error then
                  formatstatus(collecting,vcomarea.recnum);
               showreadandedit;
            end
         end
      until exitprogram or ((vcomarea.nfname='$END') and
                            (vcomarea.repeatopt=0));
      end (*collect*);
```

The showreadandedit procedure contains the rest of the complexities of the collect procedure. We begin with the pseudo-code:

```
    repeat
       initialize showreadandedit
       show the current form (vshowform)
       read the fields on the form (vreadfields)
       process the V/PLUS function key
    until not error or (vcomarea.lastkey = vprintkey)
```

The function keys in the collect and browse modes have slightly different meanings. We would like to associate symbolic names with the actual lastkey values returned by V/PLUS. We do this at the beginning of the collect procedure:

```
   const
      venterkey    = vf0;
      vheadkey     = vf1;
      vdeletekey   = vf2;
      vprintkey    = vf3;
      vrefreshkey  = vf4;
      vprevkey     = vf5;
      vnextformkey = vf6;
      vbrowsekey   = vf7;
      vexitkey     = vf8;
```

For each function key, we have a procedure to do the processing of the function. The resulting Pascal code is:

```
   begin  (* showreadandedit *)
      repeat
         vcomarea.status := 0;
         error           := FALSE;
         vshowform(vcomarea);
         if vcomarea.status <> 0 then
            verror(vcomarea)
         else
         begin
            vcomarea.showcontrol := 0;
            vreadfields(vcomarea);
            if vcomarea.status <> 0 then
               verror(vcomarea)
            else
               case vcomarea.lastkey of
               venterykey       : enterkey;
               vheadkey         : headkey;
               vdeletekey       : entryerror(errnodelete
                                            ,vomarea.recnum);
               vprintkey        : printkey;
               vrefreshkey      : vcomarea.nfname := '$REFRESH';
               vprevkey         : entryerror(errerrbadprev
                                            ,vcomarea.recnum);
               vnextformkey     : nextformkey;
               vbrowsekey       : browsekey;
               vexitkey         : exitprogram := TRUE;
               end
         end
      until not error or (vcomarea.lastkey = vprintkey);
   end (* showreadandedit *);
```

The processing for the enter key and the browse key requires some special handling. The rest are quite straightforward, so we present them here:

```
      (* Process the head key.  Reset the repeat and next form
         options, and set the next form name to '$HEAD'.
      *)

      procedure headkey;
      begin
         with vcomarea do
         begin
            repeatopt := vrepeatnorm;
            freezeopt := vclear;
            nfname    := '$HEAD';
         end;
      end (*headkey*);

      (* Process the print key.  Print the current form (including the
         data) out to the lineprinter.
      *)

      procedure printkey;
      var
         underline : int;
         cctl      : int;
      begin
         underline := 1;
         cctl      := 0;
         vprintform(vcomarea,underline,cctl);
         if vcomarea.status <> 0 then
            verror(vcomarea);
      end (*printkey*);

      (* Process the next key.  If the current form is non-repeating,
         an error message is printed.  Otherwise, the repeat option of
         the comarea is reset.
      *)

      procedure nextformkey;
      begin
         if vcomarea.repeatopt = vrepeatnorm then
            entryerror(errnotrepeating,vcomarea.recnum)
         else
            vcomarea.repeatopt := vrepeatnorm;
      end (*nextformkey*);
```

The processing for the enter key is more complicated. We have to edit the fields on the current form, finish up the form, write a batch record, and increment the record number. If an error occurs during editing of the fields on the form, we let V/PLUS highlight the field(s) and format an error message. The next call to vshowform in showreadandedit will cause the form to show the highlighted field(s) and the error message (only the first).

```
   (* Process the enter key.  Edit the data that was input, finish
      the form, and write out a batch record.  This procedure also
      causes the record number to be incremented.
   *)

   procedure enterkey;
   begin
      vfieldedits(vcomarea);
      if (vcomarea.status <> 0) or (vcomarea.numerrs <> 0) then
         verror(vcomarea)
      else
      begin
         vfinishform(vcomarea);
         if (vcomarea.status <> 0) or (vcomarea.numerrs <> 0) then
            verror(vcomarea)
```

```
        else
        begin
           vwritebatch(vcomarea);
           if vcomarea.status <> 0 then
              verror(vcomarea)
           else
              vcomarea.recnum := vcomarea.recnum + 1;
        end
     end
  end (* enterkey *);
```

The browse key causes the following to happen: the current status is saved, the mode is changed to browsing, the browse procedure is called, and, on return, the status is restored. The code, excluding the actual browse procedure, is:

```
  procedure browsekey;
  var
     saverecnum : dbl;
     savefname  : vfnametype;
   begin
     if vcomarea.numrecs = 0 then
         entryerror(errnobatchrecs,vcomarea.recnum)
     else
     begin
        saverecnum := vcomarea.recnum;
        savefname  := vcomarea.cfname;
        vcomarea.mode := browsing;
        vcomarea.repeatopt := vrepeatnorm;
        vcomarea.freezeopt := vclear;
        browse(vomcarea.recnum);     (* still to be written! *)
        vcomarea.recnum     := saverecnum;
        vcomarea.nfname     := savefname;
        vcomarea.repeatopt := vrepeatnorm;
        vcomarea.freezeopt := vclear;
        vcomarea.deleteflag:= false;
        vcomarea.mode       := collecting;
     end
  end (* browsekey *);
```

Overall Structure

The following chart describes the procedures that we have developed during this report. A procedure at a lower level is shown indented.

```
   Utilities
      initvarea, writemessage, verror, insertstr, formatrecnum,
      formatstatus, entryerror

   ENTRY
      initentry
         openform
            readline, vopenformf, writemessage
         openbatch
            readline, vopenbatch, writemessage
         openterm
            vopenterm, writemessage
      collect
         vgetnextform, vinitform, formatstatus,
         showreadandedit
            vshowform, vreadfields, headkey, printkey,
            enterkey,
               vfieldedits, vfinishform, vwritebatch
            browsekey
               browse
      completeentry
         vcloseterm, vclosebatch, vcloseformf, writemessage
```

Putting It Together

We now want to put all of our procedures together to form the ENTRY program. At the start, we used the VIEW group to gather together $INCLUDE files that would be helpful in calling V/PLUS and in coding the ENTRY program. We have changed the files somewhat, but here is a summary of the $INCLUDE files necessary for the ENTRY program:

const.view	- global V/PLUS constants.
type.view	- global V/PLUS types.
procs.view	- INTRINSIC declarations for all V/PLUS procedures.
util.view	- our utility procedures.
init.view	- ENTRY initialization procedure.
complete.view	- ENTRY completion procedure.
collect.view	- ENTRY collect procedure.
browse.view	- ENTRY browse procedure.

The complete ENTRY program can now be written:

```
   program entry(INPUT,OUTPUT);
   const
      stringmax = 20;
      inbufmax  = 80;
      version   = '(Version 1.0)';
   $include const.view
   type
      int = -32768 ..  32767;
      dbl = INTEGER;
      string = packed array[1..stringmax] of CHAR;
      inlentype = 0 ..  inbufmax;
      errtype = (errnonext
                ,errnoprev
                ,errnodelete
                ,errnotrepeating
                ,errnobatchrecs
                ,errbadprev
                );
   $include type.view
   var
      vcomarea : vcomareatype;

      error : BOOLEAN;
      exitprogram : BOOLEAN;

      inbuf : packed array[1..inbufmax] of CHAR;
   function readline : inlentype; EXTERNAL;
   $include util.view
   $include init.view
   $include complete.view
   $include browse.view
   $include collect.view
   begin   (* entry *)
      if initentry then
      begin
         collect;
         completeentry;
         WRITELN(OUTPUT,'End of ENTRY/Robelle');
      end
   end (* entry *).
```

Summary

We have not shown the browse procedure; instead we've left it as an exercise for the reader. The complete Pascal ENTRY program can be obtained from Robelle Consulting Ltd. by ordering Pascal/Robelle ($300 US for 1-5 cpus).

We have also left one question unanswered. Can we use V/PLUS in our applications? Our answer is 'YES'. Using the utilities and tools described in this report, it should be possible to access V/PLUS easily from a Pascal application. The const.view and type.view files should be useful to all Pascal programs that use V/PLUS. Several of our utility procedures (e.g., verror) should also be useful.

To write applications, we must also be able to call vgetbuffer, vgetfield, vputbuffer, and vputfield. These procedures are more difficult to use because the parameters to these procedures are variable (i.e., they are not the same from procedure-call to procedure-call).

Applications normally combine V/PLUS with IMAGE. Pascal/Robelle includes a companion technical report, "Using IMAGE From Pascal", in addition to the famous DBLOADNG program (which was originally written in FORTRAN,) completely rewritten in Pascal.

Integration of V/PLUS and IMAGE through Pascal requires more research; but that is the subject for a future report.

```
***************************************
*                                     *
*         SYSTEM DEVELOPMENT          *
*    ON HP3000 STANDARD PREMISES      *
*                                     *
***************************************
```

SYSTEM DEVELOPMENT ON HP3000 STANDARD PREMISES

by

Erik Buchwald Christensen, B.Sc.

Pi DATA A/S
Amaliegade 14A
DK-1256 København K

Telephones:	(01) 14 20 05
	(02) 36 80 25
Telex:	eidatadk

For the Proceedings of the HP3000 International Users Group Conference Copenhagen, October 1982.

ABSTRACT

System security is often considered to cover topics such as security against un-authorized data access, security against data loss and data inconsistency, secure ways of data recovery and so forth. Users as well as DP-managers, however, tend to overlook factors of risks involved in quite other areas, namely undesired dependency on certain members of EDP-staff having unpublic special knowledge as well as dependency on outside software vendors. So far systems can be secure in any traditional way of meaning, still hooking you up to certain persons and vendors. To eliminate those risks, we present a full, general concept for developing business applications using only HP3000 standard software and utilizing most HP3000 facilities, - in a way we assume they were designed for. Development is based on VPLUS screenhandling, IMAGE databases as well as KSAM and other MPE filetechniques and is independent of programming language. It provides logical using of accounts, groups and users, distinguishes between real-time, interactive system parts and background batch routines, eliminating program load time and thereby wait time between forms and dictating a friendly user interface. Furthermore, generic search on manual IMAGE master data sets, back and forth, is assigned. Guidelines for placing application subroutines in SL are discussed along with principles for fast, short and sufficient data locking not using MULTIPLE RINS. The concept has been used to develop a larger standard package of business applications written in COBOL II.

Go WITH the system, go FAR.

1. Introduction

When you enter the world of HP3000 you will soon find out that this machine is accompanied by a huge number of HP manuals describing all the basics with numerous texts on numerous pages. Sometimes, when you seek knowledge that combine two or more subjects, the answer however often seems to slip the lines.

When you ask somebody a "why" or a "how", nine out of ten will reply "why not try it out yourself?". But, goddammit, you haven't got all day to write small test programs with a project deadline ticking like a bomb under you.

So if your task is to develop a COMPLETE system from scratch and you want to do it "right" from the very beginning when the HP3000 still is new and un-covered, how do you get started?

This article tries to outline a full design concept covering all relevant aspects and problems during such a project. We will probably not present any revolutionary news to trained connoisseurs of the HP3000, but may succeed in giving a worthwhile overall view on the entire HP3000 environment.

Some two years ago we had finished developing a larger standard package of business applications to a specific mini on the turn-key market. The package was sold and even exported when we for various reasons decided to implement it on HP3000.

As you will learn below, we never develop machine independent so this was not to be yet another dull conversion project, but having the application precisely documented down to each single file, item, program, report and screen layout gave us the upportunity to perform an in-depth research in the new computer alone (the HP3000) not being disturbed by trivial problems as what was going to be the functional ends of the system.

This article is really our findings in that research.

2. Project highlights

Do not consider the following concepts as a design model for any specific type of application, but as a general framework for developing administrative EDP systems to HP3000. We will however, introduce you to some of the important assumptions and features of the project in order to give a better background for understanding many of the solutions we implemented on the HP3000.

The application is a standard accounting package known as PiCOM/3000. Briefly, it covers General Ledger, Accounts Receivables and Accounts Payables. Being a software vendor of packaged systems you have to consider the following items as extremely important:

2.1 Minimize the need for support

An accounting package will be used by ordinary clerks and bookkeepers without any special EDP knowledge. Therefore the package has to be user friendly to the extreme in order to avoid each customer buying the package hiring programmers as well as in order to minimize the time your own programmers have to spend fixing bugs on-site. Make them more productive and satisfied by developing new products, not debugging old ones.

The better the user interface the fewer the production stops will tend to be and the more likely those occurring will be fixed locally by the user himself. Note, that a good user interface is a product of many conflicting factors all concerning the common overall goal: presenting a system to the "unexperienced" user in such a way, that he can run it in an operator-less environment.

2.2 Let the programs handle available disc space

The support problem further stresses the need for built-in application control of available disc space. A software house will more often than you imagine be alarmed by a call from a user of the type "we just ran out of disc space in the middle of an update in the general ledger bookkeeping". Funny enough, those calls tends to reach you just before office closing time on Fridays, the user furthermore explaining, that his management expects updated income sheets Monday morning.

No matter how well configured the hardware seemed to be at installation time, no matter how much disc space was provided, sooner or later there will be no room for more data. Allways create discfiles fully allocated (IMAGE does that always), avoid intermediate work and sortfiles as long as you can. In this way the application's disc requirements will be easy to calculate and control, - both by the user and by the programs.

2.3 Easy backup and recovery

Of course backup and recovery procedures must be simple and easy to understand and perform. Therefore design your file structure in such a way that all non-IMAGE files can be derived and re-established from the databases thus allowing you to concentrate on logging only database transactions by means of the HP3000 IMAGE Transaction Logging and Recovery System. From a programmer's point of view this is quite simple and will offer the user an alternative recovery method to the well known "load the latest backup and start all over again" procedure.

2.4 Only use HP system software

Use only HP3000 standard software and as close to defaults as possible in order to be able to provide the user (and yourself) with normal system support agreements with HP. We see that as an important part in the overall security aspect. Furthermore as a valid sales argument, i.e. using HP software guarantees the user benefit from future extensions and upgrades developed by HP. Thus to our specific purpose we prefer plain machine vendor system software instead of "better" tools from third parties.

2.5 Sound data locking strategy

Your package (or tailored application) must rely upon a design concept where all the compromizes regarding optimizing versus security and user interface has been solved in advance. We all want to optimize our programs to do the fastest performing, but sometimes this rather technical point of view will conflict with other system requirements like the ones listed above and later on. Special considerations must be made in respect to the rather troublesome area of data locking. Everybody can make a single-user on-line system, many will have troubles when terminal number two is connected and two users are working simultaneously on the same dataset.

2.6 System executable on any configuration

Marketing a package like the PiCOM/3000 you attract the smaller companies in the low end of the HP3000 computer family. On the other hand large HP3000 installations may reserve a small corner somewhere in the hardware in which their bookkeeping department can run their Accounting System. Therefore another important design rule is to develop for smallest standard equipment in order to support any configuration with any terminal and any printer with no regard to special functions and options. To some extend, though, the design concept may provide utilization of more well equiped device models.

2.7 Use most HP3000 standard facilities

Finally, the whole project should be developed clearly on HP3000 premises. No flat industry standard COBOL with traditional filetypes and cursor addressing screen handling (accept/display). When somebody buys a HP3000 he is to some extent doing it for what is in the HP sales materials - so do not give him a bulk of programs to be :RUN in the PUB group in some account by all users, but be sure your system utilizes the HP3000 facilities in an expected way: structure the account in groups and users, use IMAGE and VPLUS and all the feautures that makes HP3000 a winner.

2.8 Application standards

To the above mentioned general topics of a business package we will briefly add some of the features that characterize our PiCOM system and which are of importance for the further discussion:

All programs are written in COBOL II.

The system is menu driven and all screen layouts are standardized. Function keys are also standardized whenever possible.

It is a true real time system without batch update. Multi company use is supported.

To provide export care has been taken to facilitate easy translation into any foreign user language.

Generic search is provided on all master datasets. We emphasize this, since it turned out to be impossible to obtain within IMAGE.

Each user may easily select and reselect output device for all his printed reports.

Extended security functions are obtained by assigning application capability lists together with rules for terminal access per operator. All operators have to be known by PiCOM/3000 and will be checked out at logon.

3. Account structure

Organize the account into the following groups:

1. PUB
2. DATA
3. <usergroup-1>
4. <usergroup-2>
 .
 .
n. <usergroup-n>

Each account equals an independent company. Thus you may run n companys on the same computer all using the same programs, but with independent data and independent lifecycles. Create <acct-1>, <acct-2>, ..., <acct-n>. Each account is organized into groups as above.

Assign users to each account in the following manner:

1. MGR (home = PUB)
2. One user BATCH with no home group
3. <user-1> with home = <usergroup-1>
4. <user-2> with home = <usergroup-2>
 .
 .
n. <user-n> with home = <usergroup-n>

Let it be the clients' responsibility to give actual names to the various accounts and groups and users. Assume as little as possible about naming so that your programs maintain the highest possible degree of flexibility regarding this aspect (avoid re-compiles).

Our concept only assumes for each account a group named "DATA" and a user "BATCH". "MGR" and "PUB" are of course automatically provided by MPE. We recommend our clients to name their users by initials and the corresponding homegroups as initials prefixed by the letter "G", e.g. user ABC, homegroup GABC. None of these names are compiled into the source code, but are placed in the PiCOM database by the client. One name, though, has to be compiled into at least one program as a fixed constant: the system reserved IMAGE password. There is no way to make that one password variable since initially the system has to open the database to get the user's IMAGE password, which then is used to re-open the base.

Placing the system reserved IMAGE password in a subprogram in SL, however, makes it relatively simple for the client to change it since a new value in the SCHEMA only needs to be followed by a subroutine compile and an ADD to SL, - not a complete re-compile of the entire system.

When creating the account only default HP3000 capabilities are used except for PH (Process Handling), if and only if the programs also has to reside within the account (which they do not have to). The reason why is explained later.

All the program files, including SL and the forms file, is placed where ever the customer prefers it, but the name on what is called the <systemacct> and <systemgroup> must be found in the database so that the system itself can determine where to run the programs. This is not for the user to be concerned about. In this way it is assured that all companys (accounts) is (can be) driven by the same application. Furthermore, new versions of the system can be tried out in some test account only by specifying another <systemgroup> and <systemacct> to the test account for new system version.

The <systemacct> must have PH.

The file contents in the various groups are as follows:

3.1 The PUB group

In PUB only a standard User Defined Command (UDC) is placed. It must be set for the account and will only contain one command, SETFIL, which establishes the nessesary file equations (will be described later).

3.2 The DATA group

This group contains all GLOBAL files in the account whether they are IMAGE databases or KSAM or other MPE files. Also database SCHEMAS reside in DATA. If the customer runs more accounts for multi company use, all schemas are identical except for the dataset CAPACITY entries.

3.3 The usergroups

Contains only LOCAL files per user. In our concept LOCAL files will never be organized as IMAGE databases.

Each user is assigned a special UDC called STARTUP which of course has OPTION LOGON so that when the user types :HELLO he is automatically forced into the PiCOM system. Also BYE is provided in STARTUP, i.e. making the operating system entirely invisible for the user never letting a colon occur except for hello's.

STARTUP is set on user in order to free MGR from it so that he eventually can enter the usergroups maintaining full MPE control.

4. File structure

Logically we distinguish between GLOBAL and LOCAL data. In order to get our point of view clear we will use examples from the PiCOM system. Since it deals with ordinary accounting problems everybody should be able to follow the terminology.

4.1 GLOBAL files

Include all databases, TAG-files and GLOBAL MPE-files.

4.1.1 The parameters and constants database

We use two databases for functional reasons, but also due to the limitation of max. 256 items per database. One is called SYSBAS and contains all global company constants, flags and tables (e.g. current year and period number, currency table, payment conditions and so on). The SYSBAS is primarily organized into a number of manual IMAGE masters (SYSM01, SYSM02, ... SYSMnn) where each table constitutes a master dataset.

A special master contains company data, such as company name (occurs on all screens and reports), report formats (page hight), the names on <systemgroup> and <systemacct>, the MPE- password for the user BATCH and on/off flags to determine whether the GLOBAL MPE-files are in use.

Also every authorized user (except for the user BATCH) must be established in SYSBAS with a lot of crucial data.

Those are:

Operator initials which is identical to the MPE username, full operator name, IMAGE password, number of logon terminal, name of MPE homegroup, name of outputdevice (LDEV), name of outputdevice for $STDLIST, an array containing application capabilities, rules for terminal access (all, specified numbers or temporarily inactivated) and finally on/off flags to determine whether his LOCAL MPE-files are in use.

4.1.2 The accounting database

The other database is called AFRBAS and contains all the traditional bookkeeping datasets (customer master, customer history, customer open items and so on). In general all master datasets are created as manual IMAGE masters whereas history and open item datasets are IMAGE details. Sorting is avoided except for open items which are sorted on duedates.

4.1.3 The TAG-files

A TAG-file is attached to each primary AFRBAS master (vendor, customer and general ledger) and is a KSAM file for sequential and backwards sequential retrieval of master entries on a variety of keys.

A TAG-file for the customer master (CUSMST), CUSTG1, will have the following COBOL definitions:

```
SELECT CUSTG1 ASSIGN TO "CUSTG1"
              ORGANIZATION IS INDEXED
              ACCESS MODE IS DYNAMIC
              RECORD KEY IS A1-KEY
              ALTERNATE RECORD KEY IS A1-KEY2
                 WITH DUPLICATES
              .
              .
              FILE STATUS IS WS-FSTATUS.

FD  CUSTG1
    LABEL RECORDS ARE STANDARD.
01  A1-REC.
    03  A1-KEY.
        05  A1-CUSTNO         PIC 9(14).
    03  A1-KEY2.
        05  A1-REVCUSTNO      PIC 9(14).
    .
    .
```

Hence, when you want to page forth and back in the customer file you just program the appropriate START's and READ NEXT's in the TAG-file using the A1-CUSTNO as key to the IMAGE master.

Note, that the reverse number is calculated as the complement value of the original number, e.i. SUBTRACT A1-CUSTNO FROM 99999999999999 GIVING A1-REVCUSTNO.

Also alfanumeric fields (e.g. customer shortname) may be reversed using a special technique when calculated (each 2-byte word has to be redefined as PIC S9(4) COMP and each then subtracted from -1).

Now reports will be produced in specified order letting the print program perform a sequential read of the TAG-file on the proper key. No temporary sortfile will be nessesary, i.e. no further uncontrollable disc space gets involved.

Note, that a TAG-file is very easy to recreate since a simple batchprogram will perform a serial read on the IMAGE master and write a new record in the TAG-file per master entry.

Use up to 16 alternate keys in a TAG-file and note the advantage of compounded keys, e.g. customergroup/customernbr, in order to produce more sophisticated sorted reports.

A TAG-file is maintained by the same program that maintains the masterfile to ensure the two files to be synchronously linked together.

4.1.4 The GLOBAL MPE-files

GLOBAL MPE-files is defined as periodically created central user files reflecting data already present in the database. After use they may be over-written. An example is the Interest Journal (INJOUR) which will be created by the user once a month during an automatic batchprocess. This process will perform the proper database update itself (DBUPDATE CUSMST, DBPUT CUSHST, DBPUT CUSOPN) and write records to INJOUR. Afterwards INJOUR is used to produce the Interest Notes and to printout the Interest Transaction Journal, but the two latter jobs with no database changes (always print without updates for easy ways of reprint due to for instance paper jam or spooler failures).

GLOBAL MPE-files can be either sequential files or KSAM files. The INJOUR file is a KSAM type, the primary key being customer number. The process creating the file will perform a serial read on CUSMST in the database, but the resultant INJOUR will automatically be in customer order ready for printout.

To summarize, the GLOBAL files are divided into IMAGE databases (SYSBAS, AFRBAS), KSAM TAG-files (e.g. CUSTG1) and GLOBAL MPE-files (e.g. INJOUR).

4.2 LOCAL files

These files are exclusively owned by each operator and thus located in his group. They are defined as the GLOBAL MPE- files except for the fact that they are created and maintained locally.

An example is the ordinary User Posting List which contains a record per posted voucher in chronological order. When an operator posts a customer invoice, PiCOM/3000 will perform a DBPUT CUSHST, DBPUT CUSOPN, DBUPDATE CUSMST and a simple write in the Posting List (PLIST). Whenever the operator wants to have his journal printed, the Posting List may afterwards be over-written.

4.3 WORK-files

A WORK-file is defined as a MPE-file being used temporarily by certain routines in one program. They will not be known by operators and not even by MGR, since they are created, used and purged invisibly.

PiCOM/3000 only uses that kind of files on two occasions and the programs themselves will calculate the needed filesizes on creation time. This is accomplished by reading the EOF's on the permanent files or sets that the WORK-files must hold (see chapter 7.9).

Thus a FILE-COMMAND followed by a COBOL OPEN OUTPUT (see chapter 5.5) will create the WORK-file in the actual user's group and ALWAYS with NUMEXTENTS,INITIALLOC = 32,32. If the disc space cannot be provided, operator is notified and the task will not begin at all.

5. Program structure

All program files, the SL and the formsfile are located in <systemgroup>.<systemacct> selected by the client. Both names are present in SYSBAS so that PiCOM/3000 can run the proper programs when an operator logs on.

You may consider your programs divided into five different types:

5.1 Subroutines

Subroutines are owned in common by the entire application and are placed in SL (Segmented Library). A programmer recognizes them by $CONTROL DYNAMIC as compiler options and they are compiled to their own unique USL-file. After compilation this USL is added to SL by SEGMENTER.

5.2 Stand alone MAIN programs

These programs constitute all the background jobs, primarily the printprograms. They will have $CONTROL USLINIT as options and are also compiled to unique USL's. After compilation, the specific USL is PREP'ed into a new programfile, the old one being purged.

5.3 Interactive SUBPROGRAMS

The interactive on-line parts of PiCOM/3000 consist of many subprograms linked together around a small basic root program (which is a MAIN program). Each on-line application is therefore just one large programfile to the HP3000 and may be :RUN in ordinary way.

A subprogram will have the same compiler options as a subroutine. When compiled, the USL-file must be the USL common to the entire on-line application. Afterwards that large USL is PREP'ed.

Note, that the small MAIN root program does not compile with USLINIT for obvious reasons.

Each on-line application is really a son process under one common father.

5.4 The FATHER process

One father process will function as an umbrella (or MASTER MENU) to the various on-line applications. The father is just an ordinary MAIN program and compiled as such except that "PH" must be specified when PREP'ed.

There is at least one good reason for subdividing the system into a number of son processes = on-line applications. That reason lies in the limitations of a programfile that HP3000 imposes upon you: 63 is the maximum number of segments in a programfile. COBOL will by default assign 1 segment per unsegmented COBOL subprogram, but you may rearrange more subprograms into 1 segment using the SEGMENTER. Even so you might as well plan from the very start to handle more than one on-line application. To do so, the father process is needed to present the overall master menu and to CREATE and ACTIVATE whatever son the operator whishes to perform. When the operator exits from that on-line application, control is automatically returned to the father, i.e. the operator gets the master menu again.

The main disadvantage by having more than one on-line application is that you will experience significant wait time on the terminals when shifting from one application to another. Once you are inside an application you will have reasonable speed between forms. Furthermore you cannot branch directly from one form in one application to another form in another application. This leads to great care when planning how to divide a logically united system into separate on-line applications.

In PiCOM/3000 we have two applications at the moment: one to handle all the maintenance on the SYSBAS database and one to handle General Ledger, Customers and Vendors. As a matter of fact we could actually gather all our subprograms in just one on-line application, but it seems more realistic to start with a divide in order to cope with future extensions in an easy way.

Note, that a division into more sons is independent of the underlying data structure and that the formsfiles do not have to follow the applications, PiCOM/3000 only has one common formsfile. Actually, you could also have one formsfile for the father and one for each of the sons.

5.5 Utilityprograms

One of the benefits of IMAGE is that it frees you from developing all the traditional support-tools to cope with installations, conversions, creations, disasters and the like.

PiCOM/3000 uses only two utilityprograms to be frequently used by the end-user:

£1 An ordinary MAIN program driven by accept/display to alter filesize on all GLOBAL non-IMAGE files.

MGR enters the DATA group and by running AFRGLOB he will be prompted for each GLOBAL file whether to create it and if so, the capacity and number of extents. All extents will always be allocated, default being 8 if omitted.

AFRGLOB will by means of the COMMAND intrinsic issue the relevant FILE command and a PURGE of the old file followed by a COBOL OPEN OUTPUT and CLOSE. Finally, the SAVE- and RELEASE-commands will make the file permanent and available to all users.

Note, that MGR may want to perform this when extending filesize to match larger IMAGE sets, but also as a means to temporarily scratch files by setting capacity = 1

£2 Another, but similiar program (AFRLOC) to alter LOCAL files, being run in the users' groups.

The two programs may be used whenever a file is in a state where it could be overwritten by the users.

6. Running the system

What will happen when a user logs on?

The only way he can do so is to type :HELLO <user>.<acct>, user being his initials, e.g. ABC, and acct being the company name on abbreviated form, e.g. Acme Company Ltd. just ACME. This person will then logon with :HELLO ABC.ACME.

The previous mentioned STARTUP UDC (set per user) will now be in effect, as it contains:

```
STARTUP
OPTION LOGON
OPTION NOBREAK
SETFIL
RUN AFR001.<systemgroup>.<systemacct>;LIB=G
RESET @
BYE
```

If the client changes the location of the programfiles, SL and the formsfile, he must accordingly alter the RUN-statement in the STARTUP UDC. The LIB parameter is set to "G" in order to search the SL in the same group where the programs are run.

The SETFIL UDC is set per account and contains all necessary file equations, e.g.:

```
SETFIL
OPTION NOLIST
FILE SYSBAS=SYSBAS.DATA
FILE AFRBAS=AFRBAS.DATA
FILE CUSTG1=CUSTG1.DATA;SHR;LOCK
FILE INJOUR=INJOUR.DATA;LOCK
```

Only the GLOBAL files (all residing in DATA) need to be equated. Note that SETFIL is independent of account name.

Since MGR has not the STARTUP UDC set he will be able to enter a user's group by for instance :HELLO MGR.ACME,GABC and perform various MPE tasks. If he is also established as an ordinary PiCOM operator he will also be able to run the system.

6.1 Logon

As you may have guessed, AFR001 is the father process in the system (the master menu).

In short this program will perform all logon routines, such as setting up the file equation for the formsfile (by getting the names on <systemgroup> and <systemacct> from SYSBAS), open the formsfile and open the user's terminal as a file. By the "WHO" intrinsic the MPE username will be obtained (e.g. "ABC") and then used as a key item to search this operator as a valid PiCOM user in SYSBAS.

For reasons to be discussed later an operator can only work on one terminal at a time in PiCOM/3000. If his data in SYSBAS already contains a value on "current active terminal", the logon routine will not allow further process.

Finally, if the operator passes all logon tests, the master menu is presented and the operator may hit his choice. This will cause the father process to CREATE and ACTIVATE the selected on-line application. Then the father closes his files which later on has to be opened again by the son and thus explaining the wait time at this point in the system.

6.2 The root program in the SONS

The on-line applications (sons) consist of many subprograms built around a small main program functioning as the root. The root program in each application represents the entry point to the entire programfile and will in many senses have functions similiar to the father program.

It will first open the SYSBAS database using the reserved IMAGE password (the one located in SL) and get the operator's data by means of the "WHO" intrinsic (as done by the father program). It will then reopen SYSBAS and open AFRBAS using this operator's IMAGE password.

Then the formsfile and the terminal will be opened by calls to the VPLUS procedures "VOPENFORMF" and "VOPENTERM" using COMAREA and the respective filename. Note, that the father has already established these file equations for the session, which means that the son only has to know the formal file designators ("AFRSC" for the formsfile, "SKARM" for the terminal).

It has been mentioned, that SYSBAS contains what is called GLOBAL FLAGS. These flags determine how PiCOM/3000 is configured for this account (this company) and thus serves as options for the clients. Some may only need the system defaults and run PiCOM in its simplest form, others may want more advanced features, for instance whether open items should be used at all. Some global flags can not be altered after configuration, others may be changed freely. All global flags reside in the SYSM01 dataset of SYSBAS and virtually every PiCOM/3000 program will have to read the value of one or more of those flags. In order to avoid excessive DBGET's on SYSM01 the root program initializes all flags on the user's data stack making them available to all subprograms through the LINKAGE SECTION.

Finally the MAIN MENU for the on-line application is presented and all global formsfields (COMAREA) and the global IMAGE area (DB-AREAS) defined in WORKING-STORAGE (the root program cannot have a linkage section).

From that point on all that the root program does is to permit the operator to walk back and forth in the menu hierarchy until reaching a point where not a new menu, but an application form with an associated subprogram is requested.

You may therefore consider the root program with the VPLUS menus to be the driver of the on-line application in the sense that the name of the subprogram being called entirely depends of the NEXT FORM NAME from VPLUS (communicated through COMAREA). The menu forms themselves does not have any associated subprograms since they only serve as paths to the operator until he gets to a point where a programmatic function has to be performed. Walking through the menus is therefore very fast since it only involves shifting of forms by the root program without any subprogram calls.

6.3 The SUBPROGRAMS

When leaving the menu level (the root program) VPLUS already knows the name of NEXT FORM and a called subprogram will therefore always perform a housekeeping routine with calls to VGETNEXTFORM, VINITFORM and VSHOWFORM. From here on the subprogram will loop VREADFIELDS, VGETBUFFER and <perform program routine>. When the operator terminates his task, he will return to his offspring, i.e. the nearest lowest menu. The name of that menu will be set as NEXT FORM NAME and an EXIT PROGRAM statement in the subprogram will return proper control to the root program, - and the menu structure is available again for the operator.

So once you are getting familiar with it, it is really not so bad to have the programs separated from the forms! As we will discuss later your system will improve, though, if you program your own interface to VPLUS, i.e. make common routines for accepting and displaying data, getting a new form and so on. Concentrate all the various VPLUS calls in these common routines, because VPLUS needs a lot of calls with a lot of testing afterwards.

Observe, that the separation between forms and programs often has the effect to the operator that the new form will appear on the screen seconds prior to the associated program being ready for datainput or output.

6.4 Having background jobs run

Quite often the operator wants to print a certain report as a background job, i.e. not preventing him from further interactive dataprocessing on the on-line applications while the report is being prepared and actually printed.

How do you handle the communication between the on-line application and the background batchjobs?

Every background program will in principles need two sources before it can be run:

£1 A STREAM file (called a "T-file" by PiCOM/3000) that among other things will issue the nessesary file equations and the ultimate RUN command.

£2 A program- or job-control file (called "WRK-file" by PiCOM/3000) which is really an old fashioned punched card since it only contains one record (more than 80 characters, though) telling the batch program exactly what to do; in which order, on which data, with which subtotals and so on.

Primarily the WRK-file will be a copy of the screen buffer that holds all the operator's instructions to the batchprogram. The GLOBAL LINKAGE FLAGS are however always added in order to free the batchprogram from those readings in SYSBAS.

Any subprogram in an on-line application therefore calls a specific subroutine in the PiCOM/3000 SL called "TFCREAT" using T-file data and WRK-file data.

The TFCREAT subroutine then gathers the remaining information to both files and first writes the WRK-file (only one job control record) on disc and then builds up and writes all the records in the STREAM file (the T-file). At last the subroutine will stream the T-file (that will !RUN the desired background program) and then PURGE the T-file afterwards.

First thing a background program will do, is to read the WRK-file with all its program instructions and issue a COBOL display of the entire record content. In this way the HP3000 JOB-list on $STDLIST always shows the commands from the T-file and the actual WRK-file data. Note, that both the WRK-file and the T-file is placed in the actual user's group, but that the job is performed by the user BATCH.

Since our design allows $STDLIST to be varied per user the JOB-list can either be directed to local printers, system printer, disc or whatever. Also LDEV is specified per user and that is really the reason why there is no printfile equations in the common SETFIL UDC since this equation is first to be solved at the time a print job is requested. The T-file contains that equation given by TFCREAT. Also the actual paper format on a certain report is variable, the information stored in the company dataset in SYSBAS and passed to the WRK-file by TFCREAT and thus given to the printprogram (which will further transfer the values to the printer interface in PiCOM/3000).

In summary, this whole printer handling concept gives the user maximum flexibility to have his printer environment configured after his own needs and not dictated by the application. In a distributed organization you may for instance choose HEAD OFF and set $STDLIST in PiCOM/3000 to "LP,2" per user. This will print only application output on all printers and thus give minimum paper waste. Note, that "LP,2" will direct JOB-lists to disc due to low value compared to OUTFENCE.

One thing to be aware of in the T- and WRK-file technique is that every user is able rapidly to request the same background program many times within a second by hitting the correct function key on his terminal over and over. This might lead to a lot of duplicate discfile names on the T- and WRK-files unless of course machine time is part of those filenames. This problem is also solved by TFCREAT that assigns the actual filenames and then equates them properly. For instance, each background program will open and read a file named WRKxxx (xxx being the number on the calling subprogram in the on-line application). The WRK-files are PURGED by a command in the T-file so that neither the T-file nor the WRK-file will be floating around on the disc after use.

An example of the TFCREAT process is shown in chapter 7.8.

6.5 Batch processing can also be real time

We deliberately use the term "background program" instead of "batch job" since you also may need some on-line batchprograms in your application, i.e. a batch process can be either background (a HP3000 JOB) or part of the on-line application (a HP3000 SESSION). When for instance PiCOM/3000 performs its period-end routines this is done on-line and with exclusive database access so that no other user can run PiCOM/3000 concurrently in that account (all terminals must log off).

This routine is then an example of an on-line batch process, the period-end request form being permanently showed on the terminal until the process is over and the resulting output produced on the printer. This type of program has nothing to do with the TFCREAT subroutine, though the technique for printfile-equation and handling may be the same.

6.6 The length of the job queue

How to cope with the time delay between a request for a given background program and the same program actually being run?

Depending on the entire system load, considerable time may elapse before the user has the wanted report in front of him on the desk. This also means to a real time package like the PiCOM/3000 that the data on the report will not reflect the status on print request time, but some later status including data changes between request and actual run.

Our approach to this problem is in fact very simple: the background printprograms will report what is in the database at run time.

Furthermore, we do not allow a background job to wait forever to get read access to data, that is if a database or some datasets are impossible to open for the printprogram in simple concurrent read mode, PiCOM/3000 simply lets the program terminate itself with a remark on the JOB-list. In an accounting package like PiCOM/3000 this is rarely to happen, but may occur, though. All GLOBAL and LOCAL MPE-files, for instance, are carrying all needed master set items themselves (such as name, address, balances) so that full printouts are produced even without database readings.

Note, that the user BATCH always gets the reserved IMAGE-password by convenience, - the operator's capability list is determining how much he in fact can ask BATCH to do, i.e. which types of reports he can run.

We do not find these rules hostile to the user since they are easily argumented and understood. Consider the design implications if any report should be absolutely synchronous with operator request: all batchperforming would have to be done within the on-line application, hooking up the screen terminal while printing occurs (as described earlier as on-line batchprocesses). Other computers actually work this way, but it would be bad usage of the HP3000 JOB technique to do so. Not to mention all the further problems with the max. 63 number-of-segments bottle-neck in the on-line applications caused by moving all the stand alone main programs up as subprograms. In our General Ledger/Accounts Payables/Accounts Receivables application more than half the programs are main programs.

But IF some printprogram by nature always MUST complete when requested, take it up as a subprogram. We recommend so for the printout of the users' Posting Lists (PLIST).

7. The application SUBROUTINES (SL)

Now we have carefully outlined the program structure concerning FATHER PROCESS (master menu), SONS = on-line applications = many SUBPROGRAMS linked to a root program and stand alone MAIN PROGRAMS (background jobs).

Remaining is the common SUBROUTINES placed in SL (Segmented Library).

By nature many of the subroutines will act as interfaces to various environments in the HP3000. Below we shall shortly discuss how PiCOM/3000 benefits from its SL.

7.1 VPLUS interface

Program your own interface to VPLUS in SL. PiCOM/3000 manages well with only three entry points for GET, READ and SHOW forms.

It is especially important that only the common subroutine actually detects what function key is depressed and decides what action to take, i.e. what further VPLUS calls to perform and/or which fields to alter in COMAREA.

Use a mode parameter with the call to your VPLUS subroutine in order to assign different functions to the f-keys.

7.2 Breaking the blockmode

Why would you want to break the blockmode in a VPLUS-driven application? That is, sometimes turn the terminal into character-mode.

One example: every time the operator requests a background job MPE will signal the job-number on his terminal using the first available input field or fields on the current form.

If there is no input fields at all on the form, the cursor will flicker in line 1 position 1 while beeping. If there is only one input field of one character, you will get the entire MPE message letter by letter.

Certainly it would be nice to control for example line 24 to such purposes.

Setting the terminal for character-mode (accept/display) is only a question of displaying the correct escape sequences to turn blockmode off. Other escape sequences have to be transmitted in order to set the blockmode back on.

Since coding of escape sequences is a bit awkward, a simple subroutine with two entry points will solve all problems:

```
ENTRY "F4ON".
    DISPLAY "<esc>W".
    EXIT PROGRAM.

ENTRY "F4OFF".
    DISPLAY "<esc>X".
    EXIT PROGRAM.
```

If you want to display the text MYMESSAGE on line 24 this is all you have to do in a subprogram:

```
    CALL "F4OFF".
    DISPLAY LINE24 "MYMESSAGE".
    CALL "F4ON".
```

LINE24 is defined in WORKING-STORAGE as PIC X(10) VALUE "<esc>&a23r00c".

7.3 Posting interface handling DATA LOCK and IMAGE LOG

Also concentrate all database updates due to regular posting in common SL subroutines. This is especially feasible in an accounting system since posting to for instance Accounts Receivables always performs exactly the same data processing routine no matter in what program or in what context the posting was done.

The far most valuable benefit from this approach is a centralized way of handling and controlling the data locking problem.

To illustrate this, let us take a look at our subroutine for Accounts Receivables posting (entry point CUSTUPD).

The call to CUSTUPD is accompanied by a MODE parameter and of course all the data the operator typed in on the bookkeeping form (such as customer number, date, voucher number, amount and so on). At this point the operator has already visually been informed that the customer actually exists by having his name and address displayed on the form, but this was only due to a simple read in the database.

When CUSTUPD is called the customer will be read again, this time covered by an unconditionally IMAGE DBLOCK MODE5 on data entry level rather than dataset (only causing wait time if that specific customer is already "busy" at another operator).

The MODE parameter specifies to CUSTUPD what type of transaction it has to perform, i.e. must a new open item be created or shall an existing open item adjust? To complete the transaction CUSTUPD has to DBUPDATE CUSMST, DBPUT CUSHST, DBPUT or DBUPDATE CUSOPN.

Since multiple datasets are affected, the following rules are implied:

£1 The lockdescriptor used with the call to DBLOCK must cover all relevant datasets in order not to use MULTIPLE RINS, - we consider it part of a user friendly concept that deadlocks ex definitione is excluded from the application.

£2 Since all changes in the database are controlled by the subroutine, let the subroutine set the DBBEGIN and DBEND marks around the complete, logical transaction in order to use the IMAGE Transaction Logging And Recovery System in the right way.

We have earlier mentioned the need for certain on-line batchprograms, for instance period-end routines. These routines consider all the postings as one whole logical transaction, but since each posting is still executed by CUSTUPD, CUSTUPD must not do the DBBEGINS/ENDS in that context, - this must rather be done by the subprogram at BOJ and EOJ. In PiCOM/3000 a minus ("-") in the signed MODE parameter tells CUSTUPD not to DBBEGIN/END.

£3 Since the subroutine knows which datasets are to be affected by DBPUTs, let the subroutine verify in advance that ALL datasets still hold free space for more entries.

This is easily done on IMAGE by calls to DBINFO and on MPE-files by the FGETINFO intrinsic. If not, that is if just one dataset or file is already full, the subroutine will return to the calling program with an error message notifying the operator that further posting is impossible until files are expanded.

Note, that on-line batch-processes cannot rely on the posting subroutines (such as CUSTUPD) checking free disc space in the affected datasets each time they are called since it is the successful result of ALL the postings that matters. Therefore these programs will have to consider the total need for free disc space, - and to check the availability of that before they will execute.

If a batch-process does not involve postings we always drop it down to a background job (a stand-alone MAIN PROGRAM). This is the case for ordinary reports and they will not involve disc space considerations since no sorting is done, when driven by TAG-files that already specifies the sorted order.

Finally it should be mentioned, that associated writings in GLOBAL or LOCAL non-IMAGE files must be accomplished by the calling subprogram which opens those files, NOT the subroutine CUSTUPD.

Furthermore, these writings should be done PRIOR to calls to CUSTUPD. IF something goes wrong, inspecting for instance last record in PLIST will determine what database transaction was the last to complete.

7.4 Disc I-O interface

An interface betwen the programs and the disc I-O will give benefits to any application:

- Safer and faster coding, the programmer being able to concentrate on logics, not how data is transferred.

- More releiable and secure system when in production due to centralized filecontrol.

- Better operator-interface by means of standardized error messages.

- More machine-independent system (what type of operating system will HP use on future 32-bit machines?).

On PiCOM/3000 the disc I-O interface consists of:

£1 One common I-O subroutine in SL that will access any IMAGE set the proper way (DBPUT, GET, UPDATE, OPEN, CLOSE and so forth) and return what has to be returned to caller using DB-AREAS.

£2 A file handler in COBOL code placed in a copy library. Each IMAGE set has its own handler to be copied into PROCEDURE DIVISION.

The handlers even accomplish all the links between an IMAGE master and its TAG-file.

To perform an I-O, all the programmer has to do is to load the key-field in DB-AREAS and then for instance PERFORM GET-NEXT-CUSMST. The handler will then deal with the proper KSAM TAG-file readings and call the I-O subroutine.

The number of code-lines in a handler is extremely low and will generally decrease sizes on source files.

Any HP3000 error message from IMAGE, MPE or KSAM is channeled through the I-O interface allowing PiCOM/3000 to present any such error in a consistent format for the user.

Furthermore, the I-O subroutine easily detects whether caller is a SESSION or a JOB and then routes the error message to either line 24 on the terminal or to $STDLIST. COBOL will interpreet the MODE-parameter returned by a call to the "WHO" intrinsic by:

```
01  WHAT-ARE-YOU             PIC S9(4)   COMP.
    88  W-SESSION            VALUE 0 THRU 7.
    88  W-JOB                VALUE 8 THRU 15.
```

Key, file- or setname, error code and error text are always provided and if a session, a dummy accept is forced to halt the process until the message is discovered.

7.5 Printer interface

Use SL to hold a central interface between all application programs and the printer(s). Do not let any program contain the traditional OPEN, WRITE and CLOSE statements, but be sure to have one SL subroutine with appropriate entry-points for those purposes.

In PiCOM/3000 these are POPEN, PWRITE, PEJECT, PCLOSE sharing a global area (PF-AREA) containing all necessary information to let the subroutine handle the further calls to the HP3000 intrinsics "FOPEN", "FWRITE" and "FCLOSE". Furthermore, our subroutine keeps track of pagenumber and linecount, freeing the printprogram from dealing with those trivials.

Since PF-AREA among other fields contains the printfilename, name on logical outputdevice, logical and physical pagehight, you may recall that the TFCREAT subroutine is providing the correct values of those fields as part of the WRK-file. When a printprogram reads the WRK-file, it will load PF-AREA with the given information in its housekeeping.

To write a line on the printer then only requires setting of the lineadvance factor followed by a call to "PWRITE" using PF-AREA and <print-image>.

A central subroutine for printer output will furthermore allow your system to adapt easily to future (and present) advanced printer options, e.g. those demanding special escape sequences being transmitted to the printer in order to invoke certain functions (compressed print, for one thing).

7.6 What about hard copy printers?

Will the PiCOM/3000 print-device approach also manage hard copy printers plugged in at the back of the screen terminal to produce application print-out?

In no way at all, but we believe this is to blame on VPLUS, rather than on the approach itself.

Since VPLUS applications open the terminal as a file you would have to close that file in order to get application output through the terminal and further out to the hard copy printer.

And by closing the terminal-file the entire on-line application is taken down in some unpredictable state from which it cannot be brought back, not resuming nicely from the breaking point, anyway.

Note, that even a hard copy printer connected with an HP264x type terminal will not produce copys of VPLUS forms using the special yellow and green function keys in copy mode.

If you are planning to use the unsupported utilityprogram "PSCREEN" to print out VPLUS forms you will be disappointed by a not very nice result.

Happily the built-in thermo printers on top of some HP262x terminals provide nice copys, but since any application should be devised with a formal procedure to print a copy of any form to any printer this apparently has to be solved using the VPLUS procedure "VPRINTFORM".

Apparently, because we have not actually tried it out yet, but it looks like this procedure will allow us to withhold our principle of independent printdevice per operator by means of file equations.

7.7 Background job interface

Determine how to link the background programs to the on-line applications and let that communication be handled by one powerful SL subroutine like "TFCREAT", mentioned previously.

One thing to add, is that TFCREAT signals the MPE job-number (£Jxxx) on the terminal. Unfortunately this is the LAST communication between the operator and the job, PiCOM/3000 therefore still lacks methods which will enable the application to monitor its own jobs.

We can illustrate the TFCREAT process by assuming the following:

At time 10:28:04:80 TFCREAT is called by the user ABC in the Acme Co. Ltd. from the printselect subprogram AFR071 to stream a printout of Gen. Ledgr. statements by the main program AFR072. ABC has his LDEV set to "LP" and $STDLIST to "LP,2". Acme gave the user BATCH the MPE-password "NOHOME". They have all their programs in the PUB group within the ACME account.

TFCREAT will perform by gathering all the above mentioned data from SYSBAS:

```
FILE WRKFILE=W0280480
OPEN OUTPUT WRKFILE
WRITE WK-RECORD
CLOSE WRKFILE
SAVE W0280480
RELEASE W0280480

FILE TFILE=T0280480
OPEN OUTPUT TFILE
WRITE !JOB J0280480,BATCH/NOHOME.ACME,GABC;OUTCLASS=LP,2
      !TELLOP **** AFR072: GEN. LEDGR. STATEMENTS ****
      !SETFIL
      !FILE W0280480=WRK071.GABC
      !FILE C0090011;DEV=LP;CCTL
      !FILE PICOM=*C0090011
      !RUN AFR072.PUB.ACME;LIB=G
      !PURGE W0280480
      !EOJ
CLOSE TFILE
STREAM T0280480

DISPLAY JOB-number on line 24

PURGE T0280480
```

By cutting off the left-most time digit the risk for duplicate filenames are as close to zero as possible.

The printprogram will open input file WRK071 and read the one record with all the operator instructions.

The printer interface always opens a printfile called "PICOM", but this would give very bad information on :SHOWOUT, since all waiting reports would have the same filename.

Therefore this naming is applied:

Any report belongs to a certain PiCOM/3000 printclass, e.g. "C" for width = 80 columns.

Any report has a unique PiCOM/3000 reportnumber, e.g. "009".

Any company (HP3000 account) has a unique number of its own, e.g. "0011", determined by the client.

By this method, :SHOWOUT will always clearly identify all spoolfiles.

7.8 Information on available disc space

We have earlier stressed the importance of proper control mechanisms within the application to investigate free disc space in various file types prior to any disc consuming transaction (whether a transaction equals one interactive single posting or whether a whole on-line batchprocess forms one logical transaction).

To that purpose one subroutine in SL will streamline the programming since this one program alone deals with "DBINFO" and all the more specialized intrinsics like "FOPEN", "FCHECK", "FGETKEYINFO", "FGETINFO", "FCLOSE" and so on.

A global communication area (FI-ARRAY) is used by PiCOM/3000 in calls to that subroutine in order to facilitate easy ways of calling the routine and have the results transmitted back to caller.

Filetype (IMAGE, KSAM, MPE), file- or datasetname and databasename must be specified and limit, eof and number of idle records or entries will be returned.

In PiCOM/3000 the "FILEINFO" subroutine also covers a comparison between the number of entries in an IMAGE master dataset and the number of active records in the associated KSAM TAG-file since many crucial programs are driven by TAG-files and since this comparison is the only feasible way to determine that the database and the TAG-files are in fact synchronous.

To minimize that risk however, the period-end routines should always recreate TAG-files from scratch.

7.9 Date routines

The PiCOM/3000 SL holds some very important date subroutines because dates always represent important data to any accounting system. Thus we have small routines (or entry points to a common program) for converting filedate format (always YYMMDD) to userdate format (individual per company by GLOBAL FLAG setting) and vice versa, for computing number of days between two dates and for adding "n" days to a certain date (computing duedate).

8. Using the RL

What to put in RL (Relocatable Library) then?

To be frank, PiCOM/3000 does not use RL at all. In that respect we entirely agree with bibliography (5), p. 51.

9. Backup and recovery

You might argue, that if the application is run with LOGGING ENABLED, what is all the fuzz about datasets running full in the middle of a transaction? All you have to do is simply to recover from the logfiles up to the latest complete transaction.

Yes and no.

For one thing, not all users will actually want to enable logging due to the management of logfiles it involves, - and the costs. Logfiles may even run full, too!

Anyway, it is still easier to expand files only instead of performing BOTH recovery AND expanding. Never use transaction logging as an excuse for built-in weaknesses in your application, but as a means to recover from outside problems, e.g. hardware or file restraints. And be pretty sure your design fully supports the recovery situation.

In case of a file structure like the one in PiCOM/3000 it is not sufficient barely to recover the global IMAGE databases, also all KSAM and other MPE-files, global or local, must easily be reconstructed to synchronize with the databases.

Any TAG-file is derived by a simple serial read on the associated manual IMAGE masters, as pointed out earlier.

Any PiCOM/3000 MPE-file is identified by a journal number which is also stored in all history entries in the database together with operator initials.

Since operator initials equals MPE username and since SYSBAS holds entries of all operators with name of homegroup, it is actually possible to perform a recovery routine laying out all LOCAL MPE-files (e.g. PLIST) in the correct usergroups based on a scan of the history datasets.

GLOBAL MPE-files, like the INJOUR, is restored similarly, but always in the DATA group regardless of creator.

If you cannot recover the ENTIRE file structure using the HP3000 IMAGE log, but have to backup just one non-IMAGE file parallel to the log by ordinary :STORE at selected break points, you are NOT using Transaction Logging at all in our point of view!

To go a bit further, it is even not enough to be able to reconstruct all the non-IMAGE files based on the database content. The ultimate recovery technique will bring any operator back to the very point in his session where the crash occured. Therefore you ALSO have to provide the status of internal sum registers per operator, so that when he enters his program again to continue the task, that program is able to get its previous memory status again from somewhere.

But since each operator has his own data-entry in SYSBAS, this is where to keep and maintain that information for possible retrieval. This however implies new updates to every interactive posting, thus presenting new conflicts as to whether a system has to be fast or robust and secure.

Really, the PiCOM/3000 operator datasets form a small database within the SYSBAS database, not just holding an entry per operator, but consisting of various sets and types of sets.

10. Programming hints

When an application design is fully developed, one may argue that all the actual programming or coding is just simple tools with which the design goals is reached and that the programmer's work is not to be concerned about by the designer.

Our experience is on the contrary that no matter how good the design may be, the end-product will not appear as the system it was ment to be, UNLESS the designer is very careful monitoring the coding phase and UNLESS he sets up certain coding rules to support the design.

We do not see how designing and coding can be totally separated and suggest that the designer is also capable of programming to some extent.

Some programming hints are hopefully already given in the previous chapters why we shall only attend a few more areas within this chapter of special interest to our design concept.

10.1 Easy translation to foreign user language

Consider English as the universal language for all software experts. Therefore keep all your code in English.

This rule seems so self-explanatory that it hardly needs to be mentioned, but it will actually impose some very important coding rules upon the programmers, rules that will ensure an end product which also foreign softwarehouses will buy and accept as a marketable system in their own country.

When you export applications like an accounting package, you have to rely on local software support and hence avoid to have to translate the sourcecode also.

Let all IMAGE SCHEMA definitions be in local language except for those items that will be addressed directly in the COBOL code.

This will allow a foreign user to perform a free standing manual SCHEMA translation of all (or almost all) IMAGE items and their definitions via EDIT/3000, thus making him able to obtain full benefit using QUERY (or the RAPID products).

In fact, the general rule is to perform all IMAGE I-O's using the "ALLITEM" mode, i.e. always process the entire data entry, never a single item. By doing so, any COBOL program needs only know the DATABASE names, the DATASET names and the names on SEARCH- and KEY-ITEMS.

Those will be the ones never to translate in the SCHEMA and hence they are created with English naming.

This concept leads furthermore to the advantage of defining all data entries as COBOL WORKING-STORAGE records in a COPY LIBRARY using COBEDIT/3000. The copy library will not have to be translated ever since all record fields are in English like everything else in the source code. Remember to assign a COBOL prefix to each record (= IMAGE entry) in order to solve the qualification problem of items belonging to more than one set.

COBOL literals like user messages must be in local language though. why exporting the sourceprograms only need a translation of them.

Unfortunately a formsfile cannot be accessed by a user written program (as far as we know) and therefore no utilities can be made to ease the replacing of screen texts by new ones in another language. Likewise for the programmed comments in the Field Processing Specifications.

On the other hand, the form fields themselves and the field names are of no interest to the end-user and we therefore suggest a naming of them according to the standard of English naming of record fields in the COBOL copy library.

Summing up, an export of the entire system will need the following tasks to be performed:

£1 Translate the IMAGE SCHEMAS using the EDIT/3000, but do not translate database, set, search- and key-items.

£2 Use FORMSPEC to rewrite all screen texts and comments.

£3 Run a utility program that will present all COBOL literals and allow new values to be replaced in the source code.

10.2 Using FORMSPEC

We have made a strong effort to design a general screen layout to be used by any form throughout the accounting system in order to produce a unique and pleasant user interface. As shown below each form will occupy a standard frame on line 1 through 5 and on line 23:

```
 1                    PiCOM ACCOUNTS RECEIVABLES
 2  RED                         PIC110-2                        23
 3 -----------------------------------------------------------------
 4  CUSTOMER MAINTENANCE              COMPANY: Acme Co. Ltd
 5 -----------------------------------------------------------------
 6
 7
 .
 .




 .
 .
22
23 -----------------------------------------------------------------
24  <error messages>
```

Module name:	ACCOUNTS RECEIVABLES
Template:	RED (colour identifies function keys)
Form name:	PIC1102 (second form used by subprogram PIC110)
Terminal number:	23
Routine name:	CUSTOMER MAINTENANCE
Company name:	Acme Co. Ltd (company in this account)
Window:	Line 24 (comments and errors)

The value of a standardized form is that the operator quickly learns where on the screen to identify the present working point within the application. The routine name will change whenever he enters a new subprogram in the application, whereas the module name first changes when entering another main branch in the menu tree structure (modules are Accounts Receivables, Accounts Payables, General Ledger). The company name is constant throughout the session, since it only changes by a new :HELLO to another HP3000 account.

Having each subprogram name identified by the form name makes it very easy to locate user reported errors simply by letting the user give you the number showed on line 2.

We also beleive the standard frame is making the operator confident towards the application since any routine or selection is presented in an already familiar way. This is also true when it comes to operating the terminal. Therefore all identical routines are assigned the same key functions.

On the older HP264x terminals this is employed using plastic templates of different colours to be placed over the eight special function keys.

Examples for Master File Maintenance (RED) and Statement of Accounts (BLUE) are:

	ENTR	f1	f2	f3	f4	f5	f6	f7	f8
RED	SHOW	CREATE	DEL	PRT	REFRESH	PREV	NEXT	MODIFY	EXIT
BLUE	SHOW	HIST/ OP ITM	NEW ACT	PRT	REFRESH	PREV	NEXT	PAGE	EXIT

A lot of forms will not conform to a standard template of course, but will use function keys in an individual way. In these cases PiCOM/3000 will display instructions on line 24 in order not to use valuable screen space within the general frame. For instance f1 = First page, f6 = Next page, f8 = Exit

Using a plastic template or displaying function key instructions on line 24 of course became ancient history when the HP262x terminals were introduced with the ability to display function key text in softkey windows. But still your application must support the older terminals.

In the screen layout line 6 through 22 is available for free design of desired texts and data fields. In this area we impose certain simple rules in order to present a quiet, non- fatiguing screen image to the operator. All texts are half bright whereas data are full bright in normal video, input fields are underlined and start/stop delimiters are invisible.

Never use FORMSPEC to freeze the top part of the screen appending lines below, as for instance would seem natural when producing statement of accounts.

This approach has the following disadvantages:

£1 Slow performance.

£2 At least on HP264x terminals each appended line causes the entire screen to tremble. What happens is that apparently all lines in turn perform a right/left shift even causing position 80 on the line to be lost!

£3 As soon as your data in the head form changes, for example when the operator wants to look at another account number, the entire screen is cleared and the head form has to be shown again and filled with the new data. This slows down performance even more.

The only advantage you get is that lines beyond the screen are appended while the upper lines roll up nicely under the frozen head form.

Instead consider one of the stronger sides of VPLUS:

Presenting new data on a current form is done rapidly and beautifully since VPLUS only shows the data which have changed.

Therefore always design full-size forms covering line 1 through 23 with REPEAT OPTION = R and NEXT FORM OPTION = C.

Even use this approach on statement of accounts and get all posting lines into the buffer before performing a VSHOWFORM. There will be no tremble and no intermediate screen clearing and the overall response time will actually be improved.

10.3 The COBOL copy library

The PiCOM/3000 COPY LIBRARY contains the following types of COBOL code:

£1 WORKING-STORAGE record layouts of all IMAGE data entries.

£2 SELECT statements for all KSAM and MPE-files. Since the SETFIL UDC establishes all file equations at logon time and since all files are created permanently by simple utility programs, the ASSIGN clause is very simple, for example SELECT CUSTG1 ASSIGN TO "CUSTG1".

£3 FD's for all KSAM and MPE-files containing the record layout.

£4 USE PROCEDURES on all KSAM and MPE-files. Since the files will be used in all types of programs (MAIN- and SUBPROGRAMS), you have to figure out a smart way to let the same USE PROCEDURE apply in both situations.

PiCOM/3000 does that by letting all USE PROCEDURES have a call to the I-O subroutine (see chapter 7.4) which will fetch the HP3000 error message, detect whether a SESSION or a JOB is running and display the message accordingly.

£5 WORKING-STORAGE record layouts of all WRK-records since each will be used by at least 3 programs: a subprogram passing the record to the TFCREAT subroutine, which writes the WRK-record on disc to be read by a background program in order to get the necessary runtime information.

£6 All GLOBAL LINKAGE- and WORKING-STORAGE areas that are used by virtually all programs in various calls. Examples are COMAREA, PF-AREA, FI-ARAY, DB-AREAS and all the global SYSM01 flags. Most of these areas serve as parameters to any subprogram call, being copied into the COBOL linkage section. Batchprograms use working-storage duplicas to copy into the source. These copys may be assigned COBOL VALUES, which is not possible for the linkage section.

27 The file handlers PROCEDURE DIVISION codings (see chapter 7.4).

10.4 How to lock KSAM- and other MPE-files

Let us review the PiCOM/3000 file structure in regard to the above mentioned file types. GLOBAL files are placed in the central DATA group and owned by all users. They are either TAG-files or GLOBAL MPE-files. LOCAL files reside in the users' groups and are accessible only by each user.

The TAG-files are granted SHARED access, but will only be subject to writes or rewrites or deletes in those programs that maintains the associated IMAGE master dataset.

These programs will by means of EXCLUSIVE/UNEXCLUSIVE statements prior and after the I-O establish the needed lock, but always immediatly after the database lock/unlock in order to satisfy the overall rule of not using MULTIPLE RINS. An EXCLUSIVE is always an unconditional lock of the entire file, - the program will automatically wait until the lock is obtained.

If a file is SHARED, EXCLUSIVE must be used. If a file is not SHARED, EXCLUSIVE cannot be used.

All other TAG-file access is read-only and thus containing no lock problems.

The GLOBAL MPE-files (like INJOUR) are not SHARED at all for logical reasons, - it has no meaning letting two users create Interest Notes at the same time in the same Transaction File.

The LOCAL MPE-files (like PLIST) are not SHARED either. This may seem natural, since they belong to one user only, but that user may very well wish to look at his file from the terminal at the same time the user BATCH is asked to perform a background printout of the file.

We chose to have both the GLOBAL and LOCAL MPE-files not SHARED in order to get a more robust application from a user's point of view. When only one program is working on a file at a time, what can go wrong? This also explains why an operator only works from one terminal at a time.

How will a program (whether SESSION or JOB) detect if one of those datafiles are in use? The reason to put that question is that very surprisingly we did not find any suitable way to let MPE make that decission, for instance by returning a FILE STATUS value to be tested upon by the programs after attempts to open. No matter what technique used, a TOMBSTONE (-F-i-l-e--I-n-f-o-r-m-a-t-i-o-n--D-i-s-p-l-a-y-) would always appear, splashing all over the forms. Not even USE PROCEDURES prevented that.

A guess is, that MR is requested or that this was a MPE III weakness.

Instead the SYSBAS database holds information on status of all datafiles ("in use" or "not in use"). The status flag for the GLOBAL MPE-files are stored in the company dataset and the status flags for the LOCAL MPE-files are of course stored together with the other data per operator.

Each program will therefore first check on the flag and set/ reset it at BOJ/EOJ. If the file is in use already, a subprogram will notify on line 24, while a background job will terminate with a display on the JOB-list according to the PiCOM/3000 rule.

Remember, that if a background job always MUST complete when requested, take it up in the on-line application as a subprogram.

The PiCOM/3000 SL holds a small subroutine, "FILTAKEN", that is used to check use-status and to set/reset the proper flag in SYSBAS.

11. Conclusion

In this article has been made a strong effort to discuss most of the questions that will face newcomers on the HP3000 developing business packages to be sold and installed turn key-wise to clients with none or little EDP-experience.

We hope somebody will find it worth studying and that he will benefit from a faster project startup than we had, - and a fewer walk-backs. Also that trained members of the HP3000 community may discover new angles to the computer.

Deliberately, description of WHAT the PiCOM/3000 package actually is capable of doing for its end-users is excluded from this discussion of design and engineering, but could be the theme on another occasion.

12. Bibliography

Having finished the first release of the system, we finally got some time to look out at the world. It was very teaching then, to be aware of the International Users' Group and the litterature it provided through annual proceedings and through the Interact publication.

From those sources we recommend the following writings as a partial list of bibliography for deeper investigation in some of the key-areas in the article:

1. Dennis Heidner, "Transaction Logging and Its Users", in HPIUG 1982 San Antonio Proceedings pp. 2-34-1.
2. David J. Greer, "IMAGE/COBOL: Practical Guidelines", in HPIUG 1982 San Antonio Proceedings pp. 4-4-1.
3. Eugene Volokh, "The Truth About Disc Files", in HPIUG 1982 San Antonio Proceedings pp. 11-17-1.
4. Robert M. Green, "MPE Internals for Neophytes", in Interact July/August 1982 pp. 58.
5. David Brown, "Maintenance Tips for User Segments in the System Segmenter Library", in Interact July/August 1982 pp. 48.
6. Peter Somers. "Using COBOL. VIEW and IMAGE. A Practical Structured Interface for the Programmer", in HPIUG 1982 San Antonio Proceedings pp. 4-12-1.
7. Bill Vaughan, "Increased Reliability at a Lower Cost", in HPIUG 1981 Berlin Proceedings.