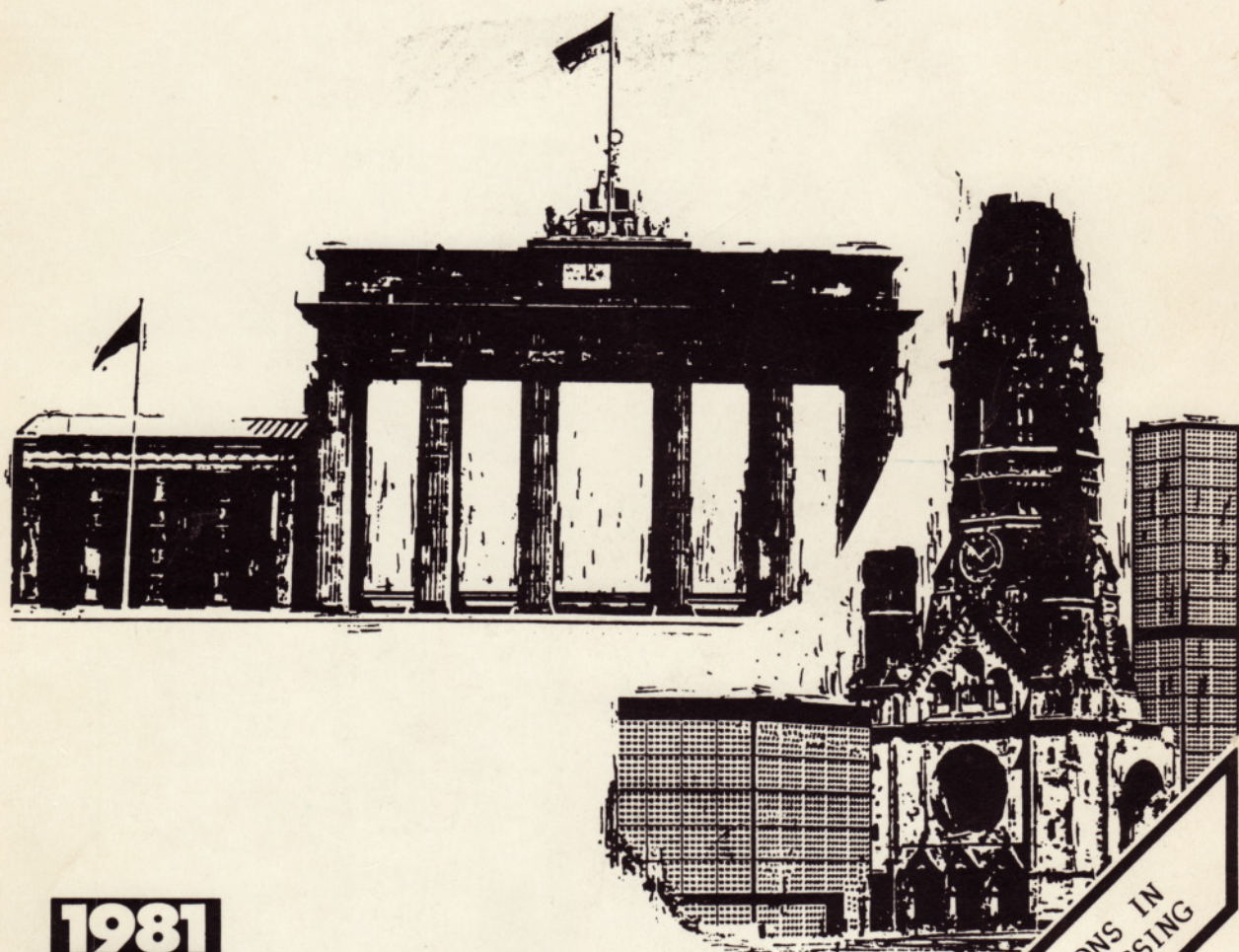


HP 3000 INTERNATIONAL USERS GROUP

1981 BERLIN INTERNATIONAL MEETING
GERMANY, OCTOBER 5-9

PROCEEDINGS



5-14-82 LIBRARY MEMBERSHIP

PROCEEDINGS OF THE HP3000 INTERNATIONAL USERS GROUP

BERLIN MEETING 1981 OCTOBER 5. - 9.

OCTOBER 5 THRU OCTOBER 9, 1981

TECHNICAL UNIVERSITY BERLIN

BERLIN / WEST - GERMANY

HEWLETT-PACKARD CO.
CORP. LIBRARY

Q11.16.2
H119 H119
19211b

1950
1951
1952

1953

1954

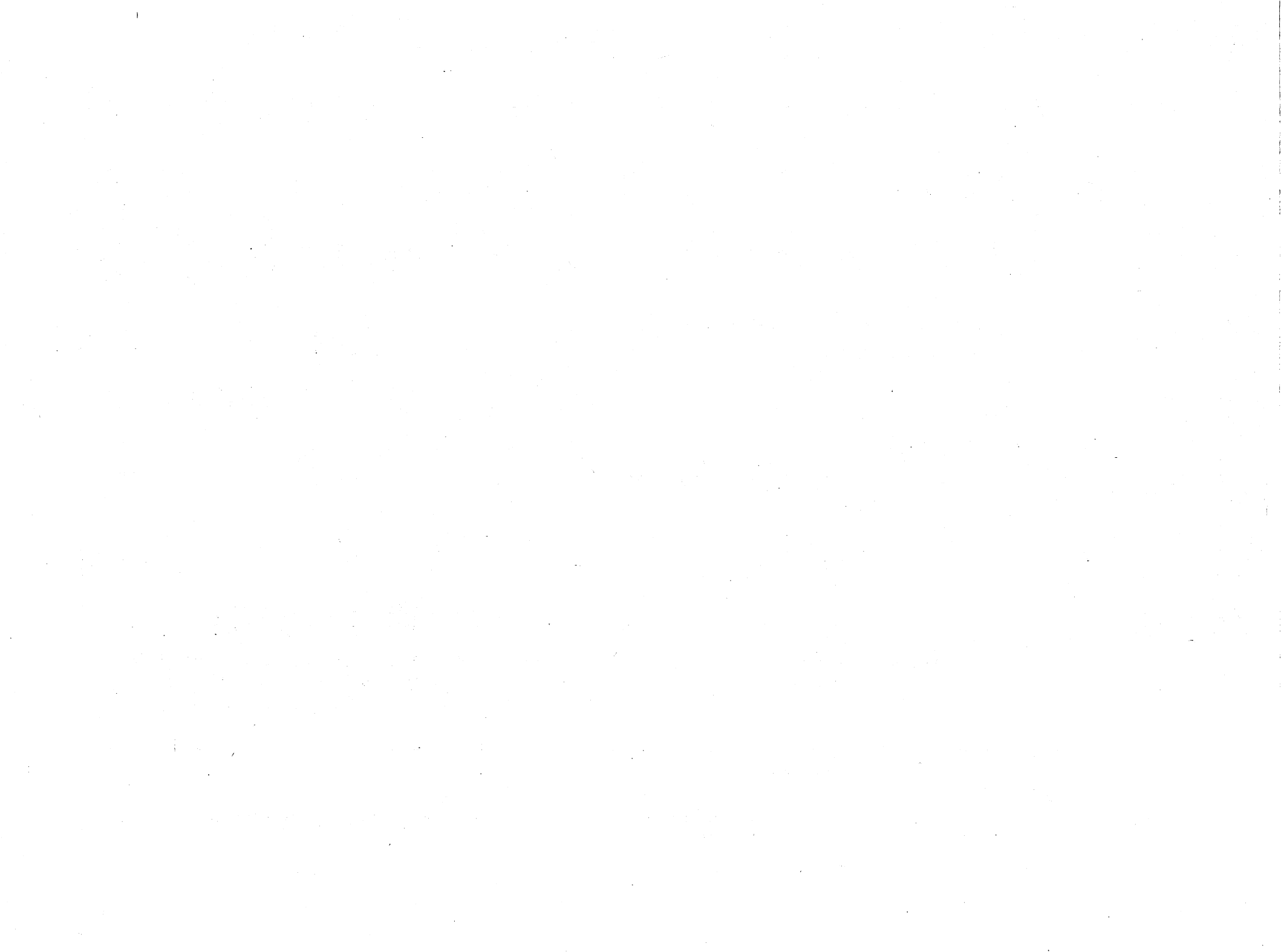
COB: TBSVA
TAAVA 500
NEATEL-UCKVD COB

PROGRAM OF THE 1981 HP3000 INTERNATIONAL USERS GROUP MEETING IN BERLIN

No.	Titel	Author(s)
Subject Category: <u>Invited Papers</u>		
B.0	On the Use of Prototyping in Software Development	Christiane Floyd
F.0	Software technology - A Future Requirement or Current Necessity	H. Blask
Q.0	Some Problems of Software Engineering	Wladyslaw M. Turski
Subject Category: <u>Business Applications</u>		
D.5	Budgeting and Profit Planning on the HP3000	Jack Damm
E.5	IPB: Interactive Planning and Budgeting	Jens Pallesen
I.5	The Use of EDP in the Freight Forwarding and Ships Agency Business	Hardy Jensen, Jorgen Rix
J.4	Using IMAGE-3000 to Establish an Order Processing--Finished Goods Inventory On-Line Database System	K. B. Sheu
K.5	A Try to Establish an Off-Line Time-Reporting & Wage Combination System	W. G. Hsia
O.5	Data Capture Systems for Real-Time Manufacturing Management	Bruce Toback
U.4	New Directions in Investment Management	Frank Helson, Richard Steck
Subject Category: <u>Data Base Management</u>		
K.2	Data Analysis - The answer to successful implementation of IMAGE	Richard Irwin
L.3	A Comparison of Relational and Network Data Base Management Systems as Implemented on the HP/3000	Christopher M. Funk, Thomas R. Harborn
T.5	Relational Database Concept, Consequences for Organization and Management-Structures	Uwe Hinrichs
U.2	How to get more from your core memory or CFS/3000: A Core Resident File System	Pierre Senant

PROGRAM OF THE 1981 HP3000 INTERNATIONAL USERS GROUP MEETING IN BERLIN

No.	Titel	Author(s)
Subject Category: <u>Data Communications</u>		
H.5	Using DS3000-1000 with HP/1000 Master Programs	Jörg Müller
I.4	A Distributed Computer System Interconnecting HP3000, HP1000, and other Mini-Computers	Björn Dreher, Hans von der Schmitt, Raymond Schoeck
L.5	Data Concentrators: In Focus for Mini-Computer Users	Peter J. Mikutta
U.3	Decentralised Processing - New horizons for Sytems Designers	Norman Midgley
Subject Category: <u>Education</u>		
R.4	RMIT Student Data Base	N. F. Riedl, E. de Graauw
Subject Category: <u>Editors</u>		
T.4	Fast Editing and Program Development using a Full Screen Editor	Jacques Van Damme
Subject Category: <u>Generators</u>		
E.3	New Approach toward System Implementation	Jean Pierre Theoret, Alan Rowan
I.1	Global Optimization: PROTOS - A COBOL Program Generator for the HP3000	Tipton Cole, Larry Van Sickle
Subject Category: <u>Graphics</u>		
J.5	Computer Graphics, a Powerful Information Tool	Sigmund Hov Moen, Frederik Major



PROGRAM OF THE 1981 HP3000 INTERNATIONAL USERS GROUP MEETING IN BERLIN

No.	Titel	Author(s)
Subject Category: <u>Papers by Hewlett-Packard Personnel</u>		
D.1	Hewlett Packard - A View From The Inside	Jan Stambaugh
D.3	The HP2680A Laser Printing System (Overview)	Jim Langley
E.1	Data Communications Strategy	Steve Zalewski
E.4	New Directions in Customer Training	Donna M. Senko
G.0	Business Computer Group Strategy	Klaus-Dieter Laidig, Werner Gamm
✓H.1	ANSI COBOL 198x: The Story behind the Headlines	Greg Gloss
H.2	HP Business Computer Group BASIC	Steve Ng
H.4	Production Management/3000	Wolfgang Bayer
J.1	Trends and Future Directions of Hewlett Packard Pheriphal Products	William J. Murphy
K.1	MPE-IV	Michael J. Paivinen
L.1	News to MPE-IV Internals	Uwe Jensen
L.4	The HP2680A Laser Printing System (Software)	Anthony Stieber
O.1	Distributed Processing: A Hewlett Packard Solution	Matthew O'Brien
O.2	Terminal I/O - An Engineering Feedback Session	Jim Beetem
P.1	HP Plus	Jef Graham
P.2	Cold Dump Analysis	Michael J. Paivinen, Sergio Mastripiari
✓R.1	Introducing the HP On-Line Performance Tool (OPT/3000)	Clifford A. Jager
R.2	ACE: Operatorress Job Scheduling and Processing	Bill Vaughan
R.5	X.21/X.25 - Data Communications	Bill Baddeley
S.1	Operator/Console Interface / MPE-IV - An Engineering Feedback Session	Michael J. Paivinen
S.3	Terminals Strategy and New Products	Richard Franklin
S.4	Interactive HP3000 to IBM Host Communications	Cynthia L. Smyth
S.5	RAPID/3000, New from HP: Relational Access, Prototyping and Interactive Development	Victor Canivell
T.1	Terminal I/O Controller for HP3000 Systems	Jim Beetem
T.2	Business Graphics: A Means to Improve Management Productivity	Chris Kocher
T.3	Increased Reliability at a Lower Cost	Bruce Wheeler
U.5	High speed digital image processing using a picture-scanning technique on incremental plotters	Ramesh Pamchal

PROGRAM OF THE 1981 HP3000 INTERNATIONAL USERS GROUP MEETING IN BERLIN

No.	Titel	Author(s)
Subject Category: <u>Installation Management</u>		
D.2	Thoughts concerning: How secure is your System?	Joerg Groessler
✓H.3	System Performance and Optimization Techniques for the HP/3000	John E. Hulme
I.2	Transaction Processor for the HP3000	Godfrey Lee
J.3	JOBLIB/3000 - an Interactive Pre-processing System of Jobs	Martti Laiho
O.4	Using the HP3000 as a Mainframe	Carl Christian Lassen
R.3	The happy Transition	Harald Henriksen
S.2	Security / Risk Management	Clifford W. Lazar, Eugene Volokh
Subject Category: <u>Software Engineering</u>		
D.4	RATFOR = FORTRAN/3000 + Elements of Structured Programming	Björn Dreher
E.2	User friendly Applications in commercial realtime Dataprocessing	Herbert Augenstein
J.2	New Software Engineering Alternatives: Notes on Selecting Software	Birket Foster
K.4	Optimisation of SPL and FORTRAN Programs	John Machin
✓U.1	Programming for Device Independence	John E. Hulme
Subject Category: <u>Word-Processing</u>		
I.3	Integrated Data- and Textprocessing with hp 3000	Joachim Geffken
K.3	Computerized Typesetting: TEX on the HP3000	Lance Carnes
L.2	A Few Well-Chosen Words Concerning a Few Chosen Ways to do Word Processing, Some Well-Chosen, Some not	Wirt Atmar
O.3	DAISY/3000 A new Approach in Text Processing	Timo Raunio



AUTHORS OF THE
1981 HP3000 INTERNATIONAL USERS GROUP MEETING IN BERLIN

Name Paper(s)

Atmar, Wirt	L.2
Augenstein, Herbert	E.2
Baddeley, Bill	R.5
Bayer, Wolfgang	H.4
Beetem, Jim	O.2, T.1
Blask, H.	F.0
Canivell, Victor	S.5
Carnes, Lance	K.3
Cole, Tipton	I.1
Damm, Jack	D.5
De Graauw, E.	R.4
Dreher, Björn	D.4, I.4
Floyd, Christiane	B.0
Foster, Birket	J.2
Franklin, Richard	S.3
Funk, Christopher M.	L.3
Gamm, Werner	G.0
Geffken, Joachim	I.3
Gloss, Greg	H.1
Graham, Jef	P.1
Groessler, Jörg	D.2
Harborn, Thomas R.	L.3
Helson, Frank	U.4
Henriksen, Harald	R.3
Hinrichs, Uwe	T.5
Hsia, W. G.	K.5
Hulme, John E.	H.3, U.1
Irwin, Richard	K.2
Jager, Clifford A.	R.1
Jensen, Hardy	I.5
Jensen, Uwe	L.1
Kocher, Chris	T.2
Laidig, Klaus-Dieter	G.0
Laiho, Martti	J.3
Langley, Jim	D.3
Lassen, Carl Christian	O.4
Lazar, Clifford W.	S.2

AUTHORS OF THE
1981 HP3000 INTERNATIONAL USERS GROUP MEETING IN BERLIN

Name Paper(s)

Lee, Godfrey	I.2
Machin, John	K.4
Major, Frederik	J.5
Mastriperieri, Sergio	P.2
Midgley, Norman	U.3
Mikutta, Peter J.	L.5
Moen, Sigmund Hov	J.5
Murphy, William J.	J.1
Müller, Jörg	H.5
Ng, Steve	H.2
O'Brien, Matthew	O.1
Paivinen, Michael J.	K.1, P.2, S.1
Pallesen, Jens	E.5
Pamchal, Ramesh	U.5
Raunio, Timo	O.3
Riedl, N. F.	R.4
Rix, Jorgen	I.5
Rowan, Alan	E.3
Schoeck, Raymond	I.4
Senant, Pierre	U.2
Senko, Donna M.	E.4
Sheu, K. B.	J.4
Smyth, Cynthia L.	S.4
Stambaugh, Jan	D.1
Steck, Richard	U.4
Stieber, Anthony	L.4
Theoret, Jean Pierre	E.3
Toback, Bruce	O.5
Turski, Wladyslaw M.	Q.0
Van Damme, Jacques	T.4
Van Sickle, Larry	I.1
Vaughan, Bill	R.2
Volokh, Eugene	S.2
Von der Schmitt, Hans	I.4
Wheeler, Bruce	T.3
Zalewski, Steve	E.1

Conference Paper:

HP 3000 International Users Group Meeting

7. Oct. 1981

On the Use of "Prototyping" in Software Development

C. Floyd

Institute for Applied Informatics
Technical University Berlin

The phrase "rapid prototyping" is currently en vogue in certain software engineering circles. The basic idea is to aid communication between software producers and software users (customers), in particular during the early stages of software development, by furnishing experimental versions of the system, to be tried out as part of requirement analysis.

In what follows I will attempt to demonstrate the role that a software-"prototype" might assume in different production settings in a manner compatible with the main line of software engineering's strive for a methodology, as illustrated for example in Prof. Turskis lecture at this meeting on October 9th (/TURSKI 81/). To begin with, however, some comments about the phrase "rapid prototyping" are in order, since this promises to be yet another unfortunate misnomer, which may well lead to serious misunderstandings, if ever this technique should be adopted by the software industry. A prototype is a well established concept in the engineering disciplines where it refers to the first functioning version of a new kind of product. In this context, a prototype is intended to exhibit all essential features of the final product and thus becomes the basis for experiments before the beginning of large scale production. This analogy does not carry over easily to software production, where we are not faced with mass production at all. Surely, if we use the concept of a prototype in software production - as I will do from now onwards, though under protest - we shall have to give it a new meaning appropriate for our purposes.

The second unfortunate term in the phrase is the epithet "rapid", which misleads us into believing that speed is the essential aspect in building a prototype. Again, this is in conflict with the engineering tradition, where the prototype is the final result of careful design, extensive calculations and field tests. I fail to see how a software prototype produced rapidly, without the careful preparations mentioned above will yield reliable answers in determining actual requirements.

In order to judge the usefulness of prototyping in software development we must find answers to the following questions:

- Why is communication about software requirements based as it is on interviews, checklists and bulky documents not sufficiently reliable and how could a prototype be helpful in this context?
- How does the software prototype relate to the final product?
- Is there one, or are there several prototypes and how are they evaluated?
- Under what circumstances can we justify the additional investment brought about by producing a prototype in the early stages, i.e. what do we hope to gain later on?
- How does prototyping relate to an orderly approach to software development, based on deriving a program from a rigorous specification according to the rules of programming methodology?

As a starting point in answering these questions we should take a close look at the well known phase-oriented approach

to software development, its merits and shortcomings (see for example /LEHMANN 80/). The phase-oriented approach was devised as a means to find contractual bases in software development and to define intermediate results in terms of documents, which form the basis for subsequent work. The phase-oriented approach relies on some important assumptions, as there are:

- that requirements, at least in principle, can be fixed in advance,
- that documents, provided that their contents are described in a sufficiently rigorous manner, are adequate as a primary means of communication, i.e. that the customer knows what he will get when he signs the contract.

Both of these assumptions unfortunately are contradicted in the daily practice of software professionals who are faced with the difficult task to base their own work on existing base-line documents, while at the same time coping with constantly changing requirements from their customers. The phase-oriented approach does of course permit to go back to earlier phases when needed, but it does not encourage the planing of profound revisions.

The phase-oriented approach provides a sound basis to limit the liability of the software producer. The product is defined by its specification and the liability of the software producer ends when he has derived a program, which is correct with respect to its specification. As Prof. Turski will point out in two days, this is a highly nontrivial activity which is well supported by modern software engineering techniques. Yet, experience shows, that even a correct program may not at all be adequate to fit the user's needs, because of far reaching misconceptions about the actual requirements.

The situation is aggravated by the fact that mistakes made early in software development are the most costly to correct. Serious mistakes in requirement analysis may well be too costly to correct at all. The user organization will have to adapt to the software - not vice versa.

There are important reasons why it may prove very hard to find out detailed software requirements for the development of large programs:

- 1) It is extremely difficult for people to visualize how seemingly minor decisions about software will later on affect their work with the system.
- 2) It is often extremely difficult to locate all groups of people who will be directly or indirectly affected by the system. Different user groups often have conflicting views about an information system (which they perceive from their own perspective), or they simply ignore each others needs.

The above mentioned difficulties do not pertain to all requirements alike, in fact the following classification of requirements helps to point out the areas where troubles most likely arise. We can distinguish:

- functional requirements describing the desired output to be produced for a given input (the relation between input and output may be highly nontrivial, but it is normally governed by a stringent set of rules; whether or not the program obeys the same set of rules can be proved - at least in principle).
- performance requirements stating the resources available to achieve these functions (it may be difficult to show the precise constraints on resources, whether or not the program meets these constraints can be measured - at least in principle).
- handling requirements characterizing the manner in which the system is to be embedded into the activities of all people affected by it.

Of these three, the handling requirements are the least well understood. Handling requirements pertain amongst others to the following areas of special concern:

- The design of man-machine interfaces in the widest sense (including conceptual models the user must have, in order to understand what the system does);
- The degree of system integration and as a consequence the possibility of interfering with or reshuffling the system's functions as needed ("conviviality" of the system according to Ivan Ilich /ILICH 79/);

- The interplay between formalized (i.e. computer-supported) and informalized work-steps permitted by the system (with the two extremes: the system enforces a working style akin to the assembly line or the system offers a tool-box to be used as needed).

This list does not claim to be complete. The examples are indicated in order to demonstrate that handling requirements will indeed lead to important decisions about software structure, that may well determine the adequacy or inadequacy of an otherwise correct program!

In the absence of a suitable theory of organizations and of sound user psychology, communication with the user, about software requirements, will continue to rely largely on experience and intuition. In this context, it is felt by many that communication is more reliable, if it is based on an already existing program which can be evaluated (albeit not systematically since there is no underlying theory how this might be done). A prototype therefore, is to be furnished in order to reduce the probability of misunderstanding requirements. The additional investment needed for its production is justified by the hope that this investment is significantly smaller than the costs that are likely to arise from the need to adapt an inadequate program later on.

It should be kept in mind, that the production of a prototype is justifiable only in the case of long-life systems, where a further expansion of the early phases will presumably lead to profits over a considerable period of time. Further, this technique is particularly relevant for programs which are embedded in technical or socio-technical environments, because such programs will have elaborate handling requirements associated with them.

How then, does a software prototype relate to the actual product in time, scope and quality? We can distinguish several feasible approaches here:

- the prototype may be intended to aid requirements analysis only or it may be intended to accompany the actual system throughout its lifetime.
- The prototype may be intended to cover essentially the same scope as the actual system or it may be intended to exhibit selected features only.

The intention of the propagators of "rapid prototyping" seems to be to produce throw-away prototypes - with the same functional scope as the actual system, but of lower quality - which precede the development of the actual system itself. This approach implies the call for new techniques, such as prototyping languages and interpreters, which reduce the effort of prototype production. I would like to point out that this approach is highly problematic:

- Since the specification does not yet exist at the time of prototype production, it is not clear what the functional scope of the prototype should be, and we find ourselves thrown back into the kind working style which was - with good reason - deplored ever since the 1960's.
- Should the specification already exist, it is not clear what is to be gained by quickly producing a system with the same functional scope, but of lesser quality than the final product. It should be remembered that the essential thing about the prototype is its evaluation, for which there is no systematic basis available as yet and which will prove to be a large effort if the prototype itself is complex. Therefore the feedback obtained by the evaluation of such a prototype will come late and will be unreliable. The production of the real system will be delayed, with no obvious gain to justify the delay.
- If the prototype is to precede, rather than to accompany, the actual system it will not be helpful in dealing with changing requirements, as will be argued below.

Requirements for software embedded in technical or socio-technical systems must be expected to change, because

- original requirements were misstated (the probability of this may perhaps be reduced with a "rapid prototype"),
- the environment evolves and develops new requirements,
- the system, once in use, transforms its environment and thus itself contributes to producing new requirements.

Because of the last two of these points, a "rapid" throw-away prototype cannot be expected to aid in reducing troubles with changing requirements in the long term.

In view of all the problems cited above it seems appropriate to drop the analogy with engineering prototypes, to generalize the concept of a "software prototype" considerably and to combine its production with an orderly approach to software production.

In the following, a prototype will designate a preliminary version of the actual system which exhibits selected features of the final product.

There may be one or a series of such prototypes, depending on the needs of the specific project. The prototypes serve primarily to aid discussions about handling requirements, i.e. whereas their functional scope may be only a fraction of the actual product's; they are carefully designed, so as to illustrate how the system can be embedded into its working environment.

This way of using prototypes implies a different view of software development, which has been termed the

process-oriented approach elsewhere (/FLOYD 81/). Rather than viewing software development as the production of one program, by going through several phases and ending up in "installation" and "maintenance", I prefer to view software development as a sequence of development cycles (re-)design, (re-)implementation and (re-)evaluation. It must be emphasized, that each development cycle is based on a specification from which the program version to be produced can be derived in an orderly fashion, thus there is no contradiction between this approach and software engineering methodology; instead the specification itself is viewed as an evolving document.

As opposed to the common phase-oriented approach, communication with the user is continuous and feedback from the evaluation of successive prototypes is incorporated into redesign at the end of each development cycle.

The specification serves as a common defining document for both software producer and user. In particular, the application model associated with the specification must be phrased so as to exhibit the embedment characteristics of the system in its working environment.

The responsibility of the user consists in providing, in each development cycle, an evaluation basis for the prototype which can be derived from the application model. In the absence of a theory we can still point to no systematic way of how to do this, but at least the new framework will allow to progress in small, meaningful steps.

In the context of the modified approach to software development described above, a prototype can assume one of the following roles:

- 1) A prototype may coexist with the actual product; it is, then, a program model of the same specification, less rigorously treated, and serves as basis of experimental changes before the program itself gets modified.
- 2) A prototype may coincide with the actual product: This is intended in version-oriented software production in development cycles, as described above. The specification is an evolving document; it may or may not change from one version to the next.
- 3) The product itself is a prototype: This arrangement is relevant to the production of standard software, which is designed to meet the functional requirements of a class of users, but where handling requirements can be decided by replugging existing components to fit individual needs.
- 4) Production starts from a prototype: Analysis and redesign of existing software can be viewed as a special case of the same approach.

Each of these arrangements may prove a valuable help to aid communication with the user in certain production settings and each of these can be combined with orderly programming methods. How much of a previous version of the program can be retained to be incorporated into a subsequent version, must be decided as part of the redesign effort following each development cycle.

In order not to create false hopes, however, we must remember that in this manner we have obtained a more flexible framework - no more. Modern software design and specification methods do not necessarily facilitate incremental partial changes, which makes the use of a specification as an evolving document awkward. We all know

that in practice the discrepancy between programs and specifications increase with time, thus making the specification obsolete long before the program is shelved. We can hope that progress in specification research will help to remedy this situation.

On the other hand, we cannot hope that communication with the user - even based on carefully designed prototypes - will significantly improve, unless we find a theory of software embedment which is based on solid grounds in both psychology and the social sciences. Such a theory will help the software designer to make intelligent choices that can be justified to the user by rational arguments, rather than by individual tastes. It will also allow for the systematic evaluation of prototypes.

Because of the serious concern for the adequacy of software systems in their working environment, research efforts in these directions must be considered one important front of software engineering research.

References:

FLOYD, C.:

A Process-oriented Approach to Software Development.
in: Systems Architecture,
Proc. 6th ACM European Regional Conf. (1981),
Westbury House 1981, pp. 285-294.

ILICH, I.:

Tools for Conviviality.
FONTANA/COLLINS 1979.

LEHMANN, M.M.:

Programs; Life Cycles and Laws of Software Evolution.
in: Proc. IEEE, Special Issue of Software Engineering
SEPT. 1980, pp. 1060-1076.

TURSKI, W.:

Some Problems of Software Engineering.
in: HP3000 International Users Group Meeting
OCT. 1981.

Presentation Abstract

Presentation Title: Thoughts concerning "How secure is your System ?"

Author(s): Jörg Grössler

Title(s): Dipl.-Ing.

Address: Technical University Berlin, Sekr. HE3,
Einsteinufer 19; 1000 Berlin 10

Abstract: (No more than 200 words)

Data security is an essential aspect of online computing systems.

It must ensure that internal data cannot be accessed by unauthorised
persons and that the file system can be rebuilt in case of a hard-
or software disaster.

In this paper components of the security system of MPE are presented
and analyzed. Weak points are highlighted and the measures necessary
to improve security are discussed.

Thoughts concerning

How secure is your System ?

Joerg Groessler; Technical University Berlin



What data security means

- o to be able to rebuild the file system
in case of a disaster

- o to restrict access on various type
of data

Standard File Backup Facilities in MPE

- o SYSDUMP, RELOAD
(based on magnetic tape)

- o STORE, RESTORE (tapes)

- o User Logging
(based on disc or tape)

- o Private volumes (disc)

Problems with Standard File Backup

- o tape read error during RELOAD
 - system cannot be started
 - next action "must be RELOAD"

measures:

- change disc packs before RELOAD
- RELOAD with 'ACCOUNTS-only' then RESTORE the remaining files (very time consuming)

- o tape read error during RESTORE

- all files stored behind error point cannot be restored

measure:

- use RESTORE- or GETFILE2-program

- o user logging causes system overhead

measure:

- consider special logging during program design

Prospects for tape-backup system

- o GETFILE-facility will be improved
- o special STORE-RESTORE system is considered (this possibly includes features like UPDATE and APPEND)

Restrictions in Data Access

- o account-system (users, groups, accounts with different passwords)
- o user capabilities (SM, PM, PH, etc.)
- o filenames with passwords
- o privileged files
- o file access capabilities on user/group- and file-level
- o RELEASE/SECURE-commands

Possible seven Ways to crack the System

1. FIELD.SUPPORT

measure:

Password on SUPPORT-account
or remove SUPPORT-account
from the system

2. Jobs in PUB.SYS-group

measure:

password on job-file or
put job into other SYS-group

3. LISTUSER @.@;LP

measure:

log-on-UDC or perform command
not in PUB.SYS-group

4. Open all files of the system

measure:

special analysis of system logging

5. Read terminal buffers (PM-capability
needed)

measure:

remove PM-capability

6. Reading tapes

measure:

keep track of all tape-transactions
also using system logging

7. FOPEN on terminals

measure: ??

8. ...

Abstract

This paper describes the HP 2680A Laser Printing System from the perspective of the HP 3000 programmer. The printer hardware is first described, then its features are explained. Concepts of downloadable character sets, electronic forms and logical pages are discussed. The implementation and use of these printer features via the system software is also covered. The impact of the laser printer in a distributed computing environment is briefly explored.

Overview

The HP 2680A is Hewlett Packards first page printer. It is based on an electrophotographic process which was licensed from Canon, a Japanese firm. The printer was designed and is manufactured by the Boise division in Boise Idaho.

Several key objectives were established at the start of the program. Reliability, flexibility, features matched to 3000 user needs, simple powerfail and paper jam recovery, very low CPU overhead, and the ability to access the printer and its features from existing programs without modification became the primary objectives of the development effort.

From the beginning the printer was designed as an extension of MPE, not an added on peripheral. This tight coupling yielded a fully integrated printing system that is fully supported by the MPE file system and spooler. In addition a powerful subsystem exists which allows complete character set and forms design. Flexible page formatting and a full complement of intrinsics provide access to all printer features.

By fully integrating the printer into the 3000 simple and reliable power fail and paper jam recovery is realized. All these benefits were achieved while the CPU overhead to drive the printer was reduced an order of magnitude from that required to drive conventional printers at comparable data rates.

Hardware

The 2680A is approximately 5.5 feet long, 2.5 feet deep and 4 feet high. It weights about 875 pounds. Power requirements are 4500 watts when printing. Throughput is 45 8.5 by 11 inch pages per minute. The equivalent lines per minute speed is 2900 lpm ranging up to 12000 lpm in reduction mode.

The paper path is short and readily accessible to the operator. It features a powered paper stacker. The fusing system is radiant, eliminating any pressure or high temperature rollers. Nothing contacts the upper side of the paper once the image is transferred from the drum to the paper, contributing to excellent reliability. The web is pulled by a programmable torque motor on the output tractors, and paper motion is gated by stepper motor driven input tractors. A solenoid powered retraction mechanism pulls the paper away from the drum when the seam

THE HP 2680 LASER PRINTING SYSTEM

(OVERVIEW)

by Jim Langley
HP 2680A R&D Project Manager
March 15, 1981

on the drum comes around. The input paper platform acts as a splice table; a vacuum is used to hold the paper onto the table when splicing a new box of paper onto the end of the previous box. The paper path can accommodate various widths up to 12.5 inches and lengths up to 17 inches. A width sensor on the input tractors allows the printer to energize only the correct width in the preheater section of the fuser. Paper which is heated produces odors, which are trapped and oxidized in replaceable filter cartridges.

The image forming process is electrophotographic. The heart of the system is a photoconductive drum about 19 inches in circumference and 14 inches long. The drum is coated with cadmium sulfide and wrapped in mylar for protection. The drum is uniformly erased and then charged to several hundred volts at the first station. The laser is then scanned across the drum perpendicular to the direction of rotation. The beam is modulated to give 180 dots per inch resolution. There are 2048 dots in one scan line, giving a printable area 11.38 inches across. The drum rotation allows the sweeping laser beam to cover an area 17 inches long. The circular dot is about .008 inches in diameter on a grid .0055 inches square. When the laser beam hits the drum the voltage is depleted. Next the drum rotates past a cloud of fine flour-like black plastic. The plastic toner is attracted to areas of no voltage by electrostatic forces. The pattern traced by the scanning laser beam is now visible as a sharp black image on the drum. Finally the paper and the drum are brought together for about 1 inch of tangential contact. The paper is correctly charged to firmly attract the toner off the drum. The small amount of residual toner not deposited on the paper is then scraped off of the drum by a urethane wiper blade and collected by a vacuum system. As the drum turns these processes are executed simultaneously at different stations around the drum.

In order to achieve high print quality over a wide range of ambient conditions the HP 2680A has two closed loop control systems. The electrostatic control system monitors the potentials on the drum just after the laser station. The voltage is measured both where the laser exposed the drum and where it did not. The microprocessor taking the measurements then controls several programmable power supplies to maintain the correct drum potentials. The readings and adjustments are made once per drum rotation. The electrostatic closed loop compensates for variations between replacement drums, drum degradation over time, humidity, temperature and altitude variations, and toner mixture fatigue.

A second closed loop system monitors the developed image on the drum to control print density on the page. By varying the amount of toner in the developer assembly which brushes the toner mixture across the drum the amount of toner on the drum and hence the final print darkness on the paper can be controlled.

The mechanical features of the printer were designed to be simple and reliable, and the operator functions are easy to learn and execute. A vacuum system in the printer contributes to cleanliness. It is used to recover toner wiped off the drum. It also is used for the splice table and to maintain good contact between the preheater pad in the fuser and

the paper. The operator loads a fresh kilogram of toner into the machine about once every eight hours of printing. Unused toner is collected with the vacuum system, trapped centrifugally and deposited in a disposable bottle which is replaced every couple of days. A new box of paper is loaded every hour. The new box can be conveniently spliced onto the end of the previous box or the new box can be easily loaded with the THREAD button.

There are two microprocessors in the HP 2680A. One is a 16 bit HP proprietary SOS device which controls all machine functions such as the operator keyboard and alphanumeric display, the paper path, the closed loop systems and internal diagnostics. The second processor is a high speed bi-polar bit slice processor which communicates with the 3000 and performs all processing on the data stream and ultimately modulating the laser beam to form the correct images at the proper place on the drum. This processor uses 256k bytes of RAM, with a second 256K available as an option. Approximately 40K bytes of this memory is used for tables and buffers, the remaining memory is partitioned dynamically during each job for character sets, forms, and page buffering.

Extensive internal diagnostics constantly monitor the state of the machine, alerting the operator if a service engineer should be called. When arriving on site the service engineer can use additional internally contained diagnostics to troubleshoot any problems. A very complete self test program is available which prints many important parameters on the machine itself. Data such as serial number, drum rotations since last PM, firmware datecodes, and all operating values are labeled and printed. The printer contains a limited amount of nonvolatile memory.

Programming Features

Page printers, even with their inherent benefits of high throughput, low noise and exceptional print quality are rarely viable as simple print and space devices because of their higher cost. However the HP 2680A is a cost effective replacement for many line printers. This is because of the flexibility and features of the printer. Electronic forms allows the inventory of costly specialty forms to be eliminated. Long lead times and form modification costs are reduced to a few hours on a terminal. Definable character sets allows the printer to be used in a wide variety of industries and applications where conventional printers are useless. In addition the print quality and crispness in conjunction with the 8.5 by 11 inch paper size means HP2680A output never needs to be copied or reduced before general distribution.

The HP2680A implements a concept called logical pages. A physical page is a sheet of paper bounded by perforations. A physical page can be divided into up to 32 rectangular areas called logical pages. Logical pages can overlap. A programmatic command to page eject moves the print to the next logical page. If all logical pages have been used, the printer goes to the first logical page on the next sheet of paper. Each logical page has several attributes such as an associated vertical format control (VFC) table, a default line spacing, and one of four orientations. In addition each logical page can have up to two forms

associated with it. When the logical page is printed the forms are automatically overlaid by the printer. Several logical pages can share the same form and VFC, the printer will automatically relocate it to the correct origin for each logical page. Logical pages are a powerful concept which particularly supports existing programs. By defining the logical page format an existing job can have its output reduced two to 1 or four to 1 or rotated without even recompiling the job. Additionally a job which currently uses preprinted forms can be switched to run on the laser printer without modification. The existing form is converted to electronic format and then the corresponding logical page is defined to use the form. The job is then printed on the laser printer and the data is merged with the form and printed.

The electronic forms capability is designed for maximum flexibility. Each form can contain horizontal and vertical lines of varying thicknesses, text written with any number of fonts in any of the four basic directions, plus areas or boxes of variable shading. Form elements can be positioned anywhere and are not restricted to certain character positions on the page as a "draw set" is. The printer can support 32 different forms simultaneously. Each logical page can use up to 2 forms as long as the total does not exceed 30. Additionally each physical page can be overlaid with up to 2 forms. Enough memory and processing power exists to create a form which is a dot per bit image of an 8.5 by 11 inch sheet of paper. Forms are easily created for the printer using an interactive program called IDIFORM.

The HP2680A printer accepts user defined character sets. Each character set contains from 1 to 128 characters. Each character has an associated cell of a specified size which contains any dot per bit representation desired. The spacing between characters and between lines can be set to any value. A character set can print in any of the four directions. Proportional character sets are supported. In this case each character has a parameter describing how far to move over after printing each character. The printer also allows the cells to be printed in any relationship to the current "pen" position. This allows centered symbols, or common base lines so different character sets can be mixed properly on a single line. When using more than one character set a primary and secondary set are defined and then selected with either shift in, shift out control codes or by setting the eighth bit of the ASCII code. HP supplies a large number of character sets of various fonts and sizes. In addition character sets and logos can be created interactively by terminal users via IDSCHAR.

Thirty two user definable VFC's are supported by the printer simultaneously. They are easily created with the IFS2680 program.

One additional feature was implemented to allow easy emulation of multi-part forms. When activated each physical page of data will be repeated up to eight times by the printer. As each copy of the page is printed, the printer will automatically overlay any two forms on the page. In this manner the same data can be repeated up to eight times, but each copy can be individually addressed to shipping, purchasing, order processing, etc.

These basic data structures provide a wide range of user features. When combined with the ability to place cells anywhere on the page and overlap at will plus the processing power to handle over 20,000 characters on an 8.5 by 11 inch sheet a truly unique printer results. The maximum number of cells on any raster scan is 255. As the cells get larger, fewer can be printed simultaneously. Character set switching, forms overlay and other features all occur at speed.

The printer's memory is allocated by a memory manager on a job by job basis. Approximately 40K bytes are used by the printer, the remaining memory is allocated to character sets, forms, VFC's and page buffering. As much memory as required is allocated to the user's character sets, forms and VFC's. All remaining memory is used to buffer pages in an intermediate linked list structure. More page buffering insures that pages are printed at speed. Insufficient page buffering causes a lower thruput rate. The programmer can add or delete character sets, forms and VFC's during the job.

Environment Files

All character sets, forms, VFC's and the logical page table and the multicopy forms table are placed in an environment file by a terminal user running IFS2680. This file is then sent to the printer at the start of a job automatically. This allows the output of a job to change appearance by changing the environment file or portions of the environment file. For example if the character set in an environment file is changed from elite to pica the next job to use the file will have output printed in pica. By simply changing the logical page description and substituting a smaller character set a job can be made to print in a 2 to 1 or 4 to 1 reduction mode. HP supplies several standard environment files to cover portrait mode pica and elite, landscape 132 column printer emulation, two to one and four to one reduction. The user can easily create additional environment files.

For new application programs the full power of the printer is available through HP supplied intrinsics. The intrinsics allow features such as writing a string to a named field on an electronic form. The form can be redesigned and rearranged without modification of any programs using the form. The data will automatically be placed in the correct field wherever it is on the page. Intrinsics also allow the pen to be moved, new primary and secondary character sets to be selected, any logical page to be turned on or off and other similar features.

System Software

Extensive system application software allows creation of character sets, forms, VFC's as well as the definition of logical pages and multicopy forms tables.

IDSCHAR provides menu driven interactive creation of character sets on graphic terminals. The program can emulate various shaped dots and grid spacings. The laser printer has round dots about 8 mils in diameter on a 5.5 mil grid. IDSCHAR also supports a digitizer to allow

easy input of character or logo outlines. The outline can then be scaled and presented superimposed on the cells grid for easy filling in. IDSCHAR supports lines, arcs, rectangular area fills plus scaling and rotation. Special logo files are supported for use on forms. An experienced graphics designer can create a complex logo in 1 to 4 hours. Generating a high quality character set takes about 40 hours.

IDSFORM provides menu driven interactive forms creation of forms on graphics terminals. It supports horizontal and vertical lines of 3 different thicknesses. Boxes can be shaded from clear to black. IDSFORM supports subforms which can be defined and then easily moved around both on the page and between different forms. Windows describe boxes consisting of headers and data fields. Data fields can be labelled to allow symbolic access allowing the form to be changed around without modifying the program. The 1040 tax form was perfectly emulated in 14 hours by an experienced user of IDSFORM.

IFS2680 is the formatting program which bundles up different character sets, forms, VFC's and a logical page table into an environment file. IFS2680 also is the program which constructs VFC's and the logical page table for the user. Overall job parameters such as the number of copies of each page desired and the multicopy forms table are specified via IFS2680. HP supplied standard environments are available from IFS2680 either as they stand or as a base to begin creating a unique environment for a special job.

A contributed program called TR2680 which interprets commands imbedded in ASCII files is available. Text editors can be used to prepare memos and reports with the imbedded commands to utilize HP2680A features such as multiple character sets, forms overlay, pen moves etc.

Once an environment file is created it is specified with a new option in the file equation :FILE PRINT;DEV=PP;ENV=FOURTO1. The environment file is automatically placed in the spool file before the data. This allows existing programs to use all of the features accessible via environment files without modification.

Power fail and jam recovery are very simple and reliable. Non volatile memory exists in the printer. When power resumes the 3000 retransmits the job from the beginning at high speed. The printer processes the data and resumes printing at the correct point in the job. The only operator invention required is to insure top of form is correctly aligned and push run. Paper jams are similar. If no paper was damaged the job can be resumed without system intervention. If the operator wishes to backup several pages the spooler is suspended, the jam cleared, and the command :RESUMESPOOL LDEV#; BACK 5 PAGES is used. This allows backing up or skipping forward an arbitrary number of pages.

Another unique concept introduced with the laser printer is the error trailer. When a program executes an illegal function such as selecting a missing character set, moving the pen off of the logical page or trying to print a character off of the logical page the printer relays this information to the 3000. This information is then printed out at

the end of the job before the trailer is printed. The error trailer describes the error in english, along with the record number and actual page number where the error occurred.

Distributed Printing

MRJE has been modified to support HP2680 environment files. If the device class is PP for page printer and the forms field is not empty then the forms identifier is used to locate an environment file.

RJE has an option which allows the translator procedure to process each record when received by the 3000. This allows complete access to the printers features from a mainframe.

One internal test site is running a Series 30 to front end the laser printer. They are printing over 130,000 pages per month. One half of the output is generated by an Amdahl 470 and sent at 9600 baud via MRJE.

At 2900 lpm the printer taxes the performance of most data communication systems. System configuration, CPU overhead and data format determine the printer utilization. The range can be from 10% to 100%. We are currently quantifying printer performance in these areas and welcome user inputs and insights.

Summary

The HP2680A laser printing system provides a cost effective solution to many computer output problems for HP3000 users. The reliability and servicability contribute to its low cost of less than 4 cents per page at 200K pages per month. The unmatched features provide capabilities unique in the industry. The complete software application package allows immediate turnkey solutions with no programming. The impact of the laser printer in the distributed network is significant and allows non HP systems to utilize the printer as well as enhancing distributed HP systems.

Björn Dreher
Institut für Kernphysik der Universität
D-6500 Mainz, West-Germany

Björn Dreher

Institut für Kernphysik der Universität

D 6500 Mainz, West-Germany

1. Introduction

RATFOR is a language introduced by Kernighan and Plauger [1] based on FORTRAN-66. In their book "Software Tools" they present a preprocessor that translates RATFOR into standard FORTRAN-IV.

In this paper I will first show why RATFOR is a very useful addition to other common languages and what we are using it for in Nuclear Physics Research and System Programming.

In chapter 3 the RATFOR syntax will be shortly described and some examples will be given.

In the forth chapter I will present our implementation of a RATFOR preprocessor and how to use it on an HP3000 system.

Conclusions about our experience with RATFOR will be drawn in chapter 5.

2. Why use RATFOR as an additional language

The primary reason for the authors of RATFOR was to make FORTRAN a better programming language. With RATFOR it is possible to write much more readable and better structured programs. This is achieved by providing additional control structures that are not available in FORTRAN-66, and by improving the "cosmetics" of the language.

The added control structures for better structuring of programs are IF-ELSE, WHILE-DO, REPEAT-UNTIL, FOR loops, DO loops, and others. An INCLUDE statement allows the inclusion of predefined code or definition sequences at certain points. The cosmetics is improved by allowing the programs to be in free-form. The end of the line marks usually

the end of the statement, but statements can easily be continued on the next line by ending with a comma or with a special continuation character. A sharp # anywhere in the line marks the beginning of a comment, thus allowing trailing comments on each line. This certainly encourages programmers to add more documentation to the source code.

Because almost all constructs of standard FORTRAN are retained in RATFOR, it is very easy for a FORTRAN programmer to learn RATFOR. There is only a very low psychological barrier to switch from FORTRAN to RATFOR.

In addition, you are not lost if you have to transfer one of your RATFOR programs to an other installation that has no RATFOR compiler available. You simply move the intermediate FORTRAN code to the other system. This is of course the way, how the RATFOR preprocessor itself is "boot-strapped" on a new machine.

In addition to the already mentioned features, RATFOR comes with a built-in macro processor, which allows not only such constructs like EQUATES and DEFINES as in SPL, but also enables you to add additional language constructs (in the form of macros) to RATFOR as you need it.

From all this you see that RATFOR is a better choice than FORTRAN in at least all those cases, where you have somewhat more complicated control paths in a program. There are only few instances where GOTO constructs are needed, and avoiding those makes programs usually more readable and better self-documenting.

Besides applications in Nuclear Physics, we are using RATFOR to implement data acquisition and measurement control subsystems as well as computer communications systems. RATFOR helps us to write these systems to a large extent in a machine independent way, burying machine dependencies in macro definitions. We are currently using three type of minicomputers in our institute: HP3000, HP1000, and PE3220.

3. RATFOR syntax

3.1 General rules

There are several characters recognized as special ones in RATFOR:

\$(or { for LEFT BRACE
\$) or } for RIGHT BRACE

Left and right braces act as delimiters for groups of statements like BEGIN and END in SPL. Other special characters are:

^ or ~ for NOT
\ or | for OR
& for AND
[* for MACRO LEFT BRACKET
*] for MACRO RIGHT BRACKET
for the begin of comments
% is the line continuation character

Any line ending in a comma will also be continued. Include files may be nested 3 levels deep. A statement which starts and ends with a quote will be stripped of the quotes and placed in column one in the output. This is useful for 'hiding' Ratfor keywords and for putting FORTRAN compiler commands (\$CONTROL ...) in column 1.

Examples:

```
'$CONTROL USLINIT,NOSOURCE'  
or  
IF(arith expr) label1,label2,label3'
```

(Note: Arithmetic IF statements are not allowed in RATFOR).

Input is free-field with only few exceptions. Capitals and small letters can be used. Embedded comments in a source line start with "#" or "!".

Blocks are one single statement or several surrounded by braces. This is similar to the BEGIN/END structure in ALGOL or SPL. The left brace "{" corresponds to BEGIN, the right brace "}" to END.

3.2 The DO statement

It resembles the well known FORTRAN DO-statement without the need to use a label at the final statement.

```
DO I=1,MAX,DELTA
  A(I)=I
or
DO I=1,MAX
  {
  A(I)=SIN(X(I))
  B(I)=A(I)**2
  }
```

3.3 The FOR statement

```
FOR (I=1 ; I<=100 ; I=I+1)
  <BLOCK>
```

<BLOCK> stands for one statement or (several statements). The three parts between the parentheses have the following meanings:

- 1: (I=1) Initialization statement. This may be omitted, thus starting with a previously defined value.
- 2: (I<=100) As long as this condition holds true, the following block will be executed. This is tested at the beginning of the block.
- 3: (I=I+1) Modification, that is performed at the end of the block.

All three clauses may be almost arbitrarily complicated, as the following example shows:

```
FOR (X=0 ; EXP(X)<=1.E70 ; X=ARCSIN(X)+10./Y)
  {
  PRINT X
  }
```

3.4 The WHILE statement

```
WHILE ( X(I) ^= 5 )
  {
  DISPLAY X(I)
  X(I)=FUNCT(X(I))
  }
```

This allows a block of statements to be repeated while a certain condition holds true, which is tested at the beginning of each step.

3.5 The REPEAT statement

This is the counterpart to the WHILE statement. A block of statements is continued until a certain condition, which is tested at the end of the block, becomes true:

```
REPEAT
  <BLOCK>
UNTIL (X==Y)
```

One may omit the UNTIL-clause to get a REPEAT FOREVER construct.

3.6 Exits

The two statements NEXT and BREAK allow to change the sequence of execution in DO, FOR, WHILE and REPEAT blocks without the need for GOTO statements (which is considered as bad programming style!) and labels.

NEXT starts over at the beginning of the currently executing block (i.e. starts again at the first statement of the DO or FOR block after the appropriate modification of the running index - or whatever was requested - has been done: corresponds to a GOTO to the CONTINUE statement of a FORTRAN DO loop)

BREAK continues behind the current block. The DO, FOR, WHILE, or REPEAT statement is terminated.

3.7 Relational expressions

The following is a table of the correspondence between FORTRAN and RATFOR relational and logical operators:

FORTRAN	RATFOR
.EQ.	==
.NE.	^=
.GT.	>
.GE.	>=
.LT.	<
.LE.	<=
.AND.	&
.OR.	
.NOT.	^

3.8 IF and ELSE clauses

This is similar as in ALGOL or SPL and many other languages:

```
IF (logical expression)
  <block>
or
IF (logical expression)
  <block>
ELSE
  <block>
```

3.9 The INCLUDE statement

The INCLUDE statements allows the inclusion of program parts, which are stored on a different file, at the point of the INCLUDE statement. INCLUDEs may be nested 3 levels deep.

3.10 The Ratfor Macro

Ratfor contains a macro processor. It is useful for simple character string replacements, string replacements with parameters, as well as for powerful extensions of the Ratfor syntax. Macros are defined with the DEFINE statement. In the following we give a few examples of simple macro definitions.

```
define(pi,3.141593)
```

Following this macro definition, every occurrence of pi (or PI) in the text will lead to the insertion of 3.141593 instead of the two letters.

```
define(maxind,200)
...
integer iarray(maxind),rarray(maxind,2)
...
do i = 1, maxind
  iarray(i) = 0
```

This is useful to define dimensions and maximum index values globally.

```
define(tan,['sin($1)/cos($1)*'])
```

This is a macro with parameters. tan(x/2) will be replaced by sin(x/2)/cos(x/2). With the macro definition, you can write the program as if "tan" were a function, but there are no function calls at run-time. For complicated expressions, however, the object code will be quite long when you call the macro often.

Macros can be globally defined for a complete source file. It is best to make the definitions at the beginning of the file, maybe with an include statement for a file containing the macros.

The following example illustrates how powerful the RATFOR macro processor can be, if you have understood its operation and syntax in detail. For instance, it is possible to write easy to use constructs for condition code checking after the call of system intrinsics:

```
IFN=FOPEN( ... )
  BEGINCC
    CCE
    << block >>
    CCG
    << block >>
    CCL
    << block >>
  ENDCC
```

There is no need to use all three conditions (CCE,CCG,CCL). If one is omitted, control continues for that case after the ENDCC. The sequence of CCE,CCG and CCL may be chosen arbitrarily.

4. Our RATFOR/3000 implementation

Our RATFOR implementation was derived from a RATFOR preprocessor originally written for the HP1000 family of computers. Therefore it is capable to produce output for the HP1000 FTN-IV compiler as well as for FORTRAN/3000.

RATFOR/3000 may be invoked by the following UDC:

For FORTRAN/3000:

```
RATFOR <in>,<out>,<list>,<opts>,<incl>,<ftnlist>
```

For FTN4/1000:

```
RAT4 <in>,<out>,<list>,<opts>,<incl>,<ftnlist>
```

the Ratfor program will be read from <in> (default \$NULL)

the intermediate Fortran code will be written to <out>
(the default is a SESSION temporary file RATTEMP)

the source listing will be written to <list> (default \$STDLIST)

options are specified by <opts> (default %17 or %13)

The options are given as an integer constant (octal or decimal).

If the most significant bit is number 0 and the least significant is number 15, the bits have the following meanings:

15 list the source, otherwise only errors are listed.
bit 15=0 is automatically set, if <list> = \$NULL;
errors are then output to \$STDLIST.

14 for future enhancements

13 =1 FORTRAN/3000 code
=0 FTN4/1000 code
automatically set by the two UDC's

12 merge all RATFOR (and other) comments into the generated
FORTRAN program

the file <incl> will be included in from of <in> (default \$NULL)

the FORTRAN compiler listing will go to <ftnlist> (default \$STDLIST)

4.1 Limitations

Due to the fact that this version of RATFOR is an adaptation from the HP1000 version there were some features in RATFOR/1000 that did not conform with FORTRAN/3000 syntax. Therefore, if in HP3000 mode some original RATFOR features are switched off. In particular, in HP3000 mode the following applies:

1. CHARACTER declarations are passed as they are, because FORTRAN/3000 supports type CHARACTER variables.
2. Character strings between quotes, e.g. "ABCDEF", are kept as they are. In non-HP3000 mode this is converted to SHABCDEF.

To allow for the use of substring designators, e.g. I[3:5], in both modes brackets are not recognized as delimiters of blocks as they were in the original version. Use braces "{}" instead!

RATFOR does NOT understand FORTRAN arithmetic IF statements. If you have to use them, e.g. to check the condition code after returning from a system intrinsic, you have to put the statement between quotes. (There is a special RATFOR macro available to check condition codes)

For FORTRAN/3000 applications it is good practice to use the following compiler command:

```
SCONTROL USLINIT,NOSOURCE
```

If you then use the default setting for the FORTRAN/3000 output (\$STDLIST) you will not get the awkward FORTRAN listing, but instead all (if at all) error messages with the line in error on your terminal. The FORTRAN line numbers are derived from the original RATFOR source line numbers, with an increment of .001 if there are more than one FORTRAN lines generated from one RATFOR line. Therefore it should be easy to find the RATFOR line, which is in error.

5. Conclusion

In conclusion, we found the RATFOR preprocessor a very valuable programming tool, especially since a FORTRAN-77 version for the HP3000 seems to be still far away. Although FORTRAN-77 adds some of RATFOR's control structures, we find the cosmetics, the appearance of the program text, of RATFOR much more appealing.



Now, what is the pay-off? Certainly compilation time is increased. In the current version, the RATFOR compiler needs about the same CPU time to transform RATFOR to FORTRAN as the FORTRAN compiler needs for its job. This can be somewhat improved in the future by sampling the most frequently used parts of the preprocessor and improving on these pieces of code.

In addition you have to be aware, that RATFOR/3000 checks only RATFOR syntax, most of the FORTRAN statements go unchecked to the FORTRAN compiler. FORTRAN/3000 will then give you the errors. Since the line numbers of the intermediate FORTRAN code are derived from the original RATFOR line numbers, it is very easy to track an error reported by the FORTRAN compiler back to the original RATFOR source line.

Regarding run-time performance, we did not find any significant difference between a RATFOR program and a corresponding version written directly in FORTRAN.

Of course, a globally optimizing FORTRAN compiler, which we are all waiting for, would improve the run-time behaviour of RATFOR programs as well as FORTRAN programs.

[1] B.-W. Kernighan, P.J. Plauger: Software Tools, Addison-Wesley Publishing Co. 1976

BUDGETING AND PROFIT PLANNING ON THE HP3000

BY JACK DAMM, PRINCIPAL, THE PALO ALTO GROUP, CUPERTINO, CALIF
(408) 725-1282

Good morning. Today I'm going to talk about budgeting and profit planning on the HP3000. This discussion will be organized around a specific example, but will not be limited to it. This is a very subjective discussion. There is certainly plenty of room for conflicting opinions. And there are many ways of putting together plans which are different from the example being presented here.

How many of you in the audience have the primary responsibility for budgeting or profit planning in your company?
How many of you are in your company's data processing department?
How many of you get no closer to profit planning than putting together your own budgets?

I hope that when this discussion is finished, that as a "D.P. person" you get a little better idea of what overall company planning is, and where your own budgeting and forecasting fits into it. As a "financial planner", I hope that you will leave with a better understanding of the the HP3000 can contribute to your own company's planning.

My discussion will proceed as follows: First, some generalities about planning. The use of the HP3000. A brief comparison of high-level financial planning languages versus manual or programmed models. Then I will discuss a typical plan.

First, a definition.
When I talk about planning, I will be speaking specifically about company (or corporate) planning. Product forecasting, departmental budgeting, and financial projections. Deciding where funds are to be spent, and analysing the impact of these expenditures.

There is a myth: That one can build a "model" of a company which can be mathematically manipulated to make "optimal" use of resources and maximize achievement of company goals. That this model needs to be complicated, and is incomprehensible to mere mortals. It is true that one could formulate a "model" of a company where the goal is to maximize profit or cash flow, with constraints involving debt-to-equity ratios, market penetration, production capacity, etc. But because of how little we actually know about the future, and how much uncertainty there is about it, to build a sophisticated model which "optimizes" the results of a business, is ridiculous. It would be fooling ourselves about how little we actually know.

There is reality: What planning really is, is sitting down and accepting that there is much uncertainty about the future. But at the same time, in order for us to be somewhat prepared for that uncertain future we must make assumptions. Best guesses about what may happen. One of the most important

BUDGETING AND PROFIT PLANNING ON THE HP3000

A presentation on financial planning given at the HP3000 International Users Group meeting in Berlin, Germany, on October 6, 1981, by Mr. Jack Damm of The Palo Alto Group.

results of the planning process is to get managers together and come to a general agreement as to how company goals are to be achieved. In a nutshell, planning is sitting down with marketing and production planning people and getting a "best guess" product forecast. Getting responsible managers to commit to realistic levels of spending within their departments. Create a plan of action which represents a generally agreed-upon approach to the direction of the business.

While it may be uncertain, a good plan at least makes it possible to rule out the "unlikely" or "infeasible" situation which might result from "seat-of-the-pants" guesses:

- Growing too fast to be supported by available funds
- Spending too much money to have any left over
- Selling more product than can be produced
- Producing more product than can be sold

So, I am going to talk about reality. Putting together a plan, what it means, and how the HP3000 can be used as an effective tool in getting the job done.

THE TOOLS: An HP3000 computer system, available for both batch and interactive use. Most of our customers use in-house (rather than dial-up) systems. They use both on-line CRT's and printing terminals, and most have access to a near-by line printer. The system is almost always accessible to the responsible manager, although some companies distribute only the reports and not the computer interaction as well. The system is used interactively for setting up reports and reviewing the results on a particular report. Reports are run in the batch mode for making multiple copies, generating a whole sequence of reports at once, and sometimes just for hard-copy output.

A high-level planning language. We work with our proprietary financial planning language, Dollar-Flow, and it will be used to illustrate the examples here. We choose to work in a "high-level language" like Dollar-Flow because:

- Planning by hand (and calculator), despite the flexibility it provides, takes too much time, involves too many opportunities for error, and is particularly undesirable when doing many iterations of a plan.
- Planning on a computer using a "procedure level" language like BASIC, FORTRAN, or COBOL takes too long to set up, is inflexible, and requires the services of a programmer.

Dollar-Flow, because it is designed for planning, enables us to focus on the problem itself, without paying particular regard to all the details of what is actually going on inside the computer.

THE PLAN: A typical financial plan might involve the following modules:

- A sales forecast
- Budgets and budget consolidations
- A profit/loss projection

- A cash flow projection
- A balance sheet projection

Most of the plans which we get involved with work on a three to five year horizon with (of course) particular attention being paid to the first year. For the first two years the plan usually is done by month, whereas the third through fifth year projections might be by quarter, by half, or on an annual basis.

SALES FORECAST: Preparing the sales forecast is the first step in profit planning. Today I am not going to go into the any techniques which can be used for forecasting. Rather, I am going to concentrate on the techniques we use for working with forecast figures which have been decided upon (in one way or another) by those responsible for the forecast. The sales forecast is the most important as well as the most uncertain part of the planning process. Since sales determine how much money is available for budgets, etc., the forecast is prepared first. And because of the uncertainty of our projections, we may re-run our plan several times with varying levels of sales as a "what-if" analysis. So we can do contingency planning.

A forecast for a product-oriented company is usually done first on a "bottoms-up" basis. That is to say, on a product-by-product basis. If we are building a plan with a three or five year horizon, the third through fifth years may only be overall sales dollars estimates. In the near term, however, the sales forecast is done by month for each product. In addition to the unit sales forecast, we need an average selling price on each product (which may vary over time), which we will show as one value here. And to enable us to generate figures for the cost of sales, we combine this data with estimates of direct labor per unit, direct material per unit, and manufacturing overhead (which may or may not be on a per-unit basis). The sales revenue and cost of sales forecast then is simply a product of the unit sales projections and the per unit price and cost factors. A typical product forecast might look like the following:

12/ 4/79

XYZ COMPANY
SALES FORECAST REPORT

PAGE 1

	AVG SELLING PRICE	DIRECT LABOR/ UNIT	DIRECT MATL/ UNIT	MFG OVHD/ UNIT	
1 WIDGETS.....	900	85	200	255	
2 GIZMOS.....	1,300	155	275	465	
3 THINGS.....	1,950	175	375	525	
4 NON-THINGS.....	2,100	575	525	1,725	
5 ZITHERS.....	2,450	560	450	1,680	
	JAN UNITS	JAN REV	JAN DL	JAN DM	JAN OVHD
1 WIDGETS.....	100	90,000	8,500	20,000	25,500
2 GIZMOS.....	50	65,000	7,750	13,750	23,250
3 THINGS.....	-	-	-	-	-
4 NON-THINGS.....	25	52,500	14,375	13,125	43,125

5 ZITHERS.....	-	-	-	-	-
6 TOTAL ALL PRODUCTS.	175	207,500	30,625	46,875	91,875

Now, a top-down analysis may be done on the total sales figures in each period (or by year) and feedback on the unit forecasts if they are too high (or low).

In businesses which build product to inventory (where sales and production levels may be very different in a given period), a manufacturing plan is required in addition to the sales forecast. That is to say, once the sales forecast has been prepared, then the production planners must sit down and determine a production plan which will meet an objective level of delivery of product and reasonable levels of inventory, given the sales forecast. The production plan would be similar to the unit sales forecast - it is simply the units to be produced by period. The result of the production plan is actual outlays for direct material and direct labor (which go into inventory). Combined with the cost of sales (what comes out of inventory) this gives us the change in inventories. A typical production plan might look like the following:

12/23/79		XYZ COMPANY PRODUCTION PLAN					PAGE 1
	JAN	FEB	MAR	APR	MAY		
6 TOTAL SALES.....	207,500	207,500	207,500	467,500	461,000		
7 TOTAL COGS LABOR.....	30,625	30,625	30,625	53,000	52,225		
8 TOTAL COGS MATL.....	46,875	46,875	46,875	96,250	94,875		
9 TOTAL COGS OVHD.....	91,875	91,875	91,875	159,000	156,675		
10 TOTAL COGS.....	169,375	169,375	169,375	308,250	303,775		
11 WIDGET UNIT PRODUCTION..	100	100	100	100	100		
12 GIZMO UNIT PRODUCTION...	50	50	25	20	10		
13 THING UNIT PRODUCTION...	-	50	100	150	150		
14 NON-THING UNIT PRODUCTION.....	25	25	25	25	25		
15 ZITHER UNIT PRODUCTION..	-	-	-	-	10		
16 TOTAL PRODUCTION (AT ASP).....	207,500	305,000	370,000	461,000	472,500		
17 TOTAL PROD LABOR.....	30,625	39,375	44,250	52,225	56,275		
18 TOTAL PROD MATL.....	46,875	65,625	77,500	94,875	96,625		
19 TOTAL PROD OVHD.....	91,875	118,125	132,750	156,675	168,825		
20 TOTAL PROD LABOR, MATL, OVHD....	169,375	223,125	254,500	303,775	321,725		
21 PROD MATL PURCHASES.....	77,500	94,875	96,625	131,625	130,250		
22 INITIAL INVENTORY.....	125,000						
23 BEGINNING INVENTORY.....	125,000	155,625	238,625	342,875	375,150		
24 INVENTORY CHANGE.....	30,625	83,000	104,250	32,275	51,575		
25 ENDING INVENTORY.....	155,625	238,625	342,875	375,150	426,725		
* END OF \$FLOW\$ REPORT *							

Now, we are not finished with the forecast. One is never finished with the forecast. But with it set up in our planning system as we are doing here, when the forecast figures change, we can re-run our model and get a revised company plan with a minimum of effort.

The next step is budgeting (which may overlap with the forecasting step). The forecast is important to the budgeting step because it tells us how much is available to be spent in the budget. In other words, the budgets may need to be revised as the forecast changes.

- A few comments about the typical budget environment.
 - There is usually a budgeting hierarchy, where there are several levels of consolidation. Budgets may be grouped at sub-department, department, product line, division, or many other levels.
 - In addition to the consolidation hierarchy, there may be service departments (or locations) which charge some costs out to other departments (data processing for example).

A typical budgeting hierarchy might have the following levels:

- Overall company budgeted expenditures
- Divisional consolidated budgets
- Product line consolidated budgets
- Departmental budgets
- Location budgets (within a department)
- Service location budgets

Now, the budget reports we usually set up identify each budgetary item by general ledger account number and description. For most of the items on the budget, the system prompts for a value to be input for each period on the budget (typically the periods are months) and allows for revisions to be made by row and/or column. Typical input lines are:

- 1001 WAGES & SALARIES
- 3001 SUPPLIES
- 5002 COMPANY AUTO
- 5003 ENTERTAINMENT

Some budget items are functions of other items. For example, fringe benefits may be set up as 20% of wages and salaries:

$$1002 \text{ FRINGE BENEFITS} = .20 \times '1001 \text{ WAGES \& SALARIES}';$$

The total budget may be calculated using the 'SUM' function of Dollar-Flow:

$$\text{TOTAL BUDGET} = \text{SUM}('1001 \text{ WAGES \& SALARIES}', '9999 \text{ MISC EXPENSE}');$$

And some budget items may use figures allocated from other budget locations:

3005 EDP EXPENSE =
 '% EDP ALLOCATION' X 'TOTAL BUDGET' OF 'D9000'/100;

A typical budget plan might look like:

12/23/79	MARKETING BUDGET FY 1980					PAGE 1
	JAN	FEB	MAR	APR	MAY	
1 1001 WAGES & SALARIES	5000	5000	5000	10000	10000	
2 1002 FRINGE BENEFITS	1000	1000	1000	2000	2000	
3 1003 PAID VACATION	100	100	100	200	200	
4 1004 SICK PAY	150	150	150	300	300	
5 3001 SUPPLIES	500	500	500	500	500	
6 3005 EDP EXPENSES	800	800	800	1200	1200	
7 5001 TRANSPORTATION FARES	2000	2000	2000	2000	2000	
8 5002 COMPANY AUTO	500	500	500	500	500	
9 5003 ENTERTAINMENT	500	500	500	500	500	
10 8001 ADVERTISING	1000	1000	5000	2000	2000	
11 8002 PROMO LITERATURE	2500	2500	2500	2500	2500	
12 8003 TRADE SHOWS	10000	-	-	20000	-	
13 9999 MISC EXPENSE	500	500	500	500	500	
14 TOTAL BUDGET	24550	14550	18550	42200	22200	
15 % FRINGE	20.0 %					
16 % EDP ALLOCATION	8.0 %					

* END OF \$FLOW\$ REPORT *

By having allocated expenses defined within our planning system as automatically flowing into each of the using locations, it is easy for us to keep all of the budgets consistent. Whenever the budget for a service department (such as EDP) is revised, the new figures flow into the departments using their services the next time those budgets are run.

One more comment about the system we are using. Within Dollar-Flow, information is organized on a report basis. That is to say, users work with one report at a time and do not have to concern themselves with the distinctions between reports, files, programs, and data. And reports are saved on the HP3000 under 8 character MPE file names so they referenced by other reports and/or re-run at some later time.

We usually organize the budget reports according to the customer's own scheme for identifying budget locations, by saving these reports with meaningful names. For example, if the EDP department has a location code number of 9000, then we might save the budget for that department under the name 'D9000'. Sub-departments locations might be coded 'D9001', 'D9002', etc. A budget which is a consolidation of other departments might be stored with a descriptive name like 'D900' or 'D900X'. With a numbering scheme which groups various budgets into group names which can be accessed using @, #, and ? symbols (according to the MPE file referencing convention), indirect file references can be used to set up the structure of a budget hierarchy. In other words, we

could define a consolidation report as summing 'D900#' which would automatically add up 'D9001' through 'D9009'. Budget consolidation reports usually have the same format as the lower-level reports they reference.

A few comments about the budgeting process. In our experience we have seen great diversity in how companies do their planning. Some companies centralize budget preparation. Distributing budget worksheets, then inputting the data and making revisions in one area. Other companies distribute the entire process, letting managers interact directly with the budget system, only processing the data when it is ready for consolidation. Some companies budget in great detail. Others take the "big picture" approach, working at higher levels of consolidation. The one thing that all of the companies which we have worked with have in common, is that the budgeting process involves many changes and several iterations. With several levels of consolidation, and allocations from several service departments, budgeting would be a massive job if it were done by hand. And a massive headache if it were done with the traditional programming approach in BASIC, FORTRAN, or COBOL.

Once the budgets have been prepared, we are ready to proceed with the profit/loss projection. With the information provided by the sales forecast and manufacturing plan, plus the departmental budgets, the P & L is little more than a recap of existing information. This is particularly easy in the system which we are using because we can 'define' relationships on our P & L report which will cause data to be read automatically from reports which have already been prepared. For example, we could read information from the manufacturing plan with rules like the following:

DIRECT LABOR = 'TOTAL COGS LABOR' OF 'MFGPLN' ;
 DIRECT MATL = 'TOTAL COGS MATL' OF 'MFGPLN' ;

where 'TOTAL COGS LABOR' and 'TOTAL COGS MATL' are lines on the production plan which has been set up and saved under the name 'MFGPLN'.

We can even reference several reports for a one-line consolidation:

MARKETING = 'TOTAL BUDGET' OF ('D2001','D2002',...,'D2009');

Of course the one-line consolidation shown here would only be necessary if we didn't have a consolidation report already set up for total marketing.

A few of the lines involve calculations with other lines on the P & L report itself:

TOTAL DIRECT COST = 'DIRECT LABOR' + 'DIRECT MATL' + 'MFG OVHD' ;
 GROSS MARGIN = 'SALES' - 'TOTAL DIRECT COST' ;

And one line, 'INTEREST EXPENSE', is a forward reference to two other lines yet to be defined:

INTEREST EXPENSE = '% INTEREST' X 'OUTSTANDING DEBT' / 1200 ;

Now, the P & L projection (combined with the cash flow) is

usually run as soon as the first pass forecast and budget reports are ready. The budgets and the forecast are then red-lined, and sent back for revision. The revisions may involve changes in budgeted expenditures, pricing (and volume) of products, or even timing of product introduction and related expenditures. The running of the P & L and the cash flow is then the "top-down" step in planning, where the forecasts and budgets are revised based on overall criteria.

A typical P & L projection might look like the following:

12/23/79		XYZ COMPANY					PAGE 1
PROFIT/LOSS AND CASH FLOW PROJECTION							
	JAN	FEB	MAR	APR	MAY		
1 PROFIT/LOSS PROJECTION:							
2 SALES	207.5	207.5	207.5	467.5	461.0		
3 DIRECT COSTS:							
4 DIRECT LABOR	30.6	30.6	30.6	53.0	52.2		
5 DIRECT MATL	46.9	46.9	46.9	96.3	94.9		
6 MFG OVHD	91.9	91.9	91.9	159.0	156.7		
7 MFG OVHD VARIANCE	(1.9)	(28.1)	(42.8)	13.3	1.2		
8 TOTAL DIRECT COST	167.5	141.3	126.6	321.6	305.0		
9 GROSS MARGIN	40.0	66.3	80.9	145.9	156.1		
10 ENGINEERING	25.0	25.0	25.0	35.0	35.0		
11 MARKETING	24.6	14.6	18.6	42.2	22.2		
12 GEN & ADMIN	25.0	25.0	25.0	25.0	25.0		
13 TOTAL PERIOD EXPENSE	74.6	64.6	68.6	102.2	82.2		
14 OPERATING PROFIT	(34.6)	1.7	12.3	43.7	73.9		
15 INTEREST EXPENSE	.9	2.0	-	-	2.3		
16 NET INCOME BEF TAX	(35.5)	(.3)	12.3	43.7	71.6		
17 TAXES ON INCOME	-	-	-	9.8	34.4		
18 NET INCOME AFTER TAX	(35.5)	(.3)	12.3	34.0	37.2		

For companies where cash flow is important (because they are growing rapidly, because margins are low, or a host of other reasons), the cash flow projection may be the most important part of the planning cycle.

Now, setting up a cash flow projection from the information we already have available requires little more than taking our P & L results and adjusting them for timing in the flow of funds. Let's start this with a very simplified rule of thumb. The difference between profit and cash flow is that:

PROFIT (like your salary) is what you pay taxes on, whereas CASH is what you have in the bank.

And we all know what a big difference that can be! Let's examine two important examples. If a company sends a shipment to a customer along with a bill for the goods, then the amount of the

bill (less the cost of making the goods) is profit. However, all the company who shipped the product has at this point is a thing called "accounts receivable", or money owed to it by the customer which has not yet been paid. This becomes cash only when the customer pays his bill. Conversely, when a company takes delivery of goods on credit from a vendor, the amount owed to the vendor ("accounts payable") only becomes a cash outflow when the bill is paid. In both of these cases, cash will eventually change hands to offset accounts receivable and accounts payable. However, the timing makes a big difference in cash flow. And, keep in mind, running a business at a loss (although undesirable) is not nearly as serious as running out of money (and unable to borrow it somewhere)! If a company is growing fast, it is very easy to run out of cash even though the business is very profitable.

Enough of the generalities. Let's look at what we do to create our cash flow. First, the "aging" of accounts receivable, or reflecting how cash payment of accounts receivable is expected to occur over time. We usually calculate a 3 month moving average of sales, then use an estimate of the number of days of sales that will be in accounts receivable at one time. We set up this number of days sales figure as an easily revised input value so that the plan can be run at varying rates of receivables collection. Combining the average sales with the number of days sales which are in accounts receivable, we project the ending accounts receivable balance for each month. With this information, all we have to do is a calculation with the following rule to determine cash receipts:

RECEIPTS = 'BEGINNING A/R' + 'SALES' - 'ENDING A/R' ;

The report lines appear as follows:

	JAN	FEB	MAR	APR	MAY
21 CASH FLOW PROJECTION:					
22 RECEIVABLES AGING:					
23 PRIOR 2 MOS SALES	180.0	190.0	-	-	-
24 3 MOS AVG SALES	192.5	201.7	207.5	294.2	378.7
25 # DAYS SALES IN A/R	45.0	45.0	45.0	45.0	45.0
26 BEGINNING A/R	300.0	288.8	302.5	311.3	441.3
27 CHANGE IN A/R	(11.3)	13.8	8.8	130.0	126.8
28 ENDING A/R	288.8	302.5	311.3	441.3	568.0
43 RECEIPTS	218.8	193.8	198.8	337.5	334.3

For accounts payable, the process is a little more involved. We take the total budgeted expenditures for all of the operating departments and subtract out the payroll and depreciation expenses from them. We add into this direct material purchases and capital equipment purchases. This gives us the amount of accounts payable we have incurred. Then, we assume that all of our bills are paid in 30 days (or 1 period in the example here). So we just take the prior period's accounts payable to determine our payment of accounts payable in the current period. To handle the first period, we add a figure for accounts payable incurred in the preceding period. This analysis appears as follows:

29 PAYABLES AGING:					
30 BUDGETED EXPENSES	164.6	154.6	158.6	272.2	252.2
31 PAYROLL	50.0	60.0	65.0	75.0	80.0
32 PAYROLL (EXCL D.L.)	19.4	20.6	20.8	22.8	23.7
33 DEPRECIATION	7.0	7.0	7.0	7.0	7.0
34 MATERIAL PURCHASES	77.5	94.9	96.6	131.6	130.3
35 CAPITAL EQUIPMENT PURCHASES	-	-	-	200.0	-
36 ACCTS PAYABLE INCURRED	215.7	221.8	227.4	574.1	351.7
37 PRIOR PERIOD A/P	190.0				
38 ACCTS PAYABLE PAID	190.0	215.7	221.8	227.4	574.1

For interest payments, we assume that they are paid in the periods that they are incurred. (We will discuss the interest calculation in somewhat greater detail in a moment.) And taxes are usually paid based on the results of the prior year. In the example here, we have assumed that no taxes were due in the preceding year.

Now, to calculate the cash flow in each period, all we do is to take the receipts and subtract the total of payroll, accounts payable paid, taxes paid, and interest paid. This gives us our net cash flow from operations in any given period.

42 CASH FLOW:					
=====					
43 RECEIPTS	218.8	193.8	198.8	337.5	334.3
44 DISBURSEMENTS:					
45 PAYROLL PAID	50.0	60.0	65.0	75.0	80.0
46 ACCTS PAYABLE PAID	190.0	215.7	221.8	227.4	574.1
47 TAXES PAID	-	-	-	-	-
48 INTEREST PAID	.9	2.0	-	-	2.3
49 TOTAL DISBURSEMENTS	240.9	277.6	286.8	302.4	656.3
50 NET CASH FLOW	(22.2)	(83.9)	(88.1)	35.1	(322.1)

Next, we consider financing. As shown here, we have constructed a model which takes the beginning balance of cash in a given period and adds into it the changes in cash due to operations. Equity financing is then added into the cash balance. Then, we set up a rule for the line 'BORROWING (REPAYMENT)' which says "If the cash balance is less than zero, borrow enough money to bring the balance back to zero. If the cash balance is greater than zero, then pay off any existing debt with whatever cash is available." We then add this borrowing or repayment into our cash balance and we have ending cash for the period. The financing lines appear as follows:

51 BEGINNING CASH	50.0	-	-	105.9	141.0
52 CHANGES IN CASH	(22.2)	(83.9)	(88.1)	35.1	(322.1)
53 ENDING CASH	27.8	(83.9)	(88.1)	141.0	(181.1)
54 FINANCING	-	-	350.0	-	-
55 BORROWING (REPAYMENT)	(27.8)	83.9	(156.0)	-	181.1
56 ENDING CASH AFTER FINANCING	-	-	105.9	141.0	-
57 BEGINNING DEBT	100.0	72.2	156.0	-	-
58 OUTSTANDING DEBT	72.2	156.0	-	-	181.1
59 % INTEREST PAID	15.0 %	15.0 %	15.0 %	15.0 %	15.0 %

One last comment about the cash flow. The 'INTEREST EXPENSE' line on the P & L statement is calculated using the '% INTEREST

PAID' and 'OUTSTANDING DEBT' lines on the cash flow. The system which we are using actually solves the circular logic involved in calculating interest, cash flow, and debt, so that these values are all mutually consistent.

The last step in our profit plan is the creation of a balance sheet. Similar to the P & L statement, the balance sheet is usually just a recap of the P & L and cash flow projection. I won't say much about the balance sheet, except to mention that we use it as a tool to make sure that our cash flow projections are working correctly. If our assets are equal to our liabilities (as they should be), then we can be pretty sure that we haven't made a mistake in the general logic of the cash flow (of course the values could still be wrong!). The balance sheet might appear as follows:

12/23/79	XYZ COMPANY						PAGE 1
	BALANCE SHEET PROJECTION						
	DEC	JAN	FEB	MAR	APR	MAY	
1 ASSETS:							
2 CASH	50.0	-	-	105.9	141.0	-	
3 ACCTS RECEIVABLE	300.0	288.8	302.5	311.3	441.3	568.0	
4 INVENTORY (NET)	125.0	155.6	238.6	342.9	375.2	426.7	
5 TOTAL CURRENT ASSETS	475.0	444.4	541.1	760.0	957.4	994.7	
6 PROP, PLANT & EQUIP	2255.0	2248.0	2241.0	2234.0	2427.0	2420.0	
7 TOTAL ASSETS	2730.0	2692.4	2782.1	2994.0	3384.4	3414.7	
8 LIABILITIES AND NET WORTH:							
9 ACCTS PAYABLE	190.0	215.7	221.8	227.4	574.1	351.7	
10 TAXES PAYABLE	-	-	-	-	9.8	44.1	
11 S.T. BANK LOAN	100.0	72.2	156.0	-	-	181.1	
12 TOTAL							
CURRENT LIABILITIES	290.0	287.8	377.8	227.4	583.8	576.9	
13 L.T. BANK LOAN	900.0	900.0	900.0	900.0	900.0	900.0	
14 TOTAL LIABILITIES	1190.0	1187.8	1277.8	1127.4	1483.8	1476.9	
15 STOCKHOLDERS EQUITY:							
16 COMMON STOCK	2340.0	2340.0	2340.0	2690.0	2690.0	2690.0	
17 RETAINED EARNINGS	(800.0)	(835.5)	(835.7)	(823.4)	(789.4)	(752.2)	
18 TOTAL							
STOCKHOLDERS EQUITY	1540.0	1504.5	1504.3	1866.6	1900.6	1937.8	
19 TOTAL LIABS & WORTH	2730.0	2692.4	2782.1	2994.0	3384.4	3414.7	
20 ASSETS - LIABS	-	-	-	-	-	-	
* END OF \$FLOW\$ REPORT *							

This completes our profit plan. I want to emphasize here that the example we have used is not a fixed or canned example. Depending on the way in which our customers think of their projections, we may alter the rules for the P & L, cash flow, and balance sheet projections. And in most cases, the forecast reports and budgets are tailored to conform to the customer's own

products and chart of accounts. The main idea here, however, is that we have a framework in which we set up analyses. Then, as required, we make changes taking advantage of the flexibility of the planning language which we are using.

Setting up a realistic plan is an excellent first step in managing a fast-moving business effectively, but it is only a beginning. What really counts is putting it into action. So, once a plan is established, then actual performance against it must be monitored. This can be done with a combination of budget variance reports, forecast versus actual sales reports, and comparisons of P & L, cash flow, and balance sheet position. In some cases, we set up variance reports within the Dollar-Flow system for this reporting. In other cases, we feed the projections which have been set up in Dollar-Flow into other reporting systems.

Let me summarize what we have covered here. Our overall company plan has encompassed a product-by-product forecast with a matching production plan to establish what we expect to produce and sell. We have established budgets which set the spending for various areas of responsibility in the company. And we have set up a profit/loss statement, cash flow projection, and balance sheet projection to examine the impact of the forecast and budgets, analyze alternative financing and management strategies, and to review the company operations on an overall basis.

I hope we have eliminated some of the mystery and myth which surrounds the concepts of financial planning and cash flow analysis. Building an effective profit plan for a company is not a trivial task, nor is it an impossible one. We feel that it is an excellent opportunity to use the computer as a tool, but at the same time is not a good data processing application. Because of the demands in planning for flexibility, immediate feedback, ease of use, and good documentation, it is simply too difficult to set up good models in languages like COBOL, BASIC, or FORTRAN. With a good financial planning language, however, the HP3000 can be a very powerful tool for budgeting and financial planning.

Thank you very much. Do you have any questions?

DATA COMMUNICATION-STRATEGY

ST. ZALEWSKI

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

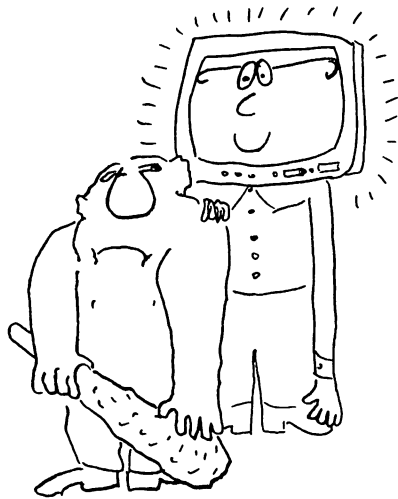
ST. ZALEWSKI
HEWLETT PACKARD

USER FRIENDLY APPLICATIONS

IN COMMERCIAL REALTIME

DATAPROCESSING

- ☞ EXPERIENCES
- ☞ SUGGESTIONS
- ☞ PROBLEMS



HERBERT AUGENSTEIN
RECHENZENTRUM
HERBERT SEITZ KG
GRÜNENSTRASSE 11/12
D-2800 BREMEN
W-GERMANY

Survey

- ☞ Introduction
- ☞ User Interface
- ☞ Application Programs
- ☞ User Training
- ☞ User Documentation

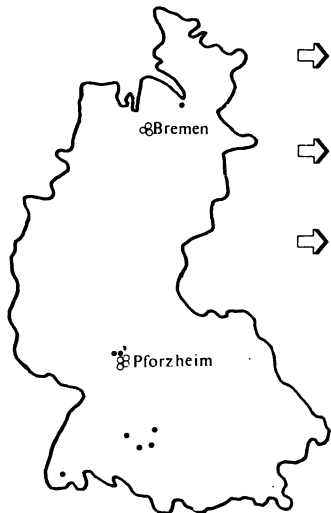


Introduction

The Herbert Seitz Company is ...

- ⇒ A REALTIME DATAPROCESSING SERVICE BUREAU
- ⇒ AND SOFTWAREHOUSE
- ⇒ AND HEWLETT PACKARD OEM

with (1981) ...



- ⇒ 7 OWN HP 3000 (SERIES III AND 44) IN OUR BREMEN AND PFORZHEIM BRANCH
- ⇒ AND 10 HP 3000 SERIES III IN ASSOCIATED COMPANIES
- ⇒ WITH APPROX. 350 TERMINALS SPREAD OVER GERMANY CONNECTED VIA HARDWIRED LEASED LINES/DIALED LINES

Location of:

- own Computers
- Associated Companies

Introduction (2)

WE PROVIDE OUR SERVICES IN GERMANY AND FRANCE FOR COMMERCIAL APPLICATIONS LIKE ...



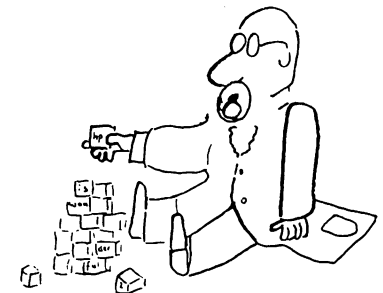
- ⇒ ACCOUNTING
- ⇒ PAYROLL
- ⇒ MATERIAL MANAGEMENT
- ⇒ SHOP FLOOR CONTROL, CAPACITY PLANNING
- ⇒ TOOLS FOR HP 3000 OPERATION, SOFTWARE-DESIGN AND DOCUMENTATION

OUR USERS ARE ...

- ⇒⇒ WORKMEN
- ⇒⇒ DATA TYPISTS, CLERKS
- ⇒⇒ MANAGERS

ONLY A FEW OF THEM ...

- ⇒⇒⇒ ARE SPEAKING (HP-)ENGLISH
- ⇒⇒⇒ HAVE DP EXPERIENCE
- ⇒⇒⇒ HAVE SEEN ANY TERMINAL BEFORE



Introduction (3)

for this kind of users we need ...

☞ A FRIENDLY **interface** BETWEEN THE USER AND HIS APPLICATION PROGRAMS

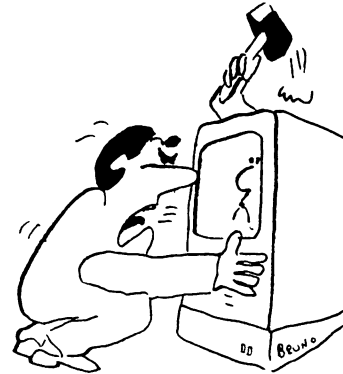
☞ EASY-TO-UNDERSTAND **application programs**

☞ A **user training** WITH REGARD TO THE STANDARD OF EDUCATION OF ITS PARTICIPANTS

☞ **user documentation** MANUALS, WHICH INVITE TO READ

Interface

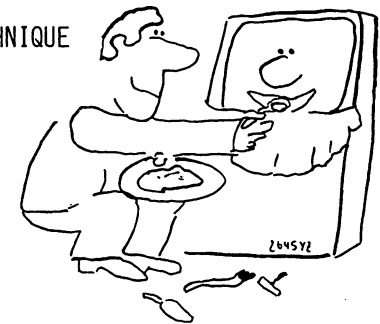
☞ KEEP YOUR USERS OUT OF MPE !



MPE IS A HIGH LEVEL OPERATING SYSTEM WITH A LOT OF POWERFUL COMMANDS, BUT IT IS NOT DESIGNED FOR THE DIRECT USE OF USERS WE ARE DISCUSSING ABOUT

☞ USE ANY KIND OF MENU-TECHNIQUE

(WE CALL OURS "USER-PROFILES")



UFS2

INTERFACE (3)

- ⇒ THE USER ONLY CAN DO THINGS YOU WANT HIM TO DO
- ⇒ BUT HE CAN DO ANYTHING POSSIBLE WITH THE HP 3000

RECHENZENTRUM HERBERT SEITZ KG

RECHENZENTRUM HERBERT SEITZ KG PFORZHEIM - BREMEN

AUSWAHLMENU 2

QUERY WITH TERMINAL- }
OUTPUT

QUERY WITH PRINT-OUTPUT } 01 = QUERY ohne LISTENAUFBEREITUNG

"LISTF" OF SOME FILES } 02 = QUERY mit LISTENAUFBEREITUNG

HARDCOPY SPOOL PRINT } 03 = ANZEIGE INHALTSVERZEICHNIS AUFBEREITETE LISTE

"PURGE" OF ONE FILE } 04 = DRUCKEN AUFBEREITETE LISTE

START OF A JOBSTREAM } 05 = LOESCHEN AUFBEREITETE LISTE

SKIP TO ANOTHER MENUE } 06 = DRUCKEN FAKTURIERUNG

07 = ZURUECK ZUM AUSWAHLMENU 1

99 = ARBEITSENDE
BITTE AUSWAHL EINGEBEN

-

PAGE: 8

UFS2

INTERFACE (2)

- ⇒ WE ONLY NEED 1 MPE COMMAND: HELLO
- ⇒ MENUES EASILY CREATED AND MAINTAINED WITH EDITOR IN ANY LANGUAGE

RECHENZENTRUM HERBERT SEITZ KG

RECHENZENTRUM HERBERT SEITZ KG PFORZHEIM - BREMEN

AUSWAHLMENU 1

01 = STUECKLISTENVERWALTUNG (P03201)

02 = AUFTPAGSVERWALTUNG (P03202)

03 = TEILESTAMMVERWALTUNG (H50001)

04 = ADRESSSTAMM-PFLEGE (ADR001)

05 = AUFBEREITEN BETRIEBSAUFTPAEGE (P03203)

06 = FAKTURIERUNG (P03207)

07 = PREISLISTEN (P03206)

08 = NEUKALKULATION ALLE ARTIKEL (P03205) DANACH FCOPY ALTSEQ/ALTDAT

09 = PREIS ETIKETTEN

10 = BRILLANTANFORDERUNG (P03208)

11 = FAKTURIERUNG LIEFERSCHEINE (P03210)

12 = LAGERARTIKEL-STAMM-PFLEGE (P03211)

13 = SOFORT - FAKTURIERUNG (P03212)

16 = AUSWAHLMENU 2

99 = ARBEITSENDE
BITTE AUSWAHL EINGEBEN

-

PAGE: 7

INTERFACE (5)

UFS2

⇒ ASK FOR RECONFIRMATION OF CRITICAL CHOICES

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 10

```

RECHENZENTRUM HERBERT SEITZ KG      BREMEN - PFORZHEIM
  FINANZBUCHHALTUNG - DIALOG        ( Auswahltable 4 )
                                      ( Druck im Hause )

01 = Drucken  DP-Liste
02 = Drucken  AP-Liste
03 = Drucken  Personenkonten
04 = Drucken  Sachkonten
05 = Drucken  Journal
06 = Drucken  Summensaldenliste
07 = Drucken  Hauptbuch
08 = Drucken  kumulierte Sachkontenwerte
09 = Drucken  Mahnungen
10 = Aufgliederung der offenen Posten
11 = Drucken  Faelligkeitsliste
12 = Drucken  Provisionsabrechnung
13 = zurueck zur Auswahltable 1
99 = Arbeitsende
BITTE AUSWAHL EINGEBEN
01
Wenn Sie diese Arbeit wirklich wollen,
dann wiederholen Sie bitte die Eingabe
Ihrer gewuenschten Auswahl. Andernfalls
geben Sie ein beliebiges Zeichen ein.
-

```

INTERFACE (4)

UFS2

⇒ CONTROL OF REQUIRED SEQUENCE OF DIFFERENT MENUE-CHOICES REFERRING TO TIME, SITUATION OR KIND OF DATA-ENTRIES

⇒ INFORM THE USER WHAT THE COMPUTER IS DOING FOR HIM

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 9

```

08 = NEUKALKULATION ALLE ARTIKEL (P03205) DANACH FCOPY ALTSEQ/ALTDAT
09 = PREIS ETIKETTEN
10 = BRILLANTANFORDERUNG (P03208)
11 = FAKTURIERUNG LIEFERSCHEINE (P03210)
12 = LAGERARTIKEL-STAMM-PFLEGE (P03211)
13 = SOFORT - FAKTURIERUNG (P03212)
14 = RECHNUNGEN IM RZ DRUCKEN

16 = AUSWAHLMENU 2

99 = ARBEITSENDE
BITTE AUSWAHL EINGEBEN
11

```

Druckaufbereitung fuer Rechnungen laeuft zur Zeit. Bitte warten Sie die Fertigmeldung ab!

Die Rechnungen sind fertig. Entscheiden Sie bitte, ob der Druck bei Ihnen oder im Rechenzentrum erfolgen soll und treffen die entsprechende Auswahl (13 = Druck bei Ihnen / 14 = Druck im RZ):

"YOUR INVOICES ARE READY, ENTER 13 FOR HARDCOPY OR 14 FOR LINE-PRINTER OUTPUT"

I N T E R F A C E (6)

AVOID ANY CONFLICTING ACTIVITIES
(AS OTHER ACTIVE SESSIONS OR JOBS,
SUCCESSFUL COMPLETION OF OTHER SESSIONS OR JOBS ETC.)

Die gewählte Arbeit kann wegen konkurrierender Zugriffe durch andere Arbeiten zur Zeit nicht ausgeführt werden !

Mit Auswahl **00** erhalten Sie wieder Ihre Auswahltablette

In **Ausnahmefällen** nach Systemabbruch koennen Sie sich ueber diese Sperre mit der Auswahl **9911** hinwegsetzen.

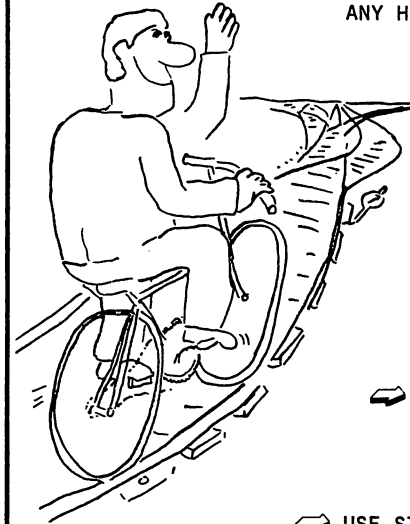
Verwenden Sie diese Auswahl nur, wenn Sie sich sicher sind. Im Zweifel geben Sie bitte 00 ein und fragen Ihr RZ.

BITTE AUSWAHL EINGEBEN

Application Programs

GENERAL GUIDELINES FOR OUR PROGRAMMERS:

- ONLY BLOCK MODE PROGRAMS (V/3000)
- TRY TO DESIGN GOOD READIBLE FORMS
- AVOID (ENGLISH) ERROR MESSAGES FROM ANY HP SUBSYSTEM
- ALWAYS POINT TO WRONG ENTRIES AND PROVIDE A MEANINGFULL ERROR MESSAGE
- ALLOW A REGULAR PROGRAM TERMINATION OR A CANCELLATION OF THE LAST ENTRY AT ANY TIME IN ANY SITUATION VIA THE F7/F8 KEYS
- IF POSSIBLE, USE THE FIELDNAMES IDENTICAL WITH THE ITEM NAMES OF THE CORRESPONDING DATA BASE
- USE STANDARD FORMS AND TRANSACTION CODES IN ALL PROGRAMS AND SYSTEMS



SOME EXAMPLES ...

TELL THE USER WHAT HE IS EXPECTED TO DO

LAGERBESTANDSFORTSCHREIBUNG				RZ HERBERT SEITZ KG.			
INMBFO				PFORZHEIM - BREMEN			
MAN	LAG	MATERIAL-NR.	S				
038	6600	903-10012	B	ZUGENTLASTUNG 3,5 X 0,9			
S=SHOW	Z=ZUG	WERKSTOFF		ALUMINIUM			
X=SBES	A=ABG	DIN-BEZ.					
Y=SDIS	B=BES	LIEFERANT		BELLVIERE S.A.		INVENTUR-DAT. 79.12.17	
I=INV	D=DIS						
E=ENDE							
		LAGERBESTAND	BESTELLBEST.	DISPOBESTAND	GREIFB.BESTAND		
GESAMTLAGER		1360,000	2200,000	3917,000	557,000-		
EINZELLAGER		500,000	0,000	700,000	200,000-		
BESTELL-DAT. 80702709							
MENGE	LIEF-NR	BEST-NR	TERM	BETERM	AUF-NR	BS	KTPD KZGRPOS
PRUEFEN SIE BITTE DEN TERMINVORSCHLAG UND ERGAENZEN SIE DIE BESTELLUNG							

"PLEASE CHECK THE SUGGESTED DELIVERY DATE AND COMPLETE THE ORDER"

PAGE: 14

SAME TRANSACTION CODES IN ALL PROGRAMS

LAGERBESTANDSFORTSCHREIBUNG				RZ HERBERT SEITZ KG.			
INMBFO				PFORZHEIM - BREMEN			
MAN	LAG	MATERIAL-NR.	S				
038	6600	AZ12-34-001	S	UHRENROHWERK MIT BRUCHSICHERUNG GRUNDKALIBER 1177			
S=SHOW	Z=ZUG	WERKSTOFF					
X=SBES	A=ABG	DIN-BEZ					
Y=SDIS	B=BES	LIEFERANT				INVENTUR-DAT. 79.12.17	
I=INV	D=DIS						
E=ENDE							
		LAGERBESTAND	BESTELLBEST.	DISPOBESTAND	GREIFB.BESTAND		
GESAMTLAGER		4360,000	9321,000	15917,000	11557,000-		
EINZELLAGER		500,000	0,000	700,000	200,000-		

SHOW - IF POSSIBLE - THE AVAILABLE TRANSACTION-CODES

PAGE: 13

APPLICATION PROGRAMS (7)

⇒⇒ HELP FACILITY AND FIELD EXPLANATION WITHIN PROGRAMS

BESCHREIBUNG DES FELDES: .DISTU AUS DEM TEILE-STAMMSATZ DISPOSITIONSSTUFE FELDLAENGE 2 STELLEN, DAVON 0 DEZIMALSTELLEN, NUMERISCH AENDERUNG IST ERLAUBT. INHALT: 1 = ENDPRODUKT 2 = BAUGRUPPE 3 = EINZEL- ODER KAUFTEIL 4 = HALBZEUG ODER ROHMATERIAL	} DESCRIPTION OF FIELD DISTU FROM MASTERFILE } DISPOSITION LEVEL } FIELDLENGTH 2, 0 DECIMALS, NUMERIC } CHANGE IS ALLOWED } ALLOWED ENTRIES: 1 = FINAL PRODUCT 2 = ASSEMBLY 3 = PART OF PURCHASE ITEM 4 = RAW-MATERIAL
---	--

★★★ OUR EXPERIENCE: THIS FEATURE IS USED VERY SELDOM AND IT IS VERY EXPENSIVE TO DESIGN AND MAINTAIN WE DON'T EMPHASIZE IT IN ALL APPLICATIONS

PAGE: 18

APPLICATION PROGRAMS (6)

FORCE USER TO CHECK HIS ENTRY, WHERE IT MAKES SENSE

EINGABESATZ ZUR BUCHHALTUNG		RZ HERBERT SEITZ KG BREMEN - PFORZHEIM	
SA 2	BERATER 7000	MAND 777	BUCH-DAT 30.04.80 JT 3 DATUM 11.02.80
BUCHUNGS-DATUM 30.04.80	JT 3	H/S <input type="checkbox"/>	ST/SK-BET ST-SCH FS
KTO-NR 0555554	BETR 500.00	BELEG-NUMMER 1234567	1.AUSGL.NR
GEGKTO 1111111	BV 01 BELEGART 01	KOST-TRAEGER	AUFN
L.AUSNR	AKZ 2 KOST	BERLIN-HILFE	MAHNSTUF KZ SCHECK
VALUTA . .	FAELLKEIT . .	NE <input type="checkbox"/> KONST	ARTIKEL-GR KZ FINANZPL
SKTO-PROZT ,	SKTO-FAEL . .	SONFELD 2	SKT.FAEB.BETRAG
ZAHL-ZIEL	SONDERFELD 1	RESERVE	
MENGEN-FELD			

Pruefen Sie bitte Ihre Buchung. Mit f7 koennen Sie stornieren

"PLEASE CHECK THE RESULT OF YOUR ENTRY, IF NECESSARY CANCEL WITH F7"

User Training

OUR PROGRAM :

⇒ INTRODUCTION INTO INTERACTIVE DATA PROCESSING



⇒ UPDATE TRAINING FOR STANDARD USERS

⇒ APPLICATION-TRAINING

⇒ QUERY FOR NON-DP PERSONNEL

⇒ UPDATE TRAINING FOR QUERY USERS

User Training (2)

INTRODUCTION TRAINING

▷ 1/2 DAY THEORY, 1/2 DAY LABS

▷ TERMINAL USE

▷ HARDCOPY-PRINTER USE

▷ HOW TO LOG ON

▷ HOW TO USE THE MENUES

▷ TRY SOME TRANSACTIONS

▷ HANDOUTS:

- SLIDE COPIES
- TERMINAL AND HARDCOPY USER MANUAL
(SIMPLIFIED AND TRANSLATED)
- CHECKLIST FOR TROUBLESHOOTING



User Training (3)

UPDATE TRAINING FOR STANDARD USERS

➤ REPEAT INFORMATIONS FROM INTRODUCTION AFTER SOME WEEKS/MONTH OF PRACTICE WITHIN 1 DAY THEORY

➤ HOW MTS WORKS

➤ HOW TO IMPROVE RELIABILITY OF DATA COMMUNICATION

➤ STRATEGIES FOR LESS RESOURCE-USAGE AND BETTER RESPONSE TIMES

➤ HELPFUL HINTS AND TRICKS FOR TERMINAL- AND HARDCOPY USAGE

➤ WHAT A COMPUTER HAS TO DO FOR A "SIMPLE TRANSACTION" (OR WHY DOES IT TAKE SUCH A LONG TIME)

➤ DIFFERENCES BETWEEN JOBS AND SESSIONS

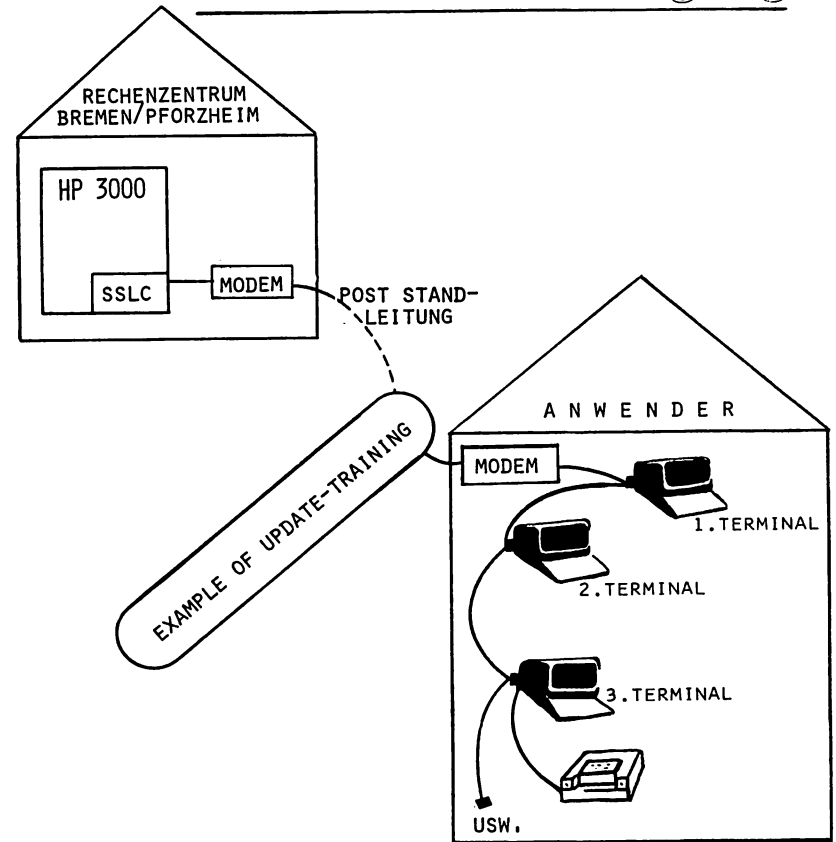
➤ REASONABLE USE OF QUERY

➤ MORE INFORMATIONS ABOUT TROUBLESHOOTING



SOME EXAMPLES ...

MTS Datenübertragung



EXAPMLE OF UPDATE-TRAINING

- DANACH ERFOLGEN EINE REIHE VON ZUGRIFFEN AUF IHRE DATENBANKEN, VOR UND NACH JEDEM ZUGRIFF MUSS IN EINER TABELLE DIE ENTSPRECHENDE KENNZEICHNUNG ERFOLGEN, DIE KONKURRIERENDE ZUGRIFFE REGELT, SIE KÖNNEN SICH DIESEN REGULUNGSMCHANISMUS WIE EINEN POLIZISTEN VORSTELLEN, DER DEN VERKEHR AN EINER VERKEHRSREICHEN KREUZUNG MIT 20(!) ODER MEHR EINMÜNDUNGEN REGELN SOLL

UPDATE	RECHENZENTRUM HERBERT SEITZ KG	PAGE: 23
--------	--------------------------------	----------

LISTEN SELBST GEMACHT - WIE FUNKTIONIERT DAS ??

DER ABRUF EINER LISTE

RECHNER

PLATTE
QL.....
LP.....

EXAPMLE OF UPDATE-TRAINING

DER "OFF-LINE DRUCK"

MIT DEM LISTEN-ABRUF WIRD EINE PLATTENDATEI ERZEUGT (KEIN PAPIER BEDRUCKT)

ERST DAS OFF-LINE DRUCKEN BRINGT DIE PLATTEN-DATEI AUF PAPIER

UPDATE	RECHENZENTRUM HERBERT SEITZ KG	PAGE: 24
--------	--------------------------------	----------

User Training (7)

APPLICATION TRAINING

- ⇒ CLASSROOM-TRAINING OR TRAINING ON THE JOB
(KIND AND TIME DEPENDS ON APPLICATION)



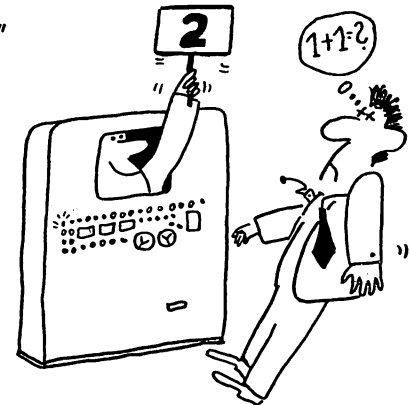
- ⇒ "TRAINING COMPANIES" FOR
TEST AND TRAINING IN ALL
APPLICATIONS

- ⇒ FREE PHONE IN CONSULTING
FOR ALL USERS

User Training (8)

QUERY FOR NON-DP-PERSONNEL

- ⇒ 3-4 DAYS WITH 7 LABS
QUERY FOR USERS, ONLY INFORMATION RETRIEVAL
(NO UPDATE AND DELETE)
- ⇒ DATABASE TERMS AND THEORY (VERY LITTLE)
- ⇒ HOW TO USE "HELP, FORM"
- ⇒ SMALL REPORTS WITH
"LIST"
- ⇒ "FIND" COMMAND
(OR HOW TO TRANSMIT
MR. BOOLE'S MESSAGE)
- ⇒ COMPLEX REPORTS



SOME EXAMPLES ...

Ü B U N G:

{ FORM
FO }

{ DATA SET NAME
DATA ITEM NAME
SETS
ITEMS
PATHS }

	RICHTIG	FALSCH
FO		
FO PATHS		
FORM KDSTAM		
FOR KDNR		
FORM		
F ITEMS		
SETS FORM		
FORM ITEMS SETS		
FOR KDSTAM		

EXAMPLE OF QUERY TRAINING MANUAL

REPORT BEISPIEL

>[D_RP01]

PROCEDURE: RP01

```

001 R
002 H1,"AUFTRAGSUEBERSICHT VOM",30
003 H1,DATE,40
004 H1,"SEITE",60
005 H1,PAGENO,65,SPACE A1
006 H2,"AUFTRG",6
007 H2,"DATUM",13
008 H2,"KDNR",19
009 H2,"KUNDE",26
011 H2,"MENGE",54
012 H2,"VK-PREIS",65
013 H2,"AUFTRGS-WERT",78,SPACE A1
014 G1,"ARTIKEL:",8,SPACE B1,SPACE A1
015 G1,ARTNR,18
016 G1,ARTKBZ,29
017 T1,R2
018 D1,AUFNR,6
019 D1,AUFDAT,13
020 D1,KDNR,20
021 D1,KDKBZ,26
022 D1,AUFMENGE,54,E1
023 E1,"ZZZZZZ?"
024 D1,ARTVKPR,65,E2
025 R1,L,AUFMENGE
026 R1,M,ARTVKPR
027 R2,A,R1
028 D1,R1,78,E2
035 E2,"ZZZZ?.99"
036 E3,"ZZZZZZ?.99-"
037 T1,AUFNR,6,COUNT
038 T1,"AUFTRAEGE",16
039 T1,AUFMENGE,54,E1,ADD
040 T1,R2,78,E2
041 T1,ARTVKPR,65,E2,AVERAGE
042 LINES=15
043 PAUSE
044 S1,ARTNR
045 END
    
```

EXAMPLE OF QUERY TRAINING MANUAL

} Druck der Artikel-Nr
und Bezeichnung bei
neuer Artikel-Nr
] Löschen Reg. 2 bei neuer Art.Nr.

} Errechnen Auftragswert
} Kumulieren Auftragswert je Artikel

User Documentation

⇒ STANDARD DOCUMENTATION FILES
(FOR MANUAL PRINTING) ARE USER ACCESSIBLE

⇒ USER MAY SUBMIT HIS ADD ON DOCUMENTATION
INTO THE SAME DOC FILE



⇒ MANUAL FOLLOWS THE TYPICAL PROGRAM
SEQUENCE

⇒ ANY NEW INFORMATION ON THE
SCREEN IS DISPLAYED WITH PICTURES

SEE EXAMPLE NEXT PAGE ...

Kunde:	DIALOG
Buchgeld nr:	ANWENDUNGEN

DISPOSITIONSERFASSUNG

WIR HABEN IN UNSERER GRUNDMASKE WIEDERUM DIE MATERIAL -
NUMMER A UND DEN VERARBEITUNGSSCHLUESSEL D FUER DIE
ERFASSUNG DER DISPOSITION EINGEGEBEN ES ERSCHEINT DIE
ABGEBILDETE MASKE

HERBERT SEITZ KG
PFORZHEIM - BREMEN

HERBERT SEITZ KG
BESTANDSFORTSCHRIFTUNG

MATERIAL-NR. 5
TESTFELD A
ERWEITERTE BEZEICHNUNG ZEILE 2
ST 3011/4508
1901/4715
LIEFERANT: KLOCKNER-HAMBURG
INLEGE-DAT: 29.07.13

MENGE	LAGEBESTAND	BESTELLBEST.	DISPOBESTAND	GREIFB. BESTAND
GESAMTLAGER	500,000	1200,000		500,000
EINZELLAGER	500,000	1200,000		500,000

DISPOSITION SA MMS
MATERIAL-NR. 5
PREIS: 2400
ST. NR. 3011/4508
123456 7654321 2338

DIS-DATE: DISPOSITIONSDATUM

FUER DIESES FELD GILT WIEDER DIE BEREITS
GENANNTRE REGEL FUER DATUMSERFASSUNG

MENGE: DISPOSITIONSMENGE (3 KOMMA STELLEN,
LOGIK WIE BESCHRIEBEN)

PREIS: IN DIESEM FELD KANN DER EINZELPREIS EINGEGEBEN
WERDEN WIRD KEIN WERT EINGEGEBEN SO HOLT DIE
MASCHINE. FUER DIE BEWERTUNG DES AUFTRAGS-
BESTANDES, DEN PREIS AUS DEM LAGERSTAMMSATZ

- CNTRL (CONTROL) DIE CONTROL-TASTE WIRD ZUR UMKEHRUNG VON NORMALFUNKTIONEN BZW. ZUR STEUERUNG VON SONDERFUNKTIONEN VERWANDT. ZUR AUSFUEHRUNG EINER SOLCHEN SONDERFUNKTION MUSS STETS DIE CONTROL-TASTE FESTGEHALTEN WERDEN UND EINE WEITERE ANDERE TASTE GEDRUECKT WERDEN. SO IST Z.B. DIE CONTROL-TASTE FESTZUHALTEN UND DIE TAB-TASTE ZU BETAETIGEN, WENN MAN FELDWEISE RUECKWAERTS SPRINGEN WILL.
- SHIFT-TASTE MIT DER FESTGEHALTENEN SHIFT-TASTE KOENNEN DIE JEWEILS IM OBEREN TASTENBEREICH ANGEZEIGTEN ZEICHEN EINGEGEBEN WERDEN. IM BEREICH DER BUCHSTABEN A - Z DIENT DIE SHIFT-TASTE WIE DIE BUCHSTABEN-UMSCHALTASTE BEI SCHREIBMASCHINEN FUR ANSTEUERUNG VON GROSSBUCHSTABEN. REGELFALL IST JEDOCH DURCH DIE CAPS-LOCK-FUNKTION (CAPS-LOCK-TASTE) DIE SHIFT-TASTE DER GROSSBUCHSTABEN EINGESCHALTET.
- LEERTASTE DIE BREITE LEERTASTE BEI DEN UNTEREN TASTATURBEREICH DER TASTATUR DER SCHREIBMASCHINE. ZUR EINGABE VON WEISSERZEICHEN. DIESE TASTE HAT WIE ANDEREN TASTEN AUCH EINE FORTSCHRITT-FUNKTION, DIE BEI LAENGEREM DRUECKEN EIN- UND BEIM LOSLASSEN DER TASTE WIEDER AUSGESCHALTET WIRD.
- DEL-TASTE NICHT VERWENDEN.
- RETURN-TASTE DIE RETURN-TASTE HAT NUR UNTER FOLGENDEN BEDINGUNGEN DIE FUNKTION EINER SENDE TASTE (ZUM RECHNER):
 - o DAS TERMINAL IST NICHT IN EINEM MEHRTERMINALBETRIEB (MTS) ANGESCHLOSSEN
 - o SIE ARBEITEN Z.ZT. NICHT IN BILDSCHIRM-MASKEN SONDERN IN ABFRAGE- ODER DRUCKPROGRAMMEN (MENU, QUERY, OFF-LINE-DRUCK)

EXAMPLE OF USER DOCUMENTATION

Presentation Abstract

Presentation Title: NEW APPROACH TOWARD SYSTEM IMPLEMENTATIONNEW APPROACH TOWARD SYSTEM IMPLEMENTATIONAuthor(s): JEAN PIERRE THEORET -ALAN ROWAN (UK)Title(s): VICE PRESIDENT PRESIDENTAddress: INFO-BOUTIQUE LTD 7575 TRANS CANADA HWYST LAURENT , QUEBEC, CANADA H4T 1V6INFO-BOUTIQUE (UK)LTD ST LEONARD HOUSE,
98 HARWOOD RD, FULHAN, LONDON

JEAN PIERRE THEORET
ALAN ROWAN (UK)

Abstract: (No more than 200 words)

From every where one can see new trend in System Implementation
Cost of hardware is going down and cost of people is going up.
This is not a new fact, but it is only recently that research
have made significant break through to adjust System Implementation to
This new reality.

New computer languages are introduced, new methods are developed to
optimize people. This is happening, not only at the level of
programmer with program generator, but also at all level of data
processing activities ie: design, documentation, training etc.....
This new approach seem to prepare data processing world to what
is realy coming next.....

J. P. THEORET
INFO-BOUTIQUE LTD
7575 CANADA HWY
ST. LAURENT, QUEBEC
CANADA H4T 1V6

A. ROWAN
INFO-BOUTIQUE(UK) LTD
ST. LEONARD HOUSE
98 HARWOOD ROAD
FULHAN, LONDON



NEW DIRECTIONS IN CUSTOMER TRAINING

for HP3000

PRODUCTIVITY PRODUCTS

and

OFFICE PRODUCTS

PRESENTATION OUTLINE

- I. Presenting. . . Customer Information Products
- II. Analysis of Customer Needs
- III. Analysis of User Types
- IV. Instructional Modes
- V. Media
- VI. Measuring Effectiveness
- VII. Training Plan

Dr. Donna Senko
Marketing Engineer
Hewlett-Packard Company
Customer Information Products

I. Presenting. . . Customer Information Products

As the use of computers and the range of technical expertise of users expands, the role of customer training and documentation is becoming a more vital part of computer system support. In turn, computer course developers and technical writers must establish direct contact with customers to ascertain technical and functional needs, likes and dislikes. This presentation should serve three purposes: 1) define HP's perspective on training and documentation: 2) inform our user base of our plans in this field: 3) solicit input from our users. The intention of this meeting is to inform and to be informed, that is, to open a direct communication link between the factory and the user.

The Customer Information Products staff of Information Networks Division provides training and documentation for two families of HP3000 software products: Productivity Products and Office Products. Productivity Products include all data management tools, such as IMAGE/3000, VPLUS/3000, QUERY/3000, and KSAM/3000. Also included are language compilers. Office Products include graphics packages, text editing/word processing, and office printing systems.

For all products serviced by the Customer Information Products group, there is an underlying philosophy of training. That is, we consider ourselves successful when we make customers successful in using the HP3000. We have found two basic ingredients necessary to do this: 1) carefully analyze customer needs: and 2) research what type of users will use each particular product.

II. Analysis of Customer Needs

The focal point of customer training is the user. In analyzing customer needs and user types, we are attempting to produce training and documentation materials which are totally user-oriented. The new materials we are developing attempt to present only relevant information to each user, in a manner best suited to that user type. We no longer try to present all information to all users, presented in the same fashion. In most cases this has become an impractical method of training.

In our analysis of customer needs we take many issues into consideration. Among these are: quick start-up, task-oriented training, cost-effective training, accurate and detailed product description, readily available training, increased system use, and increased productivity. We are particularly aware of the need for customers to use a product efficiently soon after purchase. By developing training which is task-oriented (in other words, "how to. . .", rather than a strict reference type of presentation), we are able to reach many more users and have users avoid much frustration.

The issue of cost-effective training can be presented from several points of view. Training which is streamlined to the needs of each user demands less time. Also, many of the training classes we are now developing are self-paced. This means there is no travel expense involved, and the user needn't be away from the office for extended periods. This also makes training available to many users who would not have been sent to a class.

II. Analysis of Customer Needs (continued)

Because it is a necessity for our training and documentation to be accurate and detailed, we are implementing a formal evaluation procedure as part of our course development cycle. The evaluation plan will be discussed later in this paper.

By making training accurate, specific to the user, readily available, and cost-effective, we will assist customers in increasing use of their systems and consequently increasing productivity.

III. Analysis of User Types

In analyzing user types, we address two questions: 1) who are the users: and 2) how will they use the product?

The first question (who are the users?) must be viewed from a historical perspective. Traditionally, most computer users were trained and experienced computer professionals. Until recently, most users were programmers. Today's HP3000 users include: experienced and inexperienced programmers, system administrators, data base administrators, data entry clerks, graphic designers, secretaries, office principals, and clerk/typists. Each of these user types has a different level of computer technical background and requires appropriate training.

The second question (how will they use the product?) can best be addressed with sample questions we ask in developing materials. Specifically, we try to ascertain what the tasks of each user will be. For example, will a programmer be responsible only for using a subsystem, or will he be responsible for optimizing its use? Also, is this the only training the user will receive? Will he have assistance in using the product, or should this training make him self sufficient with the product? Is this product similar to other products this user might be familiar with? Are there other subsystems which should be mentioned in the training, because of products likely to be used together? Although specific questions about each user's tasks vary from product to product, the basic questions, such as those listed above, remain the same.

IV. Instructional Modes

After considering customer needs and user types, we are able to decide on an instructional mode. That is, what type of training and documentation should we supply? Once again it is necessary to take a historical perspective. When the majority of users consisted of experienced programmers, classroom training and reference documentation served the purpose best. Reference manuals usually provide the necessary information for computer professionals to begin using a new product and supply more detailed information as they proceed. Classroom training provides users with technical information about the product, as well as application information for individual users. A well-versed instructor can usually supply necessary technical information for specific applications.

Now, as the use of computers increases, we are seeing many different types of users, more and more of whom are non-computer technical. These users usually require more training than the users of the past, and a different type of training is required. In most cases, a tutorial approach to new materials better serves the needs of these users than does reference documentation. Also, most users prefer to learn on their own, at their own pace, with new materials presented incrementally. Thus, the current emphasis is on developing more tutorial documentation and interactive self-paced instruction materials.

IV. Instructional Modes (continued)

Since some users have expressed interest in having contact with an instructor although they learn best at their own pace, we also plan to experiment with monitored self-paced instruction. With this type of instruction, students can work and learn at their own pace, but at an HP training center and with an experienced HP instructor present. The instructor will introduce the product, guide students through the self-paced course, and answer application specific questions.

But what exactly is interactive self-paced instruction? Interactive refers to using the HP3000 to learn about the HP3000. That is, all interactive instruction involves direct hands-on use of the system in the learning stages. The system is not only the object of the instruction, it is also the means. Self-paced refers to a method of allowing the user to learn at his own pace. The user can determine when he will take modules of a course, how long he will spend on each module, and whether or not he should repeat modules before going on to subsequent lessons.

V. Media

The next question is: What constitutes interactive self-paced training? Depending on the needs and users, we pick a mixture of available media. Media we are currently using or exploring are: workbooks, audio cassettes, video tape/disc, on-line HELP facilities, and total on-line training. Even media such as workbooks and audio cassettes are part of interactive training, since we always couple them with use of the system. For example, a student will use a workbook or audio cassette while sitting in front of his/her terminal and will actually use the system while reading or listening. Explicit step-by-step instructions walk the user through the procedure of using the particular subsystem.

Examples of interactive self-paced instruction with workbooks are: A Guided Tour of the HP3000 and Using DSG/3000. A self-paced course using audio cassettes is Using COBOL II. Video disc and total on-line training are currently being explored. One module of on-line training is already available as part of the System Manager classroom course. We are also currently working closely with the IND lab in designing the on-line HELP facility which accompanies many of our products.

VI. Measuring Effectiveness

In evaluating our courses, we focus on the user once again. There are two methods we employ to gather input from our customer base. The first is by meeting formally or informally to exchange general information and opinions. This is what we are attempting to do today. The second method is by including direct interface with customers in the course development cycle.

The major steps in the course development cycle are outlined in a flowchart on slide VI.3. Notice that we include three stages of materials evaluation once development has been completed: 1) materials are tested for technical accuracy and functionality at internal HP sites: 2) materials are tested for technical accuracy and functionality with customers: and 3) we check known support problems to gather information for revising materials and for planning future training. The boxes with shading on this flowchart represent the stages of course development during which customers are involved. Notice that we work with customers in the investigation stages, as well as in the test and follow-up stages. We are making an effort to work directly with customers as much as possible.

VII. Training Plan

Finally, it is time to introduce our entire training plan. The best way to do this is to present the courses available according to user type. For the sake of simplicity we have chosen to outline our courses here around five different types of users. Naturally these courses can be useful to other than the user types presented here.

At present there are two courses for the end user, that is, the non-computer technical user. These are: A Guided Tour to the HP3000, which introduces the novice user to the major subsystems of the HP3000, and Using DSG/3000, which provides the non-programmer instruction in using the interactive interface of Decision Support Graphics/3000. For the system administrator there are currently two classroom courses: System Operator and System Manager. For the data base administrator there is currently one classroom course: IMAGE/3000. The application programmer has several courses available. Among them are: the self-paced course for COBOL II, and the classroom courses: Programmer's Introduction, IMAGE/3000, VPLUS/3000, DSG/3000 Programmatic Use, 2680 Laser Printing System. The programmer analyst currently has one classroom course available: Application Design.

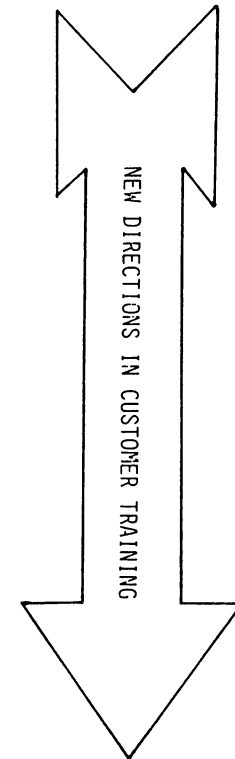
At this point we would like to get your input on existing customer courses and what you would like to see in the future.

OFFICE PRODUCTS

AND

PRODUCTIVITY PRODUCTS

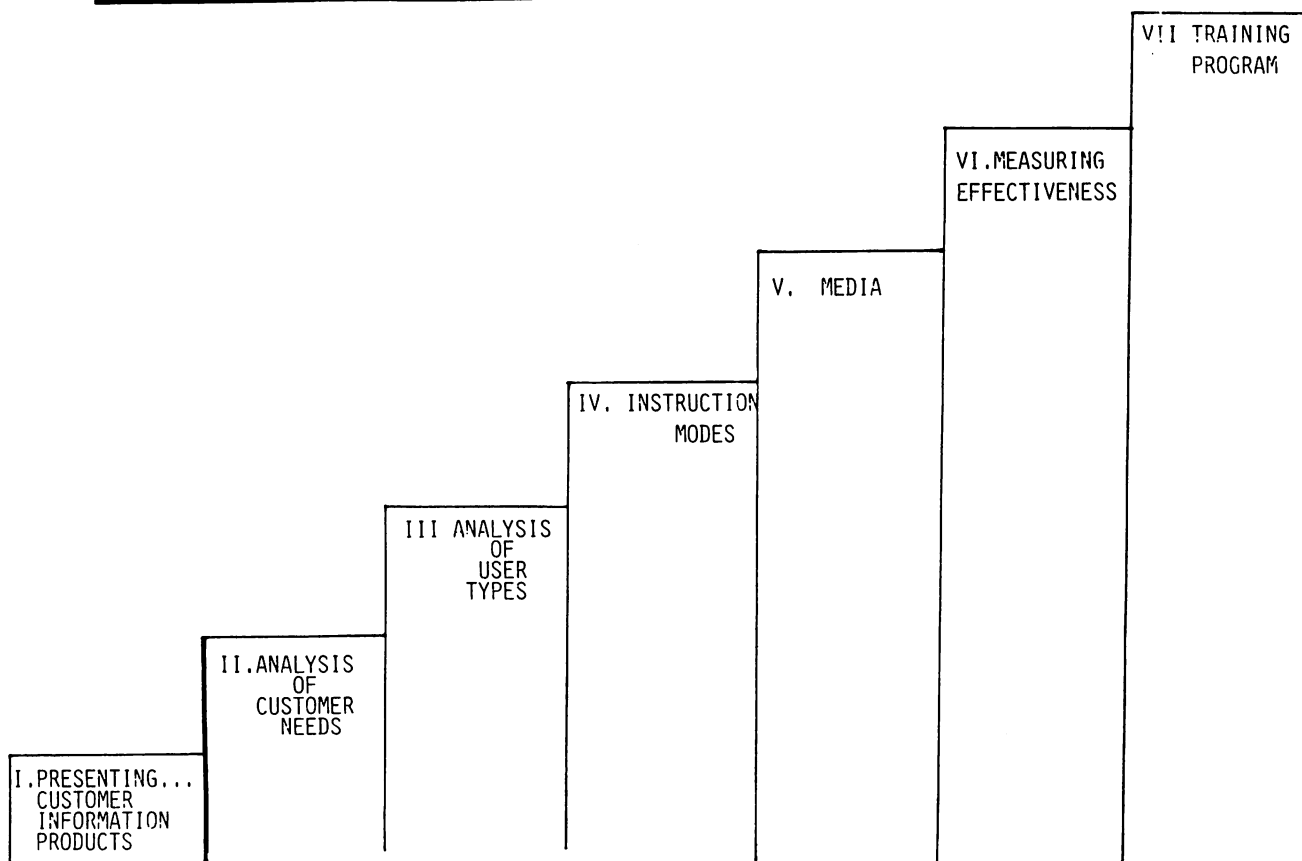
FOR HP3000



PRESENTING . . .

CUSTOMER INFORMATION PRODUCTS

I.1.



PRODUCTIVITY PRODUCTS:

- DATA MANAGEMENT TOOLS
- LANGUAGES/COMPILERS

OFFICE PRODUCTS:

- GRAPHICS PACKAGES
- TEXT EDITING/WORD-PROCESSING
- OFFICE PRINTERS

I.3.



IND CUSTOMER INFORMATION PRODUCTS

PROVIDES:

HP3000 TRAINING AND DOCUMENTATION

FOR

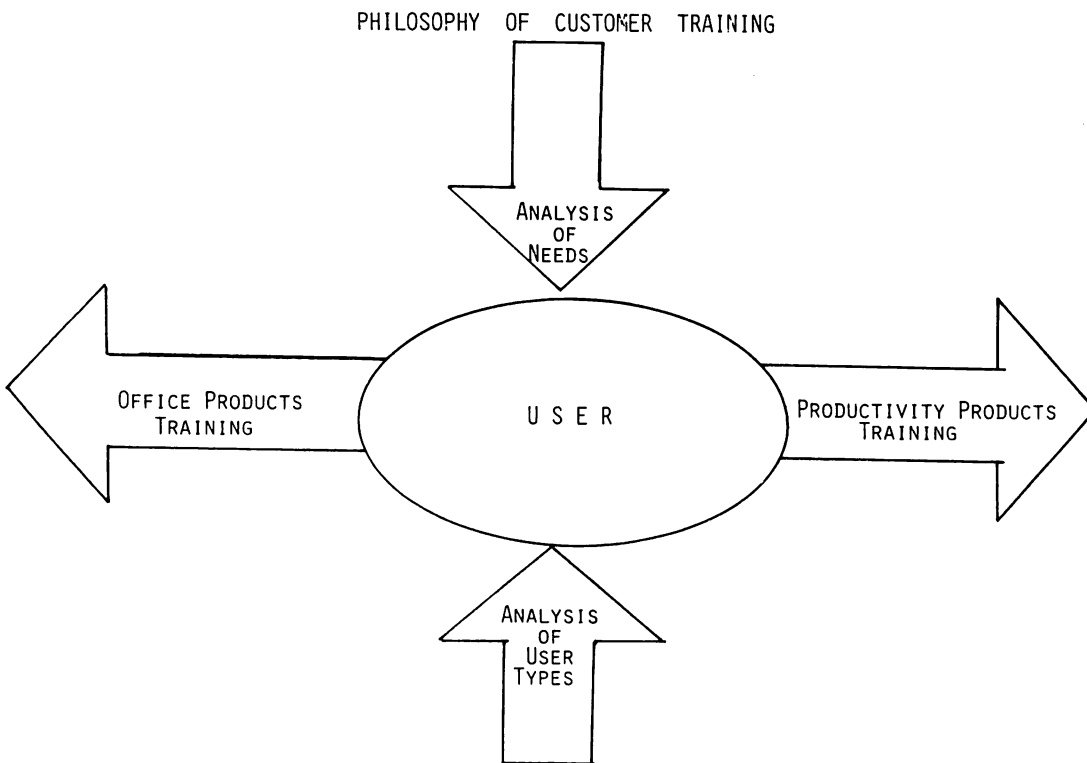
- o PRODUCTIVITY PRODUCTS
- o OFFICE PRODUCTS

I.2.



ANALYSIS
OF CUSTOMER NEEDS

II.1.

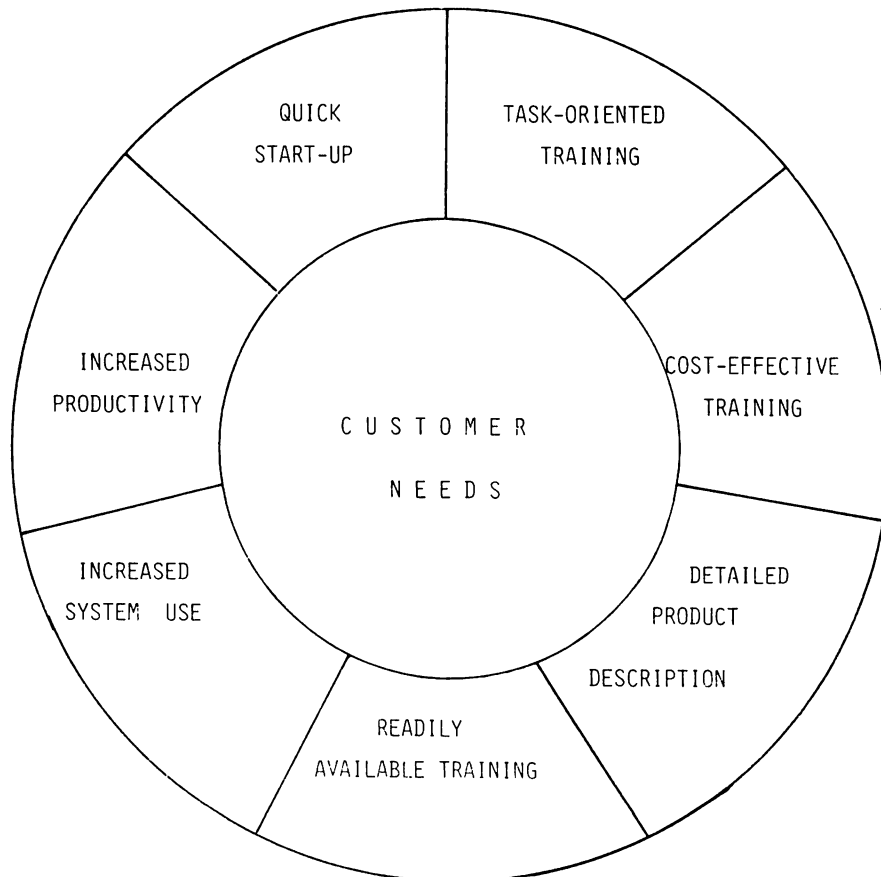


I.4.



ANALYSIS
OF USER TYPES

III.1.



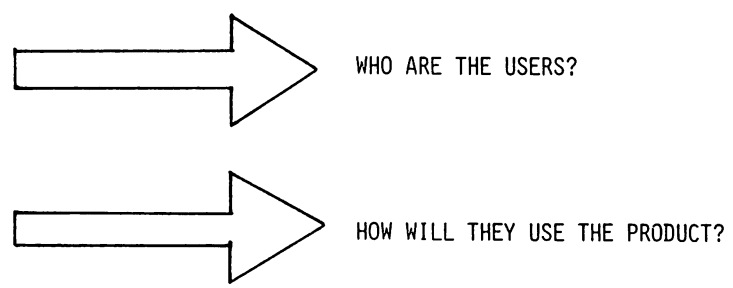
II.2.



WHO ARE THE USERS?

<p><u>UNTIL RECENTLY:</u></p> <p>TECHNICAL,</p> <p>COMPUTER PROFESSIONALS</p>	<p><u>THE TREND:</u></p> <p>MORE FIRST-TIME USERS</p> <p>AND</p> <p>NON-EDP PROFESSIONALS</p>
---	---

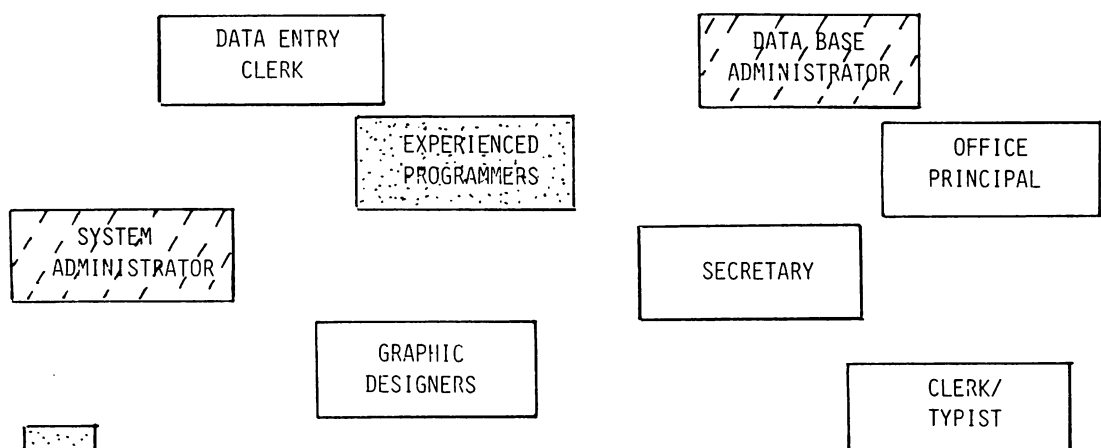
USER TYPES






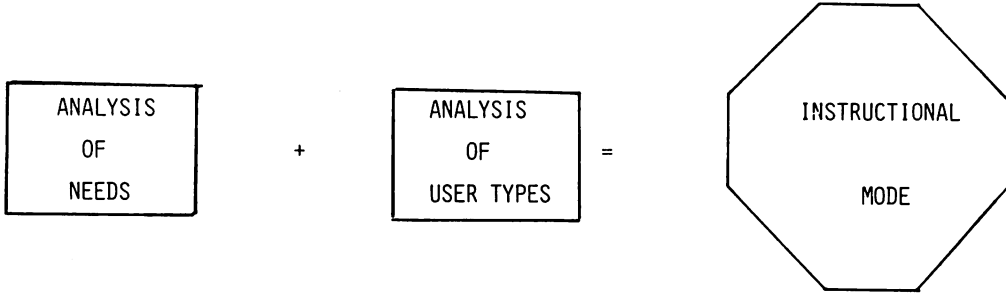
HOW WILL USERS USE THE PRODUCTS?

- O WILL PROGRAMMERS BE RESPONSIBLE FOR OPTIMIZING SYSTEM USE AS WELL AS USING THE SYSTEM?
- O DOES THIS PRODUCT REPLACE A PRODUCT THE PROGRAMMER ALREADY KNOWS?
- O HOW DOES THIS PRODUCT RELATE TO OTHER PRODUCTS?
- O IS THIS TRAINING STAND-ALONE?
- O WILL THE DATA BASE ADMINSTRATOR DESIGN NEW DATA BASES, OR SIMPLY MAINTAIN EXISTING ONES?
- O WILL THE OFFICE PRINCIPAL NEED ACCESS TO DATA BASES?
- O WILL THE DATA ENTRY CLERK NEED TO USE PROGRAMS OR UTILITIES, OR JUST ENTER DATA?

HP3000 USERS INCLUDE:



-  - TECHNICAL, COMPUTER PROFESSIONAL
-  - SLIGHTLY COMPUTER TECHNICAL
-  - NON COMPUTER TECHNICAL



IV.2.

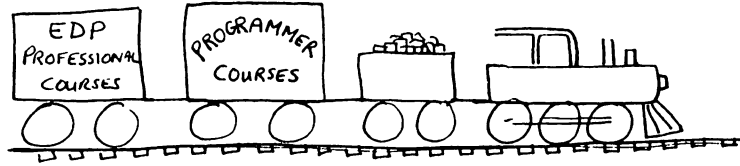


INSTRUCTIONAL MODES

IV.1.



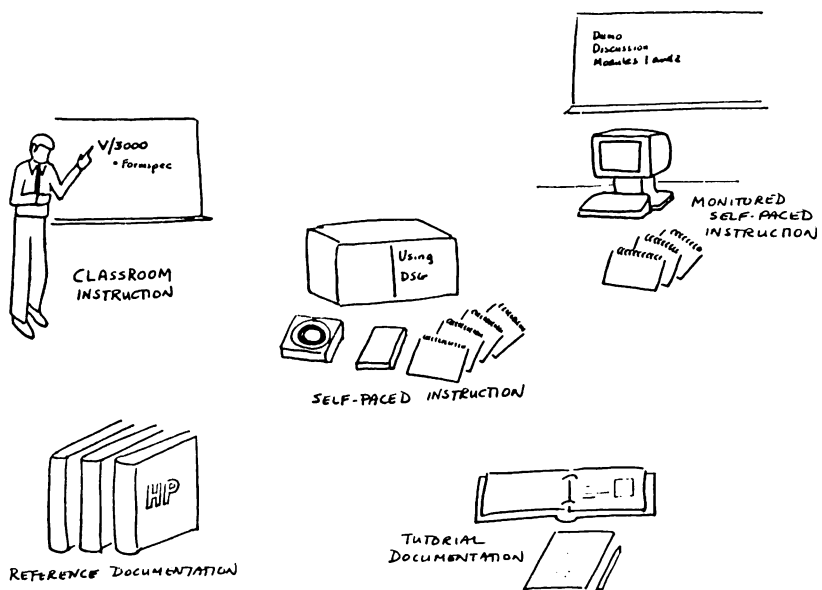
INSTRUCTIONAL MODES
WHERE WE'VE BEEN:



IV.4.



INSTRUCTIONAL MODES



IV.3.

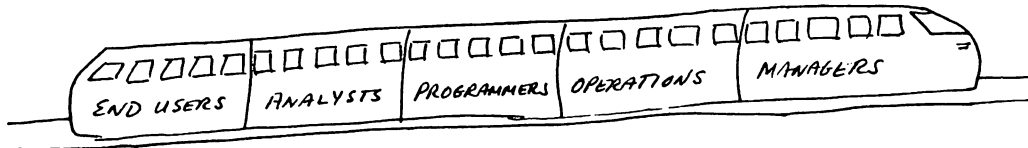


WHAT IS INTERACTIVE SELF-PACED INSTRUCTION?

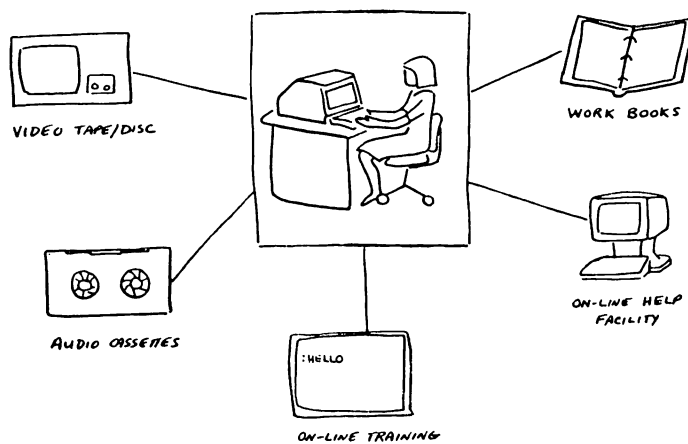
INTERACTIVE - USING YOUR HP3000 TO LEARN ABOUT THE HP3000

SELF-PACED - THE USER SETS HIS/HER OWN PACE OF LEARNING

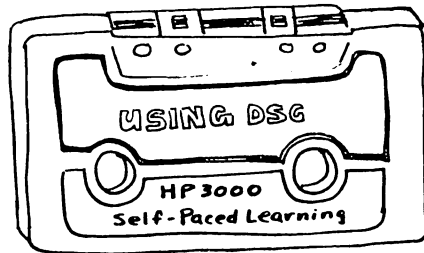
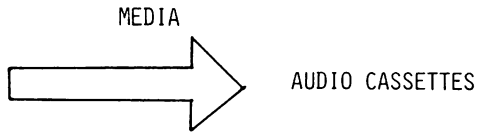
INSTRUCTIONAL MODES
WHERE WE'RE GOING:



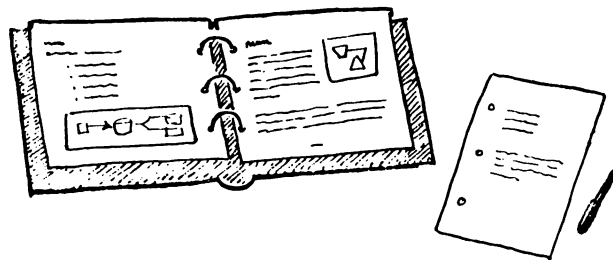
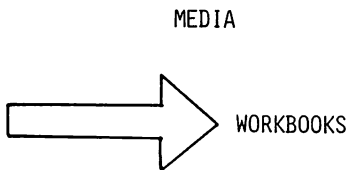
MIXED MEDIA
FOR
INTERACTIVE SELF-PACED INSTRUCTION



MEDIA

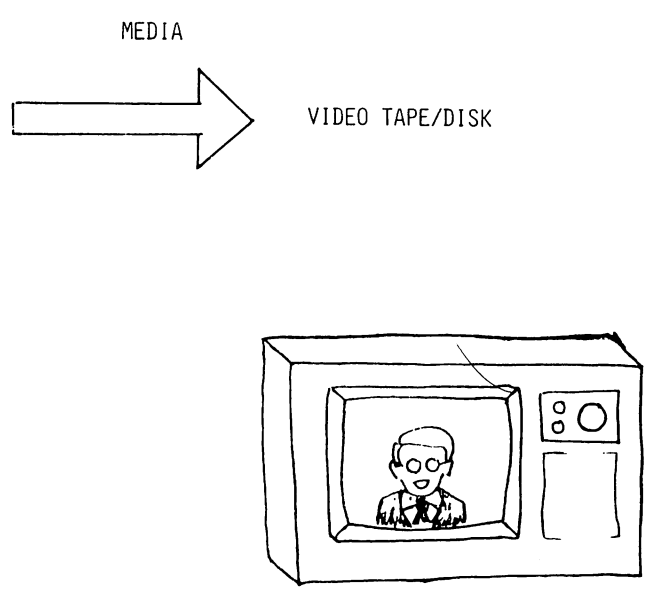
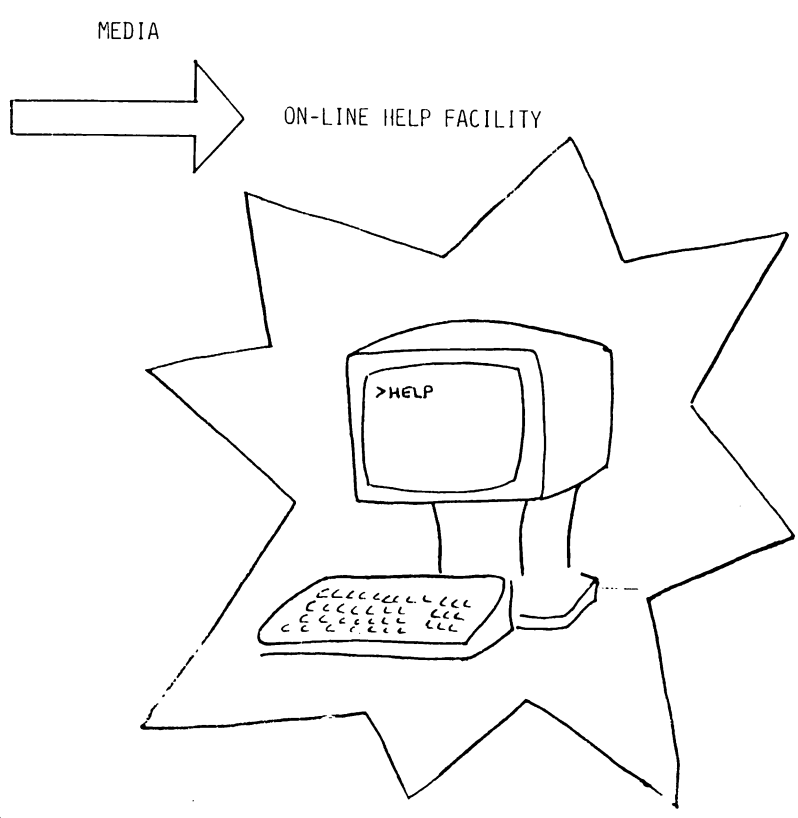


V.4.



V.3.

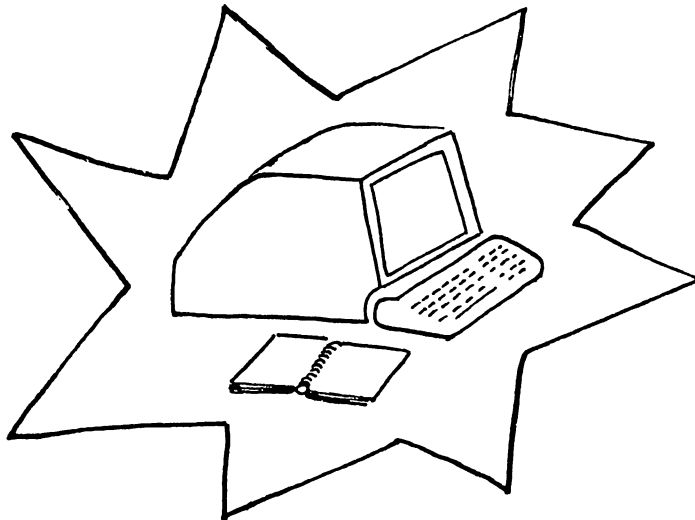
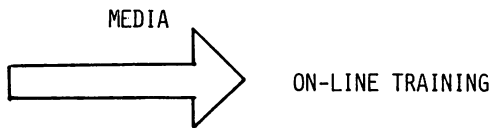




MEASURING

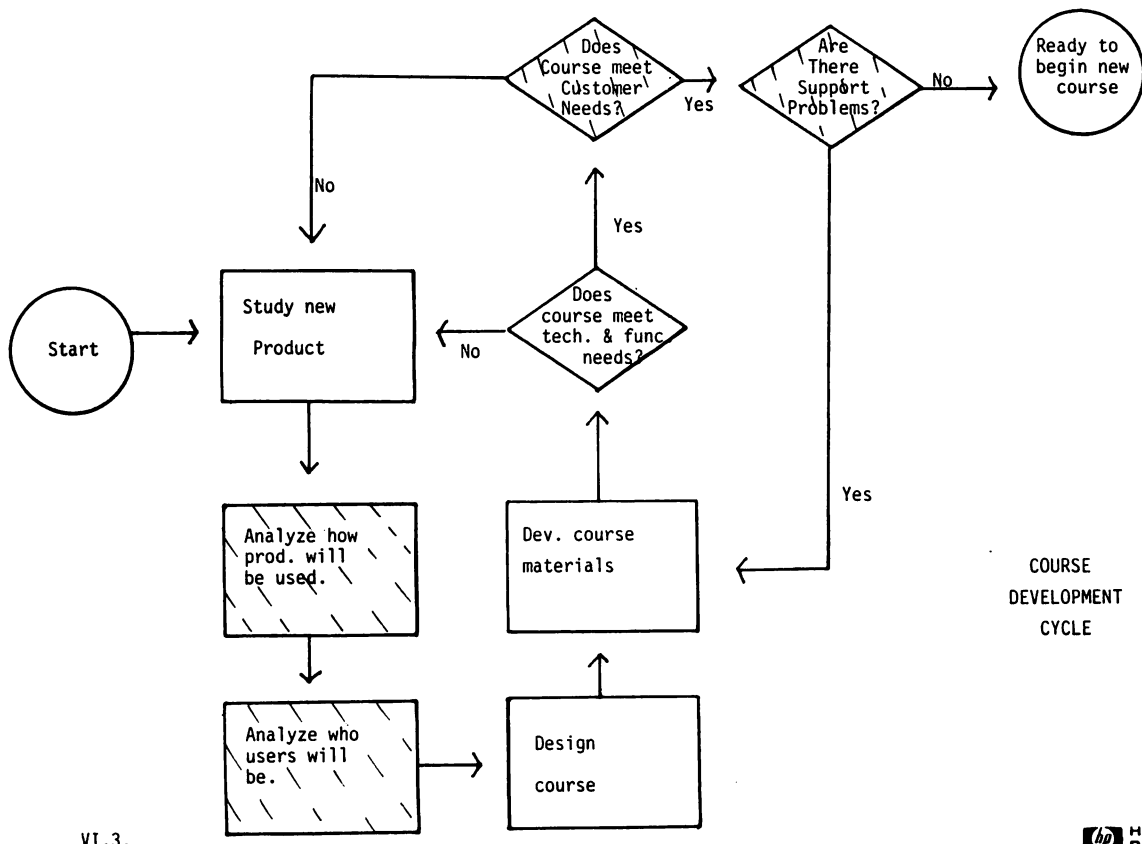
EFFECTIVENESS

VI.1.

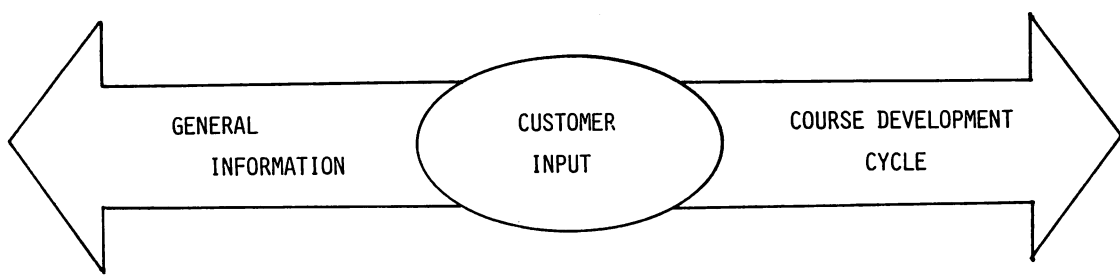


V.7.





VI.3.



VI.2.

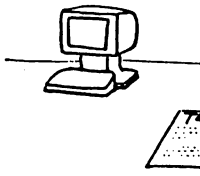


WE NEED YOUR
INPUT!

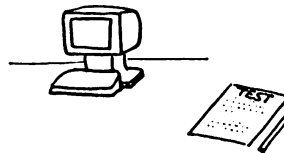
VI.5.



EVALUATION PLAN



(1) INTERNAL TECHNICAL
AND FUNCTIONAL TESTING



(2) CUSTOMER TECHNICAL
AND FUNCTIONAL TESTING



(3) CUSTOMER FOLLOW-UP FROM
SUPPORT POINT-OF-VIEW

I.4.



NEW PRODUCT TRAINING PROGRAM

FOR:

END USER

A Guided Tour
To The HP3000

Using
DSG/3000



VII.2.



TRAINING PROGRAM

VII.1.



NEW PRODUCT TRAINING PROGRAM

FOR:

DATA BASE ADMINISTRATOR

IMAGE/3000



VII.4.



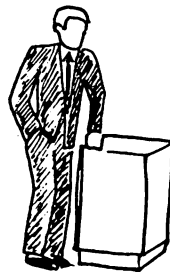
NEW PRODUCT TRAINING PROGRAM

FOR:

SYSTEM ADMINISTRATOR

System
Operator

System
Manager



VII.3.



NEW PRODUCT TRAINING PROGRAM

FOR:

ANALYST

Application Design



VII.6.



NEW PRODUCT TRAINING PROGRAM

FOR:

APPLICATION PROGRAMMER

SPL/File System

Special Capabilities

Introduction To MPE

Measuring System Performance

Learning COBOL II

Programmer's Introduction

IMAGE/3000

VPLUS/3000



ISC/3000 Programmatic Use

2680 Laser Printing System

IML/3000

VII.5.



NU-DATA APS
LYNGBYVEJ 70-72
2100 KØBENHAVN Ø

TEL. 01. 29 27 22
APS REG. 78 18
GIRO NR. 7 21 80 60

IPB

A presentation of IPB, system
for Interactive Planning and
Budgeting by NU-DATA ApS.

INTERACTIVE

PLANNING

AND

BUDGETING

IPB, system for Interactive Planning and Budgeting

Abstract:

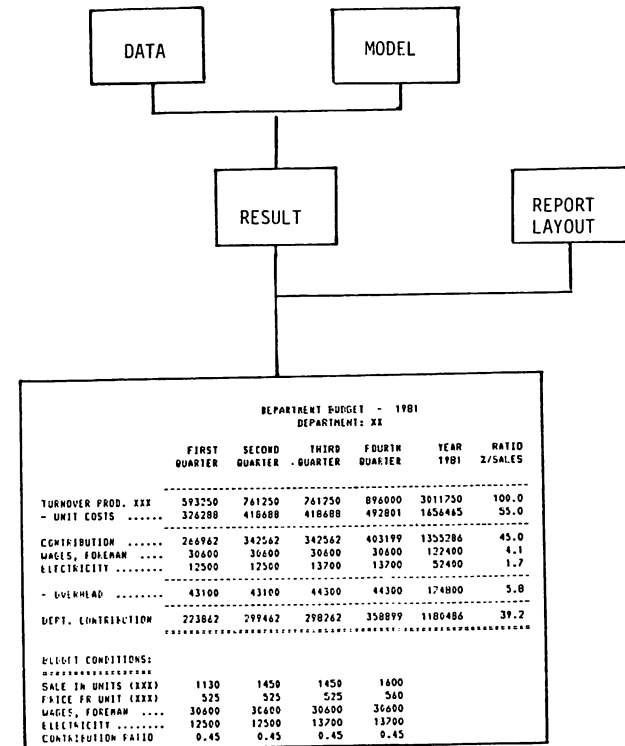
IPB is an interactive system developed for financial planning and budgeting purposes with particular emphasis on the simulation side.

The system has its own "model language" which makes it flexible and easy to use - also for persons without any kind of EDP experience. This enables the user to investigate the impact of uncertain future conditions and improves his understanding of the consequences of alternative actions.

The IPB system is an efficient tool for the manager who knows the problem of not being able to achieve information about the effects on earnings and cash-flow of some specific changes of policy, in a sufficiently quick and accurate manner.

Any accountant or financial manager who knows the problem of being stuck in daily routines, unable to deliver the analyses that management justly wants but seldom gets, can also profit from the IPB system.

The structure of all planning, budgeting and calculation can in principle be described as:



When combining DATA with certain arithmetic rules (a MODEL) a RESULT is produced. If the RESULT is combined with a REPORT layout the finished text will be written out.

IPB is made to handle exactly this structure, and the calculation and the writing out of a budget is very simple:

```
READY FOR COMMAND
1.00 BUILD RESUL FROM NODEL AND DATA

READY FOR COMMAND
1.00 WRITE RESUL AFTER FORM
```

Thus the computer takes care of the calculation/typing work and the decision makers can concentrate on alternatives. The consequences of alternatives can be illustrated like this:

```
READY FOR COMMAND
1.00 ALTERNATIVE TERMINAL
1.00 DATA1 DATA2 DATA3
1.00 D3,D7
1.00 INFLATION 0.5
```

An inflation rate of 0.5 per cent per periode will be added to the information in the three sets of data in lines 3 and 7.

If DATA1, 2 and 3 contain information about three departments three department budgets can be made using the same calculation rules on all three sets of data.

```
READY FOR COMMAND
1.00 BUILD BUDG1 FROM NODEL AND DATA1
1.00 BUILD BUDG2 FROM NODEL AND DATA2
1.00 BUILD BUDG3 FROM NODEL AND DATA3
```

The department budgets can be added up to a total budget:

```
READY FOR COMMAND
1.00 ADD BUDG1 BUDG2 BUDG3 TO TOTAL
```

The TOTAL result can now be written out according to the same report layout as the department budgets:

```
READY FOR COMMAND
1.00 WRITE TOTAL AFTER FORM
```

If several alternatives are made in relation to the base budget it is only the changes which will be of interest. If the new budget is deducted from the old one these changes will be in focus:

```
READY FOR COMMAND
1.00 SUBTRACT OLD TOTAL TO CHANG
```

When the budget is to be written out the command JUMP can be used. This entails that only the lines whose value differs from 0 (zero) are written out. The text will have the usual layout, but only factors which have been changed compared with the original ones will be included.

```
READY FOR COMMAND
1.00 WRITE CHANG AFTER FORM JUMP
```

IPB has about 60 commands and instructions of which 7 are used in the above examples. These commands/instructions make it very easy for people without EDP experience to make their own models and reports, and also registration and processing of data are facilitated.

In the following we will give a brief description of the various commands and instructions.

Summary of commands and instructions and their most important characteristics

COMMANDS

ADD file name1,, file name n TO file name.

Adds two or more data files line by line and column by column.

TERMINAL

ALTERNATIVE REFERENCE file name

These commands are used to analyse 'what if' situations. New alternatives can often be analysed by modifying existing data files (e.g. insertion of new figures, adding a constant value to existing values, change of percentage, or making an alternative inflation). By using the command ALTERNATIVE such modifications of existing data can be made in three different ways:-

- line by line
- in a sequence of lines
- in lines and files referenced in a file (if the command is REFERENCE).

AUTO file name

Executes a sequence of commands and instructions stored on the strategy file in question.

BUILD file name FROM file name (AND file name)

This command results in accomplishing an arithmetic operation defined in a model file. The result of the calculations will be stored on an existing data file or on a new data file. If the calculations require input the command (AND file name) specifies where the input can be accessed.

COPY file name TO file name

Copies files. Especially useful in connection with alternative calculations (ALTERNATIVE) if a copy of the original data is required.

DATA file name

Entry of data. The data entered will be kept on a data file of the specified name.

DIVIDE file name 1,, file name n TO file name

This command allows the user to divide data files quantity by quantity.

GET file name

Creates a new file by joining lines from one or more existing files.

MATRIX file name * file name TO file name

The command executes different kinds of matrix operations.

MERGE file name 1, (file name 2) TO file name

This command enables the user to create new data files from existing data files in any possible way.

MODEL file name

Definition of arithmetic operations. These will be kept in a file under the specified name.

MULTIPLY file name 1,, file name n TO file name

The use of this command multiplies two or more data files quantity by quantity.

PURGE file name

Deletes the file.

REFERENCE file name

In connection with alternative calculations it may often be useful to alter several lines in different data files by one operation. For instance all lines containing information of a currency will be changed by a percentage as a result of a new rate of exchange; or all lines containing information of the price of petrol per gallon will be increased by the amount XX, which is a new tax. The REFERENCE command allows the user to combine any data line in any data file by a reference.

REPORT file name

Definition of output structure, i.e. which lines and columns from which file are to be written and which layout is to be used.

SCHEME file name

This command is used to design and print schemes for data.

STOP

Terminates the program.

STRATEGY file name

Allows the user to create a chain of commands and instructions. The sequence of commands and instructions will be kept in a strategy file after which it is possible to execute the sequence by one command only. This command can be used for example when the company budget is to be gathered from several department budgets, etc.

SUBTRACT file name 1,, file name n TO file name

Subtracts data files quantity by quantity.

UPDATE file name

Any column and line in the specified data file can be updated (changed) by the use of UPDATE.

WRITE file name AFTER file name

This command prints a data file according to the structure specified under REPORT.

INSTRUCTIONS

COLUMNS

The system operates with columns 1 - 30. If the user wants to change the number of columns or if certain columns should be reserved for later use the instruction COLUMNS should be applied.

CONSTANT

Replaces specified data by a constant.

DATE

Writes today's date.

DECIMAL

Controls the number of decimals in the output.

DIVISOR

This instruction is used to reduce the values in one or more lines in a data file by a divisor.

FACTOR

This instruction is used to increase the values of one or more lines in a data file by a factor.

HEADING

Prints a heading on all pages in a report.

IF

Used as a conditional expression under GET and MERGE.

IF - - - ELSE

Logical expression which can be used in the calculations.

INFLATION

Adds inflation to defined data lines (a percentage calculation).

INTERVAL

All lines in data files, model files, report files, reference files as well as in strategy files are numbered. The system generates line numbers automatically with increments of 1. This can, however, be changed arbitrarily by the instruction INTERVAL.

LINE

Prints a line of specified characters in the report.

LIST

Gives a listing of the file which is in use.

MINUS

Subtracts a constant from data specified by the user.

MOVE

Moves one or more columns in a data file.

NEWCOL

Changes the number of active columns in data and model files.

PAGE

Changes to a new report page.

PERCENTAGE

Changes specified data by X per cent.

PLACE

Is used to copy sequences of lines as well as single lines from a data file to a model file.

PLUS

Adds a constant to data specified by the user.

READY

Terminates the use of any command; the system is then ready for a new command.

SOFTWARE TECHNOLOGY

A FUTURE REQUIREMENT OR CURRENT NECESSITY ?

DR. HARRY BLASK

DR. H. BLASK
MINISTRY OF RESEARCH AND TECHNOLOGY
BONN
WEST-GERMANY

"SOFTWARE TECHNOLOGY

A FUTURE REQUIREMENT OR CURRENT NECESSITY?"

DR. HARRY BLASK

Ladies and Gentlemen,

1. Introduction

Every time a new technical achievement is made there are a few resourceful minds who are quick to make use of it to the disadvantage and detriment of their fellowmen. So the development and propagation of information technology has "blessed" mankind with so-called computer crime. I would like to cite one of the cases reported to date which I think is of particular interest to the subject of my paper:

A few years ago, an audit was announced unexpectedly in an English bank. The result was that some employees of the data processing division disappeared overnight (and probably left the country). This practically confirmed the suspicion that irregularities had occurred. The software was then examined with special care; but in spite of an intensive search, the manipulations which had most probably been made could not be discovered.

This example throws special light on the present software situation: Even specialists fail to detect existing errors or manipulations. I would like to examine this situation further in a somewhat generalized manner before I eventually embark upon the actual subject namely software technology.

2. Situation and impact of information technology

In modern industrial societies, the generation, processing and transfer of information are playing an increasingly important role. Together with microelectronics, data processing and technical communication, information technology represents a key technology which none of the major industrialized nations can afford to neglect. Consider, for instance, the increasing integration, and the further decline in prices, of electronic components: here, future development will lead to fundamental changes for manufacturers and users of information technology as well as for private users. Since we are still at the beginning of this process, we can by no means predict all the consequences which will be brought about by information technology. Some effects, however, are already emerging today with a sometimes depressing clarity:

- The institutions using information technology and, in the final analysis, society as a whole is becoming more and more dependent on this technology. Important sectors of production engineering, the control of complex technical plants, the administration and handling of large stocks of information, all these are practically no longer possible without data processing. If data processing ever fails, this may have disastrous consequences, including even the destruction of companies. One of the main elements of the vulnerability of those who rely largely on information technology is the increasing complexity of technical systems and their applications. They make use of electronic information processing because, above all, this allows numerous links and interconnections leading to new information. The resulting trend towards increasing complexity makes the systems less transparent, less

controllable, and capable of being mastered by only a few specialists as was illustrated by the example I quoted at the beginning.

- Information technology also penetrates increasingly into sensitive areas. It controls railway and tram signalling installations; air traffic control is more and more computer-assisted. Presumably nuclear power stations, too, will soon be monitored and controlled by computers; this, however, is not yet permitted in the Federal Republic of Germany. Errors or failures occurring in these areas would result in serious accidents involving loss of life. We have still far to go before finding a solution to the problem of ensuring the reliability of information systems.

If we now consider information technology from the hardware and software aspects, we will find that the software has nothing comparable to set against the technical progress of hardware. This is probably due to the fact that hardware is the result of engineering to a much greater extent than software and that engineering uses sophisticated working techniques and theoretical elements with a long tradition. They were already known and practised long before data processing was developed. Programming, on the other hand, is relatively new. It takes place at the boundary between technology, organization and the humanities and to this day is considered an "art" and not a science. When a system is implemented the cost share of software today is already in the region of 80% and shows a tendency to increase.

3. The software crisis

Let us first of all consider the manifestations of the so-called software crisis. When comparing the software development of 20 years ago of that of today we find that little has changed. The scene is still dominated by the "free lance artist" who produces software without observing too many rules, giving free rein to his intuition. For him, it is not important that

- software be structured in a transparent and intelligible way so that later on it can be understood and, if necessary, modified or supplemented also by the other users,
- adequate documentation be provided for programmes; this applies even more to considerations of, and decisions on, design, which today are put down in writing in very few cases only,
- an extensive phase of problem analysis and definition precede the design and implementation phase,
- programme protability is achieved.

As a result, software is prone to many errors; in the case of major operating systems, for example, between 5000 and 10000 errors per year are reported to the manufacturer. It is often found that software is no longer usable soon after its developer has left the company or if development capacity is no longer accessible for other reasons.

Other symptoms of the crisis are:

- Users and developers have difficulty in communicating with each other. Having different engineering backgrounds, they do not speak the same language. Communication is further complicated by the fact that the colloquial language we normally use is informal and imprecise.
- The means of expression and description available for software development are inadequate. This applies in particular to graphical means. The large variety of programming languages available complicates standardization.
- Progress and productivity in software production are particularly difficult to control. While between 1958 and 1978 the processing productivity of computers increased by a factor of 1000, the productivity of programmers increased only by a factor of 2! Projects are managed and organized in a shirt-sleeve manner and consequently yield unsatisfactory results. A study carried out by the University of Karlsruhe involving 100 automation projects showed that, on average, software had additional time requirement of about 50%, while the costs were about 75% higher than estimated.

In addition to these qualitative characteristics of the "software crisis", there are some important economic aspects:

- a) In the Federal Republic of Germany alone, about DM 10 thousand million are spent annually on software. About 80% of the costs of a data processing application are accounted for by software. On the one hand, this is due to the fact that software is designed to fulfil

increasingly complex tasks - thus becoming increasingly complex and expensive itself -, on the other hand, microelectronics brings out price cuts for hardware, while this can hardly be said about the software side.

- b) Competition between data processing suppliers has been shifting to the software sector. This applies without reservation to operating systems, which determine computer characteristics much more than hardware does. If you just remember how many computers - especially smaller ones - use the standard components from Intel, Texas Instruments, Motorola and Fairchild, you will recognize that the net value added to a product and its identity and hence its competitive ability come from the software, including application software.
- c) Available data processing solutions are inadequately portable or not portable at all. The same problem definitions are therefore handled again and again - a practice which constitutes an enormous economic waste.

This is why we have to look for new approaches in order to reduce the cost of software development and maintenance. Just as we use high-precision tools in production engineering, we will have to use the computer itself for software production. Neither our own capabilities nor our own ability to analyse and integrate are sufficiently powerful to cope with the software crisis.

4. The software technology scene

Approaches to software production technology have developed since the late 60's and appear promising: at the present time, however, the term "technology" is rather flattering.

Today, software development and maintenance are generally based on the following phases:

- requirement analysis and specification
- rough design
- final design
- coding and implementation
- maintenance.

To include testing in this list as a phase of its own would, I think, not be correct, since testing activities occur in all the phases cited above. For all phases, there are isolated solutions with quite different capabilities. Computer-aided tools and methods for the requirement (analysis and specification) phase have so far been the least developed elements. One of the main reasons for this is probably the fact that there is a contradiction between the necessarily application-oriented character of this activity and the need for highly flexible tools for varying applications. For the rough and final design phases, the situation is better: several methods (e.g. the Jackson method, structured programming) are at least known to users by name, although they are actually used only by a few advanced software developers. In the coding phase, which is the traditional field of activity of programmers, methods and tools of software technology are more wide-spread, of course, whereas the maintenance phase is still characterized by a very small inventory of methods and tools.

The methods, techniques and tools used are often quite sophisticated and hence can be applied only by experts. In addition, the value of software technology is frequently not easy to understand. It is absolutely impossible for the normal user to estimate the value of such technological tools. As a consequence, the barriers preventing their introduction are tremendous: there is a great reluctance to introduce such tools on the part of management boards, who hesitate to invest the large funds required for the necessary training effort and for investments, as well as on the part of software developers, who are expected to give up their old familiar working methods and to acquaint themselves with new sophisticated techniques. In view of this situation it is no wonder that tools of software technology are not yet widely used either in the Federal Republic of Germany or, probably, in other countries.

5. Measures initiated by the BMFT to support software technology

In order to prepare the ground for support measures, the Federal Ministry for Research and Technology entrusted the Society for Mathematics and Data Processing (GMD) with the undertaking of a study to analyse measures for the improvement of software production and to identify research areas which, in the medium term, are, or will be, of importance for users and software producers. The study also included a detailed analysis of the market of software suppliers in the Federal Republic of Germany. Let me cite a few results of the study:

There are about 3.000 suppliers of software in the Federal Republic. About 85% of them have a capital of less than DM 250.000, which means that most of them are not able to raise enough funds of investment in tools of software technology. But also the more powerful suppliers who would be able to invest have not, as a rule, reached a satisfactory level of know-how concerning software technology. Owing to this situation, the software industry will, in the medium term, not be able to make use of the existing growth opportunities, since its technological know-how and also its economic potential are limited.

A country like the Federal Republic which - because of its dependence on imports - must be able to offer technologically advanced products cannot afford to neglect important key industries. The situation I have just described almost inevitably calls for the government to initiate supporting measures.

Under 3 successive data processing programmes from 1967 to 1979, the Federal Government gave support to research and development activities for hardware and software. A total of about 3.5 thousand million was provided to support research and development in this field. The first programme focused on hardware, the second and third on software, with an increasing emphasis on small computers. Applications of data processing in many different areas were supported (e.g. in administration, medicine, process control, computer-aided design or education). By promoting the field of data processing during the years up to 1979/1980, many of the objectives of the data processing programmes, which were designed as a pump-priming measure, have been achieved. As a consequence, the financing in particular of future products and standard applications will now be left to industry.

Present support measures concentrate on existing deficiency areas in information technology. To give you an idea of these measures, let me cite - apart from software technology - a few other important areas in which support is given predominantly to basic research work and systems know-how:

- data security
- information technology for office and administration tasks
- very-large-scale integration of electronic systems
- fundamental principles of systems engineering
- pattern recognition
- communications technology.

I do not want to go into greater detail. For those who are interested, let me point out that the support measures proposed are always announced in the Bundesanzeiger, the official organ of the Federal Government.

From the above remarks on the software crisis it is immediately obvious that much remains to be done in the field of software technology. This need is certainly not controversial. It is also, I think generally accepted that the quality of software can be improved, above all, if a systematic approach is developed and introduced for the problem definition and the design phases. But opinions differ when it comes to deciding which methods are the most efficient, since this quantitative efficiency is analysed or proved only very rarely; which means often beliefs are stated in the absence of evidence.

It would be just fine if the whole software life cycle could be covered by a few methods. But this is wishful thinking and hence unrealistic. One of the conclusions of the abovementioned study by the GMD is that the development of methods cannot be separated from the individual products to be developed.

The means of expression for design and testing certainly are - other methods may be - dependent on the kind of product to be developed (e.g. micro-systems, compilers). Nevertheless, there are certainly also a number of methods which lend themselves to general use. The variety of techniques, methods and tools required is and will be large - at least judging by the present state of the art and by present know-how. A policy aiming at the advancement of software technology and its use in the Federal Republic of Germany has to take this fact into account and consequently has to support a relatively large number of means of software technology before strategies can emerge which it would be promising to pursue with emphasis and in cooperation between several parties. We are focusing our support measures on areas where we believe the standard of know-how is particularly low. These areas are:

+ Methodology

Existing methods and tools are at present often used for only one or several software development phases. As development work proceeds to the next phase, extensive modification operations and manual adaption work are usually necessary, which would not be required if we had a coordinated and integrated system of methods. Whether eventually a system can indeed be developed which would be able to support the whole software life cycle from the requirement analysis to the maintenance phase without intermediate manual work is, I think, open to doubt. It is clear, however, that methodology is at present an area with quite considerable deficiencies. And since even relatively little progress can be expected to yield significant growth in productivity and improvement of quality, about 60 per cent of the funds available for software technology are earmarked for this area.

Individual methods in deficiency areas

- Description of requirements / Software design

Although some methods do exist, they are not all particularly satisfactory. In this area, we give specific support to graphical methods, because they seem to us especially promising for an engineering approach. The Petri networks may play an important role one day in this connection.

- Quality assurance

in particular testing. In addition to testing the code, testing (also by machines) during the first phases of the software life cycle will be of increasing importance, in particular because errors in the description of requirements and in software design are particularly expensive errors, since they are usually detected at a late stage and following extensive searches.

It is hoped that two projects which we support and which are dedicated to programme verification, i.e. proving that the code contains no errors, will be successful. If a real breakthrough is achieved here, we shall feel far less concern than before at the thought of using software also in sensitive areas.

- Maintenance

50% to 80% of the cost of a software product during the entire period of its use are caused by maintenance. Improved maintenance will be achieved automatically when better methods are used for the initial phase of the software development. But apart from that, improved methods of maintenance must also be developed. We are giving special support to activities for the development of remote diagnosis and maintenance.

Evaluation of methods

As I said before, this is an issue of special interest to the user. The trouble is that the situation with regard to methods evaluation is particularly gloomy. There are practically no measuring methods and techniques which could yield quantifiable and meaningful evaluative results. The projects we are supporting do not at present include any such activities.

One of the requirements we have fixed for all the projects we promote is that the processes and methods can be used with computer support. If they cannot be used in this way we are afraid they will mostly not be a long-term success in practice.

A second reason for this requirement is the fact that a method is not mature, logical and consistent until it can be programmed. The work we support is intended to achieve progress in software technology. The development results must be largely portable so as to facilitate more general application. Developments which are of value first and foremost to one interested party only, will not receive support. In the case of private enterprises it is our general policy to bear only 50% of the project costs.

6. Supportive measures by the European Communities

I shall now say a few words about support measures by the EC.

In 1979, The Council of the European Communities adapted a programme of supportive measures for the field of data processing covering the period from 1979 to 1983. Over these four years, the programme will provide 25 million European Accounting Units. The programme is subdivided into two parts:

a) General action

Dealing with standardization, public procurement, cooperation between research centers and organizations which advance the use of data processing, studies on employment issues; data security and data protection as well as the legal protection of computer programmes.

b) software and applications

In the field of general software, the EC Commission gives support to projects aiming at:

- the establishment and dissemination of standards
- improved portability
- improved conversion conditions
- improved efficiency of data processing systems.

The supportive measures in the field of applications are not limited to certain fields, but they have to comply with specific criteria, which I do not, however, wish to

enumerate here. The programme's terms of reference are so general that a relatively wide range of tasks can be included. More specifically, software technology projects are also eligible for promotion. In order not to raise false hopes, let me point out to you that all projects to be supported by the EC must comply with the requirements of advancing cooperation within the EC, which means that they must be carried out jointly by institutions in at least 2 EC countries. The measures supported by the Commission are announced annually in the official Gazette of the European Communities.

In addition to the steps taken by the EC Commission, other European countries, besides the Federal Republic of Germany, also have initiated action for the support of information technology. These measures differ very much from each other so that it would take too much time to go into further detail. Generally speaking, it can be said, I think, that small EC countries (e.g. Denmark, the Netherlands, Belgium) do not give financial support to industrial information technology activities, whereas larger countries like the United Kingdom and France provide quite considerable funds for this purpose. The responsibility for national support measures is assigned to different government departments in different countries, often to the department of industry, research or education.

7. Considerations on a future coordinated system technology

Before - in conclusion - trying to answer the question asked in the title of my paper, let me draw your attention to a technology of which the software technology is only one aspect. Hardware and software (by software I mean both operating systems and user software) were poorly coordinated from the first years of the great increase of computer use. As hardware development raced ahead, leaving software a long way behind, the gap between the two grew wider and will even increase as new hardware architectures are developed. The existing programming languages and techniques would make only very effective use of the possibilities offered by hardware. Consequently, it is an increasingly urgent task to make hardware and software development converge into a coordinated system technology.

8. Conclusions

To summarize, let me emphasize that the software crisis must be overcome as soon as possible. As I said before, some DM 10 thousand million are spent annually on software in the Federal Republic of Germany. Successful rationalization in software, production which reduces in particular the amount of maintenance required will soon result in major economies. If the software crisis is not overcome, the majority of staff engaged in software production would spend all their time on maintenance work and therefore be unable to tackle urgent new tasks. It is frightening to think that a very large portion of the qualified manpower engaged in data processing - which is in short supply anyway - is not available for innovation work which is urgently required by the economy. The need to reduce the great dependence and vulnerability caused by information technology by improving the structure and thus the manageability of software is another aspect which is of interest to the society as a whole.

The software producer is at present in a rather favourable situation compared with other producers. This is due on the one hand, to the extremely high demand for software, which can hardly be met, and, on the other hand, to the fact that for many customers software is still a book with seven seals, so that they have to fully rely on the producers. I am sure that this situation will, however, change during the next few years. Users will become increasingly aware of the problems involved in information technology and will endeavour to acquire at least some basic knowledge. They will then look at software with a more critical eye and will - more often than in the past - demand better quality. Software producers who are not able to meet such requirements will not be able to hold their market position.

The availability of the tools of software technology will then be crucial prerequisite for the survival of a company. Since software technology cannot, however, be introduced overnight, but probably only in the course of several years, software producers would be wise to begin immediately to tackle this task. Difficult as this task may seem in the face of such a multitude of methods, it should not be postponed, since any delay jeopardizes future opportunities.

Thank you for your attention.

BUSINESS COMPUTER GROUP STRATEGY

K.D. LAIDIG
W. GAMM

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

K.D. LAIDIG
W. GAMM
HEWLETT PACKARD

ANSI COBOL 198X: The Story behind the Headlines

Greg Gloss
Hewlett-Packard Information Networks Division

ANSI COBOL 198X: THE STORY BEHIND THE HEADLINES

GREG GLOSS

HEWLETT - PACKARD INFORMATION NETWORKS DIVISION

ABSTRACT

The ANSI (American National Standards Institute) X3J4 technical committee is in the final stages of its work on the next version of the COBOL standard. This paper will discuss some of the major new features which are expected to be included such as structured programming constructs together with those items which will no longer be allowed in the next standard. Some background explaining the reason the committee chose to make some of the changes will be included. In addition, the standardization process will be covered briefly.

BACKGROUND

The current version of ANSI COBOL was adopted in 1974. Since 1977, the ANSI X3J4 committee has been working on the next version of the COBOL standard. Since it is not clear how long this process will take, I will refer to the next standard as COBOL '8X.

OVERVIEW

The next standard will have changes in the following categories:

- a. New Features
- b. Transitional Features
- c. Deleted Features
- d. Specification Changes
- e. New Reserved Words

A significant effort has been put into incorporating structured programming constructs into COBOL. In addition, other new facilities have been added to make programming in COBOL easier. Some current features have been flagged for deletion either in the new standard or in the subsequent standard. Those features which are in the new standard, but which are not expected to be in the subsequent standard are called transitional. There have also been some changes to the rules and additional reserved words included which may affect existing programs.

STRUCTURED PROGRAMMING

The new structured programming constructs which have been defined for COBOL include Scope Terminators, nested programs, PERFORM statement enhancements, the EVALUATE statement, and the CONTINUE statement.

Scope Terminators

Under COBOL '74, conditional statements could not be included with the statement group following a conditional phrase such as AT END or ON SIZE ERROR. New reserved words have been added such that any conditional statement can be turned into an imperative statement and used as part of the conditional statement group. For example,

```
READ FILE-IN AT END
  ADD A TO B ON SIZE ERROR
  PERFORM OVERFLOW-ROUTINE
END-ADD
MOVE SPACES TO REC-IN.
```

Under COBOL '74, it is not legal to specify the ON SIZE ERROR phrase in the above example because it turns the ADD statement into a conditional statement and only imperative statements are allowed following the AT END phrase. However, with the scope terminator, END-ADD, the ADD statement with the SIZE ERROR option becomes an imperative statement and is legal in this situation. The MOVE statement is the second imperative statement to be executed if the AT END branch is taken and the period terminates the READ statement. If the READ itself were nested under a conditional such as an IF, it would be terminated by an END-READ instead of the period.

Nested Programs

The nested program facility allows programs to be contained within other programs so that global data may be easily shared and the program structure and relationships specified. In the following example, program B is contained within program A.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  A.
ENVIRONMENT DIVISION.
DATA DIVISION.
  [Global Data Declarations]
PROCEDURE DIVISION.
  [Program A Procedure Division Statements]
IDENTIFICATION DIVISION.
PROGRAM-ID.  B.
ENVIRONMENT DIVISION.
DATA DIVISION.
  [Local Data Declarations]
PROCEDURE DIVISION.
  [Program B Procedure Division Statements]
END PROGRAM B.
END PROGRAM A.
```

Program A may call program B; however, program B cannot call program A. Program B can access data in program A which is declared as GLOBAL unless program B contains a local data item of the same name.

PERFORM Statement Enhancements

The PERFORM statement has been enhanced to allow a list of imperative statements to be embedded within the statement instead of paragraph names and to allow the programmer to specify whether the UNTIL conditions are to be tested before or after the specified set of statements has been executed.

An example of an in-line PERFORM is shown below:

```
PERFORM 10 TIMES
  ADD A TO B
  ADD 1 TO A
END-PERFORM.
```

The two ADD statements will be executed 10 times.

Under COBOL '74, the UNTIL conditions are always tested before executing the specified paragraphs. The new specifications will allow the test to be made afterwards. For example,

```
PERFORM READ-LOOP
  WITH TEST AFTER
  UNTIL EOF-FLAG.
```

Control will always transfer to READ-LOOP at least once. The test option may also be specified with an in-line PERFORM.

EVALUATE Statement

The EVALUATE statement adds a multi-condition CASE construct to COBOL. The EVALUATE statement causes a set of subjects to be evaluated and compared with a set of objects. If the comparisons are all true, a specified group of statements is executed. For example,

```
EVALUATE HOURS-WORKED EXEMPT
  WHEN 0          ANY PERFORM NO-PAY
  WHEN 1 THRU 40 ANY PERFORM REG-PAY
  WHEN 41 THRU 80 "N" PERFORM OVERTIME-PAY
  WHEN 41 THRU 80 "Y" PERFORM REG-PAY
  WHEN OTHER     PERFORM PAY-ERROR.
```

The above example evaluates two data items, HOURS-WORKED and EXEMPT. If HOURS-WORKED is 0, any value for EXEMPT will be true and NO-PAY will be performed. If HOURS-WORKED is between 1 and 40, REG-PAY will be performed. If HOURS-WORKED is between

41 and 80 and EXEMPT contains "N", OVERTIME-PAY will be performed. If HOURS-WORKED is between 41 and 80 and EXEMPT contains a "Y", REG-PAY is performed. If none of the above conditions are true, PAY-ERROR is executed.

CONTINUE Statement

The CONTINUE statement is a no operation statement which indicates that no executable statement is present. It may be used anywhere a conditional statement or an imperative statement may be used. For example,

```
IF A < B THEN
  IF A < C THEN
    CONTINUE
  ELSE
    MOVE ZERO TO A
  END-IF
  ADD B TO C.
  SUBTRACT C FROM D.
```

The CONTINUE statement allows control to go to the ADD statement following the IF when A is less than C. If the NEXT SENTENCE option had been used, control would have transferred to the SUBTRACT statement instead.

OTHER NEW FEATURES

There is a long list of other new features which should make the job of the COBOL programmer easier. The more significant ones are listed here.

Reference Modification

Reference modification allows you to reference a portion of a data item by specifying a leftmost character position and a length. For example,

```
MOVE A (3:5) TO B.
```

will move the third through seventh characters of A to B.

INITIALIZE Statement

The INITIALIZE statement provides the ability to set selected types of data fields to predetermined values. Assume RECORD-1 was described as follows:

```
01 RECORD-1.
   05 EMP-NO    PIC 9(6).
   05 EMP-NAME  PIC X(20).
   05 EMP-PAY   PIC 9(5)V99.
```

```
05 JOB-TITLE PIC X(20).
```

The following INITIALIZE statements in the Procedure Division could be used to put values into the record:

```
INITIALIZE RECORD-1 REPLACING NUMERIC BY ZERO.
INITIALIZE RECORD-1 REPLACING ALPHANUMERIC BY SPACES.
```

The effect would be the same as:

```
MOVE ZERO TO EMP-NO EMP-PAY.
MOVE SPACES TO EMP-NAME JOB-TITLE.
```

De-editing

Under COBOL '74, it is not legal to move from an edited field to a numeric or numeric edited field. The new specifications will allow moving from a numeric edited item to either a numeric or numeric edited item. The edited item which is the sending item will be converted to its numeric value and moved to the receiving field.

REPLACE Statement

The REPLACE statement function is similar to that of a COPY... REPLACING except that the REPLACE statement operates on all source program text, not just text in libraries. Thus, if one of the new reserved words is used heavily in an existing program, you may want to use a REPLACE statement to change it. For example,

```
REPLACE ==TEST== BY ==TESTT==
```

will replace all subsequent occurrences of TEST by TESTT in the source program until another REPLACE statement, a REPLACE OFF statement, or the end of the source program.

Optional FILLER

The word FILLER is now optional for data items which need not be named.

```
01 A.
   05 B PIC X(5).
   05 PIC X(5) VALUE "NAME:".
```

INITIAL Attribute

The INITIAL clause in the PROGRAM-ID paragraph indicates that every time the program is called, the internal data is initialized. This function is the same as the \$CONTROL DYNAMIC option on the HP-3000.

PROGRAM-ID. SUB-PROG INITIAL.

EXTERNAL Attribute

The EXTERNAL clause specifies that a data item or file is available to every program in the run unit which describes the data item or file.

FD FILE-1 IS EXTERNAL.

SYMBOLIC CHARACTERS Clause

The SYMBOLIC CHARACTERS clause in the SPECIAL-NAMES paragraph of the Environment Division allows the programmer to equate a name to a specific character. This feature can be useful for non-printable characters. For example,

SYMBOLIC CHARACTERS BELL IS 7, CARRIAGE-RETURN IS 13.

This clause would allow a MOVE statement such as

MOVE BELL TO A.

ADD Statement Enhancement

Under COBOL '74, the ADD statement allows either a TO or a GIVING format, but a statement of the form

ADD A TO B GIVING C

is not allowed. The new specifications will allow the TO before the last operand when the GIVING option is used.

Alphabetic Tests

Two new alphabetic class tests have been defined:

1. ALPHABETIC-UPPER will be true if the data item being tested contains only A-Z and spaces.
2. ALPHABETIC-LOWER will be true if the data item being tested contains only a-z and spaces.

SET Statement Enhancements

The SET statement has been enhanced to allow the setting of external switches either on or off and condition-names to true. For example, given the following declarations:

SW0 IS SWITCH-1

01 READ-FLAG PIC 9.
88 EOF-FLAG VALUE 1.

The following SET statements could be used:

SET SWITCH-1 TO ON.
SET EOF-FLAG TO TRUE.

The second SET statement is equivalent to:

MOVE 1 TO READ-FLAG.

TRANSITIONAL CATEGORY

There are some features of the current standard which are scheduled for a phased deletion. Implementations must still support these features in the new standard, but not in the subsequent standard.

One of the most visible areas of change in the transitional category is in the realignment of file related clauses. The Environment Division is intended for machine dependent functions and the Data Division for machine independent functions. However, the placing of some clauses in the current COBOL '74 standard does not conform to this concept. Therefore, certain clauses have been moved from the file control entry in the Environment Division to the file description entry in the Data Division and vice versa. The old locations are specified as transitional elements so implementations of the new standard must support programs which contain the clauses in either the old or the new locations. The following Environment Division clauses are included in the transitional category:

FILE-STATUS
RECORD KEY
ALTERNATE RECORD KEY
ACCESS MODE

The following Data Division clauses are included in the transitional category:

BLOCK CONTAINS
CODE-SET

The Identification Division paragraphs are included in the transitional category in favor of the more general comment facility (* in column 7). Part of the reason for this change is the problem with the use of the word COPY in these paragraphs. It is not clear whether COPY in a comment entry is intended to be a COPY statement or is merely part of the comment.

The INSPECT...TALLYING...REPLACING format of the INSPECT statement is included in the transitional category since the same function can be accomplished with two separate INSPECT statements.

DELETED FEATURES

The following features are not included in the next standard:

1. The ALTER statement.
2. The ENTER statement.
3. The MEMORY SIZE clause.

The ALTER statement is being deleted because it is widely accepted as poor programming practice which causes significant program maintenance problems. The ENTER statement and MEMORY SIZE clause are being deleted from the standard because they are primarily implementor defined functions which are not necessarily meaningful on all systems and are thus not portable. Implementations will still be allowed to support these three features as extensions to the standard.

OTHER CHANGES

New status code values for file errors are being defined. These codes will cover situations which violate the standard but for which no standard status code was defined. For example, trying to open an indexed file in a program which declares it to be a relative file.

The order of the steps in a multi-conditional PERFORM...VARYING statement has been changed. Under COBOL '74, the statement

```
PERFORM PAR-1 VARYING I FROM 1 BY 1 UNTIL I>10
      AFTER J FROM I BY 1 UNTIL J>10
```

would set I to 1 and vary J from 1 to 10 and then set J to 1, increment I to 2 and vary J until 10. The new specifications will increment I to 2 before setting J to I. Thus, on the second cycle, J will vary from 2 to 10 instead of 1 to 10 as under COBOL '74. The primary reason for this change is because this statement, as currently defined, has caused much confusion because it doesn't do what most people expect and is probably not used very much. Programmers who have attempted to use this statement to do a bubble sort have usually been surprised at the results.

The new reserved word ALPHABET is required in the alphabet clause of the SPECIAL-NAMES paragraph.

```
ALPHABET ASCII IS STANDARD-1.
```

This change was required because of the desire to minimize the portability problems caused by implementor-defined reserved words. Under the COBOL '74 standard, the implementor could reserve the words used for switches, alphabet-names, and output advancing controls. The new standard will not allow these words to be reserved. This change however caused a parsing problem in the SPECIAL-NAMES paragraph because it would not be clear whether a clause such as

```
SWO IS EBCDIC
```

was specifying that EBCDIC was the mnemonic-name for switch SWO or whether SWO was the mnemonic-name for alphabet EBCDIC. By requiring the word ALPHABET in a alphabet-name clause, the ambiguity is resolved.

RESERVED WORDS

The following new reserved words have been added:

ALPHABET	END-DELETE	END-UNSTRING
ALPHABETIC-LOWER	END-DIVIDE	END-WRITE
ALPHABETIC-UPPER	END-EVALUATE	EVALUATE
ALPHANUMERIC	END-IF	EXTERNAL
ALPHANUMERIC-EDITED	END-MULTIPLY	FALSE
ANY	END-PERFORM	GLOBAL
COMMON	END-READ	INITIALIZE
CONTENT	END-RECEIVE	NUMERIC-EDITED
CONTINUE	END-RETURN	OTHER
CONVERSION	END-REWRITE	PADDING
CONVERTING	END-SEARCH	REPLACE
DAY-OF-WEEK	END-START	STANDARD-2
END-ADD	END-STRING	TEST
END-CALL	END-SUBTRACT	TRUE
END-COMPUTE		

STANDARDIZATION PROCESS

There are two committees which work on defining COBOL. The CODASYL COBOL Committee has the responsibility of developing the language. The ANSI X3J4 committee has the responsibility of standardizing the language. When working on a new standard, X3J4 can adopt specifications from either the previous standard or from the Journal of Development which reflects the work of the CODASYL COBOL Committee. If there is a problem with the JOD specifications, X3J4 must either subset the specifications from the JOD so that the problem does not appear in the standard or request that CCC resolve the problem. Both committees have representatives from implementors, users, and government. X3J4 currently has 23 members and holds six 4-day meetings each year. The work on the next standard is nearing completion as the committee has achieved the necessary two-thirds vote on its

formal letter ballot to forward the document to its parent committee, X3. X3, in turn, votes to send it out for an official public comment period of at least four months. The X3J4 committee will review all comments received during this period. After all negative comments have been processed, the X3 committee votes on sending the draft proposed standard to ANSI to be formally processed as a new standard.

During the standard revision process, X3J4 has published information concerning its work in COBOL Information Bulletins. The latest one was CIB 19 which was published in May, 1980. Comments concerning the draft standard will be officially requested during the public review period; however, comments may be submitted earlier to:

Chairperson, X3J4
CBEMA
1828 L St. N.W.
Washington, D.C. 20036

SYSTEM PERFORMANCE AND OPTIMIZATION TECHNIQUES

FOR THE HP3000

JOHN HULME

APPLIED CYBERNETICS, INC.

SYSTEM PERFORMANCE AND OPTIMIZATION TECHNIQUES
FOR THE HP/3000

John Hulme
Applied Cybernetics, Inc. \
Los Gatos, California
(408) 356-7296

INTRODUCTION

The purpose of this paper is to introduce the reader to certain techniques which can improve system performance, throughput, and run-time efficiency on HP/3000 computers. These improvements will typically reduce response time substantially and generally increase data processing productivity.

This paper will not simply tell you what to do and what not to do. In many cases there are trade-offs involved and it is more important to understand the principles behind the techniques than the techniques themselves. And because analogies often help us to learn by giving us a new perspective, we will make use of a non-data-processing illustration.

SOME BASIC PRINCIPLES

The first thing to understand is that any given computer can execute a finite number of instructions in a fixed amount of time. When that theoretical limit is reached, no amount of tuning can "squeeze" extra instructions into the computer. For the most part, however, computers do not bog down because we ask them to do too much, but rather because we cause them to trip over themselves in the process of doing it.

This leads to the second important principle: At any moment the computer is either 1) doing productive work, 2) getting ready to do productive work, or 3) waiting on some external action before it can proceed with productive work. As a program is initiated, thereby causing a certain sequence of instructions to be executed, we will call the execution of those instructions "productive work". Whether the "productive work" is really necessary or not, and whether it is efficiently or inefficiently organized, are issues to be addressed later. But a more significant fact of computer life is that usually only a small percentage of the computer's time is spent executing application program instructions.

A CRUDE MODEL

To illustrate these principles, imagine a "library for the blind". The librarian sits behind the desk waiting for a blind person to walk into the library. This is the "waiting period". When the blind person arrives, the "getting ready" period begins. The blind person tells the librarian which book to retrieve and by one method or another the book is retrieved. The librarian now begins the "productive work" phase, reading to the blind person from the selected book. When the reading is completed, the librarian may return the book to the shelf or leave it on the desk. Then a new waiting period begins.

If the library is a busy one, we can imagine that one or more assistants might be hired to transport the books between the librarian's desk and the book shelves. Let's imagine that there is one assistant for each wing of the library. The librarian can do more productive work (reading to the patrons), spending less time getting ready (still looking things up in the card catalog, but now dealing with the assistants instead of transporting books). A new type of waiting is introduced, however: waiting for assistants to bring books back.

In this analogy, the librarian represents the computer's central processing unit (CPU), by which all the productive work is accomplished. Like our imaginary library, the HP/3000 has only one CPU. To improve throughput we must maximize the CPU's productive time.

Each patron represents a log-on session or job. The librarian's desk represents the computer's main memory. It is of a limited size, merely a workspace, in comparison to the stacks of book shelves which correspond to the mass storage devices. Finally, each assistant represents an i/o channel transferring data to and from disc, for example.

While illustrating some important concepts, this analogy does not accurately model the run-time environment of the HP/3000, or any other computer. How could we refine the model to make it more realistic?

THE MODEL REFINED

At the risk of distorting the human situation, let me suggest four refinements which make our model more nearly resemble the actual computer processes:

1. The "library" should be regarded as a collection of
 - a) read only instruction manuals and reference tables (programs and constants) and
 - b) numerous loose leaf volumes (files) containing sheets of current figures and data (records) which may be periodically replaced, revised, removed, or added to.
2. The "librarian's" job should be generalized to include any type of service that can be performed on the basis of preprinted instructions and supplied data.
3. The computer always deals with a copy of whatever is stored on the disc, and usually just a few records at a time. So let's imagine that instead of asking a library assistant to fetch a particular book, the librarian will specify a limited number of paragraphs or data sheets and will ask the assistant to bring a photocopy of the desired paragraphs (colored paper for instructions; white paper for data).
4. Because the processing speeds of a computer are so great, our model operates in slow-motion by comparison. Allowing that the librarian can do in one hour what an HP/3000 can do in one second (i.e., using the scale of one hour for each second), the assistant could handle 20 to 60 requests per hour, and the equivalent of a 60-word-per-minute typist could enter one character every 12 minutes. A 2400-baud rate would be equivalent to a maximum of 5 characters transmitted per minute, and a 600-line-per-minute printer would correspond to one line every 6 minutes.

SLOW-MOTION PERFORMANCE SIMULATION

Visualize this scenario from the patron's point-of view (refer to figure 1): You walk into the library, find an empty cubicle (terminal), and make yourself comfortable. You begin to formulate and transmit your library card number and password (log-on) at the rate of no more than 5 characters per hour. (If it will relieve the agony, you may imagine that you spend the time drawing very large, very elaborate block letters). Depending on the facilities available in the cubicle, you will either transmit each letter as it is formulated or accumulate several characters (maybe even hundreds) and transmit them in a burst. In either case, you transmit each letter separately by ringing a bell, and, when you have the librarian's attention, holding up the card with the letter on it. The librarian records each character of your message on a notepad corresponding to your cubicle, then continues with his other business. Finally you send a character which means "that's the end of what I'm sending you".

The librarian eventually verifies that you are a qualified user of the library and sends you back a standard message which allows you to proceed. This process may require the librarian to send his assistant to the book shelves several times, e.g., to get a procedures manual, index of users, table of passwords, welcome message, etc.

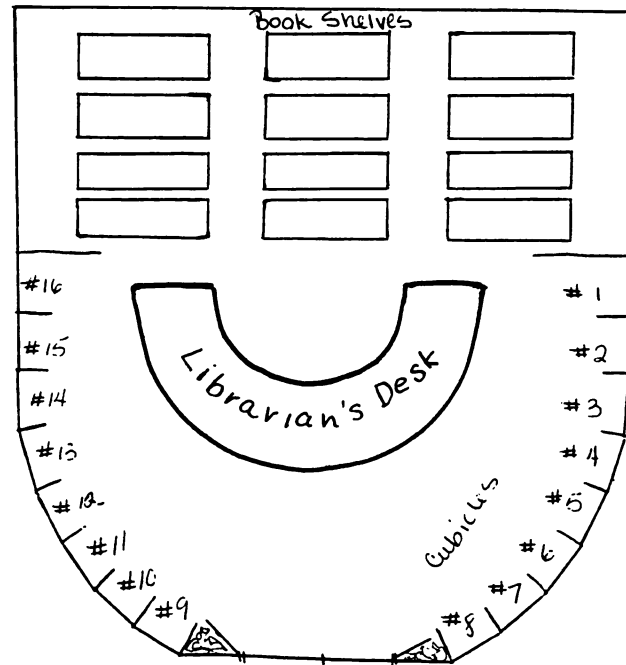


Figure 1. The Library

Next, you painstakingly tell the librarian the name of an instruction manual (program) you want him to follow in performing some service for you. He has the assistant get him a copy of the first paragraph (segment) of the instruction manual (unless a copy happens to be sitting somewhere on the desk already). He also gets a copy, your own personal copy, of a worksheet (your data stack) associated with the specific instruction manual you have specified.

In case there is not enough empty space on the desk for these papers, the librarian first clears some space by either a) throwing away one of the instruction sheets, b) having his assistant put the worksheet for some other patron in a special holding file (virtual memory), or c) having his assistant take one of the data sheets back to the loose-leaf it was copied from and replace the original with the new version.

The librarian now goes to work following the instructions you have requested. This will continue until a) he comes to a point in the instructions which specifies he is to send certain information to you and/or ask you for additional input; b) he comes to the end of the page or is otherwise instructed to refer to another page, one which is not currently on the desk; c) the instructions require that information be fetched from the book shelves, taken there to be filed, or sent to some output device; d) a predefined length of time elapses (a 500 microsecond quantum corresponds to one-half hour in our model); or e) the librarian completes his assignment and disposes of your worksheet.

In any of these cases, the librarian will go back to work for one of the other patrons, provided he has all the resources necessary to do so. If not, he will wait (until the necessary information is fetched by the assistant or transmitted by one of the patrons). Depending on what you've asked the librarian to do, and how busy he is doing things for the other patrons, it may take hours or even days before he gets back to you. But then again, it may take days for you to formulate the equivalent of one screen of input, too (at the rate of 5 characters per hour).

THROUGH THE EYES OF THE CPU

Now let's reverse roles and look at the situation from the librarian's perspective. Try to imagine yourself as a calm, unemotional, purely methodical being who is never responsible for mistakes because he does precisely as he is told. You couldn't care less if someone gets poor response time; you aren't to blame, because you only rest when there's nothing for you to do. In fact, you purposely set things aside during peak demand periods to do in your spare time. But you can't take credit for that either-- you're only following directions from the MPE handbook.

2:08:17 Ring! There's the bell in cubicle five. He's holding up the letter "R". Write it down on memo pad #5 (line buffer).

2:08:20 Here's the library assistant with the record session #12 requested. Oops! The worksheet for session #12 has been set aside (swapped out to the system disc). Send the assistant for it and wait a minute.

2:08:24 A ring from cubicle #8. That's a carriage return. Time to reinitiate session #8. Make a note to send the assistant for the worksheet when he gets back.

2:08:29 Wait some.

2:09:00 Wait some more.

2:09:16 Oh good, something to do (the observer's feelings, not yours). A ring from cubicle #3. A "7". Write it down.

2:09:20 Here's the assistant. Put worksheet #12 on the desk. Send him back for worksheet #8-- no, there's not room for it. Give him the worksheet for session #5 and send him to file it (we're waiting for input from cubicle #5). We'll send him for worksheet #8 next time.

2:09:24 Okay, now to get to work on task #12. First set the timer for 30 minutes. Now add I to J and put the result in K.

2:09:29 Move W6 to W2. Move...hold it, there's another ring from #3. Say, that's only a few seconds...must be a block-mode terminal. Write down the "9" and go back to work. Move X to Y. Call the procedure "XFORM". Oh, it's on the desk already-- it hardly ever gets thrown out, that's because nearly every program uses it.

2:09:40 Another ring from cubicle #3. This time it's a minus sign. Continue with "XFORM". Convert the first letter of Y to upper-case. Now the second letter. Now the third. Now the fourth. That's all. Return to the main program. It's still in memory. Move the new Y to P3.

2:09:52 Another ring from cubicle #3. A field separator. Resume task #12. Perform FLAG-SET subroutine. It's in another segment, one that's not in memory. Make a note to send for it. Suspend task #12 for a minute.

2:10:04 Cubicle #3 again. Just a blank, but write it down anyway. That's "7-9-minus-field separator-space", so far.

2:10:14 The assistant has finished filing worksheet #5. Send him now for worksheet #8.

2:10:16 Cubicle #3. Another space.

2:10:19 Interrupt from the printer saying the last line has printed successfully. Now reactivate the spooler job-- it's instructions are still on the desk and so is the buffer containing the print-line. Initiate i/o transfer.

2:10:26 2-second wait.

2:10:28 Cubicle #3. A third space.
12-second wait.

2:10:40 Cubicle #3. A fourth space.
12-second wait.

2:10:52 Cubicle #3. A fifth space.
12-second wait.

2:11:04 Cubicle #3. A field-separator.
5-second wait.

2:11:09 Worksheet #8 is here. Send assistant to get a copy of FLAG-SET routine. Now to process this input from cubicle #6.

Edit first field. OK. Edit second field. OK. Move first field to R1.

2:11:15 Cubicle #3. The letter "H".

Move second field to K2. Edit third field. Isn't numeric but should be. Transfer to error handler in same segment.

2:11:28 Cubicle #3. The letter "O".

Prepare output to tell cubicle #8 about error. Comment: It's a shame, but since he's in block-mode, he'll have to retransmit the whole screen again, after correcting the error in field 3. And who is to say other errors might not be detected after that? And you, the librarian, can receive those 873 characters, one every 12 seconds for nearly three hours. But you don't mind. It's only a job.

2:11:40 Cubicle #3. The letter "V".

Finish putting error message in the output buffer. Initiate transfer to cubicle #6. Mark task #8 eligible to be swapped out.

2:11:47 Cubicle #11. The letter "P".

2:11:52 Cubicle #3. The letter "E".

FLAG-SET routine is here. Continue with task #12. Move 1 to FLAG. Add 1 to COUNT. Exit back to mainline. What! The assistant had to fetch a separate segment just so we could do that?

2:11:59 Cubicle #11. Oh, oh. Two block-mode devices transmitting at once! Record the letter "I".

2:12:04 Cubicle #3. The letter "R".

Comment: Stop, I've had enough of dingy bells! This place sounds like a hotel lobby, not a library!

OBSERVATIONS

As this analogy indicates, there are three factors which reduce overall system performance:

- a. unnecessary disc i/o (most serious),
- b. unnecessary terminal i/o (too common), and
- c. unnecessary CPU usage (rarely the problem in an on-line environment).

EXCESSIVE DISC I/O

The primary cause of excessive disc i/o is inadequate main memory to hold the required work space (stack and data segments) for each concurrent process, plus all frequently referenced program segments, plus a reasonable mix of infrequently referenced program segments.

The HP/3000 is very good at handling multiple concurrent users, even when they won't all fit in memory together. In fact, the use of virtual memory, combined with a well-designed algorithm for selecting which segment to overlay, allows the system to operate efficiently even in cases where a single program exceeds the limits of main memory.

The thing to remember, however, is that code segments put a relatively small load on the system while data segments put a potentially disastrous load on the system. In the first place, code segments can be split up and made as small as the programmer wants them to be. Secondly, they do not have to be rewritten to virtual memory when the main memory space is to be re-used; they are simply overlaid. Data segments, on the other hand, tend to expand, and can be split only with difficulty. Since their contents may change, they must be rewritten each time the process is swapped out, and reread each time it is swapped back in. Finally, whatever data space is required must be repeated for each process that is active. Therefore, if you are supporting 20 terminals, any reduction in data requirements would produce 40 times the benefit that an equivalent reduction in code requirements would produce.

Aside from upgrading to a larger machine, a shortage of main memory can be averted by:

- a. reducing the number of concurrent processes (not an attractive option),
- b. reducing the average stack or data segment size,
- c. reducing the size of the average program segment,
- d. organizing program segments better so that out-of-segment transfers occur less often to non-resident segments and so that often-used code is collected in compact segments that are likely to stay in memory, or
- e. some combination of the above.

When adequate main memory is available, swapping is unnecessary, and disc accesses (which are very expensive in terms of time) will be expended strictly for data retrieval and storage. Once swapping begins, the computer's "productive" activities are at the mercy of "waiting". In the worst case, "thrashing" occurs, which means that every time a session gets a turn at execution, either the program segment has been overlaid or the session's work space has been swapped out.

It is worth noting that the use of IMAGE (or of KSAK) causes the allocation of extra data segments. Specifically, each IMAGE data base that is open requires a data segment large enough to hold one copy of the root file plus four complete data base buffers. If a program accesses multiple data bases, or if the root file or buffers are large, the memory requirements will be substantial, and with many terminals running data base applications, the memory requirements can add up very quickly. Granted, the advantages of using a powerful access method may outweigh the costs of additional memory demands, but such tools should be used carefully and not indiscriminantly.

It should also be noted that the use of block-mode requires extensive buffers in the stack (at least as large as the largest screen to be transmitted). The use of VIEW/3000 may add another 6000 bytes of buffer in each user's stack, not to mention the extra data segments created by its use of KSAK. If you have 20 users, this amounts to 120K extra bytes of memory or more.

EXCESSIVE TERMINAL I/O

Major causes of excessive terminal i/o include the following:

- a. Transmitting unnecessary characters (trailing spaces, leading zeroes, insignificant digits, etc.) to the computer, a necessary consequence of fixed-format or block-mode input.
- b. Transmitting the same data to the computer more than once, as occurs in block-mode when a whole screen is retransmitted to correct an error in a single field.
- c. Retransmitting to the computer data which has not been changed since it was received from the computer. This too is the result of block-mode transmission.
- d. Repeatedly displaying prompts at the terminal instead of using protected background forms.

Since each character of input consumes critical resources, every effort should be made to ensure that only significant data is transmitted (no extraneous zeroes or spaces and only those fields that are new or have been modified).

It is not only wasteful of computer power, but also destructive of operator morale, to wait until a whole screen of data has been entered and transmitted to the computer before discovering that the screen is invalid due to a duplicate key or an unrecognized search-item value, etc.

It is equally inefficient (for the computer, that is) to display a screen of data, have the operator update a single value and transmit the whole screen back to the computer. In an extreme case, this could amount to over a thousand characters transmitted just to change one or two characters.

EXCESSIVE CPU USAGE

Besides the costly i/o overhead, it is altogether possible that a retransmitted screen will be completely re-editted, values packed and unpacked, and fields reformatted even though only a single field was updated, and maybe even if nothing was updated. This is one cause of unnecessary CPU usage.

Most editing and reformatting done in COBOL subroutines requires excess usage to begin with, and it is far better to allow such work to be done in SPL subroutines, where it can be done efficiently. Including such subroutines in the COBOL programs also causes bulkier segments, which is likely to increase the need for swapping. The best solution is to incorporate all editing within the terminal-handling module itself, since it is already being shared by all on-line programs and is therefore likely to remain constantly in main memory. There are a multitude of factors which can unnecessarily increase the so-called "productive work" which the CPU has to do. Because computers are seldom CPU-bound in an on-line environment, few people exert the effort to truly optimize CPU performance anymore. Whenever it is a problem, more careful analysis of the program(s) in question will usually yield a more efficient method of solving the application problem.

Often, more careful analysis will also yield a better solution from the point of view of disc i/o as well, both in terms of swapping, code-segment switching, and data retrieval and storage. One word of warning, however: more efficient solutions (CPU-wise) are very often more complex, and to the extent that they increase stack space, or code-segment size, or they require more transfers from one code-segment to another, they may prove counter-productive.

One situation in which heavy CPU usage can be very detrimental is when on-line processes are competing with batch applications for CPU resources. This can be vividly illustrated by running a COBOL compile, an Editor GATHER ALL, a sort, or the BASIC interpreter at the same time on-line programs are running. Block-mode applications exhibit many of these same tendencies and can severely impede response-time for character-mode applications when both types are running concurrently.

SPECIFIC OPTIMIZATION TECHNIQUES

1. Resegment programs so that no segment exceeds %5000 words.
2. Set the blockmax parameter on IMAGE schemas as low as possible.
3. Use extra data segments where possible and free them up when finished, rather than increasing stack space for large temporary buffers.
4. Don't keep files open unnecessarily.
5. Don't abuse IMAGE:
 - a. eliminate sorted chains where possible.
 - b. carefully evaluate tradeoffs of increasing or eliminating secondary paths in detail data sets.
 - c. use "@"; or at least "*" for item lists wherever possible.
 - d. only use binary keys (in master file) when overlapping keys can be avoided.

- e. don't let synonym chains get very long.
 - f. when loading master data sets, store only primaries on the first pass, making a second pass for secondaries.
 - g. keep master data sets less than 85% filled.
 - h. periodically reorganize detail data sets that have long chains associated with a frequently-accessed path (puts consecutive records in the same physical block).
 - i. keep the number of data sets in a data base as small as practical without requiring many programs to open multiple data bases.
 - j. keep IMAGE record lengths to a minimum.
6. Have operators exit programs when not in use.
 7. Use a field-oriented terminal handler which performs standard edits for you.
 8. Use formatted screens with protected background whenever the application is appropriate to such use.
 9. Keep terminal i/o buffers small; if possible, eliminate block-mode i/o altogether. (Don't use block-mode and character-mode i/o at the same time.)
 10. Don't use VIEW without a lot of memory.
 11. Don't use DEL at all.
 12. Run CPU-intensive jobs (including compiles, preps, and Editor GATHER ALL) when on-line applications are not running, or at least run them in a lower-priority sub-queue.
 13. Set the system quantum for a shorter period than recommended in the MPE manual (but don't overdo it-- some experimentation may be necessary).



APPLIED CYBERNETICS INC.

Information Management Specialists

224 Camino del Cerro, Los Gatos, CA. 95030
408 356-7296

Production Management/3000 overview

The efficient management of production resources is made possible using the techniques of Capacity Requirements Planning (CRP) and Shop Floor Control. These techniques build upon a manufacturer's production schedule, an accurate description of the facility, and the steps necessary to fabricate or assemble each part; as well as information about the status and location of each work order. An effective method of dealing with these fundamental issues can provide a manufacturer with a sound foundation for success.

Production planning and control system

Production Management/3000 is a standard application product which helps manage the planning and control functions of a discrete manufacturing company. The objectives of the product are to minimize inventory investment while maximizing asset utilization, shipment performance, and customer satisfaction.

Production Management/3000 is a fully integrated system and addresses the following areas:

Routings and Workcenters—Maintains descriptive, cost, and planning information about each workcenter in a manufacturer's production facility and the routing sequence necessary to build each assembled or fabricated part.

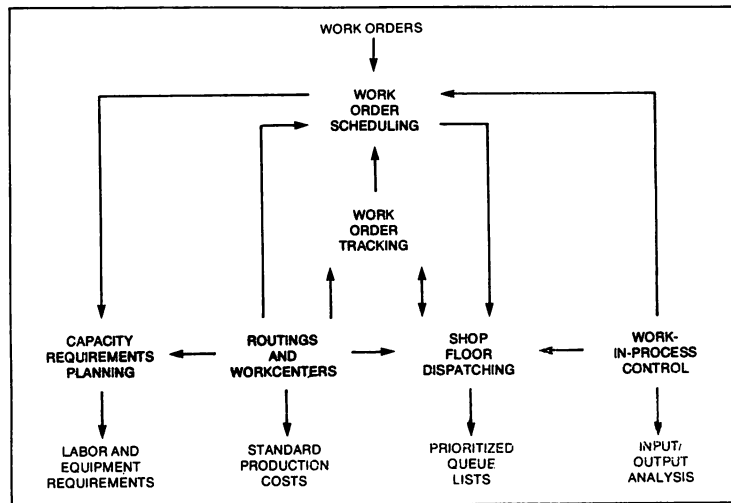
Work-In-Process Control—Provides a variety of tools to analyze the progress of the manufacturing plan and allows production managers to fine-tune the shop floor control process.

Work Order Scheduling—Calculates start and completion dates for each sequence of every production work order under the control of Production Management/3000.

Shop Floor Dispatching—Maintains production priorities to ensure that manufacturing resources are devoted to the right tasks at all times.

Work Order Tracking—Records the progress of each work order as well as related labor charges and exception information.

Capacity Requirements Planning—Anticipates labor and other manufacturing resource requirements on the basis of in-process production work orders and/or planned work orders from a Material Requirements Planning system.



Production Management/3000

PRODUCTION MANAGEMENT / 3000

Wolfgang Bayer
Product Manager

Commercial Applications Software
Böblingen General Systems Division

Hewlett-Packard GmbH
Herrenberger Str. 130
7030 Böblingen, West Germany

The shop calendar

An important factor in the creation of a detailed production plan is a definition of the work schedule for each production area. Production Management/3000 maintains a *shop calendar* which describes the work-day and shift schedules for each workcenter in the manufacturing facility. This calendar can be maintained on-line by shop management and can be modified at any time to reflect changes to the work schedule as soon as they are planned. The Production Management/3000 shop calendar describes schedules for up to three shifts per day, for as many months as necessary.

In addition to a master shop calendar, a separate calendar can be created for any individual workcenter. All Production Management/3000 scheduling and planning functions refer to the same set of shop calendars, providing a consistent basis for detailed production planning and a simple means of indicating holidays and planned work schedules.

Describing the manufacturing process

Part number definition

Each manufactured part, subassembly, and product is assigned a unique *part number*. This number is used throughout Production Management/3000 to access, update, and retrieve information about each item. Part information maintained in Production Management/3000 includes: standard yield percentage, standard run quantity, and part description. If desired, additional data items can be added through the use of the Production Management/3000 customization feature.

Defining production operations

Each basic manufacturing process can be described separately within Production Management/3000 and is assigned a unique *operation number*. Information such as average queue, setup, and unit run times as well as standard manufacturing yield is maintained for each operation along with a brief description of the procedure (i.e., drill, paint, test, etc.).

These operations form the basis for describing the individual steps involved in the manufacturing process for each manufactured part number under the control of Production Management/3000.

Add Work Station Detail				ADD WSTN
Chg Mstr	Del Mstr	Rev Mstr	Chg Mstr for Mstr	Default Values EXIT
Work Station Id	Work Station Description		Work Center Id	
WCDPILL	NUMERICAL DRILL STATION		METAL	
Capacity	14.00 hours/day	Rate	Energy 8000.00 units/hr	
Machine	24.00 hours/day	Start Up	1500.00 units	
		Unit Measure	MT	
Date Prev Breakdown	027981	Date Prev Preventive Maintenance	080181	
	mddyy		mddyy	
Queue Time Average	4.00 hrs		Work Station Status 1	
Queue Compression %	50		(Status < 1000 means available)	

Defining workstations

Basis of CAP exception reporting

Defines standard waiting time at this workstation

Routings and Workcenters

Features

- On-line data base update.
- On-line review of routing and workcenter information.
- Standard, common, and alternate routings.
- Capacity specifications of workcenters.
- Cost specifications of workcenters.
- Shop calendars.
- Standard labor and overhead cost calculation.

Description

An accurate model of the manufacturing facility and the procedures used to manufacture each product is essential to all production planning and control functions. The ROUTINGS AND WORKCENTERS module provides for on-line entry, validation, and maintenance of the information which forms the foundation for the operation of Production Management/3000.

The capability to define and maintain standard part routings and manufacturing workcenter descriptions is provided both in Materials Management/3000 and in Production Management/3000. In situations where both Materials Management/3000 and Production Management/3000 are employed together, all facility and routing specifications will be defined and maintained through Production Management/3000. In such cases, the ROUTINGS AND WORKCENTERS module of Materials Management/3000 will be disabled. Standard labor and overhead cost information will be provided to the STANDARD PRODUCT COSTING module of Materials Management/3000 from Production Management/3000.

For customers who choose to install Production Management/3000 after Materials Management/3000 is already in operation, information defined in the ROUTINGS AND WORKCENTERS module of Materials Management/3000 can be transferred to the ROUTINGS AND WORKCENTERS module of Production Management/3000 as a part of the normal system installation procedure.

Describing the manufacturing facility

Production Management/3000 views the manufacturing facility as a collection of *workcenters*, each of which shares common personnel, supervision, and/or floor space. Workcenters, in turn, are comprised of one or more *workstations* which represent labor or equipment used to perform each type of production activity. Depending upon the nature of each manufacturing facility,

individual people and/or pieces of equipment can be represented by separate workstations, or those which are more or less interchangeable with one another can be grouped together.

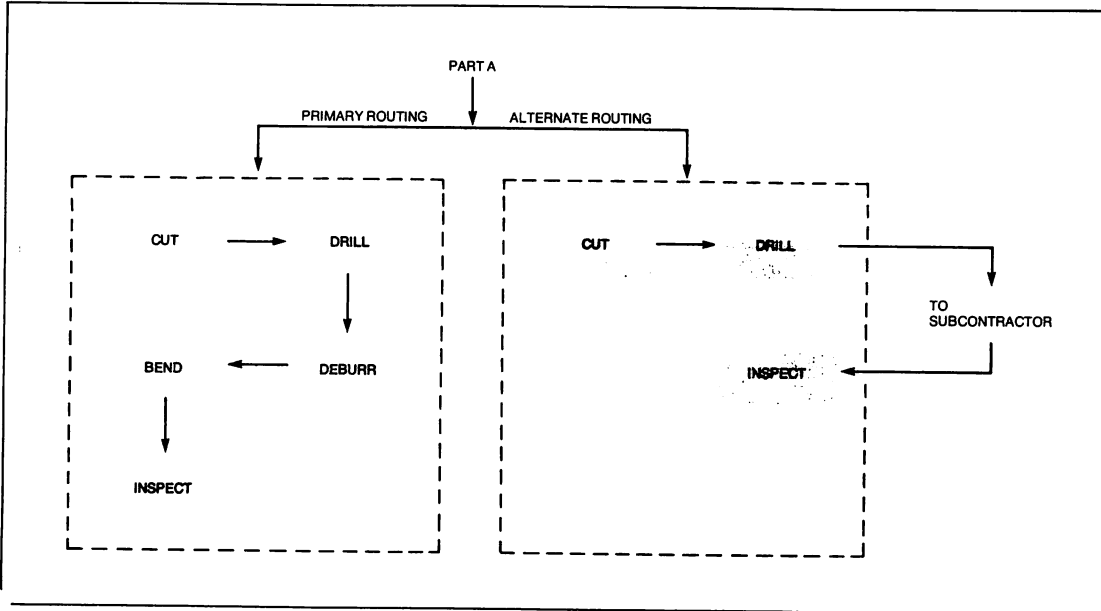
A workstation may be thought of as a location at which people and/or machines perform a particular type of activity needed to produce each part, assembly, or product. Information maintained for each workstation includes descriptive and control information such as labor and equipment capacity, average queue delay time, and the percentage of the normal queue which can be compressed for high priority work.

Additional information including average wage rates, production capacity, and the names of responsible managers can be maintained at the workcenter level.

The information maintained for each workstation and workcenter is essential for realistic work order scheduling and tracking, as well as for the calculation of accurate standard labor and overhead cost information.

Alternate part routings

Alternate routings designed to take advantage of various equipment configurations or subcontractors can be defined in advance. Each alternate routing specified for a part must be complete and is independent of all other routings. When work orders are actually created, any of the specified routings can be selected.



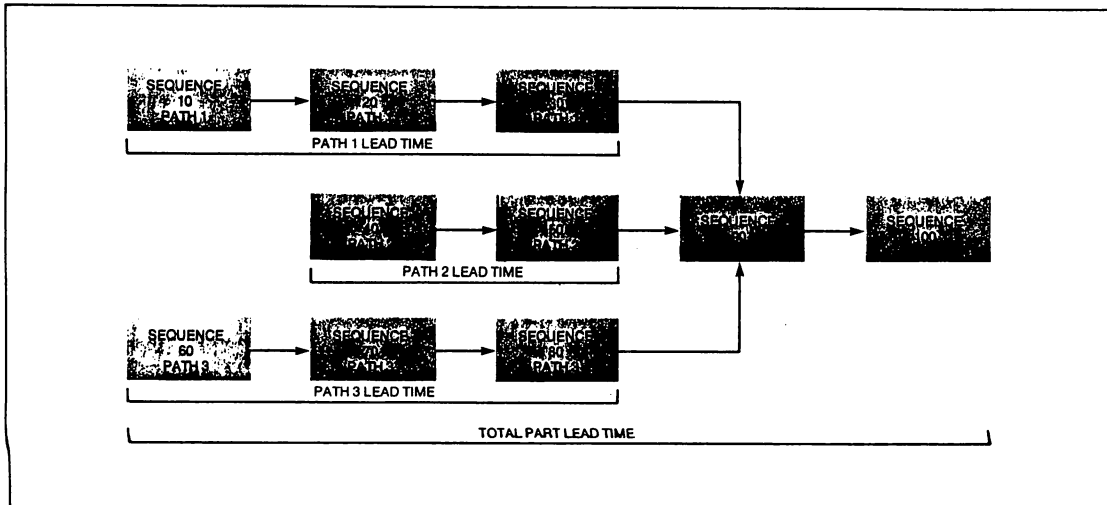
Alternate routings

Parallel routing sequences

Production Management/3000 provides the capability to schedule and control multiple parallel production paths for each individual part number. This allows related subassemblies to be manufactured on a single production work order and then to be combined without using intermediate inventory levels. Any number of parallel paths can be specified for each part, and each can include as many production

sequences as necessary. Each production path is identified by a user-defined parallel code attached to each routing sequence, and by a specification of the sequence at which paths "join" for assembly.

Production Management/3000 provides for the scheduling and control of parallel production paths to minimize overall manufacturing lead time, and to keep work-in-process inventory levels to a minimum. In certain types of manufacturing environments, the use of parallel production paths can result in significant reductions in inventory carrying costs.



Parallel routing paths

Employee information

Employee information in Production Management/3000 is used primarily for the validation of direct labor charges. An *employee identification code* must be assigned to each employee who will be charging labor time to work orders. As labor is reported, Production Management/3000 verifies that a valid ID code is entered with each transaction. Additional information such as location code and overhead account can be maintained for each employee, for use as reference information.

Standard Product Costing

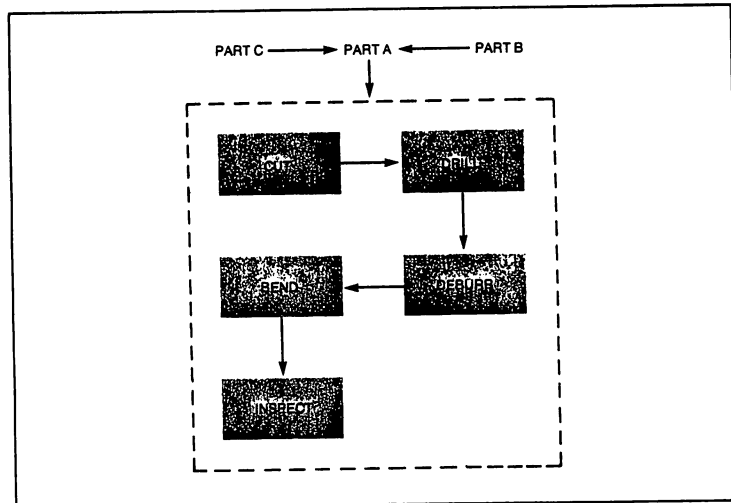
Production Management/3000 calculates standard labor and overhead costs for each part under its control. These costs are calculated using the run quantity for each part, labor setup and run times from the standard

routing, and overhead and average wage rates from each workcenter. If more than one routing exists for any part, the primary routing is used (i.e., any alternate routings are ignored). This information is available to the STANDARD PRODUCT COSTING module of Materials Management/3000 for use in the standard cost roll-up. For a detailed discussion of STANDARD PRODUCT COSTING, please refer to the chapter describing Materials Management/3000 earlier in this document.

Common part routings

Production Management/3000 allows any part to use the routing defined for any other part as if it were its own. All Production Management/3000 functions can automatically refer to the *common* routings for such a part. This feature eliminates the need to define identical routings for each of a number of similar parts which utilize the same basic manufacturing process.

An additional advantage of this approach is that when the common routing changes in any way, only one *master* routing needs to be modified in order to effect the change for all parts which refer to it. This can represent a substantial reduction in effort and can help to ensure the consistency of manufacturing specifications.



Common routings

Input/Output analysis

The Input/Output analysis report provides a means of evaluating recent performance both in absolute terms, and in the light of previously planned production requirements. *Planned* input and output are presented alongside actual or *observed* input and output for each recent time period, and cumulative changes in backlog are calculated for easy analysis. Early

identification of production bottlenecks through the use of Input/Output analysis allows corrective action to be taken before the overall manufacturing plan can be seriously effected.

-PERIOD-		-INPUT-				-OUTPUT-				-BACKLOG-	
DATE	PLAN	OBSD	DEV	CUM.DEV	PLAN	OBSD	DEV	CUM.DEV	OBSD	PLAN	
03/24/81	16.0	14.0	-2.0	-2.0	16.0	15.5	-1.5	-1.5	42.6		
03/26/81	13.5	13.5	.0	-2.0	16.0	15.5	-.5	-2.0	40.6		
03/28/81	17.0	20.0	3.0	1.0	16.0	17.0	1.0	-1.0	43.6		
03/30/81	16.0	12.5	-3.5	-2.5	17.0	12.5	-4.5	-5.5	43.6		
04/01/81	16.0	11.5	-4.5	-7.0	17.0	13.5	-3.5	-9.0	41.6		
04/03/81	16.0				17.0					40.6	
04/05/81	16.0				17.0					39.6	
04/07/81	14.0				17.0					36.6	

Standard labor hours scheduled to arrive at this workstation

Standard labor hours actually arriving at this workstation

Cumulative deviation from the production plan at this workstation

LEGEND: '<' = Out of Tolerance

Input/Output report

Work-In-Process Control

Features

- Input/Output analysis.
- On-line work order status and history reporting.
- Work order priority control.
- Work order partialling.
- Work order routing modification.
- Data archiving and retrieval.

Description

WORK-IN-PROCESS CONTROL provides a set of reports and functions which allow production managers to monitor the progress of the manufacturing plan and to override the automatic scheduling and control functions of Production Management/3000 in order to accommodate special situations.

When factors beyond the scope of automatic planning and control arise, Production Management/3000 allows priority and routing decisions to be included directly into the manufacturing plan. This capability removes the need to "work around the system" in special situations; thus integrating the best of automatic scheduling and control with management discretion.

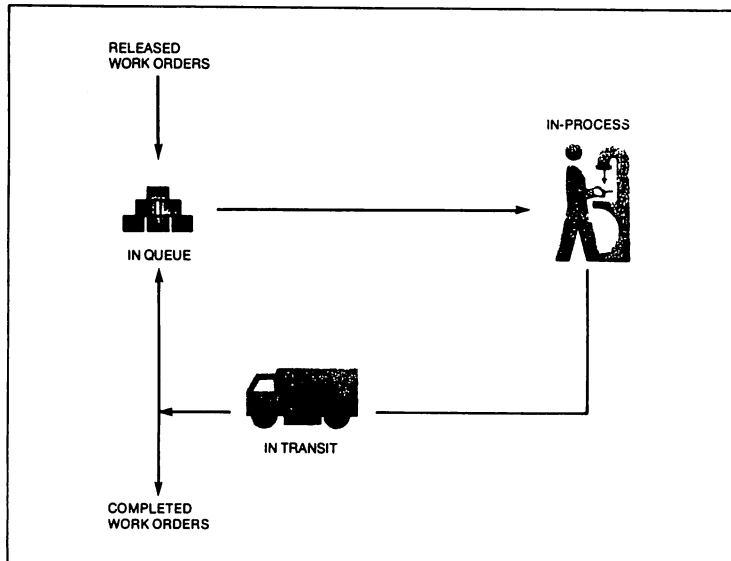
Life cycle of a work order

All planning and control functions of Production Management/3000 revolve around individual production *work orders*. Taken together, all planned and open work orders constitute the current *production schedule*. In

order to ensure that the detailed production schedule supports the due dates of all known requirements, Production Management/3000 assumes that infinite production capacity is available at all times. CAPACITY REQUIREMENTS PLANNING evaluates all such requirements over time in order to identify schedules which call for more capacity than is actually available. By evaluating the

resource requirements stemming from each work order, *capacity requirements* and *production priorities* can be determined so that specific actions can be recommended by Production Management/3000.

Each Production Management/3000 function uses the current production schedule in a coordinated way as individual work orders proceed through the following life cycle:

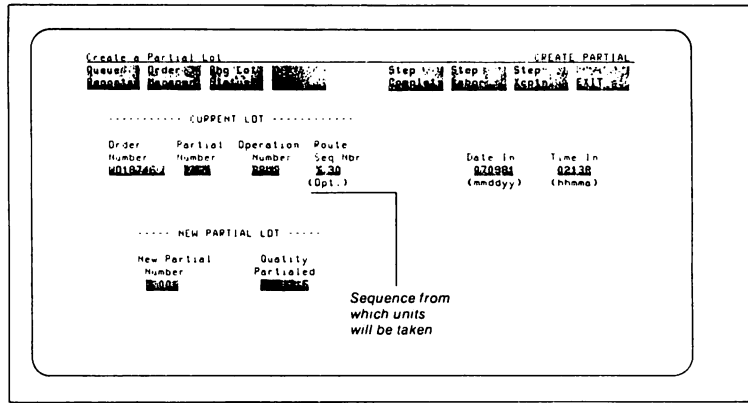


Production life cycle

Creating partial work orders

Production Management/3000 provides for the separation of work orders into multiple *partial work orders* at any time. Partial work orders are created by specifying the number of units from the *parent* work order which are to be separated, and the production sequence from which they are to be taken. A separate copy of the work order is then automatically created which has a routing identical to that of the parent.

Once created, partial work orders become independent of their parent work orders and can be assigned separate due dates and/or priorities. For accounting and material planning purposes, all *related* work orders carry the same order number and are distinguished from one another on the basis of the *partial number* assigned to each. Any number of partials can be associated with a single work order number, and partial work orders can be further sub-divided if necessary.



Creating a partial order

Labor reporting and maintenance

Direct labor hours reported on the shop floor can be reviewed by employee, work order, or workcenter. Modifications can be applied to any individual charge, if necessary, prior to the utilization of these records for job costing or other reporting purposes.

Work order status reporting

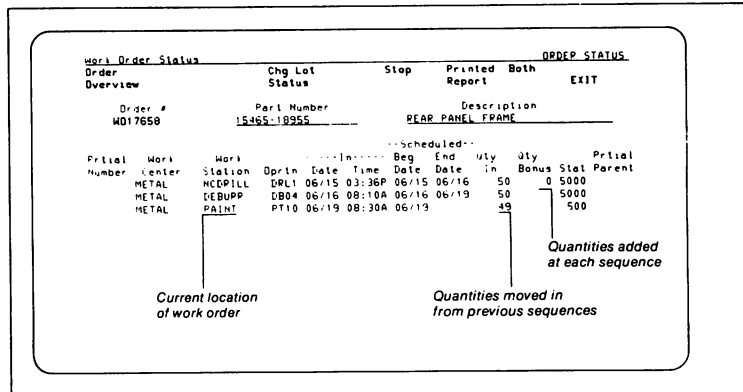
The current location and status of each work order in process is available at all times for on-line analysis and review. Work-in-process status can be reported either on CRT screens or printed reports by:

- Individual work order.
- Workstation or workcenter.
- Operation number.
- Part number.

Order routing modification

Production Management/3000 provides the capability to modify the *order routing* for any single work order to accommodate special situations (such as subcontracting) without changing the *standard routing* for the associated part. Such changes are applied to the copy of the routing which is made at the time each work order is *scheduled* (see the section on WORK ORDER SCHEDULING later in this chapter).

Existing *production sequences* can be modified or deleted and new production sequences can be added at any point in the routing. Work orders which are modified in this manner are immediately rescheduled in order to ensure that all changes are accurately reflected in the detailed production schedule.



Reporting work order status

Infinite capacity assumption

For purposes of work order scheduling, Production Management/3000 assumes that infinite production capacity is available at all times. Each work order is scheduled independently and requirements in excess of available capacity are then identified by the CAPACITY REQUIREMENTS PLANNING module (see the discussion of CRP later in this chapter). This procedure ensures that detailed production schedules will support the overall master schedule, and that modifications to the plan based on capacity constraints will be made in a coordinated manner through the materials management function.

The scheduling calculation

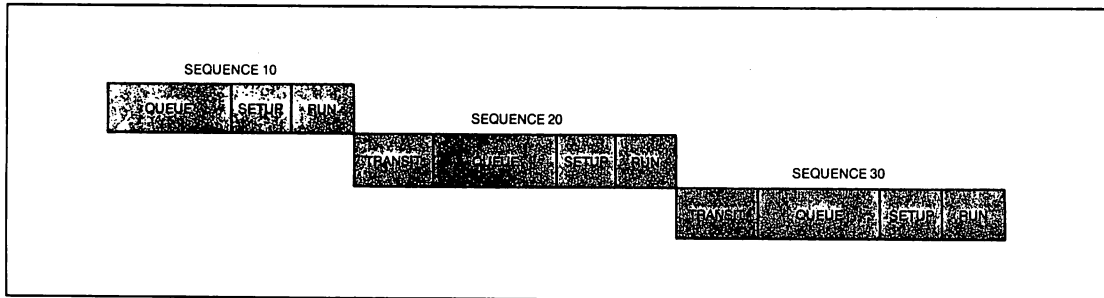
Lead time requirements for each work order are calculated by accumulating the specified transit, queue, setup and run times for each production sequence.

Backward and forward scheduling

Once sequence lead times have been calculated, *should-be* start dates can be assigned to each production sequence either by "backward scheduling" from a user-supplied order due date, or by "forward scheduling" from a user-supplied order start date. The dates are established by calculating actual elapsed time required to perform any production sequence and by examining the shop calendar for each workcenter to determine the number of working hours scheduled for any given date and shift.

Parallel production paths

When parallel production paths are defined in the routing for a part (see the ROUTINGS AND WORKCENTERS section earlier in this chapter), each path is scheduled independently so that all will be ready for the joining sequence at the same time. In this way, inventory levels at all points in the manufacturing cycle can be maintained at the lowest possible levels.



Components of lead time

Work Order Scheduling

Features

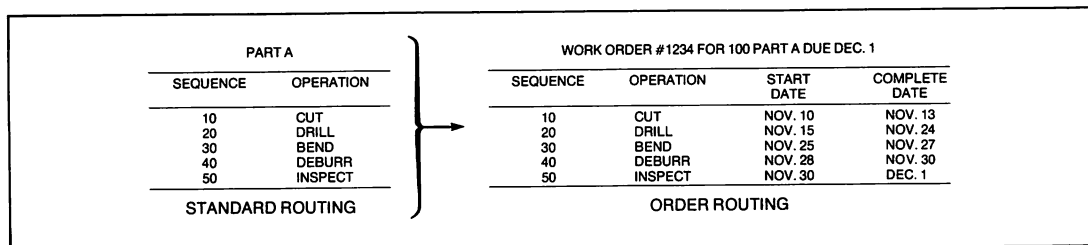
- Backward or forward scheduling.
- Infinite capacity scheduling.
- Work order specific routings.
- Overlapped, compressed, and split sequence scheduling.
- Partial work order scheduling.
- Parallel path scheduling.

Description

Work orders can be created as a result of transactions from Materials Management/3000, and/or directly within Production Management/3000. As each work order is created, a unique copy of the standard part routing for that part is made and is associated with the order. This order routing can then

be modified as necessary to indicate unique deviations from the standard routing (refer to the discussion of "Order routing modification" in the WORK-IN-PROCESS CONTROL section earlier in this chapter).

Using the due date of the work order as a basis, scheduled start and completion dates/times are calculated for each production sequence. Work orders are automatically rescheduled whenever their due dates or other specifications are modified. These schedules form the basis for the evaluation of production priorities as well as other planning and control functions.



Standard Routing and Order Routing

Shop Floor Dispatching

Features

- Routing Lists.
- On-line review of workcenter status.
- Dispatch lists by workstation or workcenter.
- On-line priority calculation.
- Multiple dispatching rules.
- Manual priority override.

Description

The SHOP FLOOR DISPATCHING module provides the tools to ensure that manufacturing resources are devoted to the right work orders at all times. The SHOP FLOOR DISPATCHING module evaluates production priorities dynamically on the basis of up-to-date status information and makes these priorities visible when needed on the shop floor.



Overlapping, splitting, and queue compression

Production Management 3000 provides for three methods of reducing the manufacturing lead time for a part below that which results from the scheduling calculation described above:

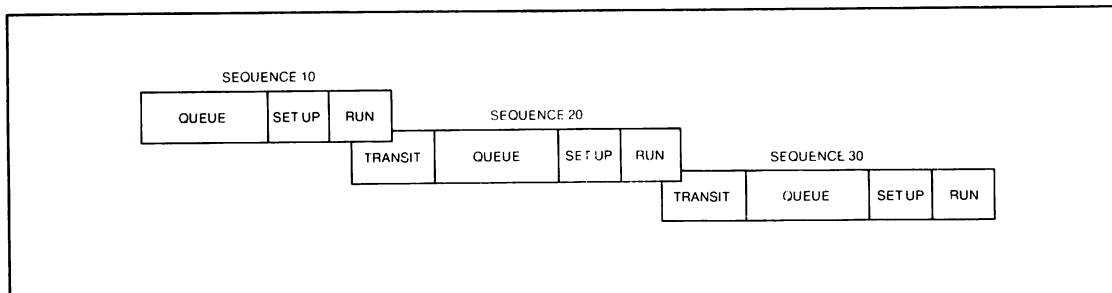
Overlapped sequences—Are those where some part of a work order is moved on to begin the next production sequence before all units

on the work order have been completed. The effect is to “overlap” process times at two sequences and to reduce the overall elapsed time required for completion of a work order. Overlap can be specified either for individual routing sequences or for all sequences in a standard part routing.

Split sequences—Are those where more than one person or machine performs work concurrently at a single production sequence. The number of such people or machines can be specified for any sequence of a part’s routing, which has the effect of reducing the *processing* portion of elapsed lead time.

Queue compression—Is a means of pre-expediting a work order by assuming less than standard queue times when calculating

scheduled start dates for each production sequence. Since these dates form the basis for the assignment of relative priorities among work orders in queue, this has the effect of assigning the *compressed* work order a higher priority at each sequence than it would normally have. This is accomplished by specifying a “queue compression factor” for a work order which indicates the percentage of the standard queue which is to be removed for scheduling purposes.



Overlapped scheduling

Work order priority calculation

The relative priority of each work order in the queue of each workstation is evaluated on the basis of a user-specified system value which defines the *facility dispatching rule*. Two such dispatching rules are provided by Production Management/3000:

Scheduled start dates—The scheduled start dates assigned to each production step by the WORK ORDER SCHEDULING module are compared with one another, and higher priorities are assigned to work orders with earlier dates.

Critical ratio—Priorities are based upon the ratio between the remaining lead time for each work order, and the elapsed time remaining until the due date for that work order. Work orders with *lower* critical ratios have higher priorities.

In either case, these priorities can be overridden, as necessary, by user-specified priority assignments.

When personnel or machines become available, a dispatcher can obtain a list of jobs currently waiting at any workstation or workcenter in priority order. This list can be produced either on-line or in printed form.

Workcenter Queue List										QUEUE WCTR	
Queue Part	Queue Dprtn	Chg Lot Status	Stop	Printed Report	Both	EXIT					
Wctr Id	Wstin Id	(leave blank to see all work stations)									
METAL						SHEET METAL SHOP					
Order Number	Prll	Dper Nbr	Schd-Date	Date	Part-Number	Stat	Qty Left				
MCDRILL	WD012934		DRL1 01 07/01 07/02	06/01	3920-0394	1000	50				
Tool-Id:		MCTP-1302		Drwg-Id:							
MCDRILL	WD918728		DRL1 58 06/25 06/25	06/23	3874-0192	1000	75				
Tool-Id:		MCTP-1982		Drwg-Id:							
PAINT	WD193289		PT11 50 06/26 06/27	06/24	5178-9948	1000	120				
Tool-Id:		PT10 50 06/27 06/28		06/23 1478-3844		1000		10			
Tool-Id:		PT11 50 06/28 02/28		06/23 4409-2894		1000		55			
PAINT	WD287389		PT11 50 06/28 02/28	06/23	4409-2894	1000	55				
Tool-Id:		DB01 50 06/28 06/30		06/23 2298-0198		1000		100			
Tool-Id:				Drwg-Id:							

Selecting work from queue

Releasing work orders

Once shop management has determined that all materials and documentation necessary to begin production activity on a given work order have been assembled, the order is *released* to the queue of the first production sequence. At that time a *Routing List* is printed which describes the routing for that order in detail. This Routing List is added to the shop packet and travels with the work order through the production process.

As work order movement or schedule changes take place, the Production Management/3000 data base is modified immediately to reflect the current status of work-in-process at all times. This dynamic model of the shop environment provides the basis for *dispatching* work according to the latest priorities whenever manufacturing resources become available.

HAPPY PEDALER BICYCLE WORKS										06/15/81 08:43A	
Work Order Routing List										Page 1	
Order #	Part Number	Description									
WO000012	3940-0498	BACK-PLANE CABINET FIXTURE									
AltRte#	RteSeq#	Pll Cde	NxtSeq#	Opn#	Wrk Cntr	Wrk stn	Rpr Lvl	OrdTyp			
0010			DRL1		METAL	DRILL	0	WO			
Drawing Id		Tool Id TL001224									
Current Transit Time		---Current Set-Up---			---Current Unit-Run---						
8.0		Labor	Machine	Code	Labor	Machine	Code	Labor	Machine	Code	
		2.0	2.0	L	.25	.25	L				
AltRte#	RteSeq#	Pll Cde	NxtSeq#	Opn#	Wrk Cntr	Wrk stn	Rpr Lvl	OrdTyp			
0020			DB01		METAL	DEBURR	0	WO			
Drawing Id		Tool Id									
Current Transit Time		---Current Set-Up---			---Current Unit-Run---						
8.0		Labor	Machine	Code	Labor	Machine	Code	Labor	Machine	Code	
		1.5	0.0	L	.05	.5	M				
AltRte#	RteSeq#	Pll Cde	NxtSeq#	Opn#	Wrk Cntr	Wrk stn	Rpr Lvl	OrdTyp			
0030			PT11		METAL	PAINT	0	WO			
Drawing Id		Tool Id									
Current Transit Time		---Current Set-Up---			---Current Unit-Run---						
8.0		Labor	Machine	Code	Labor	Machine	Code	Labor	Machine	Code	
		4.0	4.0	L	.5	0.0	L				

Work order Routing List

Work Order Tracking

Features

- On-line work order status review.
- Interactive step completion reporting.
- Factory Data Capture terminals.
- Partial work order tracking.
- Rework and scrap reporting.
- Exception reporting.
- Automatic load balancing.
- Labor collection.

Description

In order to effectively establish production priorities and provide shop managers with the information necessary to properly manage the manufacturing environment, Production Management/3000 must maintain an accurate picture of the status and location of each work order (or partial work order) at all times. Work order status information is collected on the shop floor at the time production activity actually takes place. Either simplified HP Factory Data Capture terminals or full-screen CRTs can be used for this purpose.

Work order status can be modified in either of two ways:

- 1) As a result of completing an operation. When a production *step completion* is reported, the status of the associated work order is automatically changed from *in process* at the current sequence to *in transit* to the next sequence.
- 2) Through the use of a specific transaction which modifies the status of a work order sequence to indicate that it has changed from one status to another (e.g., *in transit* to *in queue*, or *in queue* to *in process*).

Work order status reporting

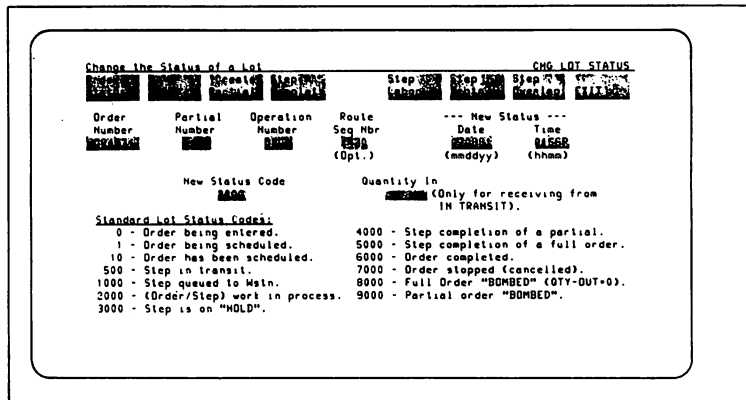
The current location and status of each work order in process is available at all times for on-line analysis and review. Work-in-process status can be reported either on CRT screens or printed reports by:

- Individual work order.
- Workstation or workcenter.
- Operation number.
- Part number.

Selecting work orders from available queue

As each work order is selected from queue for processing, its status is changed from *in queue* to *in process* using the WORK ORDER TRACKING function to indicate that it is being worked on.

Dynamic shop floor dispatching ensures that priorities are up to date at all times, reflects changes to the overall production plan quickly, and can reduce or remove the need for manual expediting of hot jobs.



Changing the status of a lot

Production exceptions

Information describing material failures, scrap losses, rework activity, or other exceptional conditions associated with each work order sequence can be collected and stored for analysis and review. This information can be entered at the time work orders are being tracked from one production sequence to another, or separately when exceptions are actually detected.

Exceptions can be recorded either as *free-form comments*, or in terms of predefined codes which can be analyzed on a statistical basis. *Exception codes* and their definitions can be described at any time and can be either *global* or *local* in scope. *Global exception codes* can be used to describe unusual events regardless of where in the manufacturing facility they occur, while *local exception codes* apply only to those workcenters with which they have been specifically associated.

```

Enter Production Step Exception Information
-----
Order  Rev All  Step  Step  Step  Chg Lot  Step  Step
Status  WC  Reptn  Complet  Labor  Status  Overlap  EXIT
-----
Order  Partial  Operation  Route  Exception
Number  Number  Number  Seq Nbr  Date  Time
M019847  2528  DPL2  30  070581  0200P
              (Opt.)  (mmddy)  (hhmm)

----- Exception -----
Code  Quantity  Component Part  Comment
BP1H  4  1339-0936  LOCK-WASHERS DAMAGED
              AND UNUSABLE

Standard Exception Codes:
BP1H-Material arrived broken      F002-Housing damaged.
MS1-Wrong number of parts received F003-Performance not within specs.
F000-Component does not          F004-Pins broken/missing.
work (indeterminate).           F005-Incorrect Option Received.
F001-Switch Failure              F006-Broken trace.
    
```

Recording exception information

Pre-defined exception code

Free-form comment

Production sequence completions

As soon as each production sequence has been completed, the results can be reported to Production Management/3000 by the people who actually perform the work. The number of units completed, scrapped (lost), bonused (found), and reworked at each sequence can be entered via an HP Factory Data Capture terminal or a full-screen CRT. Related information such as direct labor charges, material failures, or other exceptions can be collected either all at one time as a part of the *step completion* transaction, or separately through individual transactions designed specifically for each type of information.

Labor collection

Production Management/3000 provides the means to capture and record direct labor hours applied to each work order sequence. Labor hours are reported directly by shop personnel on the shop floor where work is actually being performed. As each entry is made, Production Management/3000 ensures that a valid employee ID and work order number have been provided.

Labor hours can be entered at the time work orders are being tracked from one production sequence to another, or through separate transactions as work is actually performed. Labor charges can be modified at any time and accumulated hours can be reported by employee ID, work order number, or workcenter.

```

Enter Production Step Completion Information
-----
Order  Rev All  Step  Step  Step  Chg Lot  Step  Step  Run
Status  WC  Reptn  Complet  Labor  Status  Overlap  EXIT  Hours
-----
Order  Partial  Operation  Route  Date Out  Time Out  Run
Number  Number  Number  Seq Nbr  (mmddy)  (hhmm)  Hours
M000012  2528  DPL2  30  082886  0138P  006.000
              (Opt.)  (mmddy)  (hhmm)

--Completion?--
(Normal, Repair)  Quantity  Quantity  Quantity
( or Bomb )  Bonused  Reworked  Scrapped
              1  0  0

Employee  Hours
Number  -----
Direct Labor: 18703 000  0542128  005.00

----- Exception -----
Code  Quantity  Component Part  Comment
M001  1  1339-0936  REWORKED ONE UNIT TO
              SHOOT SHIPPED CORNER.
    
```

Completing a production sequence

```

Enter Direct Labor for an Operation
-----
Order  Rev All  Step  Step  Step  Chg Lot  Step  Step
Status  WC  Reptn  Complet  Labor  Status  Overlap  EXIT
-----
Order  Partial  Operation  Route  Date  Time
Number  Number  Number  Seq Nbr  (mmddy)  (hhmm)
              (Opt.)

Employee  Set-up  Labor
Number  Hours  Hours
Direct Labor: 18703 000  0542128  005.00
    
```

Charging time to a work order

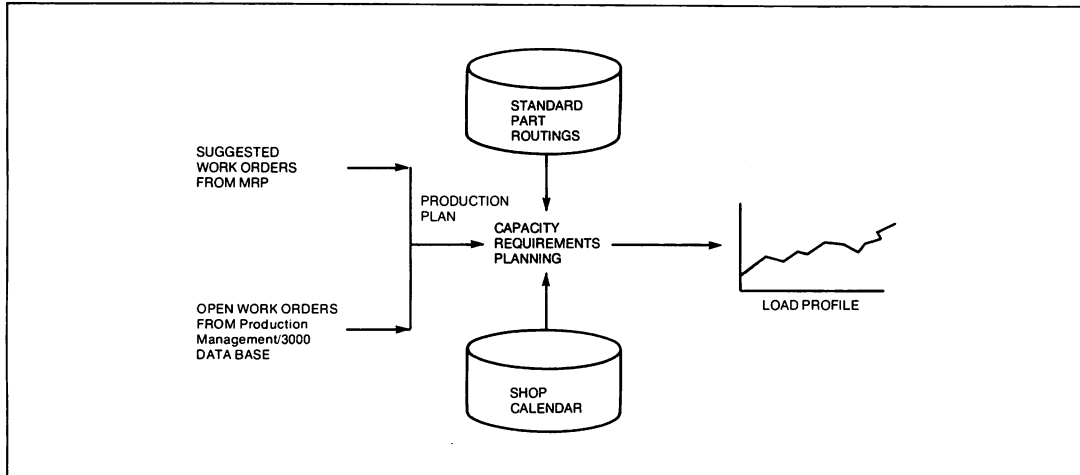
Capacity Requirements Planning

Description
 CAPACITY REQUIREMENTS PLANNING can be used to plan labor and other manufacturing resource requirements over time at the workstation and workcenter level. It can use both existing work orders

and production requirements suggested by Materials Management/3000 or another similar system. It provides a tool to highlight potential capacity constraints and provides information that can be used to help ensure that appropriate resources are available when needed and to smooth the workload for a manufacturing facility.

Features

- Load projection by workstation.
- Exception reporting.
- Summary workcenter reporting.
- Alternate load measures.
- User-defined reporting periods.
- Load detail reporting.

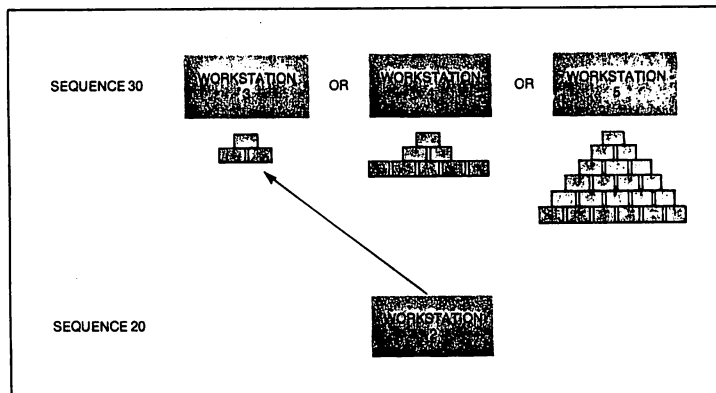


Capacity Requirements Planning

Balancing load among alternate workstations

When successful completion of a production sequence is reported, the status of the work order is modified to indicate that it is in transit to the next production sequence. In cases where more than one alternate workstation has been specified for the next production sequence, Production Management/3000 will examine the amount of work in the queue of each alternate workstation. The work order will then automatically be made available in the queue of the workstation with the least hours of work waiting to be done and the user will be notified which workstation has been selected.

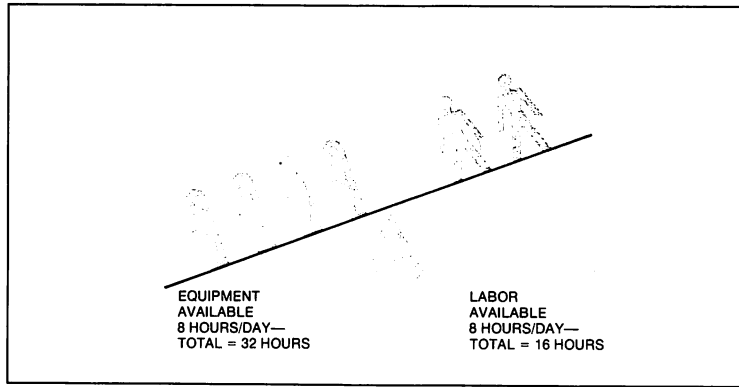
This technique for *load balancing* provides for the even distribution of work among alternate workstations, and consequently for the reduction of lead times and inventory levels.



Load balancing

Workcenter summary reporting

In situations where direct labor personnel are trained to move among several different workstations in response to changing workloads, the true labor capacity of a workcenter may be less than the sum of the capacities of all workstations within it. In addition to workstation load reports, Production Management/3000 provides summary load reports for each workcenter which can help to identify resource constraints which might not be visible at the workstation level.



Workcenter labor capacity

Calculation of requirements

The calculation of manufacturing resource requirements is accomplished by scheduling each work order in the production schedule using part and order routings and the shop calendar to determine detailed load requirements for each work order. Each work order is scheduled independently of all others as if infinite capacity were available at all times. These detailed load requirements are then accumulated by workstation for each user-defined time period (days, weeks, months or quarters), forming a load profile for each workstation.

Alternate load measures

Although labor is often the capacity constraint in manufacturing operations, this is not always the case. CAPACITY REQUIREMENTS PLANNING can calculate resource requirements both in terms of labor hours and one alternate unit of measure (e.g., machine hours, cm2, etc.) which can be specified for each workstation. Both load measures are accumulated and reported separately for each workstation, so that they can be analyzed

independently of one another or together. This capability provides the flexibility to evaluate resource requirements in terms most appropriate to each individual workstation.

Load profile reporting

After all time-phased load profiles have been calculated, the resulting resource requirements are compared with the standard production capacity specified for each workstation. In cases where requirements differ from standard capacity by more than a user-specified percentage, a workstation load report is printed which describes the load profile of that workstation and indicates the time periods in which overload or underload conditions are anticipated.

PRINT DATE 04/02/81 15:52		HAPPY PEDALER BICYCLE WORKS										PAGE 1						
RUN DATE 04/02/81 15:00		EXCEPTION CAPACITY PROJECTION REPORT																
		WORK STATION																
WKCTR	METAL			D	W	BW	M	Q	SA	A								
WKSTH	DRILL	UPPER TOLERANCE		115	120	120	125	1	8	150	150							
MANAGER-J. FERGUSON		LOWER TOLERANCE		90	85	85	80	60	60	50								
---PERIOD---		---MACHINE---		---LABOR LOAD---							---UTILIZATION OF SCHEDULED CAPACITY---							
DATE	LEN	LOAD	CAP	RLSD	OPEH	PLAN	SUGG	TOTAL	LABOR	CAP	CUR	CUM	%	OUT	50.	100.	150.	200.
04/02/81	1	0	0	6	4	0	0	10<	8	2	2	125	10	XXXXXXXXXXXXXXXXXXXXXXXXXXXX				
04/03/81	1	0	0	5	3	0	0	8<	8	0	2	125	10	XXXXXXXXXXXXXXXXXXXXXXXXXXXX				
04/04/81	1	0	0	7	3	0	0	10<	8	2	4	150	35	XXXXXXXXXXXXXXXXXXXXXXXXXXXX				
04/05/81	1	0	0	12	0	0	3	15<	8	5	9	213	98	XXXXXXXXXXXXXXXXXXXXXXXXXXXX				
04/08/81	1	0	0	2	6	0	0	8<	8	0	9	213	98	XXXXXXXXXXXXXXXXXXXXXXXXXXXX				
04/09/81	1	0	0	0	7	0	0	7<	8	-1	8	200	85	XXXXXXXXXXXXXXXXXXXXXXXXXXXX				
04/10/81	1	0	0	0	4	2	2	8<	8	0	8	200	85	XXXXXXXXXXXXXXXXXXXXXXXXXXXX				
04/11/81	1	0	0	0	0	6	0	6<	8	-2	6	175	60	XXXXXXXXXXXXXXXXXXXXXXXXXXXX				
04/12/81	1	0	0	0	0	10	4	14<	8	6	12	250	135	XXXXXXXXXXXXXXXXXXXXXXXXXXXX				
04/13/81	7	0	0	0	12	46	0	58<	40	18	30	175	60	XXXXXXXXXXXXXXXXXXXXXXXXXXXX				
04/20/81	7	0	0	0	48	10	58<	40	18	48	220	105	XXXXXXXXXXXXXXXXXXXXXXXXXXXX					
04/27/81	7	0	0	0	36	0	36<	40	-4	44	210	105	XXXXXXXXXXXXXXXXXXXXXXXXXXXX					
05/04/81	7	0	0	0	38	2	40<	40	0	44	210	105	XXXXXXXXXXXXXXXXXXXXXXXXXXXX					

Workstation load profile

Calendar days in reporting period

Total labor hours planned for each period

Percent by which planned load is outside of stated tolerances

Load detail reporting

In situations where examination of the load profile report identifies overload conditions which need to be corrected, a *load detail report* can be requested for any individual

workstation. The load detail report indicates the specific work orders which contribute to the accumulated load profile during each time period, and the amount of workload which each represents.

On the basis of this information, individual work orders can be rescheduled into earlier or later time periods to arrive at a load profile which better fits the available manufacturing resources.

PRINT DATE 04/02/81 15:52		HAPPY PEDALER BICYCLE WORKS										PAGE 1					
RUN DATE 04/02/81 13:49		DETAIL CAPACITY PROJECTION REPORT															
		WORK STATION															
WRKTR	METAL	D	W	BW	M	Q	SA	A									
WRKSTN	DEBUHR	UPPER TOLERANCE		115	120	120	125	140	150	150							
MANAGER	J. FERGUSON	LOWER TOLERANCE		90	85	85	80	60	60	50							
---PERIOD---	---PART NUMBER---	PL	-----ORDER-----	---	SCHEDULED DATE-	OPER	MACHINE-	LABOR--	---	SCHEDULED----	---	WORKSTATION--					
DATE	LEN		NUMBER	PRTL	TP	QTY	IN	START	OUT	NBR	LOAD CAP	LOAD CAP	DEV	CUM LDR	OUT	PREV	NEXT
4/02	1	3940-0498	WO200168	OP	32	04/02	04/02	04/02	04/02	DB03	.0	1.5				DRILL	PAINT
		8498-0498	WO200527	OP	50	04/02	04/02	04/02	04/02	DB01	.0	2.2				DRILL	PAINT
		9487-1152	WO200886	OP	100	04/02	04/02	04/02	04/02	DB01	.0	2.5				DRILL	PAINT
		8376-8274	WO201245	OP	110	04/02	04/02	04/02	04/02	DB01	.0	3.0				DRILL	PAINT
		0938-1784	WO201604	OP	75	04/02	04/02	04/02	04/02	DB01	.0	1.5				DRILL	PAINT
		7467-6662	WO201963	OP	90	04/02	04/02	04/02	04/02	DB01	.0	4.5				DRILL	PAINT
		8578-8391	WO202322	OP	100	04/02	04/02	04/03	04/03	DB01	.0	2.2				DRILL	PAINT
		6473-0395	WO202681	OP	100	04/02	04/02	04/02	04/02	DB07	.0	1.4				DRILL	PAINT
		8495-1722	WO203040	OP	225	04/02	04/02	04/02	04/02	DB11	.0	1.0				DRILL	PAINT
		4095-2883	WO203399	OP	400	04/02	04/02	04/02	04/02	DB01	.0	2.5				DRILL	PAINT
		7459-0384	WO203758	OP	430	04/02	04/02	04/02	04/02	DB01	.0	3.0				DRILL	PAINT
		9847-7844	WO204117	OP	100	04/02	04/02	04/03	04/03	DB01	.0	2.2				DRILL	PAINT
		1872-1846	WO204476	OP	48	04/02	04/02	04/02	04/02	DB01	.0	3.0				DRILL	PAINT
		8732-3704	WO204835	OP	48	04/02	04/02	04/02	04/02	DB01	.0	2.5				DRILL	PAINT
			TOTAL								.0	35.0	40.0	5.0			

LEGEND:	'<' = Out of Tolerance	'D' = Daily	'W' = Weekly	'BW' = Biweekly	'M' = Monthly	'Q' = Quarterly	'SA' = Semiannual	'A' = Annual
---------	------------------------	-------------	--------------	-----------------	---------------	-----------------	-------------------	--------------

Starting date of reporting period	Planned labor requirements for each work order
-----------------------------------	--

USING DS 3000/1000 WITH HP 1000 MASTER PROGRAMS

JOERG MUELLER

J. MUELLER
SWS SOFTWARE SYSTEMS
BERN, SWITZERLAND

by Joerg Mueller
SWS SoftWare Systems
Bern, Switzerland

Introduction

HP/3000 is a very efficient general purpose computer, which gives users and programers a lot of flexibility on the software side. Now that newer HP/3000 models support HP-IB devices flexibility has been extended to the hardware. Nevertheless users would be ill advised to use HP/3000 as a measurement data logger or to do any other form of fast data acquisition. Interrupts at the rate of thousands or even only hundreds per second should not be sent to a general purpose computer.

Of course Hewlett Packard offers a solution for fast data acquisition and processing with the HP/1000 computers. An important additional advantage of the HP/1000 systems is a price tag which favourably compares to the /3000-line. Users already owning an HP/3000 see the disadvantage though of a largely incompatable, different 16 bit computer system. Architecture, operating system, language syntax and file structure are all different on the two system types. Except for trivial cases it is not even possible to define a common Fortran subset, so that the same source code could be used on both computers. The choice of having both types of systems within one company may nevertheless pay off, because a good price to performance ratio can be achieved. Once the decision has been made to purchase both HP/3000-

and HP/1000-computers it is obviously attractive to link the systems with a DS3000-1000 communication line. In fact in certain cases it may even be possible to have a memory resident HP/1000 operating system without any mass storage of its own thanks to the DS3000-1000 downloading facility.

The HP/1000 systems have evolved and this allows to use them also as general purpose computers. Nevertheless it is advisable to restrict the use of the /1000 and to do most of the programming on the HP/3000, if the resources such as the present workload and the communication line allow it, because software maintenance on the HP/3000 often is easier and has to be done for other applications anyway, so that the programer staff usually will be more familiar with the /3000. However the environment may impose a more important HP/1000 configuration. Some problems with DS3000-1000 which may occur under such circumstances will here be discussed. Program to program communication (PTOP) will be analysed primarily in this paper, but the other forms of communication also will be discussed briefly so as to see their respective merits.

1. Respective merits of the DS3000-1000 features

1.1 Remote operator commands

Most of the RTE-IVB operator commands can be issued from a HP/3000 remote session. It is possible to start and analyse the state of the RTE-system. In fact a RTE-IVE memory based system may entirely be controlled by a HP/3000 session. The HP/1000 is working as a front end computer in such a case, a configuration, which may be very useful.

It is interesting to note that on the HP/3000 a Cross-Assembler of the HP/2100 is available (which in effect allows a subset of the HP/1000 instructions) and a Cross Loader also exists. A Fortran crosscompiler exists in the contributed library now. It therefor now should be possible to develop a complete HP/1000 application on the /3000 and simply download by DS3000-1000. The HP/1000 does not need to have any storage device in such a case.

1.2. Remote EXEC calls

An EXEC call is familiar to all HP/1000 programmers. The HP/3000 equivalent are the system intrinsic calls. The system resources may programatically be used by such calls. Input/Output, program scheduling, segment loads and time requests are possible. It is therefore feasible for a HP/3000 process to use all HP/1000 peripherals. Only the actual driver of the device is residing on the HP/1000 in such a case.

The opposite type of calls, i.e. the use of HP/3000 system intrinsics by a HP/1000 program, is not possible however.

1.3. Remote file access

Although HP/1000 files are not easily accessible by HP/3000 session commands, it is however fully possible to use HP/1000 files by a HP/3000 program. Such files cannot only be read and written to, but they can be created, opened and closed, renamed or also purged. The structure of the files remain of the HP/1000 type of course.

Besides different file structures the two computer systems also have different real number representation. This must be remembered for binary files which are commonly used by HP/1000 and HP/3000 programs. Except for a little snag in case of a zero the SPL-routines CR1R3 and CR3R1 contained in the contributed library allow correct conversion between the two types of real variables.

Again it is not possible for a HP/1000 program to access HP/3000 files. One obvious problem in this context is the maximum length of a name, which may only have up to six characters for a file on the HP/1000 instead of 8 letters as on the /3000.

1.4 Program to Program Communication (PTOP)

The most powerful communication tool available within DS3000-1000 is the PTOP-package. It consists of a set of nearly identical procedures available on both computers.

For a PTOP user application to work a pair of programs must exist. The program initiating the communication is the so called master program. The other program is called a slave program and is automatically scheduled by the respective operating system, when the master program initiates the communication. The master program may be on the HP/3000 or on the HP/1000. Since a HP/3000 program can practically control all resources of the HP/1000, it will not often be necessary to have HP/3000 master programs. Usually the other forms of remote access will be sufficient. Whenever a HP/1000 program wants to access the /3000 there is no other choice however than to use PTOP communication.

The master program initiates communication by the POPEN procedure. The slave receives all information by a GET call and returns its information by ACCEPT or REJCT. On the master side the following other procedures are available:

PREAD to receive a buffer of information,
PWRIT(E) to send a buffer of information,
PCONT(ROL) to exchange a "tag"-field i.e. a buffer of
 20 words,
PCLOS(E) to abort the slave program,
PCHECK (/3000) to analyse the last PTOP transaction.

Although the procedure calls are very simple, PTOP communication can present some difficulties to a beginner. Whenever multiple master programs initiate the same slave or automatic scheduling is wanted, problems may increase even more. But on the other hand applications can be tackled, which otherwise could not be realised at all. In fact once DS3000 is installed, PTOP can prove to be a very useful tool for concurrently running programs within the same computer system.

2. Situation of CIBA-GEIGY Photochemie (Fribourg, Switzerland)

The computer configurations to be discussed here as a practical case have the following simplified outline (Fig.1).

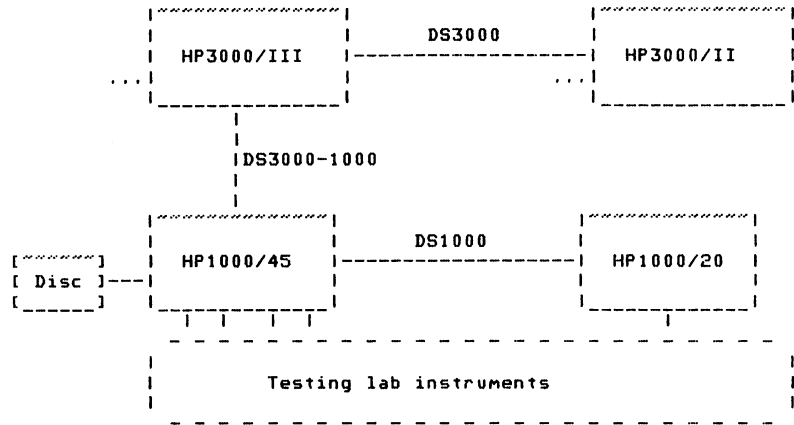


Fig. 1
Configuration with hardwired connections.

The HP3000/II system serves primarily as development system, the HP3000/III supports materials control, production management, quality control and distribution and is thereby fully loaded. The HP/1000 systems perform measurements, control the instruments and process the data for Quality Control. The results are intermediately stored on the HP/1000 disc and usually daily transmitted to a HP/3000 data base.

Had the HP/3000 systems not been so heavily loaded, a configuration without disc-system could have been envisaged. Since this solution was out of the question, the benefits of largely independent HP/1000 systems were fully exploited. System operation still had to be simple though since no actual computer operator could be afforded for the /1000-systems.

It is here that local PTOp processing on the HP/1000 systems proved to be very useful. Concurrently running programs could thereby easily be synchronised. The PTOp programs running with the HP/3000 are not of such a nature as shall be shown later.

The usual contact between the HP1000/45 and the HP3000/III during the normal working hours restricts itself to a few occasional /3000-sessions initiated on the /1000 or the fetching of a source file as will be seen later. The daily evening transmission sends all relevant measurement results for complete evaluation and storage to the /3000. Since the /3000 systems are continuously operator controlled this is also the place all plotting is done through batch jobs streamed by the transmission program.

The transmission programs contain automatic maintenance functions so as to eliminate measurement results older than two days on the

/1000-disc, if transmission and storage on the /3000 were faultless.

The remaining important function of the DS 3000-1000 line enables to set up the HP/1000 master data files from scratch. This is done after every disc copy (usually once a month), whenever a new software revision is installed and when the master sets are updated on the /3000 side. The only updating mechanism for those /3000 master sets allowed to the user go by way of the /1000. Thereby data integrity usually is conserved eventhough the information is stored on both systems. Of course there always will be a hurried smart user ruining the concept.

Except for the occasional problems with ignorant new users the system has been successfully operational in automatic mode for several months now.

3. DS3000-1000 PTOP programs

For the three types of transmission operation mentioned three pairs of PTOP programs had to be written:

- 1) PUGE: to put or get source files
- 2) PUTSE: to transmit and maintain
the measurement result files
- 3) FMDEN: to maintain the master data sets

All three applications obviously aim to do programatic remote file access from the /1000 to the /3000. As it was shown earlier this function only can be achieved by PTOP communication. Had the amount of initial problems been foreseen it probably would have been more economical to sacrifice the requirement of at least semi-automatic operation and have the user manually start a /3000 session and a /3000 program, which then could directly do remote file access without PTOP communication. Now the PTOP pairs are operational they offer easier use though.

3.1. PUGE (PUT and GET)

Source file maintenance, i.e. creating back ups and only keeping source files, which are being worked on directly on disc, is a housekeeping problem common to all computer systems. Having DS3000-1000 this problem could elegantly be solved with some

additional benefits.

The programmers disc cartridges containing his source files are declared a scratch area and are usually entirely erased every week or two. Whereas such a way of operating a HP/3000 probably would create a programmers revolution, the HP/1000 with a DS-link can be maintained in such a manner, because

```
:RU,PUGE
+GET source
```

will rapidly give the file back (it takes 8 seconds to get the + prompt and e.g. 25 seconds for a source file of 300 lines to be available on the /1000). Of course eventually the source files are not available on line on the /3000 either anymore. In that case the programmer has to have the source reloaded from mag tape on to the /3000 and can then fetch it again, obviously a more tedious operation.

The additional benefit is the ease of maintenance on the HP/1000. A procedure using PUGE is all that has to be called, whereby all source files are transmitted to the /3000 and there renamed to suit the file naming conventions of the /3000 (a source &SOURC.JM will become SSOURCJM.HP1000 on the /3000). The whole scratch area then is simply purged.

3.2. PUTSE (PUT SEnsitometric results)

The program pair to transmit measurement data essentially operates in

the same manner as PUGE as far as the communication is concerned. One of the apparent differences is the automatic streaming of batch evaluation jobs by the HP/3000 slave.

The measurements of a day are usually transmitted within about 15 minutes.

3.3. FMDEN (Fetch Master sets for DENsitymetry)

By this program pair either a complete dump of all relevant master sets from the /3000 to the /1000 is realised or individual data is updated on both computers simultaneously.

A complete dump takes about one minute and fills 805 sectors of the 7906 disc. This corresponds to a practical speed of 25 kBaud (in the software sense).

4. Design of PTOP program pairs

As a first general rule it can be stated that PTOP communication should be avoided, whenever other means such as remote file access or remote EXEC calls can give the same result, because the application will be operational more rapidly.

There are situations though, as seen earlier, when PTOP communication

is the only good solution for the end user. Even between two HP/3000 such program pairs may be useful as well described in the DS3000 reference manual.

Once it has been decided to use PTOP communication, programing is usually done in such a way that the computer initiating the communication has the master program and the other one the slave. DS3000-1000 appears to have a bug though and it is currently not advisable to write HP/1000 master programs and HP/3000 slaves. Once the programs are in working order there are no big problems to be expected. Testing proves to be very disagreeable though, because an abort of the slave on the /3000 on bounds violation, integer overflow, an inexistent file or other such normalities in a testing phase systematically provokes a system down of MPE. Obviously the poor programmer provoking such a small disaster on a production oriented heavily loaded HP/3000 will have plenty of niceties to hear.

If a /3000 slave has to be written there are a few options which increase the chance of MPE-survival. An obvious solution is to make the slave as short and simple as possible. Delegate more intricate work to previously or later executable programs. One of the possibilities in this context is the use of streamed subsequent jobs. Another solution can consist of a very simple PTOP pair, which only initiates the communication and then is followed by another pair, for which the master is on the /3000 side. In fact it is possible for a slave program itself to become master of another. Probably this would not help to solve the abort problem (it has not been tested), because it appears to be related to the original, atypical scheduling of the slave.

Other obvious means of avoiding the slave to abort consist in all the necessary tests to intercept the potential error. This requires additional code though and thereby makes the testing phase longer. The final way out for testing again is disagreeable: night shift. Wait until all those backlogged batch jobs are out of the system and then learn to warmstart MPE!

There are other potential problems with PTOP. A programmer glancing through the available procedure calls intuitively assumes that communication starts by POPEN and terminates by PCLOSE. The assumption is correct for POPEN, but PCLOSE not only stops communication, but also aborts the slave, even if it is concurrently working for another master. Except for emergency exits a master program will therefore contain at least one POPEN, but usually no PCLOSE.

A slave must be careful by what process it is called. All transactions of the master are received by the GET procedure. By program logic maybe a PREAD is expected to be calling, in some cases it may though be a POPEN from another master or, in fact, from the same master, which terminated and restarted for some reason.

From all that has been said it primarily can be concluded that PTOP communication should be kept as simple as possible, especially if the programmer is inexperienced with DS3000. Hopefully the system down bug will be out before MPE revision Zephyr (or whatever) arrives and then PTOP communication can be a very useful tool even between a /1000 and a /3000.

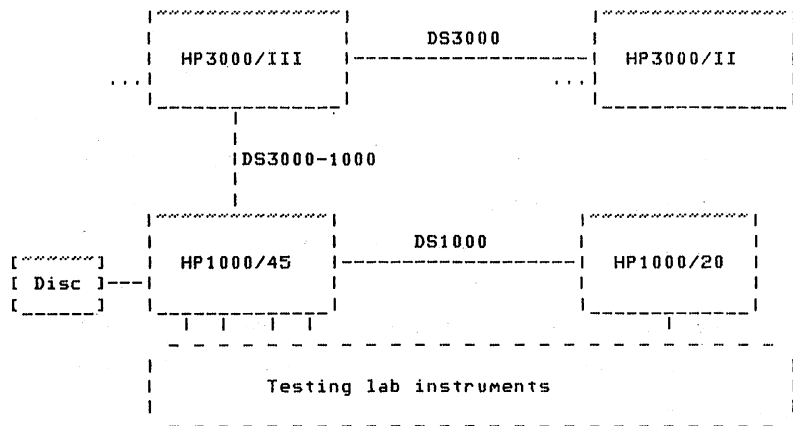


Fig. 1
Configuration with hardwired
connections.

GLOBAL OPTIMIZATION

(ABSTRACT)

PROTOS - A COBOL PROGRAM GENERATOR FOR THE HP3000

By

J. TIPTON COLE

TITLE: Global Optimization

**SPEAKER: J Tipton Cole, Cole & Van Sickle Company, Inc., Austin, TX
P.O. Box 15085 8900 Shoal Creek 404 Austin, Texas 78761 USA**

ABSTRACT: Global optimization of any computer application for business is facilitated by observing a few basic rules: 1) work on the right problem, 2) optimize only where it helps, 3) optimize for the total lifetime cost of the project, and 4) use proven methods and tools. Technical workers and managers often lose sight of business objectives. Worse yet, despite all of the talk about software engineering, we neglect fundamental engineering principles and techniques.

This talk stresses the importance of concentrating on the business purposes behind the project and the importance of implementing the applications now rather than later.

This presentation is designed for DP managers, non-DP managers who have responsibility for DP or who must deal with the DP department, and DP staff members with managerial aspirations.

J. TIPTON COLE

COLE & VAN SICKLE COMPANY, INC.,

AUSTIN, TX

***** IF YOU FEEL ASPIRATIONS FOR A COMPLETE TEXT, ASK THE
AUTHOR - UNTIL NOW THERE IS NONE FOR THE PROCEEDINGS -
(EDITOR) ***

(ABSTRACT)

TITLE: PROTOS - A COBOL Program Generator for the HP3000

SPEAKER: J Tipton Cole, Cole & Van Sickle Company, Inc., Austin, TX

ABSTRACT: PROTOS is a COBOL program generator written exclusively for the HP3000 family of computers. It is designed to work naturally and efficiently with IMAGE and V/3000. It writes COBOL code that takes advantage of the strengths of HP's best software and of the strengths of the HP hardware. The COBOL code follows the best programming practices. It is easy to read and understand; it is well-structured. In fact it is superior to hand written code in quality as well as cost.

This talk covers the reasons that business organizations need tools such as PROTOS, and the benefits that PROTOS brings to any business application project. Discussion of specific features and demonstration of PROTOS in action are left to the question and answer period and individual meetings.

This presentation is designed for DP managers, non-DP managers who have responsibility for DP or who must deal with the DP department, and DP staff members with managerial aspirations.

***** NOTES SEE ABSTRACT BEFORE (EDITOR) ***

QUASAR SYSTEMS

QUASAR SYSTEMS

TRANSACTION PROCESSOR FOR THE HP3000

I am sure that each of us has had the need to manipulate files, or perform bulk updates of an application database, and found that the existing methods are either incomplete (i.e. FCOPY) or too troublesome (i.e. COBOL) to use. Most application systems involve several standard batch functions which require custom programming. Yet the task involved is so standard one should be able to specify it in a simple, logical and straightforward manner.

These functions can include:

- : Daily, weekly, or monthly rollovers.
- : Reformatting a file.
- : Producing a summary file.
- : Selectively copying based on some condition.
- : Copying elements from one file to another.
- : Reformatting a database.

File manipulation tasks are a common requirement in developing as well as running most application systems. While excellent productivity tools now exist for large segments of application development and maintenance, batch processing programs still have to be prepared in the same tiresome manner.

The paper introduces the concept of a powerful batch-oriented data manipulation tool called a TRANSACTION PROCESSOR, which will keep pace with and interface with current state-of-the-art productivity tools.

The TRANSACTION PROCESSOR will be called QTP and will complete Quasar Systems' family of application generator products, which currently include QUIZ for reports and QUICK for screen-based input. With this family, users will be able to generate entire applications in a consistent easy-to-use style.

This paper will discuss:

1. QTP in relation to an application dictionary
2. Design objectives
3. QTP in operation (some examples)
4. QTP in the production environment
5. Design considerations.

TRANSACTION PROCESSOR FOR THE HP3000

Godfrey Lee
Product Development Specialist

Quasar Systems Ltd.
Software Products Group
275 Slater Street, 10th Floor
Ottawa, Canada K1P 5H9
(613)237-1440
Telex: 053-3341

Quasar Systems Inc.
1460 Maria Lane, Suite 250
Walnut Creek
California
94596
(415)943-7277

QUASAR SYSTEMS

TRANSACTION PROCESSOR AND THE APPLICATION DICTIONARY

The transaction processor will be able to operate as an independent product in association with Quasar Systems application dictionary. In essence, QTP will have two components:

1. QSCHEMA

The schema processor compiles a description of data files and element characteristics including data validation and display specifications. The compiled schema functions as an application dictionary, providing central administrative control and freeing users of QTP from a great deal of repetitive programming.

2. QTP

Under the control of specification statements which can be used by both programmers and non-programmers, QTP will carry out two major functions:

- . standard batch applications
- . file manipulation.

QUASAR SYSTEMS

DESIGN OBJECTIVES

The design objectives of QTP are:

- . to support the standard maintenance functions of add, change and delete against all data permanently on file
- . to support standard editing of input including, type checking, value range checking, and pattern matching
- . to support the copying of elements from one file to another
- . to allow the reformatting of files and databases
- . to be able to produce summary files
- . to support these summary options: sum, count, average, maximum, minimum, percentage and ratio
- . to be able to specify the sorting and selection of input files
- . to support any combination of IMAGE, KSAM and MPE files
- . to reference the structure, composition and elements of files in a central independent schema
- . to use concise specification statements in simple free-form syntax.

QUASAR SYSTEMS

THE TRANSACTION PROCESSOR IN OPERATION

To show the scope of the QTP in operation, here are four short examples of situations which occur frequently and which normally require specially written programs.

1. New product number

The manager of inventory control wants to assign a new product series "M" to all series "S" product numbers greater than 6000. With QTP, this task could be accomplished by entering the following statements:

```
> ACCESS PRODUCTS
> SELECT IF PRODUCT-CODE = "S" AND PRODUCT-NUM > 6000
> FILE PRODUCTS UPDATE
> ITEM PRODUCT-CODE FINAL "M"
> GO
```

The ACCESS statement specifies which file(s) are to be read--in this case, the file PRODUCTS. The SELECT statement then restricts the selection of records from the product file to those records to be changed. The FILE and ITEM statements specify the changes to be made to selected records. The GO statement causes the QTP request to be executed.

2. Organizational change

The San Francisco branch has been reorganized and is now part of California branch. All reference to San Francisco is to be deleted and all records for San Francisco employees are to be updated to reflect their new status as records of California branch employees.

```
> ACCESS BRANCHES LINK TO EMPLOYEES LINK TO BILLINGS
> SELECT IF BRANCH-NO OF BRANCHES = "SF"
> FILE EMPLOYEES UPDATE
> ITEM BRANCH-NO FINAL "CA"
> FILE BILLINGS UPDATE
> ITEM BRANCH-NO FINAL "CA"
> GO
```

The ACCESS statement in this example illustrates multi-file access. Keyed linkages between files can typically be performed automatically, using information in QSCHEMA. The ITEM statements in this example set BRANCH-NO to "CA" in the selected EMPLOYEES and BILLINGS records.

QUASAR SYSTEMS

3. Culling obsolete data

A company wants to streamline their customer file and delete anyone on their mailing list who hasn't corresponded for over a year.

```
> ACCESS MAIL-LIST
> SELECT IF 365 < (DAYS (SYSDATE) - DAYS (RESPONSE-DATE))
> FILE MAIL-LIST DELETE
> GO
```

The FILE statement in this example deletes all records of MAIL-LIST that have satisfied the condition in the SELECT statement.

4. Reformatting a file

QTP will be ideally suited to problems involving the reformatting of files. Assume for instance, that the old customer file shown in Figure 1 is obsolete. The "PYR-SALES" (previous years sales) item is to be dropped; "YTD-SALES" (year to date totals) is to be expanded for larger dollar volumes; item "CUSTOMER-ID" is to be expanded; an item "SALESMAN-CODE" is to be added; and all items are to be re-ordered.

Figure 1

OLD CUSTOMER MASTER		NEW CUSTOMER MASTER	
CUSTOMER-NAME	X(20)	CUSTOMER-ID	X(10)
CUSTOMER-ID	X(6)	CUSTOMER-NAME	X(20)
CUSTOMER-ADDRESS	X(60)	CUSTOMER-ADDRESS	X(60)
PYR-SALES	9(6)	SALESMAN-CODE	X(6)
YTD-SALES	9(6)	YTD-SALES	9(10) COMP

The steps needed to format the new customer file are:

(a) Unload the master file.

```
> ACCESS CUSTOMER
> SUBFILE TMP OUTPUT CUSTOMER
> GO
```

(b) Change the schema, purge and recreate the customer file (details not shown).

QUASAR SYSTEMS

(c) Reload the new master file.

```
> ACCESS *TMP
> FILE CUSTOMER ADD
> GO
```

(d) Purge the temporary file.

```
:PURGE TMP
```

The SUBFILE statement creates an ad-hoc file containing specified information. Subfiles automatically contain their own schema and are therefore self describing. In this example SUBFILE creates a temporary file TMP containing a copy of the customer master file.

QTP automatically performs the following manipulations for commonly named items in the two files:

- . changes item type
- . changes item size
- . changes item order.

QUASAR SYSTEMS

QTP IN THE PRODUCTION ENVIRONMENT

The following two examples look in detail at how QTP might handle two common month-end production situations.

1. Adding 1% interest to all invoices over 30 days due

To expand on a typical accounts receivable situation, assume a company has reached the due date for monthly accounts receivable. The account manager wants to add 1% interest to all outstanding accounts and update the master file. QTP performs this task in fewer than 20 specification lines.

```
> ACCESS ACCOUNT-MASTER LINK TO ACCOUNT-DETAIL
> SORT ON ACCOUNT-NO, INVOICE-NO
> TEMPORARY INVOICE-DATE RESET AT INVOICE-NO &
>   INITIAL DATE OF ACCOUNT-DETAIL &
>   IF TYPE OF ACCOUNT-DETAIL="INVOICE"
> TEMPORARY INVOICE-BALANCE RESET AT INVOICE-NO INITIAL 0
> SUM AMOUNT OF ACCOUNT-DETAIL INTO INVOICE-BALANCE &
>   IF TYPE OF ACCOUNT-DETAIL = "INVOICE" OR &
>   TYPE OF ACCOUNT-DETAIL = "INTEREST"
> SUM AMOUNT OF ACCOUNT-DETAIL INTO INVOICE-BALANCE NEGATIVE &
>   IF TYPE OF ACCOUNT-DETAIL = "PAYMENT"
> FILE ACCOUNT-DETAIL ALIAS INTEREST ADD AT INVOICE-NO &
>   IF SYSDATE > INVOICE-DATE + 30
>   ITEM AMOUNT FINAL INVOICE-BALANCE * 0.01
>   ITEM TYPE FINAL "INTEREST"
> FILE ACCOUNT-MASTER UPDATE AT ACCOUNT-NO
> SUM AMOUNT OF INTEREST INTO BALANCE OF ACCOUNT-MASTER
> GO
```

The account details are accessed and sorted on account number and invoice number.

Two temporary items, INVOICE-DATE and INVOICE-BALANCE are created to hold the date and accumulated outstanding balance of each invoice.

The two SUM statements accumulate the outstanding balance.

The first FILE statement together with the following two ITEM statements create a new detail record for the interest if the invoice is past due.

The last FILE statement and following SUM statement update the account balance to reflect the new interest change.

2. Standard Batched Update

The standard batch update is probably the most universal QTP application. At the end of each day, a company wants to total all money received and prepare for the next day's transactions. With QTP, this assignment could be performed in ten specification lines.

```
> ACCESS BATCH-HEADER LINK TO TRANS
> SELECT IF TOTAL-ENTERED = TOTAL-CALCULATED
> SORT ON ACCOUNT-NO
> FILE ACCOUNT-DETAIL ADD
>   ITEM TYPE INITIAL "PAYMENT"
> FILE ACCOUNT-MASTER UPDATE AT ACCOUNT-NO
> SUM AMOUNT OF TRANS INTO BALANCE OF ACCOUNT-MASTER
> FILE BATCH-HEADER DELETE
> FILE TRANS DELETE
> GO
```

The ACCESS, SELECT and SORT statements retrieve transactions from balanced batches and sort them by account number.

The FILE statement for ACCOUNT-DETAIL creates payment records from the transactions.

The FILE statement together with the following SUM statement update ACCOUNT-DETAIL to reflect the new payments.

The final two FILE statements delete all processed batches and transactions.

TRANSACTION PROCESSOR STATEMENTS

The major specification statements used in QTP will be as follows:

ACCESS	specifies the files to be read, the order in which they should be read and linkage between files.
BUILD	takes all requests defined up to the BUILD statement and saves these requests into a named MPE file for future use.
CHOOSE	specifies an explicit set of data by key for retrieval.
DEFINE	used to define a frequently used expression.
EDIT	specifies that input files be edited according to editing defined in QSHEMA.
FILE	defines an output action to be performed on a file. These actions are: ADD add if record does not exist UPDATE add if record does not exist, else replace REPLACE replace if record exists DELETE delete if record exists.
ITEM	indicates specific items to be assigned initial or final values, or to accumulate totals.
RESET	resets status control options to original status.
SELECT	restricts selection of records for processing to those which satisfy a condition.
SORT	specifies the order in which records are sorted.
SUBFILE	creates a sequential file (MPE). Does not require the file to exist in the QSHEMA. Will produce its own schema information in the header of the file, and be accessible to both QTP and QUIZ.
TEMPORARY	creates a temporary-item which does not exist in the database.

QUASAR SYSTEMS

DESIGN CONSIDERATIONS

To arrive at a smoothly functioning product, certain design considerations were uppermost in the thoughts of the development team.

- The desirability of a specification based language to insulate users from procedural constructs.
- The need to support complicated production runs as well as ad-hoc file maintenance functions.
- The need for efficient run time performance. Since QTP will run repetitively against bulk volumes of data, its design will differ significantly from a data entry system which requires a high degree of user interaction.
- The need for effective interaction with QUICK to allow class data changes in conjunction with data entry.
- The automatic insulation of users from migrating secondary and other file positioning problems inherent in IMAGE and KSAM file updates.

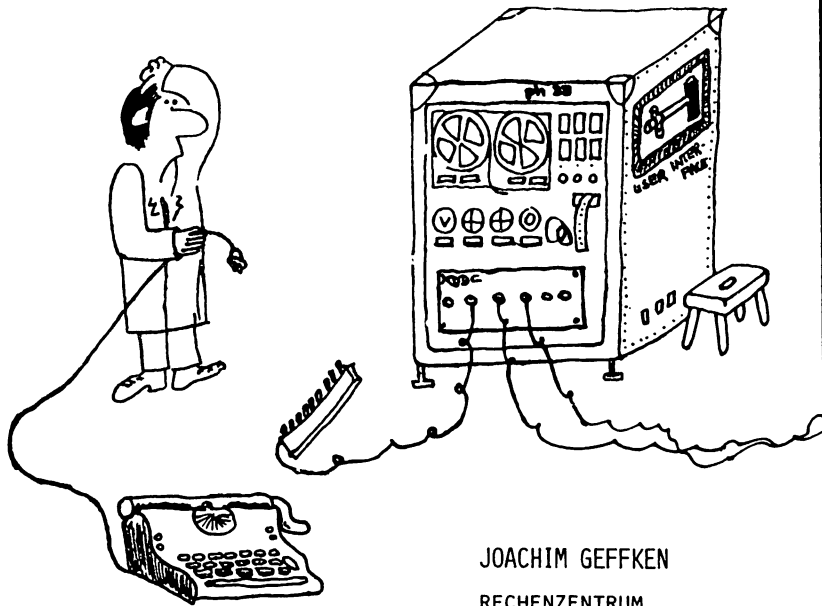
QUASAR SYSTEMS

SUMMARY

Application systems on the HP3000 are typically composed of numerous data entry screens, numerous reports and a relatively small number of batch processes which run on a regular basis at day-end, month-end and year-end. Significant progress has been made towards eliminating the need to custom program data entry and reporting functions. Very little attention has been paid to the development of productivity tools to perform standard batch processing functions. The transaction processor is designed to perform most standard batch operations as well as a wide range of file manipulation functions. QTP, together with its companion products QUIZ, QUICK, and QSCHEMA, will form a complete application generator for the HP3000. Complete systems can be built using these components with major savings in programmer resources applied to development and maintenance, and with real gains in data integrity, system consistency and flexibility.

Integrated Data - and

Textprocessing with hp3000



JOACHIM GEFFKEN
RECHENZENTRUM
HERBERT SEITZ KG
GRÜNENSTRASSE 11/12
2800 BREMEN 1

Survey

→ **Introduction**

→ **File access**

→ **Dataselection**

→ **Correction aid**

→ **Customizer**

→ **Supervisor**

→ **Interface**

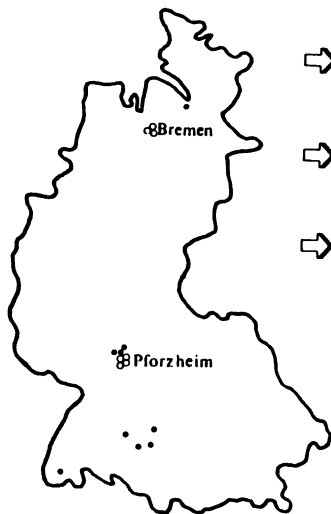


Introduction

The Herbert Seitz Company is ...

- ⇒ A REALTIME DATAPROCESSING SERVICE BUREAU
- ⇒ AND SOFTWAREHOUSE
- ⇒ AND HEWLETT PACKARD OEM

with (1981) ...



- ⇒ 7 OWN HP 3000 (SERIES III AND 44) IN OUR BREMEN AND PFORZHEIM BRANCH
- ⇒ AND 10 HP 3000 SERIES III IN ASSOCIATED COMPANIES
- ⇒ WITH APPROX. 350 TERMINALS SPREAD OVER GERMANY CONNECTED VIA HARDWIRED LEASED LINES/DIALED LINES

Location of:

- own Computers
- Associated Companies

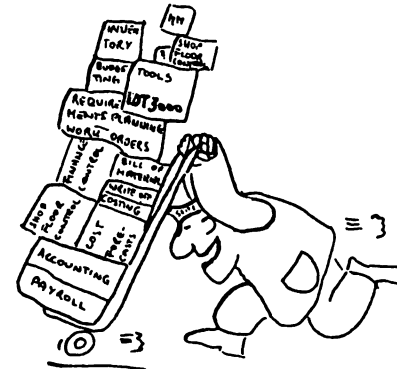
IDT

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 3

Introduction 2

WE PROVIDE OUR SERVICES IN GERMANY AND FRANCE FOR COMMERCIAL APPLICATIONS LIKE ...



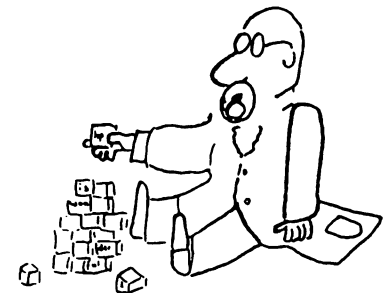
- ⇒ ACCOUNTING
- ⇒ PAYROLL
- ⇒ MATERIAL MANAGEMENT
- ⇒ SHOP FLOOR CONTROL, CAPACITY PLANNING
- ⇒ TOOLS FOR HP 3000 OPERATION, SOFTWARE-DESIGN AND DOCUMENTATION

OUR USERS ARE ...

- ⇒ WORKMEN
- ⇒ DATA TYPISTS, CLERKS
- ⇒ MANAGERS

ONLY A FEW OF THEM ...

- ⇒ ARE SPEAKING (HP-)ENGLISH
- ⇒ HAVE DP EXPERIENCE
- ⇒ HAVE SEEN ANY TERMINAL BEFORE



IDT

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 4

Introduction 3

THIS PRESENTATION IS A GENERAL DESCRIPTION OF SOME TECHNIQUES OF INTEGRATED DATA- AND TEXTPROCESSING ON HP3000 COMPUTERS AS THEY ARE IMPLEMENTED IN IDT3000. THIS IS NOT A COMPLETE PRODUCT OVERVIEW.

IDT3000 IS A DATA- AND TEXTPROCESSING SOFTWARE PACKAGE DESIGNED BY HERBERT SEITZ KG WITH:

- AN IMAGE TEXTDATABASE AND DICTIONARY
- POWERFUL TEXTEDITING AND FORMATTING FEATURES FOR BUSINESS LETTERS AND REPORTS
- FILE ACCESS TO IMAGE-, KSAM- AND MPE-FILES
- A SELF LEARNING DICTIONARY AND AN ON-LINE CORRECTION AID
- MULTILINGUAL SCREENS, MESSAGES AND HYPHENATION ALGORITHMS
- A CUSTOMIZER FOR CHARACTER SETS, TERMINAL- AND PRINTERTYPES, DATAFILES, DATADEFINITIONS AND OPERATING ENVIRONMENTS
- INTERFACES TO DATAPROCESSING AND DATACOMMUNICATION
- A DAILY REPORT OF THE OUTGOING LETTERS AND REPORTS
- A TRACKING MECHANISM FOR RENEWED SUBMISSIONS
- A BATCH PROCESSING INTERFACE

CORRECTION AID / DICTIONARY (1)

GENERAL CONSIDERATIONS:

- THE USE OF A DICTIONARY MAKES ONLY SENSE IF THE VOCABULARY IS SUFFICIENT
- A VOCABULARY OF APPROXIMATELY 150.000 WORDS IS A REASONABLE COMPROMISE (VOCABULARY, DISC SPACE AND ACCESS TIME)
- A STATISTICAL EVALUATION OF THE VOCABULARY DURING THE SELECTION-PROCESS IS ADVISABLE
- THE GENERAL RULE OF THUMB FOR DATABASE CAPACITIES SHOULD BE KEPT IN MIND IN ORDER TO GAIN REASONABLE RESPONSE TIME (I.E. CHECKING OF 100 WORDS IN 2 - 3 SECS.)
- UNTIL THERE ARE BETTER ALGORITHMS AVAILABLE FOR PARSING, INTERPRETING AND UNDERSTANDING TEXT IN HIS CONTEXT IT IS NECESSARY TO MAKE THE DICTIONARY

⇒ USER ACCESSIBLE

AND

⇒ SELF LEARNING

SEE NEXT PAGES ...

IDT

CORRECTION AID / DICTIONARY (3)

Form 2 Maintenance correction aid/ hyphenation exceptions IDT 3009

- (1) Vocabulary into correction aid from 'text name' (A)
- (2) Only from section: (B)
- (3) Print correction aid totally
- (4) Print hyphenation exceptions totally
- (5) Vocabulary from MPE-file _____ (C)
into correction aid
- (6) Vocabulary from MPE-file _____
into hyphenation exceptions file
- (7) Copy correction aid into MPE-file (D)
- (8) Copy hyphenation exceptions into MPE-file

Printer No.

Please enter required activity and press ENTER !

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 8

NEW VOCABULARY MAY BE ENTERED FROM THE TEXTDATABASE (A) OR ONLY SOME SECTIONS (B) OR MPE-FILES (C). IF THE TEXT IS CAREFULLY CHECKED IN THE FIRST PERIOD OF USE THE DICTIONARY WILL BECOME MORE AND MORE SUFFICIENT FOR THE SPECIFIC APPLICATION. THE DICTIONARY MAY BE RESTORED TO MPE FILES (D) FOR BACKUP PURPOSES OR STATISTICAL EVALUATIONS.

IDT

CORRECTION AID / DICTIONARY (2)

Form 4 Maintenance correction aid

word : userfriendliness (A)

(X) Copy display to printer

Please enter new word for Your dictionary !

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 7

THE USER MAY ENTER OR UPDATE DICTIONARY ENTRIES WHENEVER NECESSARY (A).

CORRECTION AID / DICTIONARY (5)

Form 3
Maintenance of hyphenation-exceptions
IDT 3000

Word : ABBREVIATION (A)
 Hyphenation at :

(^) Normal hyphenation after this column
 (K) for ck to k-k hyphenation (special feature for german)
 (V) for consonant duplication (special feature for german)

existing definitions will be displayed

(X) Copy screen to lineprinter

Please mark where You want a hyphenation

THE USER MAY ENTER OR ALTER EXCEPTIONS FOR THE HYPHENATION ALGORITHM (A) WHEN NECESSARY. IN THIS WAY THE STANDARD PRECISION OF APPROX. 95% MAY BE INCREASED >99% FOR A SPECIFIC APPLICATION WITH ITS TYPICAL SET OF VOCABULARY AND SIZES OF THE FORMATTED TEXT.

CORRECTION AID / DICTIONARY (4)

Form 43
Text maintenance
IDT 3000

Text name Section Password
 (1) text entry (4) insert at Text-File
 (2) print on printer (5) modify from
 (3) renumber (6) display from line
 (X) Correction aid (7) delete all, or from line to
 (8) duplicate line / from to
 (9) search pattern / from to
 (0) copy / to

1 11 21 31 41 51 61 71

Start (C)

This is an example of the IDT3000 "self learning" dictionary. The words not contained in the dictionary are displayed in inverse video and the user is asked to check this words.

(D)

Please check your text and correct errors !

THE CORRECTION AID CAN BE ENABLED THROUGH THE USER (A) AND WORKS DURING TEXTENTRY OR UPDATE (B). UNKNOWN (NOT NECESSARY WRONG) WORDS ARE MARKED (C) AND THE USER IS PROMPTED FOR RECHECKING AND CORRECTING (D).

FILE ACCESS (1)

THE INTEGRATION OF DATA- AND TEXTPROCESSING REQUIRES:

- COMFORTABLE ACCESS TO DIFFERENT FILE TYPES (IMAGE, KSAM, MPE)
- USER ACCESS TO ALL FILES WITH READ ACCESS
- UNLIMITED USER ACCESS TO THE CONFIGURATION OF THE OPERATING ENVIRONMENT
- EASY MODIFICATION OF DATA FORMATS AFTER LAYOUTCHANGES
- DATA ACCESS VIA DATA DICTIONARIES

SEE EXAMPLES NEXT PAGES ...

IDT

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 12

CORRECTION AID / DICTIONARY (6)

Form 6 Pattern Maintenance IDT 3000

No. Pattern (A) (B)
300 \$(IDT 3000 the integrated Data- and Textprocessing Software\$)

Existing patterns will be displayed !

(X) Copy screen to lineprinter

Please enter or modify pattern!

COMPLEX TEXT EXPRESSIONS MAY BE DEFINED AS TEXT PATTERNS (A), IF NECESSARY WITH TEXT FORMATTING COMMANDS (B). PATTERNS ARE CALLED WITH THE COMMAND &nnn (nnn = PATTERN NO.),

IDT

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 11

IDT

FILE ACCESS (3)

Form 5 Customizing data access (address-file and master-file 2) IDT 3000

A (A) Customizing address-file (B) Customizing master-file 2

In this form You may customize Your Address- file

You will have to describe each field (1 - 50). Existing descriptions will be shown.

Field No.	Fieldname (B)	Start Data (1)	Size (2)	Format (3) (F)
12	(A) Name	019	28	

(1) P4-P12 packed numeric item (C) (D) (3) 'X' place for 1 alphanumeric character
 C2-C10 binary coded numeric item 'Z' supress zeros
 F alpha-numeric item

(2) Size : the last digit must be a sign character!
 you may need one digit for a sign character!

(X) Copy screen to lineprinter

Please enter fieldformat!

UP TO 100 FIELDS PER USER AND/OR SESSION CAN BE CONFIGURED (A). THE NAME (B), POSITION (C), DATA TYPE (D), SIZE (E) AND OPTIONAL EDIT MASKS (F) CAN BE DEFINED AND MODIFIED WHEN NECESSARY.

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 14

IDT

FILE ACCESS (2)

Form 2 File Configuration IDT 3000

Specification of Address-File: Filename KOSTAM

Filetype

(1) IMAGE-DB ADRSTA Password MGR
 Searchitem ADRNR (Detail Dataset)

(A) (2) KSAM-File (Access via Primary-Key)
 (3) MPE-File (sequential, only for Serial Letters)
 (4) Address-File not used

Specification of the Order File : Filename [STTD]

Filetype

(1) IMAGE-DB PRODO1 Password MGR
 Searchitem [DTNR] (Detail Dataset)

(B) (2) KSAM-File (Access via Primary-Key)
 (4) Second Master-File not used

(X) Copy screen to lineprinter

Please enter data and press -ENTER-

THE USER MAY DEFINE OR CHANGE FILENAME AND TYPE FOR AN ADDRESSFILE (A) AND ANOTHER USER SELECTABLE FILE (B).

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 13

FILEACCESS (5)

Form 4 Businessletters IDT 3000

Text name Section Password

(1) From unformatted text or (2) formatted file to printer No.

(1) A letter to addressee: (B)

(A) (2) without accessing the address-file

(3) A letter to all addresses of the address-file

(4) Only selected addresses with field field

(X) Using letterhead: _____ dated for renewed submission : _____

(X) With data of Order - Master-file of (C)

(X) Margin alignment _____ Column (only printer 2501)

Your sign _____ Your letter _____ Our sign _____ Date _____

Please enter the required processing/selection options and press -ENTER- !

THE FILEACCESS (AND THE DATA SELECTION) CAN BE DONE DURING THE PRINTING /
 SELECTING OF BUSINESS LETTERS (A). DATA MAY BE SELECTED AND INSERTED FROM
 THE ADDRESSFILE (B) AS WELL AS FROM THE MASTER FILE 2 (C).

IDT

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 16

FILEACCESS (4)

Form 2 Formatting the text IDT 3000

Text: name Section Password

(1) Formatting text (text file -> formatted text file) Columns 55

(A) (2) Print formatted text on printer No.

(X) Insert addressee of 1000003 (B)

(X) Add from master-file2 Orders from 935-001/21C (C)

(X) Blockformat required

Please enter the required options and press -ENTER- !

THE FILEACCESS (AND THE DATA SELECTION) CAN BE DONE DURING THE TEXT FORMATTING (A).
 DATA MAY BE SELECTED AND INSERTED FROM THE ADDRESSFILE (B) AS WELL AS FROM THE
 MASTER FILE 2 (C).

IDT

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 15

CUSTOMIZER (1)

Form 1 Customizer for Hardwareconfiguration IDT 3000

(B) Char.Set : (1) ASCII/UK (2) Deutsch(ISO) (3) Francais (4) Espanol
 (C) Language : (1) English (2) Deutsch (3) Francais (4) Espanol

(X) Auditfile used. (E) addressee in field No. (D)
 (X) Database for Hyphenation-Exceptions used
 (X) Correction-Aid used

Printer	Type	Device-Class	Printer	Type	Device-Class
1		P2601 (A)	2		P
3			4		
5			6		P2601
7		34	8		35

(A) Terminal-Printer (B) Lineprinter 2513/2617/2619
 (C) Matrixprinter 2608/2631 as Hardcopy (D) as Device
 (E) Daisywheelprinter 2601 as Hardcopy (F) as Device
 (H) Olympia ESW100RD as Hardcopy (G) Laserprinter 2680
 (X) Copy screen to lineprinter

Please enter data and press -ENTER-

THE INTEGRATION OF DATA- AND TEXTPROCESSING REQUIRES THE OPTION OF TEXTPROCESSING WITHIN THE EXISTING DATAPROCESSING (HARDWARE-)ENVIRONMENT (A) INCLUDING CHARACTER SETS (B), LANGUAGES (C), DATA LAYOUTS (D) AND THE PROCESSING ENVIRONMENT (E) + (F).

IDT

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 20

DATA SELECTION (3)

Form 183 Text maintenance IDT 3000

Text name DEMOBER0 Section 82 Password Text-File

(1) text entry (4) insert at
 (2) print on printer (5) modify from
 (3) renumber (6) display from line
 (7) delete all, or from line _____ to _____
 (8) duplicate line _____ / _____ to _____
 (9) search pattern _____ from _____
 (X) Korrektur (0) copy _____ / _____ to

11 21 31 41 51 61 71

-Start* (A)
 This is an example of IDT300's datainsertion feature.
 Dear Mr. &A12.
 Thank You for Your letter and Your interest in our new &B31. The price of &B35 is very attractive.
 Our local representative Mr. &D\$SALESMAN will contact You within the next few days and provide further information for You.
 Sincerely Yours \$R\$R&Vsigner\$R&Vtitle (B)

DATA ELEMENTS CAN BE DEFINED BY IDT3000 INTERNAL FIELD NUMBERS (A) (REFERRING TO THE CUSTOMIZED FILE-ENVIRONMENT) OR BY DATA ELEMENT NAMES OF A DATA DICTIONARY (B) (DICTIONARY 3000).

IDT

RECHENZENTRUM HERBERT SEITZ KG

PAGE: 19

A DISTRIBUTED COMPUTER SYSTEM INTERCONNECTING HP3000,

HP 1000 AND OTHER MINI - COMPUTERS

BJÖRN DREHER, HANS VON DER SCHMITT, RAIMOND SCHOECK

INSTITUT FÜR KERNPHYSIK DER UNIVERSITÄT D-6500 MAINZ

WEST GERMANY

A Distributed Computer System Interconnecting HP3000, HP1000,
and other Mini-Computers

Björn Dreher, Hans von der Schmitt, Raymond Schoeck
Institut für Kernphysik der Universität
D-6500 Mainz, West-Germany

1. Introduction and early history

The Institut für Kernphysik der Johannes Gutenberg-Universität at Mainz, West-Germany, is a medium size institute for basic research in the field of Nuclear Physics. We have an 340 MeV linear accelerator for electrons to perform research in the nuclear structure area using electromagnetic interaction. Currently an 175 MeV two stage c.w. race-track microtron for electrons is under construction, which is controlled and operated with the help of two HP1000 computers. The first of the two stages is operational since 1979.

Until 1976 the control of experiments and the data acquisition was performed by a (today still operational) CDC1700 "minicomputer". In addition a substantial part of the data analysis was also done on this system. By that time we noticed that the computing power and the tools for program development of the CDC1700 were no longer adequate for our tasks. Therefore we decided to purchase a new powerful minicomputer system (HP3000) for the data analysis part of the work and to connect to it several smaller systems (HP1000) as front-ends for the real-time applications.

Since at that time the cost for disc memories was higher than today and memory-resident operating systems were still around, we wanted to be able to perform the program development to a large extent on the HP3000, even for the front-ends. Operating systems were to be generated on the HP3000 and then downloaded to the small system.

In addition, to reduce the cost for the front-ends, these should be equipped only with experiment related peripherals, such as CAMAC interfaces, and not necessarily with expensive magnetic tape transports or line printers. A concentration of those devices at the HP3000 would promise a much higher utilization and availability to all front-end computers.

At that time DS3000/DS1000 was not yet available, but HP had available a product called "Programmable Controller", which was an HP2100 (or at that time already an HP21MX) computer connected to the HP3000 via a 16-bit parallel link using Universal Interfaces at both ends. With this product came cross software that enabled the user to generate RTE-C operating systems, to assemble HP1000 Assembler programs, bring it into an absolute form using a Cross Loader (XL2100) and download it to the front-end computer. There existed also an (unsupported) Cross FORTRAN compiler that was compatible with those days' FTN-IV compiler of RTE-III. Our final configuration is shown in figure 1. The two interconnected HP1000 systems are today running RTE-IV operating systems, the third one used to work with RTE-C and we are currently in the process of rewriting our applications to be compatible with RTE-IV. For the CDC1700 we built an interface to let it appear to the HP3000 like an HP1000 computer, so that we could connect it to a third Universal Interface card.

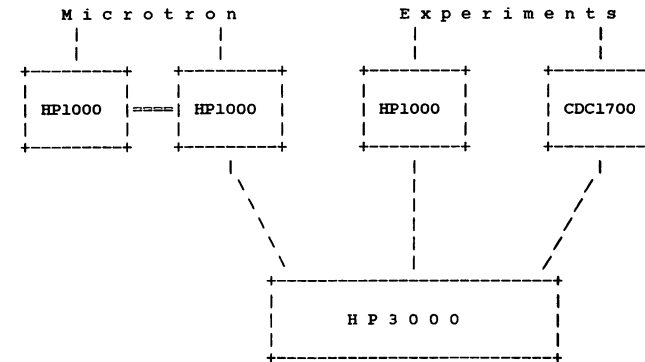


Fig. 1: Overview of our distributed computer system (the CDC1700 will be replaced by a PE3220 system).

There existed already some communications software for a similar, but in some essential points different, distributed system at the Technical University in Berlin. This had been written jointly by HP Frankfurt and the Technical University of Berlin. It was kindly made available to us and constituted the basis for our current communications system between the front-end computers and the HP3000.

2. The current system

In the process of getting the initial system running it turned out that both communication drivers in the HP3000 (IOREMO) and in the HP1000 (DVR63) were not suited for our problem. Therefore we had to modify both, especially the HP1000 side. Now the communication between the two computers is really interrupt driven on both sides without the need to loop on a status request to see whether the other side is willing to send, to receive, or to do nothing at all.

Today three computers are connected in a star configuration to the HP3000 and a fourth one will be added in the near future. The peripherals of the HP3000 are accessible to the small computers through the file system or directly via special communication drivers in the HP1000 in the case of magnetic tape units, line printer, and plotter. Direct program to program communication between a program on the HP1000 and one on the HP3000 is also available.

In the following we give an overview about the possibilities a user on one of the front-end computers has in the current system:

a. Access to the entire file system of the HP3000

- Read and write sequentially or in direct access
- Position to records, write filemarks, space filemarks, rewind, etc.
- Obtain the status of a device

A supervisory program in the HP3000 (CENTRAL) keeps track of all files opened by programs in the front-end computers, so that only those programs in the front-end computers may access the files who "own" it. When a program closes the connection to the HP3000, all files opened by it are automatically closed. In addition there is the option to open files globally, so that they can be accessed by more than one program simultaneously.

b. Initiate batch jobs

c. Create and activate processes

d. Program to program communication between programs in the front-end computer and programs in the HP3000.

e. Perform MPE commands

f. Generate an RTE-C operating system on the HP3000 and download it into the target machine. Develop application software in FORTRAN-IV or HP1000 Assembler, bind it into the target system and download it dynamically into the target machine.

g. Transparent use of peripherals of the HP3000. Magnetic tape units and the lineprinter are accessible from the front-end processors as if they were connected directly to them, e.g. through FORTRAN READ/WRITE statements or EXEC-calls. The plotter is available through standard (Calcomp-) calls. This concept is easily expanded to other peripherals.

According to the initial demands to the system, communication is normally initiated by one of the front-end computers. Each request consists of a pair of messages of variable length. The first message contains the request type and the necessary data. The second (return) message contains possible error codes and the resulting data. The interface on the HP1000 side is a set of subroutines (SMTL) that allow the programmatic execution of all features mentioned above, or a direct EXEC call to the drivers of the (virtual) peripheral devices attached to the HP3000. An interactive program (KOPPL) allows to exercise all requests to the HP3000 and to transfer files from one machine to the other.

The receiving process in the HP3000 is one program (CENTRAL), which performs all necessary operations to satisfy the individual requests. This process runs as an always present batch job (in the CS queue), one job per link to a front-end computer. Each front-end computer can have up to 5 programs communicating with the HP3000 at the same time, each of which may have up to 10 files simultaneously open. Those numbers are more or less arbitrarily chosen and can be easily increased when the need arises. Requests from different programs can be freely intermixed.

3. Need for additional functions

As one can see from the previous chapter, the current system constitutes a fixed master/slave relationship between two communicating computers, where the front-end processor is the master and the HP3000 is always the slave. This was satisfying for the first few applications, but soon it turned out that a dynamic establishment of the master/slave relationship and a more direct communication between

programs/processes in the various machines would make many applications easier to implement. In particular, it should be possible to start an exchange of messages from any computer.

A general process to process communication across the entire distributed system seemed to be the most attractive solution. Of course, the existing higher-level functions of our current system should not be touched.

4. The HP1000 message system

Fortunately a system that fulfilled many of those needs had been implemented in 1978/79. It was developed in our institute for the inter-process communication (IPC) among the various on-line control programs which are distributed within the two HP1000 computers that control the new microtron mentioned in chapter 1.

The communications protocol of the process-to-process layer is based on the exchange of messages via just two operations: SEND and RECEIVE. Messages have the same form whether being exchanged locally in one computer or between the two computers. The participants of the communication are addressed by 10-bytes symbolic names. These are mapped to target computer and process by the system.

Messages are transmitted as sequences of fixed-length packets in a store-and-forward fashion. Each packet is 64 bytes long consisting of a header (essentially sender and receiver addresses) and the data. The packet transmission constitutes the computer-to-computer protocol, which is thus very simple and easy to standardize.

In addition, I/O requests to devices attached to remote computers are transparently handled by using IPC between I/O drivers.

In summary, this message system enabled us to build a modular distributed control system. The modules (programs) are explicitly portable between the two computers by using the symbolic addressing scheme and by the transparent I/O system. Thus programs can be freely redistributed on demand in a growing system, as our microtron control system is.

5. The second generation communications system

Based on these ideas, we are currently in the process of implementing a second generation of our communications system uniformly throughout our distributed system. It will interconnect one HP3000 with three HP1000's and one PE3220 computer.

The new system will also be an IPC system transmitting messages decomposed into packets. However as compared to the above application, the system must be capable of higher data flows since the number of data bytes per message will be larger on the average. Therefore the packet size will be increased to 128 bytes. On the other side there is less need for a fully symbolic addressing capability and other overhead-generating features of the HP1000 message system. Thus a higher data throughput may be achieved. Some essential features of the new system will be given below.

From an architectural viewpoint, messages between two processes will be transmitted in our system by a store-and-forward packet switching mechanism.

Splitting messages into a number of smaller packets has important advantages:

- a. Long messages do not necessarily monopolize a communication line. They can be interleaved with packets from other (more urgent) messages.
- b. By using fixed length packets as the vehicle of message transfer, it is easy to buffer messages prior to the actual transmission and incoming messages before they are collected by their recipient. Buffering space will be taken from a global packet pool common for all ports of a particular computer.
- c. Buffering separates nicely the lower communications layer, that controls just the traffic of incoming and outgoing packets, from the next higher layer, that consists essentially of the decomposition of messages into a series of packets (SEND operation), the reconstruction of messages from a series of packets (RECEIVE operation), and the mapping of symbolic addresses.
- d. A store-and-forward capability comes as a by-product from the buffering.

Besides packeting/de-packeting, packet transmission, and buffer management, the system has to perform additional monitor functions on the participating processes.

- i. Processes must be suspended and re-activated in the course of SEND and RECEIVE requests.
- ii. Time-out conditions may arise in RECEIVE as well as in SEND requests; the former when an expected message is not received in time, the latter when a transmitted message is not consumed by the recipient in time.

From the hardware point of view we will still have point-to-point connections between the various computers in a star configuration (with the exception that two HP1000 systems are connected directly with each other), the HP3000 being the inner node. We will use the existing hardware (16 bit parallel plus some control lines) with Universal Interfaces in the HP-computers on both sides. However, as can be seen from point c above, the kind of hardware is not that important for the designed function of the whole system.

6. Implementation

To reduce programming time and to improve maintainability and documentation of the system as well as portability we decided to write as many as possible routines in a high-level language. For the non-HP3000 systems we decided to use RATFOR [1], which is a FORTRAN dialect that adds structured elements to FORTRAN-IV. The output of the RATFOR preprocessor is standard FORTRAN-IV, which serves for the portability of RATFOR programs. Only few routines on the HP1000 systems are written in HP1000 Assembler.

The same approach will be taken for the PE3220 system, which will replace the CDC1700 computer.

Since it is expected that the HP3000 will have the highest message traffic of all computers and since several routines have to perform their tasks in privileged mode, we decided to write the HP3000 side in SPL, at least the inner kernel with the most time-critical parts of the system. This allows us to make the best use of the HP3000 architecture and its instruction set, thus lowering the time overhead introduced by the packet switching architecture.

7. Conclusion

In summary, even the current (first generation) system and the dedicated HP1000 message system have proved very valuable in many daily applications. The second generation system, that will be available on all our in-house (mini-)computer systems, will have an even higher impact on many current and future applications. After 4 years of experience with an own, custom designed, communications system, we feel that in our case this was the right way to go. Even today, there is no communications system commercially available that would fulfill exactly all our needs. Having all the knowledge about the system in house allows us quite easily to add new required function to the system, as they arise. The fact that most modules of the system are written in a high-level portable language makes it easy to put the system on other computer families and integrate them into our distributed system.

[1] B.-W. Kernighan, P.J. Flauger: Software Tools, Addison-Wesley Publishing Co. 1976

THE USE OF EDP IN THE FREIGHT FORWARDING

AND SHIPS AGENCY BUSINESS

HARDY JENSEN

JORGEN RIX

H. JENSEN

J. RIX

DATA ADVICE A/S

NIELS BOHR'S VEJ 3

DK-6000 KOLDING

CALL FOR PAPERS

Presentation Abstract

Presentation Title: The use of EDP in the freight forwarding and ships agency business

Author(s): Hardy Jensen, Jørgen Rix

Title(s): Sales Manager, Sales Director

Address: Data Advice A/S, Niels Bohrs Vej 3, DK-6000 Kolding

Abstract: (No more than 200 words)

Data Advice A/S, is an OEM- and software house in Denmark, which have specialized in the freight forwarding and ships agency business.

Data Advice A/S has therefore designed and implemented a total freight forwarding and ships agency package for use on HP3000 machines.

In this package Data Advice A/S have integrated the forwarding and the book-keeping procedures so that double data registration has been avoid, also is the system build so, that is uses the advantage of an on-line machine i.e. information typed in by one person is at once accessible to all users of the system.



Abstract: (No more than 200 words)

DA
DATA ADVICE

DATE:

SAG:

SIDE:

02

5

Briefly the system, among other things, contents of following functions:

- Import and export routines from the point where the order is taken through part-/full-load bookings, way-bill handling, haulage planning, loading lists, unloading lists, manifesting, customs clearance (both import and export), automatic invoicing to total financial accounting including book-keeping on all relevant accounts.

TRANS AND FUTURE DIRECTIONS OF HP PERIPHERAL PRODUCTS

W. J. MURPHY

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

W. J. MURPHY
HEWLETT PACKARD

NEW SOFTWARE ENGINEERING ALTERNATIVES

BIRKET FOSTER

NEW SOFTWARE ENGINEERING ALTERNATIVES

(ABSTRACT)

notes on selecting software by Birket Foster, m.b.foster associates ltd.
2755 draper place, ottawa phone (613) 820-5067

The objective of this talk is to help hp3000 users in the software selection process. It is recognized, that many applications and utilities could never be written using in-house expertise or resources (adager, quiz,qedit, 8image, etc) there is a growing community of HP3000 software vendors to which many HP3000 sites are turning to speed the systems development life cycle.

For most sites the process of selecting software is a new one. This paper presents a framework to help hp3000 users avoid the pitfalls of purchasing someone elses software to process their application.

The areas, covered in this paper are:

1. introduction
- 2.the selection framework
 - a) scope
 - b) features
 - c) rating system
 - d) selecting committee
3. souces of supply
4. other information required
5. the decision
6. the legal contract
7. conclusion

***** SORRY, BUT DUE TO POSTAL STRIKE THIS PAPER DIDN'T REACH US
IN TIME, ASK THE AUTHOR FOR COMPLETE DOCUMENTATION - (EDITOR)

JOBLIB/3000 - AN INTERACTIVE PRE-PROCESSING SYSTEM

Martti Laiho
OY PORASTO AB
Toolontullinkatu 8
SF-00250 Helsinki 25
FINLAND
telex: 125194 PSTO SF

MARTTI LAIHO

JOBLIB/3000 - an Interactive Pre-processing System

Introduction

This presentation is a progress report on the development work made at Oy Porasto Ab since 1977 to make better use of the automatic data processing capabilities in HP3000 systems. For earlier reports refer to /1/ and /2/.

Most automatic processing is in batch mode, but the old way of batch using "fixed" stream files suffers from some severe drawbacks:

- security of passwords on JOB stream files
- troublesome updating of parameters and optional parts of streams when using general purpose editor programs, which often leads to mistakes, and
- many copies and versions of same stream files on disc
- documentation problems of stream updating with editors in transferring the JOB preparation and initiation work to end users of the system
- a typing error in the filename of a :STREAM command may start an undesired JOB.

Too often these problems are avoided by letting end users do the work from time to time, making them wait for the execution of a series of tasks on-line, even if all responses could be supplied beforehand.

These problems can be solved easily by pre-processing techniques. That is by updating the stream file under control of a general purpose pre-processing program, according to stored processing rules for the particular job stream (or task). To provide best support for the user this pre-processing must be interactive.

MARTTI LAIHO
Oy PORASTO AB
TOOLONTULLINKATU 8
SF-00250 HELSINKI 24
FINLAND

JOBLIB/3000 system

The problems listed above are solved by JOBLIB/3000, general purpose pre-processing system for HP3000, in the following ways:

In the JOBLIB system, job streams are gathered as "job templates" into special library files, leaving all information concerning user, account, group and passwords out of the !JOB-lines.

To initiate a job stream, a user simply starts the program JOBGEN, provides the name of job-library (default is JOBLIB) and the particular JOB required. Passwords are either supplied automatically from some secure data base (this is a new option in JOBGEN 5.5) or they are prompted from the user.

Updating of parameter values in the stream is made by character string replacement operations, using **mnemonic string variables**, the only data format for variables in JOBLIB command language - job generation language (JGL), the lines of which are inserted in the job templates to control the flow of pre-processing. The string variable processing is implemented in a very powerful way providing full text processing capabilities found in many programming languages, and they can be used also as numeric variables if the character string value assigned to them happens to be numeric. Of course this can be ensured by some data checking facilities of JGL.

In many pre-processing systems, the value assignment for string variables is made by commands such as
DEFINE (identifier, "value")
(see /3/ p. 251), but in JOBLIB system we use automatic assignments of the type

```
..SET identifier="value"  
..COMP identifier=(arithmetic expression)  
and interactive assignments of type  
..DISP <advice>  
...  
..READ identifier . ="default"; checking rules .
```

where the preceding ..DISP commands are used to provide the user with further information concerning the prompted parameter value.

Values of string variables can be any character strings of 0 to 80 ASCII characters in length and restricted only by the data checking rules, if used. Currently there can be up to 150 different string variables used in one job definition.

The assigned string value will then be replaced at every occurrence of the variable name, preceded by "&", in any lines of the job template that the name occurs. These are then copied into JOBGEN work memory before they are interpreted and/or copied into the final stream file.

Values of string variables can also be used to control the flow of pre-processing by the structured commands

```
..IF , ..WHILE and ..CASE
```

where the values of string variables can be tested in the condition part of the command using relational and logical operators (which will be fully implemented in release 5.5). This provides the possibility to select alternative or optional parts in the final stream automatically, according to parameters supplied by the user.

To achieve full flexibility and automaticity in assignments, replacements and conditions, we have implemented substring operations and string processing functions such as: \$LEN(s), \$UPS(s), etc.

For maintenance of parts common to many JOBS and to make use of generalized parts, we have also implemented a **procedure library** facility in JGL. This feature has been especially appreciated by those users of JOBLIB system, who are familiar with the procedure library facility of IBM's operating systems. The procedure library provides an easy way to:

- use parametrized QUERY reports included in different JOBS
- maintain tables of parameter values in separate library files
- define generalized parts of JOBS for making backups or for other routines, etc.

The procedure calls (..INCL -commands) can be hierarchical and even recursive. The outer pre-processing can be secured by using local string variables inside the procedures, and using string variables as parameters in procedure call. Values can be transferred into and out of the variables used in procedures, just like in programming languages.

Instead of a procedure, a whole file can be included during the pre-processing. This provides possibilities for including for example

- stream parts generated by some other programs into the final stream
- source programs with inserted JGL control lines and string replacements into a compilation stream, making use of interactive pre-processing for COBOL programming. This goes even far beyond the capabilities of COBOLII.

Benefits

The JOBLIB system makes batch jobs easier to use, and it provides a comprehensive technique for system personnel to submit job preparation work to operators and even to end users. All the user has to know is the name of the JOBLIB, the name of the job, and how to answer the questions presented in the job template.

For system personnel, the JOBLIB system provides the possibility of tool-jobs, a flexible test environment, a procedure library for common parts, and a good on-line documentation technique. From elements of macro processing and programming languages, we have created a new and efficient "productivity tool" that provides a dynamic and flexible extension of traditional job streams. The JOBLIB system is not just a library system, but a new approach to batch processing in an on-line environment.

The benefits of JOBLIB system are clear: it saves human and computer resources, time and money, eliminating errors and increasing security and productivity.

Future plans

We are currently (August 1981) using release 5.4 of the JOBLIB/3000 system and we are developing release 5.5.

As the main trend of development we are making JOBLIB/3000 as "open ended" a system as possible. This means that it can be used in different kinds of system environments, it can be integrated with different software, and it will be more and more customizable. Examples of this are:

- We have separated all program messages of JOBGEN into a separate catalog file, so that they can be easily customized or translated into any other national language that can be written using ASCII letters.
- We have added the option of using a fully customizable data base, to supply required passwords into !JOB-lines of stream files. This is delivered with sources of interface modules and using pre-processable installation streams.
- Currently JOBGEN supports SLEEPER as the scheduling monitor, but this will be made more open to other systems using a separate :STREAM -time reservation module delivered on source level.
- JOBGEN will also be accessible as a son process of customer's own application monitor programs and it can be controlled directly by a message file of procedure type, generated by the father process.

Availability

We at Porasto believe that the JOBLIB system is of use in almost all HP3000 installations and we have made it available on an yearly rental basis. Currently it has been installed in 20 HP3000 systems in Finland and Sweden. For further information, please contact **Oy Porasto Ab**, att: Martti Laiho, Toolontullinkatu 8, SF-00250 Helsinki 25, FINLAND.

References

- / 1 / JOBLIB - Generalized Job Library and Interactive Job Generating System, Martti Laiho, Proceedings of HPIUG Fall Meeting 1980, pp 9.5.1-9.5.10
- / 2 / JOBLIB/3000 - Job-library and Pre-processing System, Simon Harryman et al, HPIUG Spring Meeting 1981
- / 3 / Kernighan, Plauger: SOFTWARE TOOLS Addison-Wesley, 1976, ISBN 0-201-03669-X.

(This presentation has been written with DAISY/3000 text processing system)

USING IMAGE-3000 TO ESTABLISH AN ORDER PROCESSING--
FINISHED GOODS INVENTORY ON-LINE DATA BASE SYSTEM

K.B. SHEU

WALSIN LIHWA CABLES
ELECTRIC WIRE & CABLE CORPORATION

THE WALSIN BUILDING
219 CHUNG HSIAO E. ROAD, SEC 4
TAIPEI TAIWAN R.O.C.

Speaker: K. B. Sheu

**WALSIN LIHWA
CABLES**

SHEET NO

ABSTRACT (1)

TOPIC:USING IMAGE-3000 TO ESTABLISH AN ORDER PROCESSING--FINISHED
GOODS INVENTORY ON-LINE DATA BASE SYSTEM
WALSIN LIHWA CABLE CO.LTD.,APPLY IMAGE-3000 TO ESTABLISH AN ON-LINE
DATA BASE SYSTEM WHICH INCORPORATES 4 MODULES----PRODUCT DATA MODULE,
CUSTOMER INFORMATION MODULE, ORDER PROCESSING MODULE AND FINISHED
GOODS INVENTORY MODULE. THESE MODULES ARE LINKED BY SEARCH-ITEMS.
USING THE CAPABILITIES OF IMAGE-3000, A TWO LEVEL HIERARCHICAL DATA
BASE SYSTEM CAN BE IMPLEMENTED AND SUITABLE FOR ON-LINE OPERATION.
SEVERAL OTHER TRANSACTION FILES ARE ALSO RELATED AROUND TO THIS
SYSTEM TO FORM A TOTAL INTEGRITY.

***** WAITING FOR THE FULL TEXT OF THIS PAPER .. (EDITOR) ***

WALSIN LIHWA ELECTRIC WIRE & CABLE CORPORATION
The WALSIN Building 219 Chung Hsiao E. Road Sec. 4 Taipei Taiwan R.O.C.
Tel 771 2121 (20 Lines) P. O. BOX 22926 Telex 11516 WALSIN Taipei
Cable WALSIN Taipei

COMPUTER GRAPHICS, A POWERFUL INFORMATION TOOL

SIGMUND HOV MOEN
FREDERIK MAJOR

SIGMUND HOV MOEN A/S SYDVARANGER, KIRKENES, NORWAY

FREDERIK MAJOR THE SHIP RESEARCH INST. OF NORWAY TRONDHEIM, NORWAY

COMPUTER GRAPHICS, A POWERFULL INFORMATION TOOL.

Authors: Sigmund Hov Moen, A/S Sydvaranger, Kirkenes, Norway
Fredrik Major, The Ship Research Inst. of Norway,
Trondheim, Norway

Information transfer from/to human beings is a central part of a computer system. We will here show how Computer Graphics can make man machine communication extremely much more efficient. A case study of a Norwegian mining company, A/S Sydvaranger, will illustrate this fact. This company uses Computer Graphics for information transfer between people at all levels.

The graphic system in use for the above mentioned application is G P G S - F (General Purpose Graphic System in Fortran) which is the standard graphic system for Norwegian users.

This standardization has been pushed through by NORSCOD, The Norwegian Cooperation in Computer Graphics, formed by Norwegian users.

GP GS-F which here will be briefly described (A more detailed description in (1)) is implemented for 12 different computer types with device drivers for 17 different graphic devices. There are approximately 70 installations of the system today throughout the world, 10 of these are HP 3000 installations.

BRIEF DATA ON AKTIESELSKABET SYDVARANGERs PRODUCTION

J5 4

THE BASIS for the company's existence are the very substantial iron ore deposits situated on the Kirkenes-peninsula combined with excellent deep sea harbour facilities in Kirkenes, only 5 miles from the main orebody at Bjørnevattn.

AKTIESELSKABET SYDVARANGER was founded in 1906. The company today has 2300 shareholders. The Norwegian government owns 51% of the shares.

THE PLANT is today designed and built for a total production capacity of 2.5 million metric tons iron pellets (minimum 65% Fe).

THE MINING in Bjørnevattn is carried out as an open cast operation. The open pit is considered large by Norwegian standards. The present production plans operate with a stripping ratio of 2.5:1, i.e. 15 million tons of waste have to be removed to enable recovery of our annual production of approx. 6 million tons of magnetic ore which contains about 30% iron.

DRILLING is mainly carried out by means of Bucyrus Erie 60 R rotary drills producing holes of 12 1/4 inch diameter.

BLAST CHARGING is done with ANFEX (Ammonium nitrate-fuel explosives) and TNT-slurry. Each hole is charged with approximately 1 ton of explosives. Total blast size varies between 75.000 tons and 750.000 tons of material broken, and the equivalent amount of explosives used varies between 30 and 300 tons.

LOADING OF MATERIAL is carried out with track mounted P & H electric shovels with 7 m³ bucket capacity and one Marion 191 and one P & H 2100 shovel, each with 11 'm (15 'yds) bucket. In addition to this several wheel loaders are used, the largest of which is a 10 'yds Caterpillar 992 and Dart 600 12 'yds.

ORE AND WASTE TRANSPORT is taken care of by a fleet of 100 and 150 tons Lectra Haul and Haulpack trucks.

PRIMARY CRUSHING is done in Bjørnevattn by means of (two) 54" Nordberg gyratory crushers (54" wide intake opening) which is normally set at 5 1/2" closed side, giving ore crushed down to approx. minus 5".

ORE COBBING is carried out on conveying it over large magnetic drums separating most of the waste (15% - 20 %) from the ore before it is discharged into the ore storage silo.

THIS CRUSHED ORE is transported by our own railway in 60 tons capacity cars and dumped in large ore bins at our secondary crushing plant. Each train has 20 cars and thus carries approx. 1200 tons.

OUR TWO STAGE SECONDARY CRUSHING PLANT uses 7" Symonds cone crushers and reduces the ore in size to approx. minus 1 inch.

PRIMARY GRINDING of this material is carried out in a two stage ball mill process where water is

added as a carrying agent. After this grinding approx. 30% of the material is separated out as waste over series of magnetic drums. This product or concentrate, is ground further to make it suitable for pelletizing.

This concentrate contains approx. 68% Fe.

PELLETIZING, a process which transforms the concentrate into the shape of small 1/2" diameter balls is carried out by adding approx. 1% of a binding agent called bentonite (basically a special, dry, finely ground clay). This mixture is rolled in large inclined drums which produce small balls (green pellets).

These are dried over a travelling grate, preheated and sintered. The «pellets» will have a temperature in excess of 1000° C when leaving the grate. From here they enter a rotating oven or kiln where they are heated further to approx. 1350° C. The hot pellets are now discharging into a cooler section where they are cooled down to 20-30° C by a forced air draught.

This product is stored in a large 400.000 tons capacity silo blasted into the mountain.

THE HARBOUR in Kirkenes is a good natural harbour which has little problem with ice in winter, in spite of its location at 70° North. The harbour facilities has a capacity for loading ships up to 150.000 tons at a rate of approx. 4.000 t/h.

AKTIESELSKABET SYDVARANGER today has approx. 1250 employees in Ser-Varanger

1 COMPUTER GRAPHICS IN A MINING COMPANY

1.1 A/S Sydvaranger

"Raw data".

A/S Sydvaranger is a mining company situated i Kirkenes, a small town on the russian border far up north an far east in Norway.

The mine is an open pit one and contains low grade iron ore which are concentrated to the proper quality in a conventional crushing/milling/separating plant. Our final product is iron pellets which is small balls (about half an inch in diameter) containing 65% iron.

Further information about the company and the process is given on the next pages.

J5 3

1.2 An EDP pioner in Norway

A/S Sydvaranger installed its first EDP equipment in 1963. At that time a computer was a scarcity even at the Norwegian universities.

The first machine was an IBM punched-card-based 421.

Part of the companies staff-department-routines were implemented in the first place and soon afterwards a first version of a stock controlling and accounting system.

In 1966 a new and more powerful computer was installed (IBM 360/20) and the main part of the companies accounting system was "computerized".

An information system for the mining operations was also developed.

In 1974 the 360/20 was scrapped and after a short interludium where different alternatives were tested, the company in 76 bought and installed a HP 3000.

Today A/S Sydvaranger heavily depends on the computer in the following areas:

- . Production reporting
- . Accounting
- . Stock control
- . Payment of wages

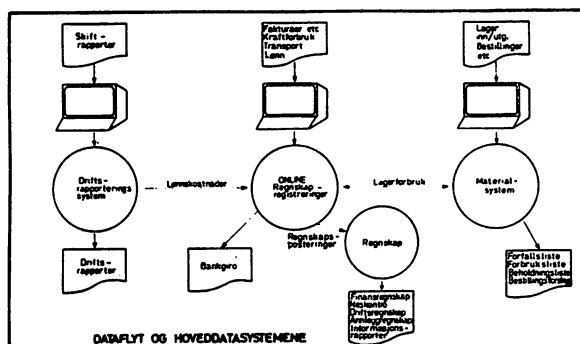


Fig. 2. Datablitt og hoveddatasystemene.

The companies data flow

Within a year a process control machine (Siemens R 30) will be installed to improve production quantum- and quality control in the pelletizing plant.

We are also testing out a mineral evaluation system (Mineval), which is a program based on first generation graphical methods.

The total investment in EDP equipment so far amounts to approximately 5 mill. kroner. (1 mill. \$)

The hardware configuration consists of approx 30 terminals of all kinds - six of them at the companies headquarters in Oslo.

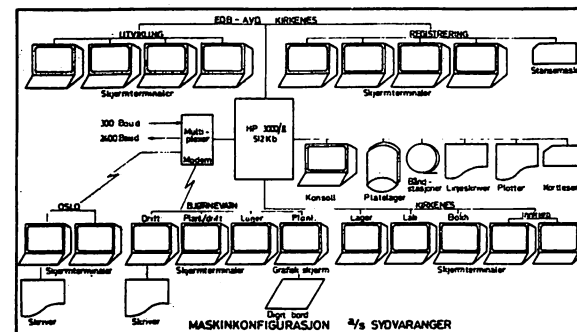


Fig. 1. A/S Sydvarangers maskinkonfigurasjon

The EDP-department today counts 11 persons, 7 of them are programmers.

Our main effort is to improve the information contained in the great variety of EDP-reports, and also get the computer and computer methods accepted everywhere in the company.

Considering the fact that EDP less than 20 years ago was a rarity, it is not surprising that use of the modern microprocessorpowered computers have created a lot of discussion, - both seen from the economic and social point of view.

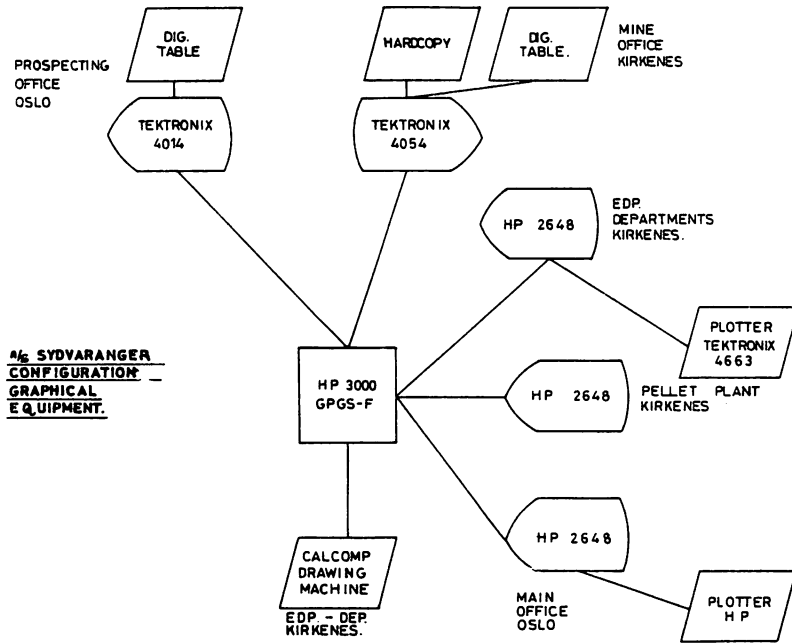
What is the outcome of the millions that have been invested in EDP-equipment the manager is asking. What will happen with my job is the workers question.

One of our projects to reassure both parts is to supplement the numerical reports with better understandable ones. Today this is possible not least due to computer graphics.

Drawing machines, plotters, graphical screens, digitizers etc. has moved the computer a big step towards the people.

1.3 Graphic Hardware

The hardware is a mixture of a number of units as shown below.



All units is online to the HP3000. Some of them are multiplexed on medium capacity telephon lines.

1.4 Graphic Software

Hardware is mentioned - computer journals are full of visualization units of all kinds. Less is spoken about software. How do you program a 3 dimensional rotation picture; how do you program a histogram? Do you hav to write a special program for every graphical output unit?

So many questions. And so many traps to be caught in.

One way to get rid of most of the difficulties is to choose a standardized graphical software package.

The GPGS-F is such a system. Standardized by NORSIGD, a norwegian organization formed by people interested in graphics.

GPGS-F is fortunately implemented on the HP3000 system. The system basis is a set of fundamental routines drawing lines, circles, character strings, numerics etc..

Besides the routines already mentioned the system contains a number of special drivers for the most popular and common graphical units.

The system is very well documented and easy to learn.

1.5 System philosophy

The graphical system is an online one. On the user's commands it are fetching data from the big databases containing up-to-date information about the stock, the production, the accounting etc.. The extracted data is then presented as diagrams..

The user's of the system are not computer specialists, they might not even be interested in computers. They are mine foremen, geologists, production planers, clerks, directors etc.. Therefore the user interface to the system is very important. The online dialog should be adapted to the user's language and way of thinking.

This is obtained by giving the user a series of commands to choose between.

A valid command leads to a conversation like the following

- . command name identifying the user's main topic of interest
- . additional information f.ex. machine number
- . period of interest
- . output media

Not valid input at all levels are followed by a very friendly-sorry - and a recommended continuation.

The language is plain and simple norwegian, not a word is reminding the user of the bits and bytes, the full duplex, the recursive procedures, the interlaced memory - and whatsoever.

As a first aid the HELP command gives an overall information about the system.

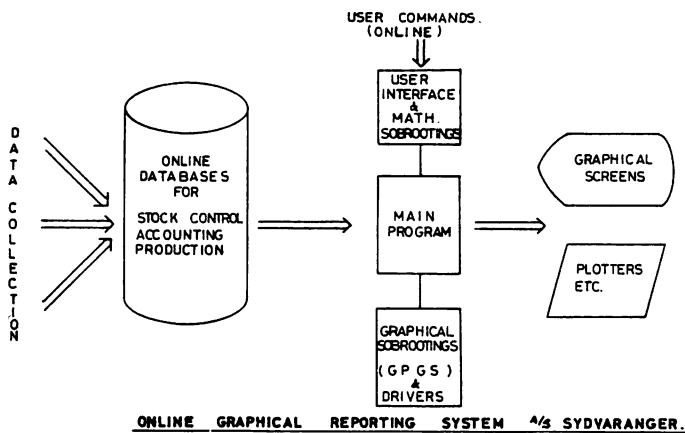
1.6 Data collection

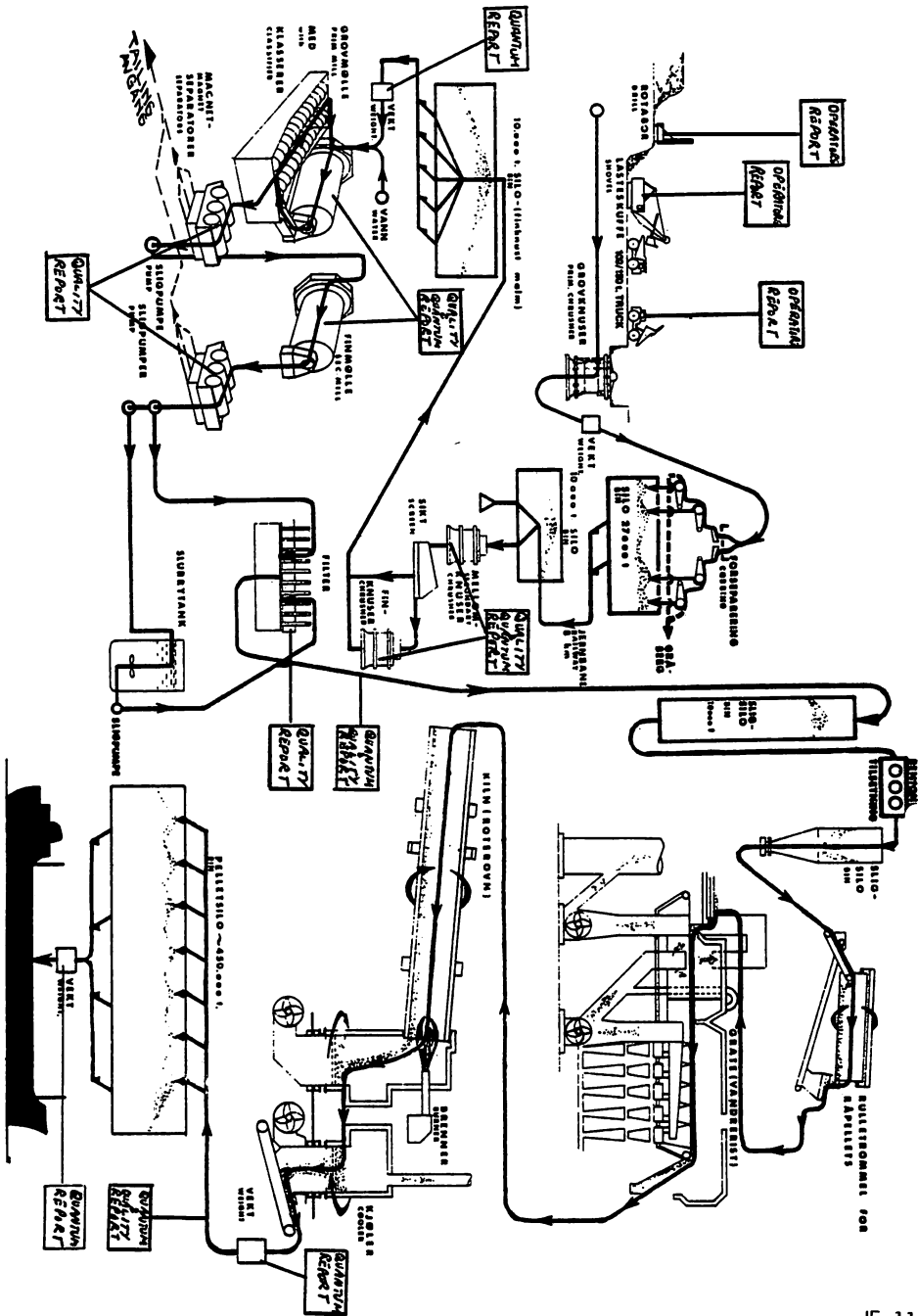
The systems practical value fully depends upon the quality of the information put into the databanks. It has therefore been necessary to establish a data reporting program.

A great number of reports is collected and registered into the computer.

This is also done online by means of masking technics.

On the next page the data collection system for the production data is shown.





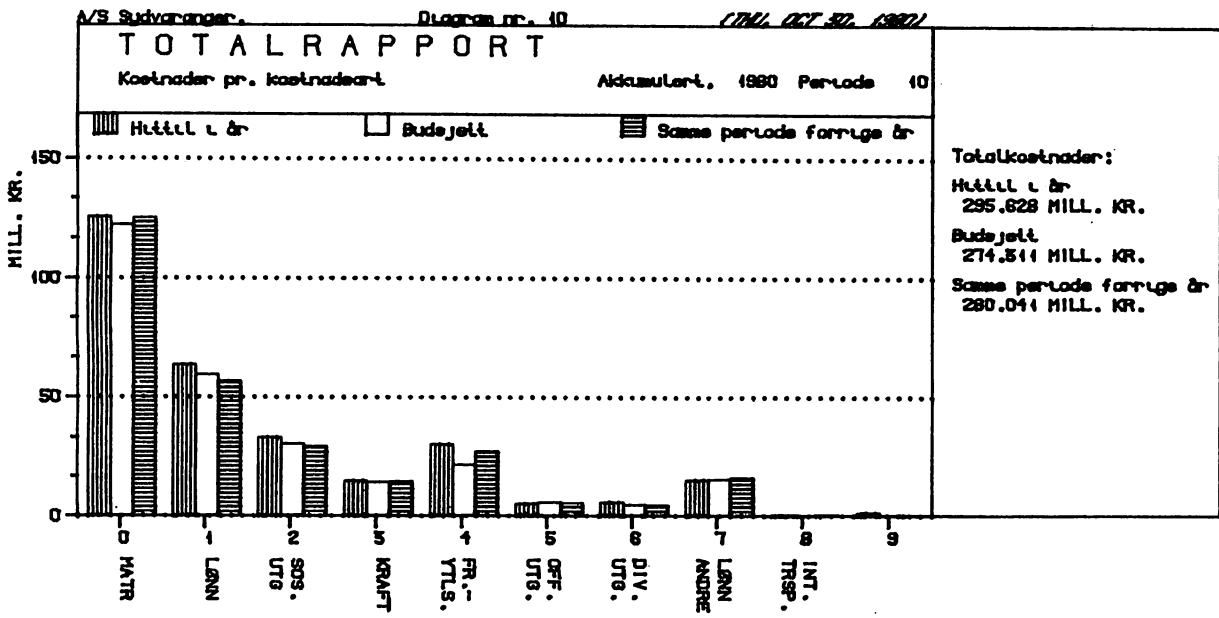
1.7 Examples

The example section is divided in three parts.

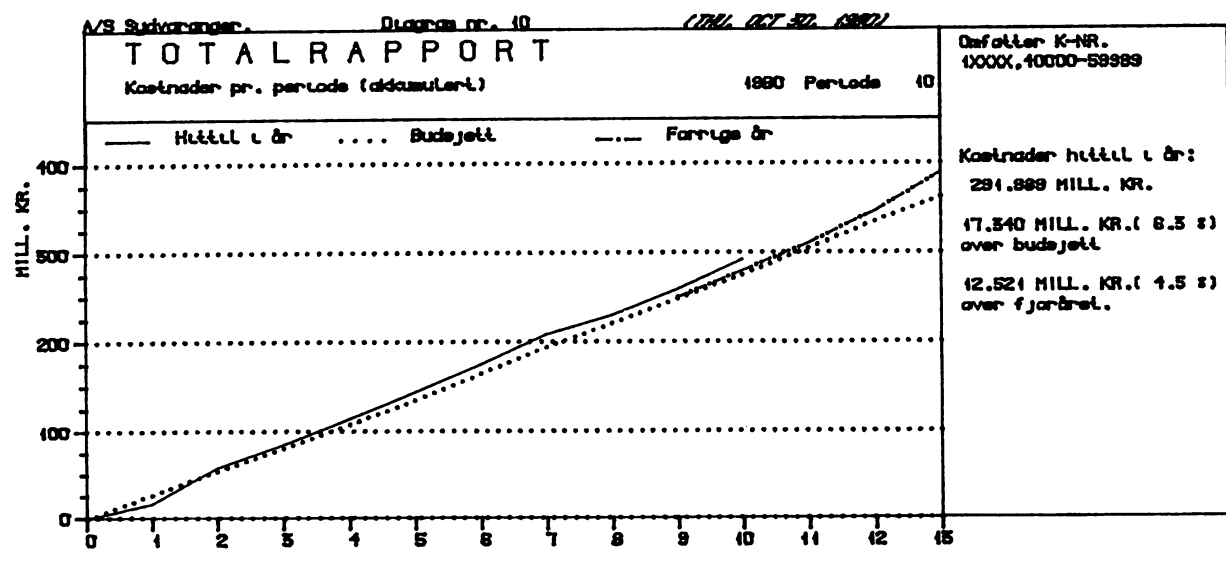
Part one contains examples from the economics reporting system (diagram 1 - 3).

Part two shows some mining reports (diagram 4 - 6).

Part three shows how the pelletizing plant has been run (diagram 7 - 8).




2
↓

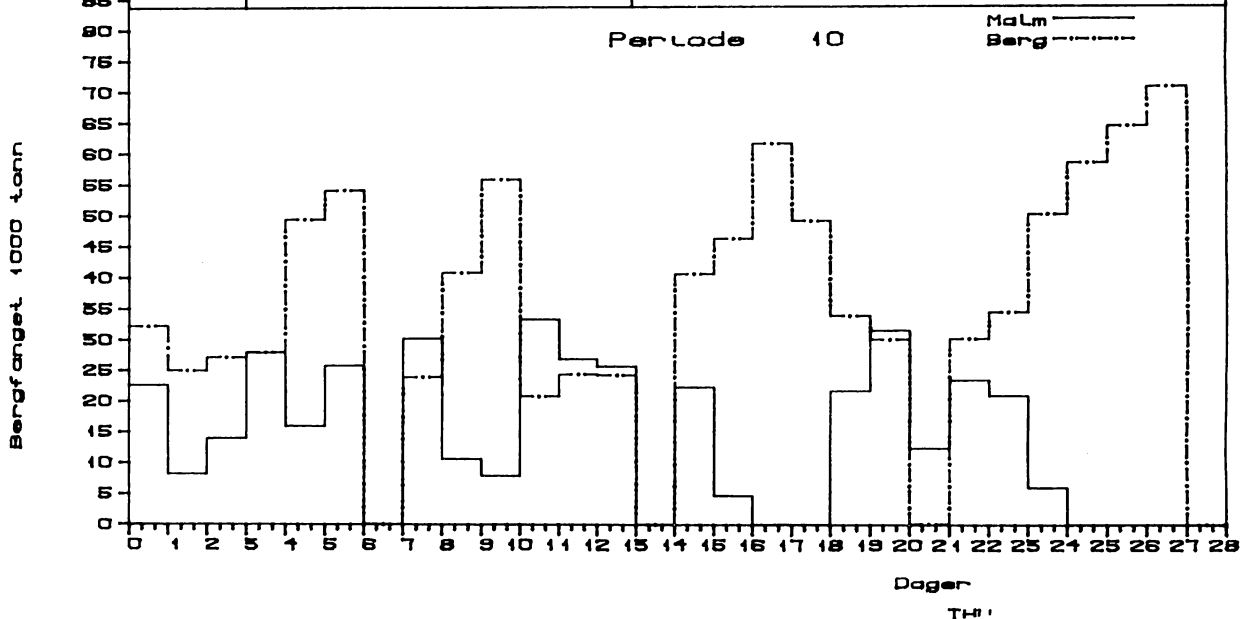


A. 3 Sydvaranger.

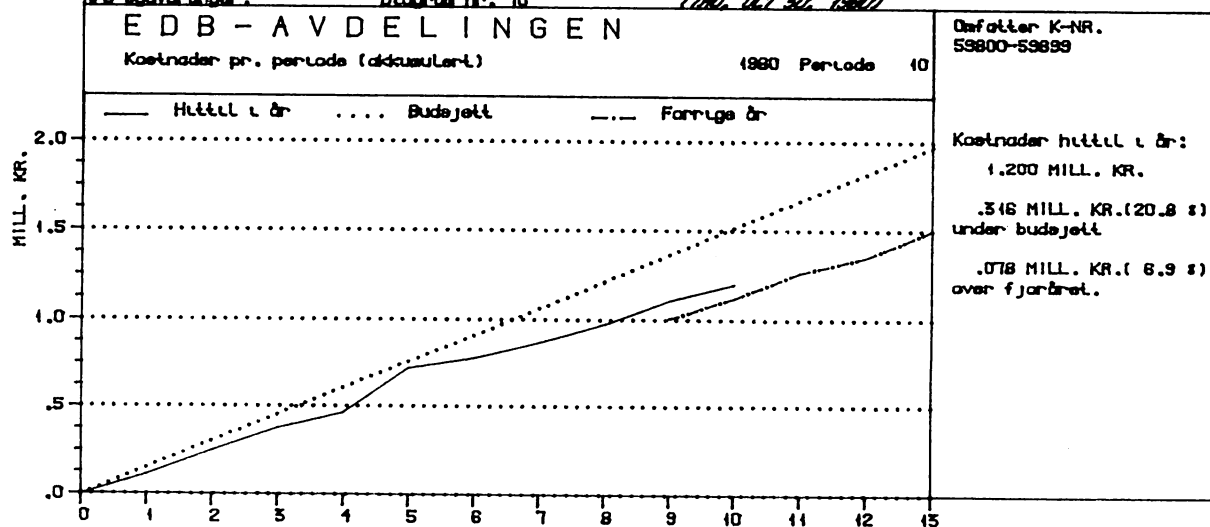
Driftsrapportering, Bjarnevann

Produksjonsdiagram

	Bergfanget (1000 tonn)		Fordeling i % :		Berg/malm :	
	Malm	: 590.965	Opplag	: 2.40		: .0
	Berg	: 976.270	Bjarnevann N	: 7.42		: 1.5
	Total	: 1566.255	Bjarnevann S	: 47.02		: 8.4
	Gjen/dagn	: 58.928	Søstervann	: .00		: .0
	Berg/malm	: 2.400	Tverrdalen	: .00		: .0
Stip. anseprod. : 20115.000		Fleketund	: 45.14		: 1.7	



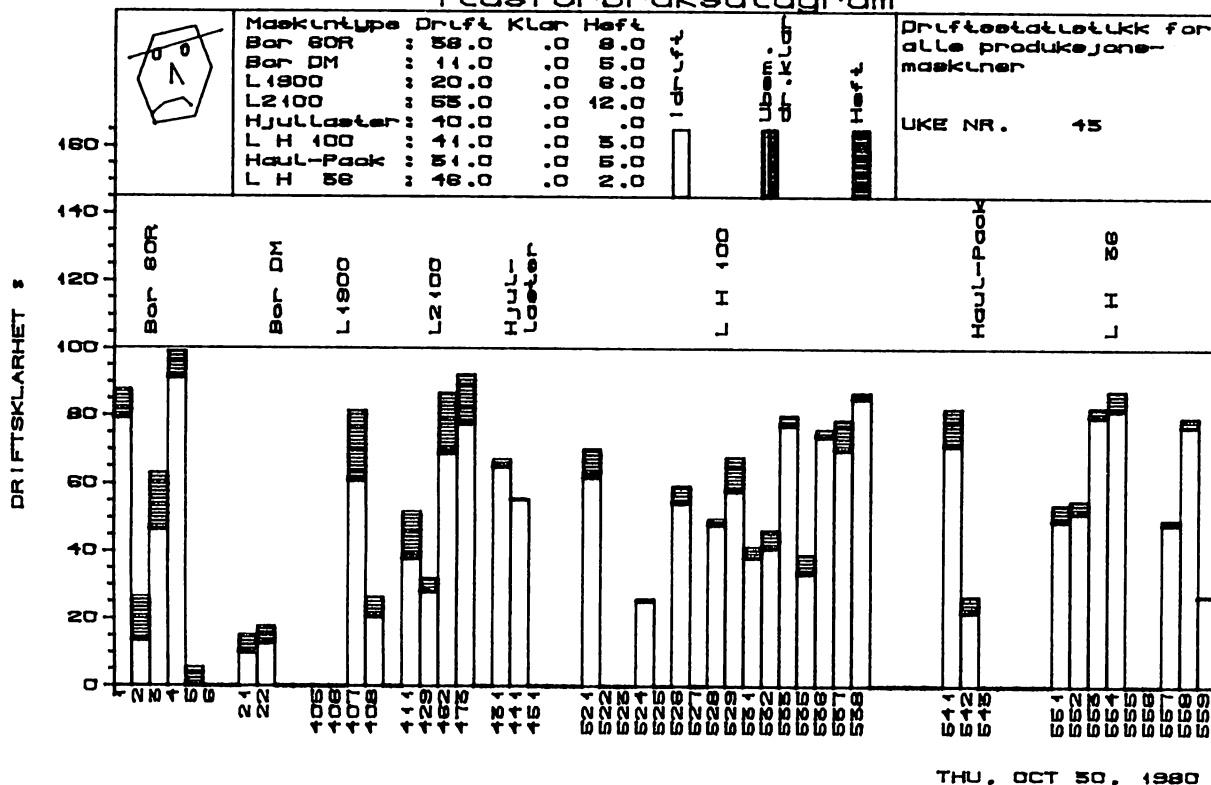
A/S Sydvaranger. Diagram nr. 70 (1980) (1980) (1980)



A. 3 Sydvaranger.

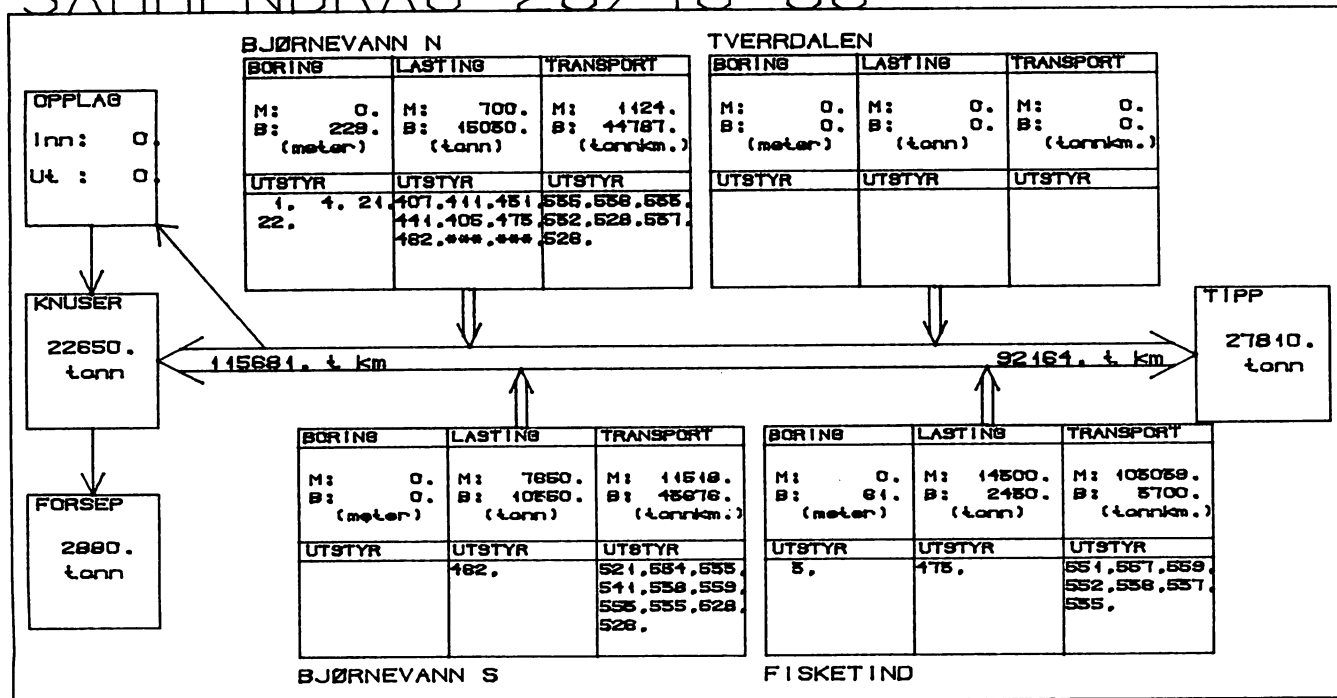
Driftsrapportering, Bjørnevann

Tidforbrukediagram



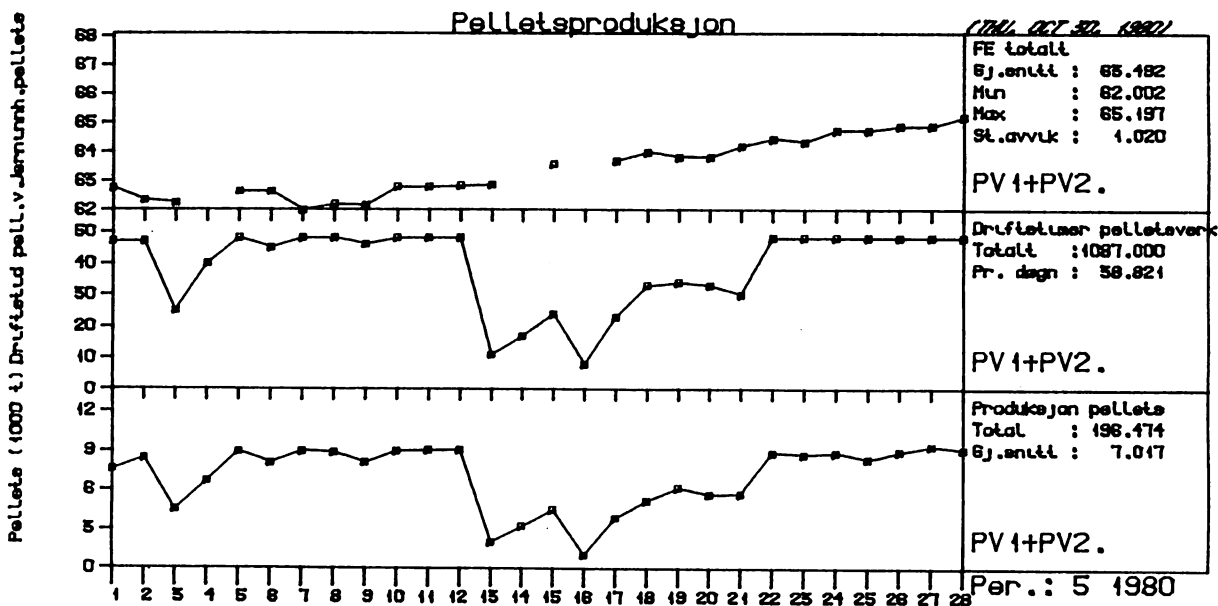
GRUBEN

SAMMENDRAG 28/10-80



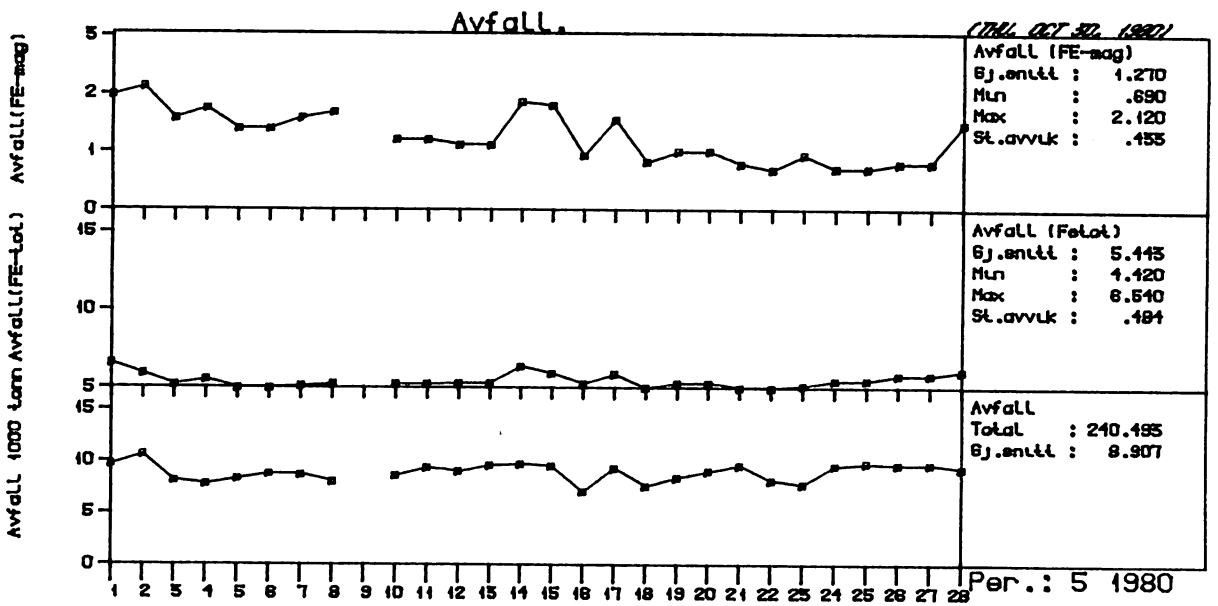
A/S Sydvaranger .

Driftsrapportering, Kirkenes

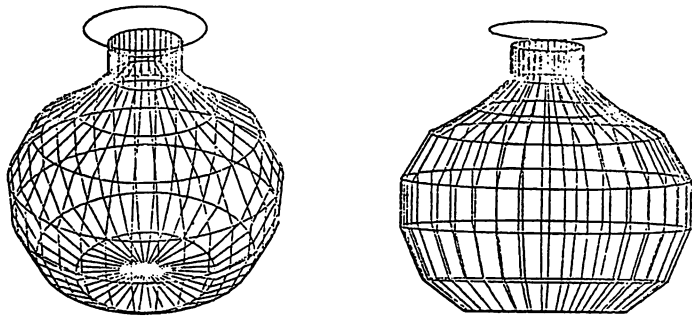


A/S Sydvaranger .

Driftsrapportering, Kirkenes



2 GPGS-F, A PORTABLE DEVICE INDEPENDENT GRAPHIC SYSTEM



2.1 Background

During the early seventies, computer graphics was introduced in the research and commission work of The Ship Research Institute of Norway (NSFI). In this process lack of suitable standardized basic software was very badly felt.

About the same time similar problems arose in other research institutions and in the industry.

A national special interest group in computer graphics grew up from the loose attempts of cooperation between the involved parts.

NORSIGD (The Norwegian Special Group in Computer Graphics) was founded in 1975 and has since then formed the basis of a very good cooperation between most Norwegian computer graphics users.

The main task for NORSIGD has been developing a standard graphics software system for Norway.

2.2 History

The GPGS system was originally designed by Rekencentrum, Delft University of Technology, The Netherlands and Science Faculty, Catholic University Nijmegen, The Netherlands in 1972. A version of the system written in standard Fortran has been developed by NORSIGD. The first Fortran based version was released in 1975 and named GPGS-F. GPGS-F has been under continuous development since the first version was released. This work has been guided by annual user meetings. The result of these meetings has been new features and minor changes to the system.

2.3 Device control

GPGS-F provides device independent programming with choice of graphic device at run-time.

Figure 1 shows how the device independency is obtained. The device independent part produces the same picture code for all devices, and the device driver(s) translates this code to the bit pattern required for the actual device. The device independent code is put on such a level that advanced graphic devices may be used in an efficient way. Examples of such functions are character, circle and marker generation. GPGS-F will also be able to take advantage for the functions of more advanced refresh display such as hardware scaling, rotation and depth quing.

Table 1 shows the current implementation status for GPGS-f.

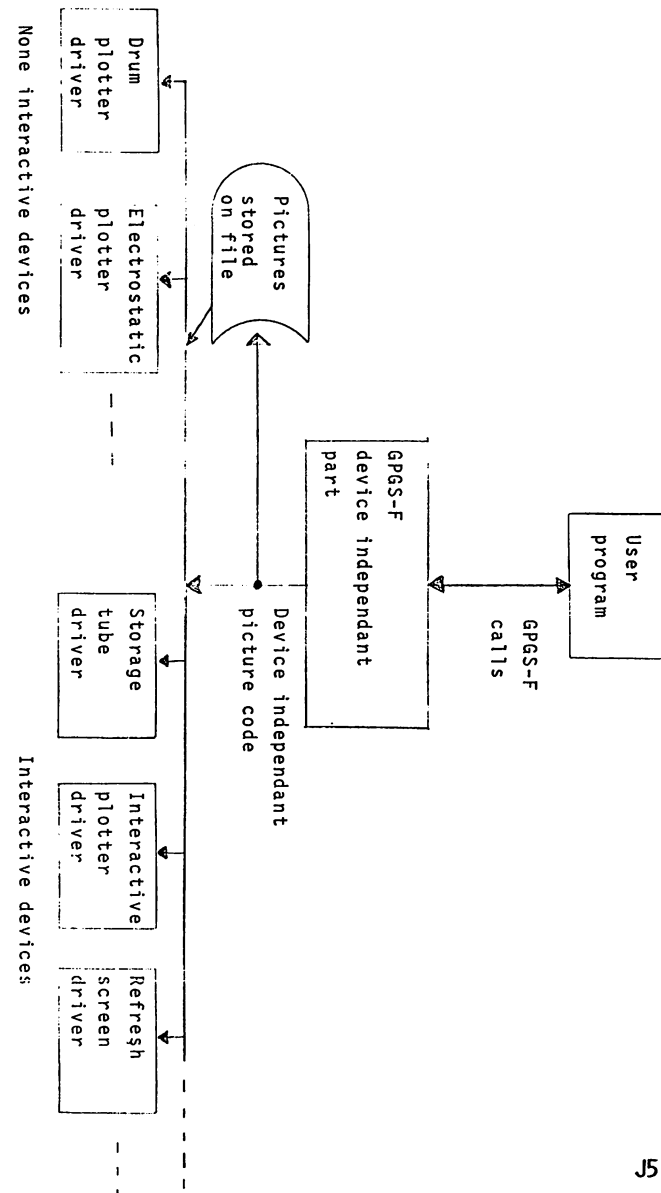
The status for GPGS-F implementations as of August 1, 1981 is shown below:

(X=implemented, O=under implementation)

	UNI- VAC 1100	HP- 3000	VAX 11	CDC CYBER	DEC 10	IBM 370	NORD 10	PDP 11	PRIME	LSI 11
GPGS-F	X	X	X	X	X	X	X	X	X	X
FILSYS	X	X		X		X	X	X		
GRAPHISTO	X	X	X	X	X	X	X	X		
SURRENDER	X	X	X	X	X	X	X			
CALCOMP	X	X	X	X	X	X	X	X		
KINGMATIC	X			X			X	X		
PRINTER	X	X	X	X	X	X	X			
HP 7221	X	X	X			X	X			X
TEK 4662	X	X			X	X	X			X
TEK 4663	X	X					X			
VERSATEC	X						X			
TEK 4010	X	X	X	X	X	X	X	X	X	
TEK 4014	X	X	X		X	X	X	X	X	
HP 2648	X	X	X				X			
TEK 4025	X						X			
TEK 4027	X						X			
RAM 6002	X									
ALPHADUS	X			X	X					
PSEUDO	X	X		X	X	X	X			
FILE	X	X		X		X	X			
ICAN										X

Unapproved (preliminary) implementations: Honeywell Bull
 MYCRON } micros
 CROMEMCO }

Fig. 1. MAIN STRUCTURE OF GPGS-F.



2.4 Graphic elements

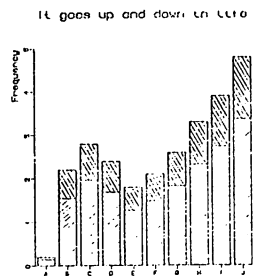
By single calls to the basic routines of GPGS-F, the following graphic elements may be generated:

- straight lines
- set of straight lines
- circles/circle areas
- text
- markers
- functions

All graphic elements may be hardware generated by the graphic device.

The elements are defined in a user defined coordinate system (2-or 3-D Kartesian) and are fully transformable.

2.5 GRAPHISTO - Graph and Histogram plotting



GRAPHISTO (b) is a subroutine package which using the basic routines of GPGS-F can produce curve-, bar- and piecharts. The package was designed to remove programming effort from the tasks of producing standard plots. GRAPHISTO is aimed at easy presentation of one variable data. An advantage when using this package is that it can be entered at different levels. When the user wants to produce one of the standard charts of the system this can be done by one simple call. If this standard chart does not satisfy the requirements for the plot, user can enter the system on a lower level, composing the plot he wants. The user can even go down to the basic routines of GPGS-F and mix these with the GRAPHISTO calls.

GRAPHISTO provides 4 so-called 'chart' routines that will in answer to a single call draw a complete diagram with annotated axis, texts on axis and datapoints.

The types of plots provided through these 4 routines are:

- Histogram with labels under each bar and linear or logarithmic axis in x and y.
- Table of lines with straight lines between plots.
- Smooth curve through specified points.
- Pie chart.

These 4 routines use the basic GRAPHISTO routines as axis drawing, range computation, 'nice' value computation and curve plotting. The lower level routines are also available to the user and offers possibilities for sophisticated non-standard plots like multiple axis, marking special data points etc. Appended to this chapter are some plots to demonstrate the use and possibilities of GRAPHISTO.

Figure 7 Shows some plots produced with GRAPHISTO.

Available facilities are:

Chart plotting:

- Simple and smooth (cubic spline) curves in different linestyles.
- Histograms. They may be plotted with or without hatching of bars in any angle.
- Pie charts with texts and percentage of total pie.

Axis drawing

- Near to plot on either side of plot.
- Through any data or page value.
- Several parallel with different units.

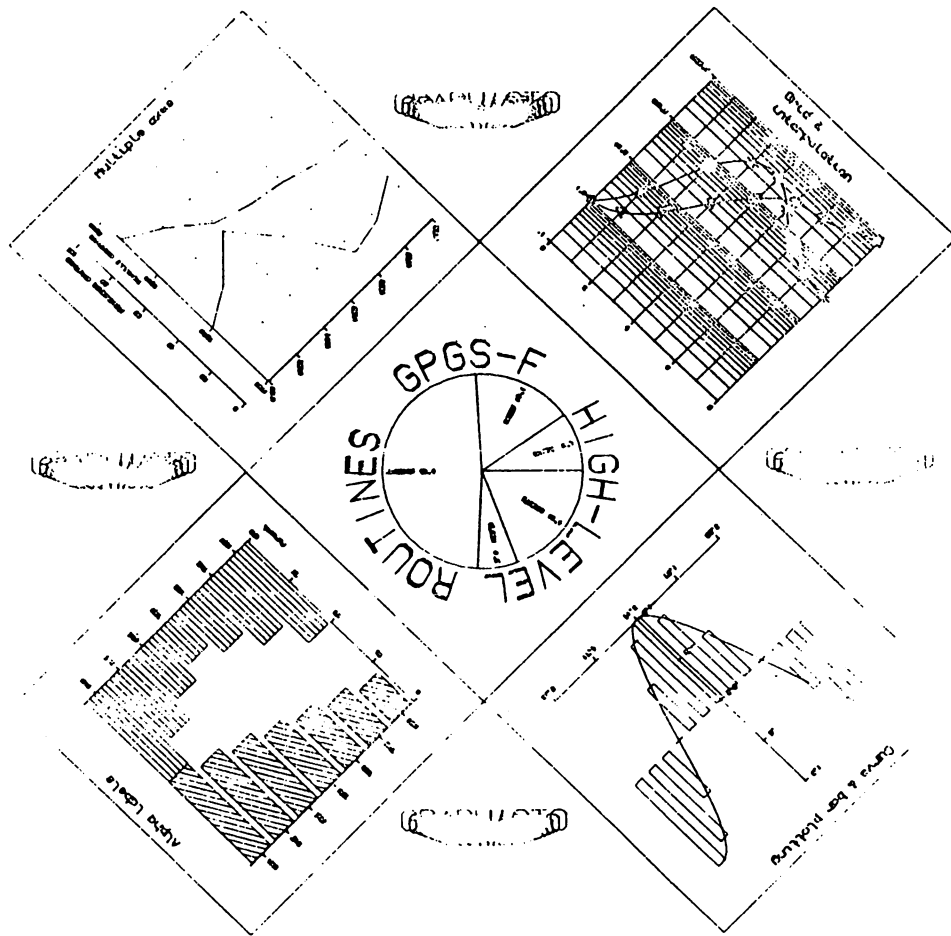


Figure 7. GRAPHISTO example plots.

Axis annotation:

- On either side of axis
- Numeric or text labels
- Any character size and angle
- Upper and lower case
- Extra tick marks
- Title

Grid:

- Along x- or y-axis
- Linear or logarithmic
- Any line type or '+' at grid crossings

Dataplotting:

- Table of values in x- and y or functions
- Simple connected points
- Interpolated curve through points
- Markers at points
- 'Undefined' points
- Automatic data indexing
- Automatic data incrementing

Page layout:

- Centered heading
- Positioning of dataplotting area in users window
- Bar and curve legends
- Frame

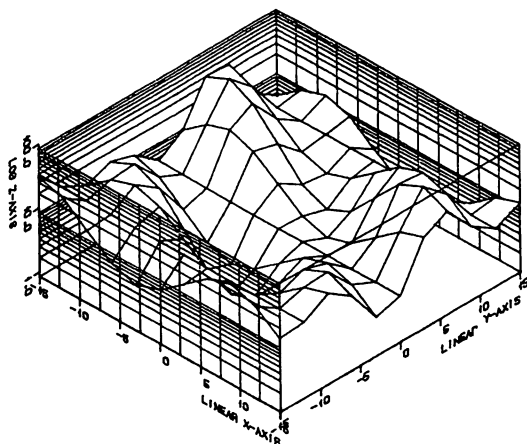
Miscellaneous:

- 'Best-fit' range computation of data contained in array or function.
- 'Best-fit' label format computation.
- Conversation between coordinates in users window and in plotting coordinates.

Error Reporting:

The routines in the GRAPHISTO system uses the error system of GPGS-F. This includes parameter checking and print of routine number, where error occurred. Also the number of calls made to GRAPHISTO routines and wrong parameters are printed. This makes it easier for the user to find place and reason of error.

10. SURRENDER, 3-D SURFACE PLOTTING



SURRENDER (S) is a subroutine package for drawing bivariate surfaces in 3 dimensions. GPGS-F basic routines are used for line drawing and GRAPHISTO routines for axis drawing and curve smoothing.

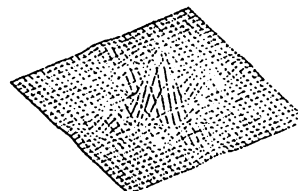
Base for all SURRENDER plotting is a rectangular x-y grid (matrix) with z-values in each node. A surface with M grid points in x-direction and N grid in y-direction will be stored in a Fortran DIMENSION ARRAY (N,M).

This grid may be rendered as a 3-D perspective plot of the grid (Isolines any combinations of x,y and z) with hidden lines removed or as a 2-D contour map (Isolines for any of x,y or z). Also other usefull facilities like drawing axes, marking points etc. are available and will be further explained later.

The package is built up much the same way as GRAPHISTO with some routines that makes a complete plot in one call, and others to add features for a more sophisticated plot.

Example of minimum effort plot:

```
DIMENSION IWORK(200),VP(4),ZMAT(31,31),IGPT(3)
DATA IDEV/8/,VP/C.0,C.0.3,C.0.0.3/
DATA IBLUE/20/,IRED/60/
C
C COMPUTE THE FUNCTION 'SIN(X)*SIN(Y)/(X*Y)'
C
DO 4000 IY=-15,15
  Y=FLOAT(IY)
  SINYY=1.0
  IF (IY.NE.0) SINYY=SIN(Y)/Y
DO 1000 IX=-15,15
  X=FLOAT(IX)
  SINXX=1.0
  IF (IX.NE.0) SINXX=SIN(X)/X
  ZMAT(IX+16,IY+16)=10.0*SINXX*SINYY
1000 CONTINUE
C
C MAKE MINIMUM EFFORT PLOT
C
CALL NTIDEV(IDEV)
CALL BGNPIC(1)
CALL PLOWA3(ZMAT,31,31,-15.,15.,-15.,15.,IWORK,200)
CALL ENDPIC
```



Hidden line removal:

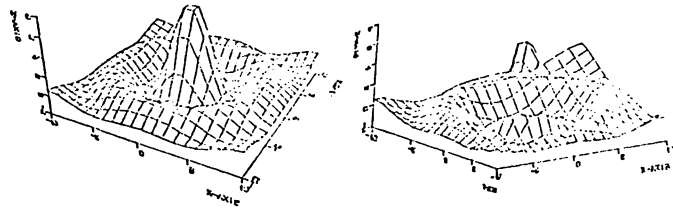
By default the hidden lines will be removed. To do this the system uses a working array to be supplied by user. As the needed size of this array is dependant upon the number of points to plot, this method gives no restrictions about number of points.

Setting focal point and eye position:

The surface may be seen from any point in space and some routines are used to set this point either using cartesian or spherical coordinate system. The viewing may be either axonometric or perspective.

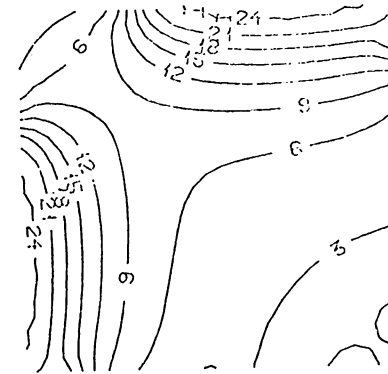
Adding axes to the plot.

Axes may be added by a call to a single routine giving standard axis annotation or by several calls to GRAPHISTO Axes routines for special labels and format.



Contour plots.

Contour plots consists of isolines in the z-direction. The range and number of contour lines may be given, and they may be added to the perspective plot or plotted as a separate 2-dimensional plot.



Default contour plot.

References

1. Major, F. ,
Martens, D. GPGS-F, a portable device independent
graphic system.
Presented at the HPGSUG meeting, Lyon 1979.

2. NORSIGD: GPGS-F User's Guide,
4.edition, TAPIR forlag 1980

3. NORSIGD: GPGS-F Course Notes,
RUNIT 1977

4. Zachrisen, M, GRAPHISTO, User's Manual
Torgersen, J.E.: RUNIT report STF14 A79023

5. Zachrisen, M: SURRENDER - a subroutine package
for rendering bivariate surfaces.
RUNIT report STF14 A79020

M.P.E.-IV

M. S. PAIVINEN

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

M.S. PAIVINEN
HEWLETT PACKARD

© DATABASE CONSULTANTS EUROPE B.V., 1981
Keizersgracht 557
1017 DR AMSTERDAM
Netherlands
(020) 224243

August 1981

DATA ANALYSIS - The answer to successful implementation of IMAGE

by RICHARD IRWIN

Presentation to the HP3000 International
Users Group

1981 Berlin International
Meeting Germany

ABSTRACT

IMAGE is now considered to be technically one of the most successful DBMS's. However, as with other DBMS's it suffers badly when poor design tools and methods are used. Data Analysis is an essential part of a concept that is growing rapidly in Europe and America.

This paper describes the Data Analysis process; how to begin. Defining data areas and data resources. The concept of entities attributes and relationships. The degree of relationships, one-to-one, one-to-many, many-to-many. Types of relationship, e.g. optional, involuted, multiple. The use of the data model as a learning aid to the analyst and to communicate with non-dp users. Life cycle, sub-type and time dependent entity roles. The interactive part of data modelling. Mapping to the logical database. Distinction between conceptual, logical and physical stages of development. Overcoming general structural limitations of IMAGE.

This paper is written and presented by Richard Irwin, a Senior Consultant with Database Consultants Europe BV who has spent the past five years analysing, designing and implementing IMAGE/3000 systems.

1. INTRODUCTION

This paper is primarily concerned with the system development cycle in a business environment. This does not mean however that data analysis should not be performed in other environments, e.g. - scientific - but in order to demonstrate its usefulness, a specific area has been chosen. The paper has also concentrated on the logical construction of IMAGE and using an HP3000, but many aspects can be seen to apply to other file systems and ranges of hardware.

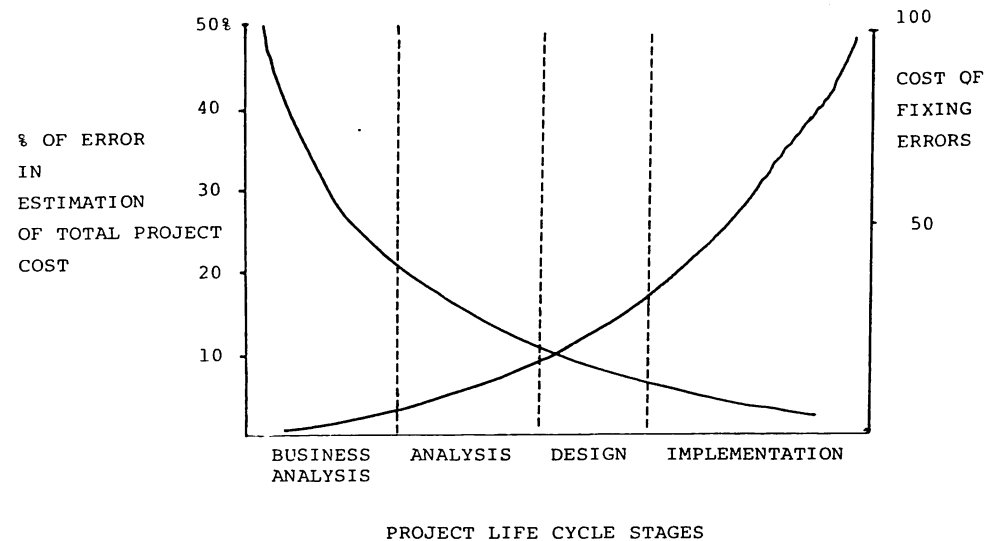
2. WHY USE A METHODOLOGY?

A 'method' is a procedure for carrying out a certain task. A 'methodology' is an integrated set of procedures, founded on consistent basic principles, which provide a complete framework within which a given task can be performed. A methodology is used to perform these basic functions.

2. 1. Highlighting of problems at an early stage.
The development process is structured to allow critical management, user and technical decisions to be taken at the right time, i.e., as early as possible. (see Figure 1 over)

Figure 1

(C.G. Davis "Requirements Problems in large real-time systems development")



2. 2. Providing a means of communication.
Check point facilities provide a means of communication between all levels of personnel concerned with the project.
2. 3. Proof of progress.
DP management is under constant pressure to show results. Without a methodology all we do is push for early system completion, thus instead of the project being time-shared as in Figure 2 (over),

3.

we save time in the analysis, resulting in Figure 3.

Figure 2 IDEAL

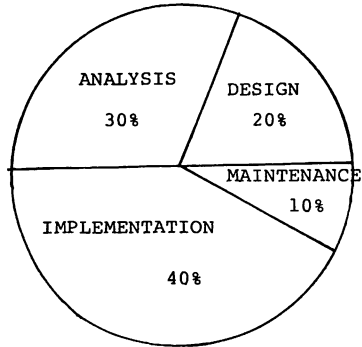
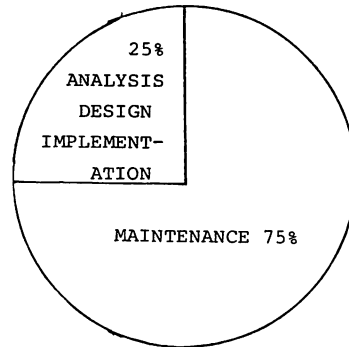


Figure 3 REAL



With a methodology however, we can prove our progress at each step by producing checkpoint documents. A methodology must therefore provide:

- guidelines which ensure that we don't overlook things (not rules as they are too inflexible)
- an approach which is top-down or outside-in and modular
- easily understood diagrams for communication
- standards for use and documentation

3. DATA - A VALUABLE RESOURCE

For years the value of data was grossly under-estimated. This meant that the emphasis was on the application approach, where, for each application, the data would be defined again resulting in the following difficulties:

3. 1. Duplication of data

- inconsistencies of value, timeliness and meaning
- cost of storage

4.

3. 2. Consolidation across applications

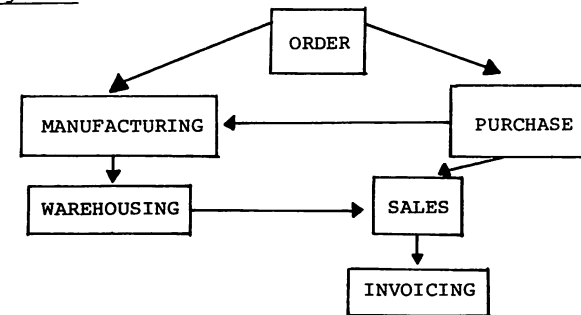
- file collation
- proliferation of work files
- integration of processing

3. 3. Lack of control

- satisfying new application requirements
- availability and use of data

When the introduction of the database philosophy came, it was not necessarily a philosophy centred around Database Management Systems but more the acceptance of the need to share data. Data is the basis of information flow across the functional boundaries within an enterprise as represented in Figure 4.

Figure 4



The definition of a database should be:

an organised, integrated collection of data which

- is structured to reflect the real world of the enterprise

- is stored independently of programs which use it
- satisfies the requirements of multiple user application

or all of the above may be summarized quite simply by defining a database as "a common pool of shared data".

However, new problems soon became apparent to the designs of early database systems:

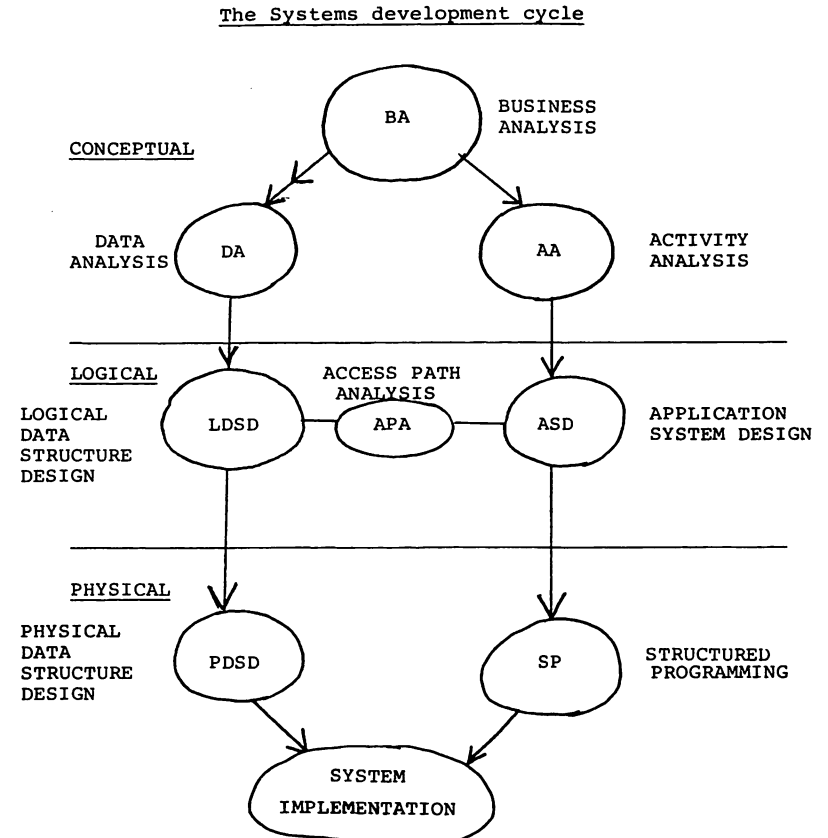
- lack of procedures to make and document critical design decisions
- development of single-application oriented databases
- adoption of a bottom-up approach to data analysis meant that the designs became inflexible
- failure to fully exploit the rôle of data dictionary/directories
- no allowance (or design) for database recovery or re-organisation
- the database project was usually seen as a file conversion exercise

To summarise - the database philosophy evolved from a need to share data, but whilst there are many benefits for the use of a database there are also pitfalls in the development stages.

4. WHAT IS DATA ANALYSIS?

Data analysis is part of the systems development cycle as shown in Figure 5.

Figure 5



The objective behind each part is as follows:

<u>METHOD</u>	<u>OBJECTIVE</u>
Business Analysis (BA)	To define the business area boundaries for analysis needs, at a high level
Data Analysis (DA)	To analyse the data resources
Activity Analysis (AA)	To define the users' information handling processes
Logical data structure design (LDS)	To map the data model to the logical data structure
Application system design (ASD)	To translate the user information handling processes into a technical application system design

The most important factor to note is the early split between data and functions. Most of the emphasis in other system development methodologies has been on the functional side, typically on the programming effort. Although programming errors are one direct cause of costly and inflexible systems many of the errors can be traced back to errors in the analysis and design stage.

What is fundamentally wrong with many approaches is that no method exists for analysing and describing in a concise, user-oriented way, the business data and how it operates, divorced from any considerations of how the system will eventually be designed.

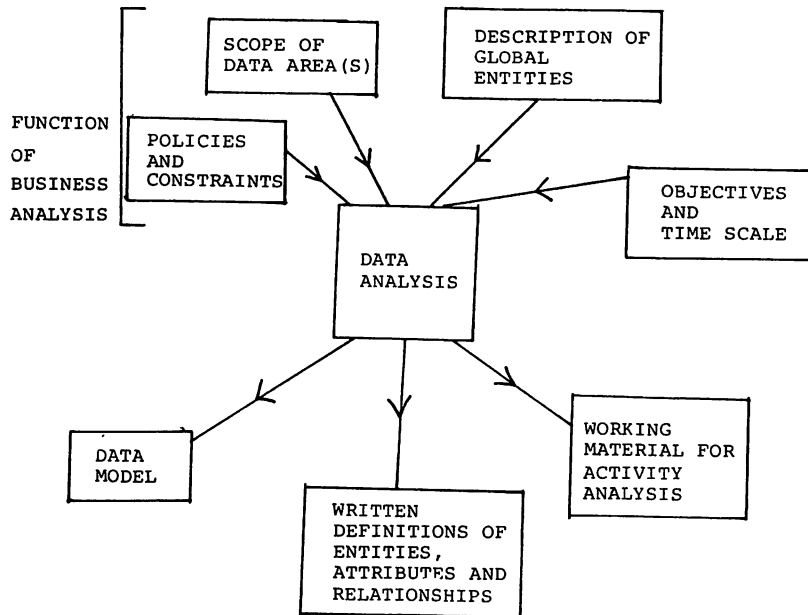
It is often desirable, and should be possible to analyse a business without any prior constraints on how parts of the business are to be computerised and which business functions will form the basis of computer systems.

In many approaches other than Data Analysis, the emphasis is placed on determining and analysing the "output required", (i.e. listings, reports, computer files, etc. - a dangerous practice in itself, as information requirements are never static) and then expressing the results in terms of computer files, English narrative descriptions (often long and complex) of the "processes required", and technical flowcharts of the flow of data through the system. The results of this approach are apparent - inflexible systems which are not resilient to change and whose development is often unco-ordinated and fraught with problems. The underlying cause is that the business was never fully understood before the design stage.

One approach which seeks to remedy this lack of an analysis methodology is known as data analysis. Database Consultants Europe BV (DCE) has successfully used this technique for a number of years during which time the initial concept has been developed into a complete analysis and design methodology.

Data analysis is a method used to understand and document a complex environment in terms of its data resources. The results of data analysis are summarised in a diagram known as a data model. Detailed results are documented on specially designed forms. The input and output of Data Analysis can be summarised as in Figure 6.

Figure 6



5. WHERE TO BEGIN?

To initiate the task of Data Analysis the following tasks should be performed:

5. 1. Gaining support
 - from management
 - data processing staff
 - user departments involved
5. 2. Definitions
 - of the objectives
 - of the timescale
 - terms of reference
5. 3. Design and acceptance
 - Data Analysis standards as compared with existing standards
 - Documentation
5. 4. Anticipation of possible unfortunate discoveries
 - irreconcilable coding systems
 - inconsistent existing data
 - incorrectly interpreted reports
5. 5. Education and training
 - theory and methodology
 - detailed procedures.

6. DATA AREAS AND RESOURCES

When choosing the data area for analysis, it should be small enough to be manageable and not too complex that an overview cannot be attained. There should be clear cut definable boundaries with a minimum of interaction with other areas. It should be independent of, or well defined in terms of, specific applications. There should also be a business requirement for applications to be implemented in that area, or for improvements to be made to existing application. The data resources can be categorised in the following way:

- Personal knowledge and ideas
- Clerical records
- Manually produced reports
- Correspondence
- Computer files
- Other computer readable data
- Computer produced reports

7. DEFINITION OF ENTITIES, ATTRIBUTES AND RELATIONSHIPS

Within data analysis, there are three major components:

7. 1. Entities

An entity is something of fundamental importance to a company. It is thus something about which

data will probably be kept in an information handling system, e.g. objects, people, places or abstractions such as events.

7. 2. Attributes

An attribute is a basic unit of information which describes an entity. Within the company environment, an attribute cannot usefully be sub-divided into other units of information.

An entity must have attributes if it is of interest to the company, e.g. the entity "insurance policy" could have the attributes policy number, date policy started, person's name, person's date of birth.

7. 3. Relationships

A relationship is an association between entities, e.g. the entity 'order' is related to the entity 'order line', the entity 'car' is related to the entity 'part'.

There are no theoretical rules which can be applied to decide when an object is worth being an entity or attribute until the company environment is known to the analyst. If the object is not of fundamental importance to the enterprise then it is not worth keeping information about it. For example, can a building be an entity? In the case of a company which simply exists in one building the answer could be 'no'. However, in the case of a construction company or an electrical installation company the answer would most definitely be 'yes'.

A similar example for attributes is a 'person's weight'. In the case of a vacuum cleaner sales company the answer would be that a 'person's weight' would not be an attribute but for a hospital it would.

8. ANALYSIS OF ENTITIES

In order to recognise entities it is important to ask what data or objects are within the chosen area(s). A first pass of definitions of entities and preferably their distinguishing or key attributes is made. The key point to remember is that the focus is on entities not processes. However it is important to ask what events take place (e.g. job offers) in order to identify entities which are abstractions. It is advisable to check all input and output reports for other possible report entities.

9. ANALYSIS OF ATTRIBUTES

Once the major entities have been identified, determination of relevant attributes is performed by examining:

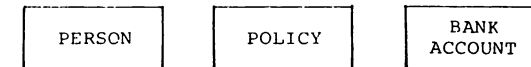
- Manual files
- Documents
- Decision criteria (to identify implicit selection or distinguishing attributes)
- Computer files

For new entities it will be necessary to ask when data is needed to be kept about that entity.

10. PICTORIAL REPRESENTATION

10. 1. Entities

Entities are represented by a rectangular box with the name of the entity written inside the box.



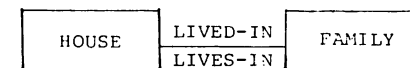
In the early global data analysis phase, only the entity name is written inside the rectangle. However, when doing the detailed data analysis it is sometimes convenient to include also the names of the identifying attributes.

10. 2. Relationships

Relationships are shown by drawing a line between the entities, also showing the degree of the relationship. An abbreviated name of the relationship can be written alongside the line. (NB. entity names and relationship names are read in a clockwise direction).

10. 2. 1. One to One (1:1)

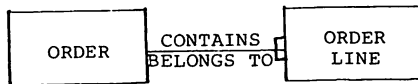
The One to One degree of relationship is represented by



i.e. One house is lived in by one family and one family lives in one house.

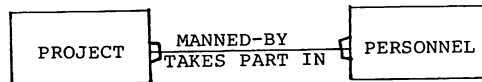
10. 2. 2. One to Many (1:n)

One to many degree of relationship is represented by



i.e. One order contains many order lines and one order line belongs to one order.

10. 2. 3. Many to Many (n:n)

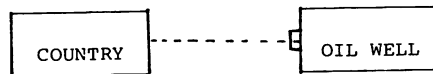


One project is manned by many personnel and one person can take part in many projects.

11. FORMS OF RELATIONSHIP

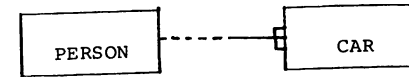
Relationships can take many forms

11. 1. Optional Relationships



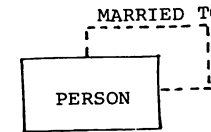
i.e. A country may or may not contain oil wells.

11. 2. Partially optional



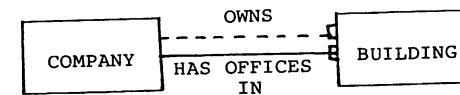
i.e. A person may own none, one or many cars but a car must be owned by a person.

11. 3. Involved relationships



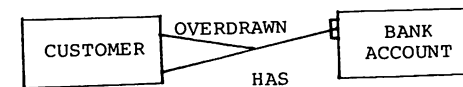
An involved relationship is a relationship between occurrences of the same entity type, e.g. a person may be married to another person.

11. 4. Multiple relationships



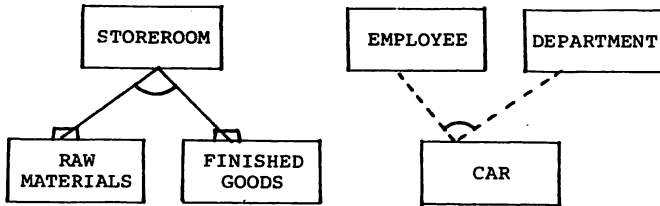
i.e. A company may own many buildings and must have offices in many buildings.

11. 5. Inclusive relationships



An entity can participate in the one relationship - "overdrawn" only if it also participates in the other relationship - "has".

11. 6. Exclusive relationships



An entity occurrence may participate in any one, but not more than one, of a number of alternative or exclusive relationships, e.g. a car must be owned by either an employee or a department, not both.

12. THE DATA MODEL AS A COMMUNICATION TOOL

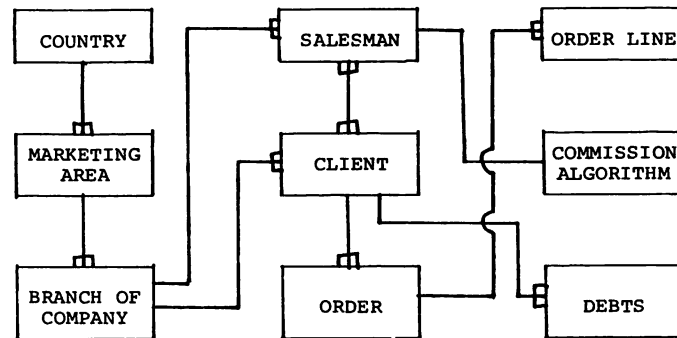
It is imperative for any methodology to be able to use the analysis results for communication purposes. Since communication must take place between the user of the information and the technician who is performing the analysis, it is also necessary to keep the pictorial representations as simple as possible to avoid misunderstandings and to enhance co-operation. No user has the time or inclination to sit down and discuss 50 or 60 pages of documentation. It should also be remembered, therefore, that only those parts of the picture which pertain to the user's direct area of interest should be brought to him for discussion.

At all times the picture should represent the real world of data and its relationships.

A situation which must be avoided is the tendency to think in computer terms instead of user terms. It is also important to keep the representation in the simplest possible form.

Figure 7

Example data model - first pass



It should be noted that a hierarchical structure has been deliberately avoided so that relationships can be thought about more easily. Obviously models become messy and have to be redrawn when too many lines of relationships are involved.

13. PRACTICAL PROBLEMS

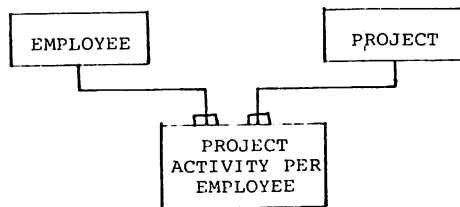
13. 1. Many to many relationships

When a many to many relationship occurs it usually means that another entity can be identified between the two entities.

Example

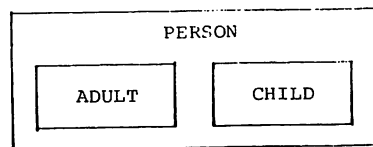


i.e. An employee works on many different projects and projects have many different employees. Because it is useful to have the attribute 'length of time of employee on project' we are obliged to create the following situation:



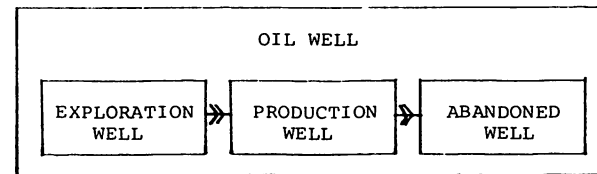
13. 2. Entity rôles

Entities which are similar but which have slightly different attributes depending on the value(s) of certain classifying attribute(s), are probably entity 'roles'. An entity rôle is a sub division of an entity type which is difficult to separate from the entity type with which it is associated. E.g. A 'person' entity may be subdivided into 'ADULT' or 'CHILD'. This would be represented in the following way:



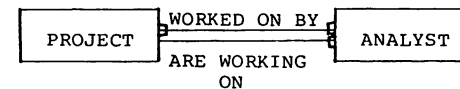
13. 3. Life-cycle rôles

Life-cycle rôles are special cases of sub-type rôles in which a sequence exists between the sub-types. This sequence is shown by a double arrow on the connecting line.



13. 4. The Problem of Time

Most time problems can be represented simply by showing the multiple relationships of present and past, e.g.



14. ACCESS PATH ANALYSIS

Access path analysis could easily be a large enough topic for discussion within a separate paper. In this paper the subject is summarised to demonstrate its relevance. To perform access path analysis the following tasks are performed.

14. 1. For each access path, the entity types are listed in the order they are needed for a particular function.

14. 2. The selection criteria are recorded in terms of relationships and attributes.

14. 3. It has to be recorded whether each entity attribute type is retrieved, modified, created or deleted.

14. 4. It is also necessary to record any relationships created, modified or deleted.

Example of Access Path Analysis.

ACTIVITY ORDER ENTRY

ACTIVITY DESCRIPTION

Function 1 An order is received by telephone. The depot that will make the delivery is selected depending on whether the goods are bulk or packaged. The order is recorded, and related to the delivery point and the depot.

Function 2 The goods specified in each order line are validated. The order lines are recorded, linked to the goods and to the order or back-order as appropriate.

RESULT

Function 1
First entry point Delivery point- retrieved, selected by delivery point name.
Depot - retrieved, selected by bulk or package relationship.
Order - Stored, related to delivery point and depot.

Function 2 Product - retrieved, selected by product code.
Second entry point

Stock - retrieved, selected by relationship with product and depot and updated.

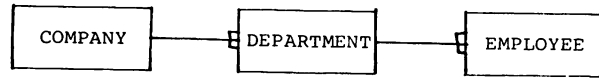
Order line - stored, related to order and product.

15. MAPPING TO IMAGE

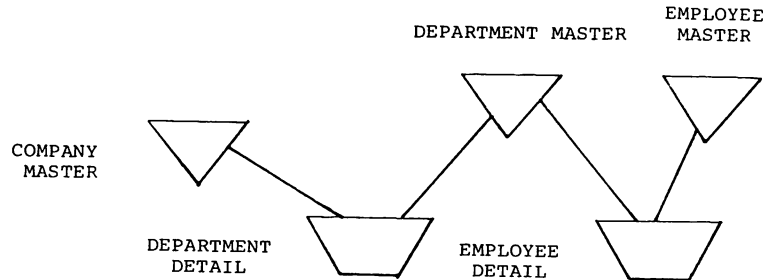
The activities of data modelling and activity analysis are performed as far as possible without reference to implementation techniques available in IMAGE. The process of access path analysis does, however, identify alternative methods of achieving the required result, besides showing that occasionally the data model is deficient or inconvenient for handling some functions. At this point it helps to understand the range of logical structures available, in order to consider the alternative methods of representing entities in the Database, together with their attributes and relationships. IMAGE provides the capability to implement relationships through the use of PATHS though in practice there can be many restrictions on using that PATH. The simplest way to convert entities to logical records is to create one record type for each entity type, i.e. creating a DATA SET or a number of DATA SETS for an entity. Attributes become DATA ITEMS. It may be necessary to divide attributes across more than one DATA SET to provide efficient access to the most frequently used attributes or to combine several entities into one logical record. Any such changes should be checked against the data model.

16. TYPICAL STRUCTURAL PROBLEM

A typical example of a problem presented by IMAGE is the following.



This would have to be represented in IMAGE by



Although it is an inconvenience to have to adjust the design to fit into these kind of circumstances, IMAGE has proved to be very simple for interpretation when looking to the ease of the DBSCHEMA.

17. PERFORMANCE CONSIDERATIONS

Performance considerations are quite often outside the scope of data analysis particularly as performance is usually dependent upon volume. In a high transaction volume system it is worth considering the possibility of splitting entities into sub-entities to reduce the volume in each set and perhaps even reducing the necessity for an access PATH.

18. CONCLUSION

Data analysis provides a good communication tool between the user wishing to understand his system and the analyst wishing to understand the user data. It provides a methodology which is flexible enough to adjust to new environments, not a checklist of standards which are inflexible to change.

IMAGE is successful mainly because it is simple to understand. With an easy-to-understand methodology and DBMS, implementation of systems becomes a smoother process with involved, motivated users and a database ready to cope with future demands.

COMPUTERIZED TYPESETTING: \TeX ON THE HP3000

COMPUTERIZED TYPESETTING: TEX ON THE HP3000

Lance Carnes
Independent Consultant
163 Linden Lane
Mill Valley, California 94941 U.S.A.

LANCE CARNES

September 1981

ABSTRACT: \TeX is a program which allows the ordinary user to produce professional quality typeset output. \TeX was developed by Donald E. Knuth of Stanford University and is currently used throughout the world for typesetting both technical and non-technical material. This paper will describe the use of \TeX and show some examples of its output. The transportable version of TEX, written in Pascal, has been successfully moved to the HP3000. The second part of the paper describes the tasks involved in this process.

I. INTRODUCTION

1. What is \TeX ?

Tau Epsilon Chi (\TeX) is a system for typesetting technical books and papers. It can also be used for ordinary non-technical material. The system does not require the user to have a knowledge of typesetting rules or conventions.

The original \TeX system was developed at Stanford University by Donald E. Knuth. Frustrated in his attempts to print a second edition of *The Art of Computer Programming* in the same printing style as the first edition, he looked for alternatives in the area of computerized typesetting. Finding nothing that suited him, he embarked on a project which was to become the \TeX system. This system is described in detail in his informative and humorous book, *\TeX and METAFONT* [Knut79].

The \TeX system is currently used throughout the world, partly for technical work in mathematics and physics, and partly for various other uses. The *Journal of the American Mathematical Society* now accepts \TeX input files for publication. Some major corporations and universities use it for typesetting their internal documentation, user manuals, newsletters, etc. The \TeX Users Group accepts articles and letters for their Journal in \TeX format.

2. How does it work?

The \TeX program accepts an input file consisting of text and control sequences, and generates a device independent output file (DVI file) which contains commands for driving a raster printing device. Once \TeX has processed the input and produced a DVI file, it is up to a device driver program to interpret the commands in the DVI file and produce

LANCE CARNES
INDEPENDENT CONSULTANT
163 LINDEN LANE
MILL VALLEY
CALIFORNIA 94941 USA

printed output. This sequence of events is shown in Figure 1.

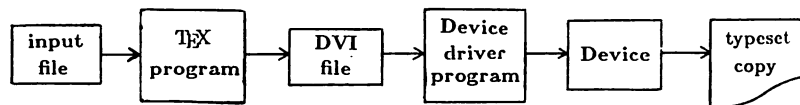


Figure 1. Functional diagram of the TeX system.

Most of the typesetting is done by TeX automatically. TeX operates on many levels, composing pages, paragraphs lines and words. All of these are interrelated, with the intention of producing professional quality printing. In cases where TeX needs to be guided, for example in printing the TeX logo, the user intervenes by specifying a control sequence (see 3. below).

The TeX system does not typeset a single word or a single line at a time. Rather, it typesets a page or more at a time. This is done for a variety of reasons. Mainly, we want the printed page to consist of pleasantly spaced paragraphs, lines and words. Also we want to avoid other unwanted phenomena, such as “widow” lines. A widow line is the first line of a paragraph appearing at the bottom of a page with the paragraph continuing on the next page. To eliminate widows, TeX returns to the paragraphs already layed out and expands them slightly so as to use one more line on the page. This forces the widow line to the top of the next page.

Paragraphs are composed to reduce the number of hyphenations and so as not to leave a single word stranded in the last line. In addition, the spacing between words is equalized throughout the paragraph.

Lines of text are composed of words and other symbols (e.g. mathematical formulas) with the space between words equalized.

Words are typeset with the letters placed one character width apart. Unlike standard computer printers which print all characters in the same width (usually 1/10 inch), typesetting separates characters by the exact width of the character, depending on the “font” or character style used. In addition, TeX will place characters closer together or farther apart in accordance with traditional typesetting rules. For example, when typesetting the word “AVIATOR” the “A” and “V” are placed closer together; this is called “kerning”. Notice in the word “find” that the “f” and “i” are pushed together to form the “ligature” fi. These typesetting conventions and more are known to TeX, freeing the user from having to memorize them.

The basic concepts TeX uses are “boxes” and “glue”. A box contains something which is to be printed, and glue specifies the spacing between boxes. For example, a character is a box, a word is a collection of character boxes, a line is a group of word boxes, a paragraph is a collection of line boxes, and a page is a box composed of paragraph boxes. The space between boxes can expand or contract by carefully defined amounts, called the stretchability or shrinkability of the glue. For example, when TeX composes a paragraph that has a hyphenation it tries to back up and redistribute the spacing of the words in the paragraph to avoid the hyphenation. It does this by increasing or shrinking the space or

glue between the boxes by allowable amounts.

For further details on the inner workings of TeX, see [Knut79] or [Spiv80].

3. Submitting an input file to TEX

The input file for TeX is edited using any text editor. The text and any control sequences are contained in this file. When TeX is run, this file is designated as the input file.

Basically, text is entered in a standard fashion with spaces between words, and one blank line between paragraphs. The input need not be formatted in any particular manner beyond this. Control sequences are defined as a “\” followed by a word or symbol. They allow the user to specify a special command. For example, “\it IMPORTANT” would cause the word IMPORTANT to be set in italic font.

The TeX system can be run in either interactive or batch mode. In interactive mode, if TeX finds an error the user is allowed to make modifications on the fly. For example,

```
!Undefined control sequence
\iy
  IMPORTANT
↑
```

The TeX program is indicating that it does not know the control sequence “\iy” and shows what it has scanned on the first line, and what it has not yet scanned on the following line. At this point the user may correct the input by typing “i” to erase one symbol or control sequence, and then “I” to insert the correct sequence “\it”. Any corrections made in this manner are recorded in an errors file for future reference.

As TeX is processing the input, it is writing to the DVI file. After the input is successfully processed, the DVI file is ready for the device driver program.

Two other important facilities are available with TEX. These are alternate input files and macro definitions. Alternate input files are TeX input files which are read in conjunction with another input file. For example, if a paper has an abstract and three sections, and each is in a separate file, a main file would draw them all together as follows:

```
% paper on TEX for the HP3000
\input basic % basic control sequences
\input texabs % abstract file
\input sect1
\input sect2
\input sect3
\end
```

Each of the alternate input files could have had \input commands also. The maximum nesting depth is nine.

Macro definitions allow the user to specify a common sequence by defining it and giving it a name. For example, the logo TeX was specified by inserting “\TEX”. \TEX was previously defined as

```
\def\TEX{\hbox{lowercase{\:a \uppercase{T}\hskip-2pt\lower1.94pt
\hbox{\uppercase{E}}\hskip-2pt \uppercase{X}}}}
```

It is much easier to write “\TEX” than to insert the above expansion.

4. Fonts

A font is a specific design of an alphabet and associated symbols. For example, this paper is set in a font called Computer Modern Roman. Most typewriters have Pica or Elite type fonts. The different “balls” or “daisy wheels” on some printers allow the user to change fonts.

The T_EX system allows up to 64 different fonts to be specified within the same job. A control sequence is given to switch from one to the other. Naturally you must have a device which can support all of these different fonts.

Knuth also wanted to define his own fonts and created a system called METAFONT to do this. Using METAFONT one can design a font which is coded into a file for use by T_EX. For more information on METAFONT see [Knut79].

5. The DVI file

The DVI file consists of a series of 8-bit codes which tells a device driver how to typeset the job. The format of the DVI files is given in Appendix B.

Basically, a DVI file command is of the form “set the letter d and advance the character width” or “change to font 3” or “advance vertically 12 rsu’s”. No inherent intelligence on the part of the device is assumed. In fact, T_EX gets along best with devices which have no internal programming, such as proportional spacing or typesetting firmware.

6. Device drivers.

The assumed printer is a raster scan printing device. This implies that all spacing between characters and lines is user specified. A typical computer line printer is not a raster device since it will always print 10 characters/inch, and six lines/inch (or some variation of this). Most of the daisy wheel terminals available now can be used as raster devices. The actual device T_EX is aimed at is a commercial computer-driven typesetting device, such as a Xerox Graphics Printer, a Mergenthaler Linotron 202, or an HP2680 Laser Printer.

The T_EX program has no knowledge of any particular printing device. It creates the same DVI file regardless of the output device. It is the job of the Device Driver program to interpret the DVI commands and produce output on a specific device. While there is only one T_EX program, there will be one Device Driver program for each output device.

II. T_EX on the HP3000

1. T_EX in Pascal

The T_EX system was originally written in a language called SAIL (Stanford University Artificial Intelligence Language). The SAIL compiler and the original T_EX system ran

on the DEC-20 computer only. T_EX is in the public domain, but was not even remotely transportable. Due to the popularity of the system, a project was undertaken to translate the T_EX system into a computer language which was available on most modern computer systems.

The language chosen for the transportable system was Pascal. The method for translating the system was as follows. First, a well documented pseudo-Pascal source was developed. This source has only a slight resemblance to a Pascal program and was intended to serve mostly as documentation, and to give all the algorithms. This file is often referred to as the DOC file.

The second step was to produce syntactically correct Pascal source code from the DOC file. There is a program called UNDOC which performs this step. The resulting Pascal source is distributed anyone wanting to transport T_EX to another computer.

The DOC file is actually typeset, and a photocopy is provided with the distribution tape. The Pascal source is almost unreadable, but will compile. Examples of both of these files is in Appendix C.

2. Moving T_EX to the HP3000.

The Stanford T_EX-in-Pascal project brought the system to a point where it could be transported to other computer systems. The transportation process, however, requires a good deal of time and a patient systems programmer.

At the time of writing, this author has successfully transported T_EX to the HP3000. The project was by no means trivial, as will be shown.

Bringing T_EX to the HP3000 had a lot of problems right from the outset. First, there was no supported Pascal compiler at the time this project was begun. Second, the design of the T_EX program assumes a large address space, something on the order of 600K words of addressable memory.

The tasks broke down as follows:

- a. Edit the Pascal sources. While the system was translated to a “Standard” Pascal, there are still many variations and assumed extensions which had to be accounted for. With 23,000 lines of Pascal source this took considerable time and effort.
- b. Rewrite the “System dependent” routines. These are the procedures and functions which interface T_EX with the file system, terminal I/O and other traits particular to the host system. About 25 routines had to be modified or rewritten.
- c. Implement a virtual memory scheme. T_EX references several large arrays throughout, some as large as 50,000 elements with 4 32-bit words per element. An addressing scheme was developed to allow the array contents to reside in secondary storage.
- d. Revise the Pascal compiler to allow 32-bit integers and to compile large array references. T_EX assumes 32-bit integers throughout, and the Portable P4 compiler from the HP Users Contributed library was modified to allow them.
- e. Optimize the performance of the system. When the above tasks were completed and the system first ran on the HP3000, it was the incredibly slow. Where

the original \TeX system at Stanford processed a document in less than two minutes, the initial HP3000 \TeX took 40 minutes. By analyzing \TeX 's operation, some optimizations have been made reducing the run time to about 6 minutes. Additional optimizations will be made to allow the system to run as fast as possible. One tool which has been particularly useful for identifying inefficient code is APC/3000 from Wick Hill Associates.

3. Device drivers.

A device driver for a daisy wheel printer has been developed for use on the HP3000. While only one font is available at a time with this device, satisfactory results have been obtained. The output is suitable for internal documentation, and for proofing a document. Future plans are to develop a driver for the HP2680 Laser printer.

However, it is not necessary to have a high quality printing device on-site. There is one commercial printing house in San Francisco which uses \TeX for typesetting on a Mergenthaler Linotron 202; the output from this device is camera ready. DVI files produced by \TeX on the HP3000, once proofed on the daisy wheel printer, will be sent to this commercial printer.

III. Conclusions

This is a truly remarkable system. It gives the ordinary person the ability to print professional quality copy. The user will not have to explain to a typographer what is wanted, but will have personal control.

The HP3000 implementation of \TeX will be a boon for any organization desiring to improve the quality of documentation, user manuals and other printed materials. Good results can be obtained with an inexpensive daisy wheel printer. Where camera ready copy is desired, several higher quality devices are commercially available.

Hopefully more organizations will begin to use \TeX for documentation, manuals, annual reports and newsletters. Perhaps one day soon the HP General Systems Users Group will accept papers for publication in \TeX format.

ACKNOWLEDGEMENTS.

My thanks to Prof. Luis Trabb-Pardo and Charles Restivo of Stanford University for their assistance in learning the \TeX system; and to GENTRY, INC. of Oakland, California, for providing time on the HP3000.

REFERENCES.

[Knut79] Donald E. Knuth, \TeX and METAFONT. New Directions in Typesetting. Digital Press, 1979.

This is a beautifully printed book, an acknowledgement of the \TeX system. Don Knuth's writing style is at once brilliant and witty. It contains a User's Guide to the \TeX and METAFONT systems and a paper on Mathematical Typography.

[Spiv80] Michael Spivak, *The Joy of TEX*. A Gourmet Guide to Typesetting Technical Text by Computer. Version -1. American Mathematical Society, 1980.

This is a real book. It gives a lighthearted introduction to the use of AMS- \TeX , the version of \TeX used by the AMS.

TUGboat. The \TeX Users Group Newsletter. Published by the American Mathematical Society.

The \TeX Users Group is small currently, but enthusiastic and helpful. For information on membership write to

TEX Users Group
c/o American Mathematical Society
P.O. Box 6248
Providence, Rhode Island 02940 USA

`\noindent (\bf ABSTRACT:) \TEX` is a program which allows the ordinary user to produce professional quality typeset output. `\TEX` was developed by Donald E. Knuth of Stanford University and is currently used throughout the world for typesetting both technical and non-technical material. This paper will describe the use of `\TEX` and show some examples of its output. The transportable version of `TEX`, written in Pascal, has been successfully moved to the HP3000. The second part of the paper describes the tasks involved in this process.

`\vskip 0.4 cm`
`\noindent (\bf I. INTRODUCTION)`

`\vskip 0.3 cm`
`\noindent (\bf 1. What is \TEX ?)`

`\vskip 0.1 cm`
`(\it Tau Epsilon Chi) (\TEX)` is a system for typesetting technical books and papers. It can also be used for ordinary non-technical material. The system does not require the user to have a knowledge of typesetting rules or conventions.

The original `\TEX` system was developed at Stanford University by Donald E. Knuth. Frustrated in his attempts to print a second edition of `(\it The Art of Computer Programming)` in the same printing style as the first edition, he looked for alternatives in the area of computerized typesetting. Finding nothing that suited him, he embarked on a project

ABSTRACT: `\TeX` is a program which allows the ordinary user to produce professional quality typeset output. `\TeX` was developed by Donald E. Knuth of Stanford University and is currently used throughout the world for typesetting both technical and non-technical material. This paper will describe the use of `\TeX` and show some examples of its output. The transportable version of `TEX`, written in Pascal, has been successfully moved to the HP3000. The second part of the paper describes the tasks involved in this process.

I. INTRODUCTION

1. What is `\TeX` ?

Tau Epsilon Chi (`\TeX`) is a system for typesetting technical books and papers. It can also be used for ordinary non-technical material. The system does not require the user to have a knowledge of typesetting rules or conventions.

The original `\TeX` system was developed at Stanford University by Donald E. Knuth. Frustrated in his attempts to print a second edition of *The Art of Computer Programming* in the same printing style as the first edition, he looked for alternatives in the area of computerized typesetting. Finding nothing that suited him, he embarked on a project

APPENDIX A. A portion of the `\TeX` input file for this paper.

Command Name	Command Bytes	Description
VERTCHAR0	0	Set character number 0 from the current font such that its reference point is at the current position on the page, and then increment horizontal coordinate by the character's width.
VERTCHAR1	1	Set character number 1, etc.
	:	:
VERTCHAR127	127	Set character number 127, etc.
NOP	128	No-op, do nothing, ignore. Note that NOPs come between commands, they may not come between a command and its parameters, or between two parameters.
BOP	129 c0[4] c1[4] ... c9[4] p[4]	Beginning of page. The parameter p is a pointer to the BOP command of the previous page in the .DVI file (where the first BOP in a .DVI file has a p of -1, by convention). The ten c's hold the values of <code>\TeX</code> 's ten \counters at the time this page was output.
EOP	130	The end of all commands for the page has been reached. The number of PUSH commands on this page should equal the number of POPs.
PUSH	132	Push the current values of horizontal coordinate and vertical coordinate, and the current w-, x-, y-, and z-amounts onto the stack, but don't alter them (so an X0 after a PUSH will get to the same spot that it would have had it had been given just before the PUSH).
POP	133	Pop the z-, y-, x-, and w-amounts, and vertical coordinate and horizontal coordinate off the stack. At no point in a .DVI file will there have been more POPs than PUSHes.
HORZRULE	135 h[4] w[4]	Typeset a rule of height h and width w, with its bottom left corner at the current position on the page. If either h ≤ 0 or w ≤ 0, no rule should be set.

APPENDIX B. DVI commands.

VERTRULE	134 h[4] w[4] Same as HORZRULE, but also increment horizontal coordinate by w when done (even if $h \leq 0$ or $w \leq 0$).	Y3	147 n[3] As above.
HORZCHAR	136 c[1] Set character c just as if we'd gotten the VERTCHARc command, but don't change the current position on the page. Note that c must be in the range [0..127].	Y4	146 n[4] As above.
FONT	137 f[4] Set current font to f. Note that this command is not currently used by TeX—it is only needed if f is greater than 63, because of the FONTNUM commands below. Large font numbers are intended for use with oriental alphabets and for (possibly large) illustrations that are to appear in a document; the maximum legal number is $2^{32} - 2$.	Y0	149 Guess.
X2	144 m[2] Move right m rsu's by adding m to horizontal coordinate, and put m into x-amount. Note that m is in 2's complement, so this could actually be a move to the left.	Z2	152 m[2] Another downer. Affects vertical coordinate and z-amount.
X3	143 m[3] Same as X2 (but has a 3 byte long m parameter).	Z3	151 m[3]
X4	142 m[4] Same as X2 (but has a 4 byte long m parameter).	Z4	150 m[4]
X0	145 Move right x-amount (which can be negative, etc).	Z0	153 Guess again.
W2	140 m[2] The same as the X2 command (i.e., alters horizontal coordinate), but alter w-amount rather than x-amount, so that doing a W0 command can have different results than doing an X0 command.	FONTNUM0	154 Set current font to 0.
W3	139 m[3] As above.	FONTNUM1	155 Set current font to 1.
W4	138 m[4] As above.	:	:
W0	141 Move right w-amount.	FONTNUM63	217 Set current font to 63.
Y2	148 n[2] Same idea, but now it's "down" rather than "right", so vertical coordinate changes, as does y-amount.		

25. Find the equation of the plane passing through the end points of the three vectors $\mathbf{A} = 3\mathbf{i} - \mathbf{j} + \mathbf{k}$, $\mathbf{B} = \mathbf{i} + 2\mathbf{j} - \mathbf{k}$, and $\mathbf{C} = \mathbf{i} + \mathbf{j} + \mathbf{k}$, supposed to be drawn from the origin.

26. Show that the plane through the three points (x_1, y_1, z_1) , (x_2, y_2, z_2) , and (x_3, y_3, z_3) is given by

$$\begin{vmatrix} x_1 - x & y_1 - y & z_1 - z \\ x_2 - x & y_2 - y & z_2 - z \\ x_3 - x & y_3 - y & z_3 - z \end{vmatrix} = 0.$$

Solution. Let $w = f(x, y, z) = x^2 + y^2 - z$, so that the equation of the surface has the form

$$f(x, y, z) = \text{constant},$$

36. The procedure *Print* takes an integer as argument and prints the corresponding *stringpool* entry both in the terminal and in the errors file.

```

procedure Print(mes : integer);
var i : integer; { index in the string }
    c : asciiCode;
begin i := string[mes]; c := stringpool[i];
while c <> null do
begin terOut := chr(c); errfil := chr(c); put(terOut); put(errfil);
Increment(i); c := stringpool[i]
end;
end;
procedure PrintLn(mes : integer);
{ Like Print, but beginning at a new line. }
begin terOut := chr(carriageReturn); errfil := terOut; put(terOut);
put(errfil); terOut := chr(lineFeed); errfil := terOut; put(terOut);
put(errfil); Print(mes)
end;

```

```

1214 PROCEDURE PRINT(MES: INTEGER);
1215 VAR I: INTEGER;
1216     C: ASCII_CODE;
1217 BEGIN I := STRING[MES]; C := STRINGPOOL[I];
1218 WHILE C <> NULL DO
1219 BEGIN TEROUT := CHR(C); ERRFIL := CHR(C); PUT(TEROUT); PUT(ERRFIL);
1220 INCREMENT(I); C := STRINGPOOL[I]
1221 END;
1222 END;
1223 PROCEDURE PRINTLN(MES: INTEGER);
1224 { Like PRINT, but beginning at a new line. }
1225 BEGIN TEROUT := CHR(CARRIAGERETURN); ERRFIL := TEROUT; PUT(TEROUT);
1226 PUT(ERRFIL); TEROUT := CHR(LINEFEED); ERRFIL := TEROUT; PUT(TEROUT);
1227 PUT(ERRFIL); PRINT(MES)
1228 END;

```

CONTENTS

OPTIMIZATION OF SPL AND FORTRAN PROGRAMS

John Machin

Campbell & Cook Computer Services

Melbourne, Australia

1.	INTRODUCTION	1
	1.1 Scope	1
	1.2 Terminology	2
	1.3 Basis	3
	1.4 Background: The Machine Architecture	3
	1.5 Background: The Compilers	5
2.	GENERAL OPTIMIZATION PRINCIPLES	6
	2.1 Do it at compile-time	6
	2.2 Do it only as often as needed	7
	2.3 Do it in the registers	8
	2.4 Don't do it at all	10
	2.5 Don't do it with a procedure call	11
	2.6 Be wary of long reals	15
	2.7 Avoid data-type conversions	17
	2.8 Multiply instead of dividing	18
	2.9 Exploit special hardware features	19
	2.10 Avoid unnecessary memory references	21

OPTIMIZATION TOPICS SPECIFIC TO FORTRAN	23
3.1 Multi-dimensioned arrays	23
3.2 MORECOM	25
3.3 \$INTEGER*4	29
3.4 Character Variables	31
3.5 Indirect Indirection	34
3.6 \$CONTROL BOUNDS	37
3.7 The Formatter	38
3.8 \$CONTROL INIT	38
3.9 Use of SPL Routines	39
4. MISCELLANEOUS	40
4.1 Slow programs: prevention	40
4.2 Slow programs: cure	40
4.3 Know your machine	42
4.4 Future Shock	43
REFERENCES	44
APPENDIX A: CLEARMANY PROCEDURE	45
A.1 Purpose	45
A.2 Calling sequence	45
A.3 Source	46

1. INTRODUCTION

1.1 Scope

This paper discusses general principles and specific techniques for making SPL and FORTRAN programs use less CPU time on HP3000 computers.

There are three things which affect the CPU speed of a program:

- (a) **Hardware:** Once one has purchased a particular machine, nothing can be done to increase its CPU speed.
- (b) **Manufacturer-supplied compilers and run-time libraries:** The quality of compiler-generated code can have a potent effect on the speed of a program, as can the efficiency or otherwise of the library routines it calls. Later sections of the paper highlight language features and compiler features which necessarily execute slowly, in order that the programmer might avoid them, where

possible. Other features, where the necessity is improbable or dubious, are listed for temporary avoidance by the programmer until the suggested compiler enhancements are implemented.

- (c) User-written code: Given that suitable algorithms have been chosen, the way in which the user programs them can be of considerable consequence.

1.2 Terminology

To avoid much repetition, abbreviations are used for the names of the four main numerical data types, as shown below:

<u>Abbreviation</u>	<u>Expansion</u>	<u>SPL Equivalent</u>	<u>FORTRAN Equivalent</u>
SI	short integer	integer	integer[*2]
LI	long integer	double	integer*4
SR	short real	real	real
LR	long real	long	double precision

1.3 Basis

Statements made in this paper in relation to code emission by the compilers are based on the Athena (1918) software release.

Timings were obtained on an HP3000 Series III.

1.4 Background: The Machine Architecture

- 1.4.1 It is of course stating the obvious to say that the HP3000 architecture, at the "machine instruction" level, is scarcely traditional. Many of the features of the machine ease considerably the task of generating machine code for medium-complexity languages such as SPL and FORTRAN. Some features however cause difficulty in emitting code, and this difficulty sometimes lead to suboptimal code.

- 1.4.2 The low limits on direct data addresses (e.g. DB+255,Q+127) can cause problems with programs requiring many variables. This leads to an extra level of indirection in FORTRAN programs (see "MORECOM" and "Indirect Indirection" later) and surgery in SPL programs.

1.4.3 The BR (branch) instruction post-indexes (like all other memory-reference instructions) when it should pre-index. The code generated for computed GO TO, CASE and SWITCH statements is cute but is about 3 times as much as would otherwise be necessary.

1.4.4 The elegant simplicity of the stack machine is rudely violated by the instructions which handle long reals. Far from the zero-address "stackops" used for arithmetic on other data types, these instructions operate on the addresses of one operand (negate), two operands (compare), or three operands (add, subtract, multiply, divide). Given this startling departure from orthogonality, together with the fact that there are no specific instructions for loading or unloading the stack four words at a time, the number of references to long reals in later sections of this paper should cause little surprise.

1.4.5 The low limits on direct code addresses (e.g. P-255, P+255 for LOAD, BR, MTBA etc and P-31, P+31 for the test-and-branch instructions) cause two problems in code generation:

- (a) Is a branch to be direct or indirect?
If indirect, where should the indirect word be placed?

- (b) Where should constants be placed?

Some solutions to these problems are good; occasionally however, an indirect branch is used unnecessarily, and a constant is dumped immediately (with a branch around it) instead of being carried forward.

1.5 Background: The Compilers

Neither the SPL compiler nor the FORTRAN compiler is represented by Hewlett-Packard as being an optimizing compiler. However, on perusing a check list of optimization techniques described in the literature, one finds that some of these are used (at least partially) and that the use of others is obviated by the machine architecture.

It is generally accepted that of the languages available on the HP3000, SPL is the most "efficient", closely followed by FORTRAN. Looking at directly comparable features of the two languages, it is found that sometimes SPL generates better code, and other times FORTRAN does. Both compilers would benefit from a cultural exchange.

2. GENERAL OPTIMIZATION PRINCIPLES

2.1 Do it at compile-time

While it may look better to code:

```
PARAMETER PI = 3.14159...
...
CIRCUM = 2.0 * PI * RADIUS
(or the SPL equivalent)
```

it will run faster if you write

```
PARAMETER TWOPI = 6.28318...
CIRCUM = TWOPI * RADIUS
```

Neither compiler will simplify expressions involving constants; if you write

```
A:= 1 + 1;
```

that is exactly what you get.

2.2 Do it only as often as needed

2.2.1 Both compilers perform limited elimination of common sub-expressions within a statement. This is done only with respect to subscripted array references. The seemingly wider scope stated by Splinter [1] (expressions in parentheses) does not prevail.

The method used is to load the index register once only with the value required for the offset into the array(s).

Three qualifications must be met for the compilers to perform this optimization:

- (a) The subscript expressions must be lexically identical.
- (b) The array(s) must not be long real.
- (c) (SPL only) The subscript "expressions" can only be simple variables or constants.

Examples of FORTRAN statements where the elimination is done are:

```
A(I + J) = B(I + J) + C(I + J)
D(I + 3, J + 5) = E(I + 3, J + 5)
```

No elimination is done in:

```
A(I + J) = B(J + I)  (not lexically same)
K = (I + J) * (I + J) (not in array reference)
```

2.2.2 Where there are common sub-expressions not fitting the above criteria, time and code-space can be saved by using a temporary variable.

2.2.3 Invariant expressions can be moved outside loops. This seems obvious, almost too trivial to mention, but such cases can be "hidden" by the high level language: see section 3.1.1.

2.3 Do it in the registers

2.3.1 The HP3000 architecture does not offer quite the same scope (or the necessity!) for optimizing the use of registers as does a machine of the "umpteenth general purpose registers" variety.

2.3.2 The index (X) register has a limited arithmetic capability, and may also be used as temporary storage. Of course the X register is used for several things other than array subscripting, and so it is dangerous to merely equate some name to the X register and write code as though an ordinary variable was involved.

In particular the use of the X register in and around statements involving long reals is perilous.

E.g. LONG A, B;

```
LONG ARRAY C(0:10), D(0:10);
INTEGER X = X;
...
A:= B; << SETS X TO 1
        IF B IS A PROCEDURE ARGUMENT BY
        REFERENCE>>
...
C(X):= D(X); << CHANGES VALUE OF X >>
```

2.3.3 The assignment operator can be used in SPL to replace a load from memory with a faster stack-duplicate operation.

Instead of

```
C:= D + E;
```

```
A:= B + C;
```

write

```
A:= B + (C:= D + E);
```

Warning: for long reals, the compiler generates worse code for the latter case.

2.4 Don't do it at all

2.4.1 When you write

```
FOR I:= J STEP K UNTIL L DO .....
```

the SPL compiler must generate code which checks whether the loop is to be entered at all.

When you write

```
FOR I:= 0 UNTIL 9 DO .....
```

the loop body must be entered, but the compiler still goes through the motions.

If this latter loop is nested within others, this is a waste of time, which can be saved (together with 2 or 3 words of code) by writing

```
FOR* I:= 0 ....
```

2.5 Don't do it with a procedure call

2.5.1 As is well known, there is a reasonable overhead involved in a call to a procedure in the current segment, and a greater overhead in a call to a procedure in another segment (especially if the called segment is not present in memory).

While splitting a program into procedures or subroutines aids greatly in structuring a program, care should be taken to avoid frivolous procedure calls.

2.5.2 The SPL subroutine, although offering a somewhat more austere environment than a procedure, has the advantage of faster invocation and faster return.

It may sometimes be worth the waste of code space to change a procedure into a subroutine and include it in each calling procedure.

The following "dirty trick" allows a single copy of a subroutine to be shared by several procedures in the same segment:

(a) Include the source of the subroutine ("SHARED'SUB") in an "initialising" procedure.

(b) The initialising procedure should include:

```
SUB'ADDR:= @SHARED'SUB;
where SUB'ADDR is global.
```

(c) Invocation of the shared subroutine is done by

```
<< stack arguments, if required >>
TOS:= SUB'ADDR;
ASSEMBLE (SCAL 0);
```

2.5.3 Less obvious than explicit procedure calls (coded by the programmer) are implicit procedure calls generated by the compilers.

Usually there is a rationale for an implicit procedure call: the language feature is not directly supported by the microcode, and in-line code would take up too much space.

2.5.4 In Fortran, all "basic external functions" are handled by procedures. Turning to the "intrinsic functions", which also look like function calls at the source level, we find that some of them are in fact handled by in-line code. Almost all the numerical functions are in this category; among the exceptions are AINT, JDINT, DDINT, AMOD, and the MAX/MIN family.

Of course the MAX and MIN procedures cater for a variable number of arguments; but there is a case for using in-line code for the frequent case of two short integer arguments. The code currently generated for $J = \text{MAX}(K, L)$ is

```
LOAD K
LOAD L
LDI 2
PCAL MAX0'
STOR J
```

whereas in-line code would take only two more words:

```
LOAD K
LOAD L
DDUP, CMP
BGE P + 2
XCH, NOP
DEL, NOP
STOR J
```

In a degenerate case such as $J = \text{MAX}(K, J)$, the programmer can instead write

```
IF (J.GT.K) J = K
```

which is better than the in-line code above, especially if the probability of J exceeding K is low.

- 2.5.5 In both SPL and FORTRAN, procedures are generally used for exponentiation. The exception is that FORTRAN emits in-line code for the exponentiation of short integers and short reals to the short integer powers 1, 2, 3 and 4.

This means that, contrary to the advice given by H-P [3], it is better to write A^{**2} than $A*A$, where A is short (integer or real). The converse applies when A is long (integer or real).

It is curious that optimization of the unlikely expression A^{**1} is done (not very well: $B = A^{**1}$ generates e.g.

```
LDD Q + 1, I; NOP, NOP; STD Q + 2, I)
```

whereas no effort is made with the equally unlikely expression A^{**0} .

2.6 Be wary of long reals

- 2.6.1 As mentioned earlier, the non-stack nature of the instructions for handling long reals makes life hard for compiler writers, occasionally leading to the emission of rather peculiar code.

- 2.6.2 In the code generated for $A = B + C$, the FORTRAN compiler loads the address of A last instead of first, then does CAB, CAB to put it into the right place. The SPL compiler avoids this waste of time and code space.

2.6.3 The FORTRAN compiler always uses the MOVE instruction for simple assignments, and usually achieves reasonable code, e.g. nine machine instructions for $A(J) = B(J)$. On the other hand the SPL compiler eschews the MOVE instruction and in desperately trying to simulate 4-word loads and stores, requires 22 instructions to encode $A(J) := B(J)!!$

2.6.4 As the long real machine instructions work with addresses, not values on the stack, it follows that when expressions force the compilers to put temporary results on the stack (the natural method with other data types), the results will be sub-optimal.

One way of avoiding this is to use variables to hold often used constants.

It is better to write

```
ONE = 1D0
...
A= A + ONE
B= B + ONE
```

than

```
A= A + 1D0
B= B + 1D0
```

Another method is to simulate the code which would be emitted by a compiler for a 3-address machine:

Instead of

```
A= B + C * D
```

write

```
TEMP = C * D
A= B + TEMP
```

2.7 Avoid data-type conversions

2.7.1 Unlike SPL where the programmer must explicitly code a type conversion, FORTRAN automatically emits type conversions in "mixed-mode" expressions. As these conversions take time and code-space, they should be avoided where possible.

2.7.2 Particularly wasteful is the habit common to some programmers of using short integer constants in an otherwise real expression. The code generated by $A = A + 1$ runs at about 70% of the speed of that generated by $A = A + 1.0$.

2.7.3 Conversion from long integer to long real and vice versa requires a procedure call; all others are done in-line.

2.7.4 Intriguing code is generated by the FORTRAN compiler for the conversion from long integer to short integer.

The instructions used to do it on the stack are:

```
DTST, NOP
DASL 16
DEL, NOP
```

followed by a test for overflow which is not done in SPL. The SPL compiler uses only one word of code instead of the 3 above:

```
DTST, DELB.
```

2.8 Multiply instead of dividing

2.8.1 Multiplication is faster than division, so it should be substituted where possible.

2.8.2 Care should be used when replacing division by multiplication when a constant is involved. For example 10.0 can be represented exactly as a short real, but 0.1 cannot be. Coding $A=B*0.1$ instead of $A=B/10.0$ may result in loss of precision.

2.9 Exploit special hardware features

2.9.1 Testing for a true or false value is actually reduced by the machine to a test for odd or even. Consequently we may obtain a test for parity in the guise of a "logical" test.

In SPL, the condition

```
I MOD 2 = 1
```

can be re-written as

```
LOGICAL (I)
```

and in FORTRAN, the similar condition can be re-written as `BOOL (I)`.

2.9.2 In SPL the construct `I <= J <= K` uses the CPRB (compare range and branch) instruction and it is better to use this than

```
I <= J AND J <= K.
```

However, when the lower bound is zero, it is much faster to use

```
LOGICAL (J) <= LOGICAL (K)
than 0 <= J <= K.
```

2.9.3 The hardware condition code is not affected merely by testing it, nor by branches. Where a logic path is required for each of the results of a comparison (<, +, >), the test does not need to be performed twice.

Instead of

```
IF I = J THEN ...
ELSE IF I > J THEN ...
ELSE ...;
```

write

```
IF I = J THEN ...
ELSE IF > THEN ...
ELSE ...
```

2.9.4 The CMPB (compare bytes) instruction can be induced to report the residual count of uncomparred bytes, as well as the addresses of the unequal bytes and the condition code.

A classic case is scanning off trailing blanks. Although it is easier and clearer to write

```
WHILE LEN > 0 AND BUF(LEN-1) = " " DO LEN:= LEN-1;
```

the following code runs much faster:

```
IF BUF(LEN-1) = " " THEN BEGIN
    IF BUF(LEN-2) = BUF(LEN-1), (1-LEN), 0 THEN;
    LEN:= - TOS;
    DDEL;
END;
```

2.10 Avoid unnecessary memory references

The standard practice of the FORTRAN compiler and the normal usage of the SPL programmer is to address arrays indirectly through a pointer. To obtain the contents of an array element, the contents of the pointer cell must first be obtained.

2.10.2 The SPL programmer can avoid this, when sufficient primary address space is available, by coding "=DB" or "=Q" in the array declaration. As only the "zero'th" element of the array must be in the direct address range, one large array may use direct addressing.

2.10.3 The FORTRAN compiler makes no attempt to use direct addressing, even in the simple case when all the local arrays and variables would fit in the range (Q+1, Q+127).

Although optimal allocation of addresses might require $n!$ iterations where n is the number of local arrays and variables, some optimization would be better than none.

3. OPTIMIZATION TOPICS SPECIFIC TO FORTRAN

3.1 Multi-dimensional arrays

3.1.1 Given the declaration

```
DIMENSION A(3, 4, 5), AX(60)
EQUIVALENCE (A, AX)
```

when the programmer writes

```
DO 100 K = 1, 5
100 T = T + A(I, J, K)
```

the effect is as though the following had been written:

```
DO 100 K = 1, 5
100 T = T + AX(((K - 1) * 4 + J - 1) * 3 + I)
```

Obviously part of the offset calculation need be done once only, before the loop is entered.

It is possible to recode this as:

```
IX = (J - 5) * 3 + I
DO 100 K = IX + 12, IX + 60, 12
100 T = T + AX(K)
```

Such rewriting is of course error-prone. Having the compiler do it would be preferable, but this is one of the more complex optimization algorithms.

- 3.1.2 Where one or more of the subscripts is a constant, no cognisance is taken of this, and the effect is ludicrous. For example, when the user writes A(1, 1, 1), the code emitted is as though he had written

```
AX(((1 - 1) * 4 + 1 - 1) * 3 + 1)
```

instead of AX(1)!!

There is a glaring need for compiler enhancement in this case.

- 3.1.3 The programmer should evaluate carefully his perceived need for multi-dimensioned arrays. Often such an array can be replaced entirely by an array of one dimension, or an array of one dimension can be equivalenced to it and used for some of the manipulations required (especially those which operate on every element of the array).

- 3.1.4 One special case is where an array has two dimensions, but in references to the array, one of the subscripts is always constant. This array should be split up into several arrays of one dimension.

For example:

```
DIMENSION CASH(3, 10)
...
NET = CASH(1, J) - CASH(2, J) - CASH(3, J)
is better written as:
DIMENSION SALES(10), EXPENSES(10), TAX(10)
...
NET = SALES(J) - EXPENSES(J) - TAX(J)
```

This is much clearer as well as much more efficient.

3.2 MORECOM

- 3.2.1 As mentioned earlier, the limit of 255 on direct DB-relative addressing causes problems with FORTRAN programs with many variables and arrays in COMMON.

The compiler option \$CONTROL MORECOM was introduced to alleviate this problem.

When this option is in effect, instead of one word of primary DB being allocated to point to each variable and array in COMMON, one word is allocated to point to each COMMON block. Consequently indexing must be used to address variables within COMMON.

3.2.2 Assume the following declarations:

```
REAL A, B, C
INTEGER I, J
COMMON/BLK/I, A, J, B
```

Without the MORECOM option in effect, the statement `C = B` will generate code such as:

```
LDD DB + 3, I
STD Q + 1
```

which is as good as one will ever get.

With MORECOM, however, the same statement will generate:

```
LDXI 2
LDD DB + 0, I, X
STD Q + 1
```

The variable B above is at an even offset (4) from the start of the block - this allows the use of double-word indexing in the LDD instruction.

Things can be worse: the variable A is at an odd offset (1) from the start of the block - what happens is this:

```
LDXI 1
LOAD DB + 0, I, X
INCX, NOP
LOAD DB + 0, I, X
DTST, NOP
STD Q + 1
```

To add insult to injury, the DTST instruction above is quite redundant.

3.2.3 A similar effect is observable when arrays are used; however when the offset is zero, the (longer) code for the odd case is used!!

Timings are shown below for repeating `A(I) = B(I)` 5,000 times, where A and B are REAL arrays.

<u>Addressing for</u> <u>A & B</u>	<u>Time</u> <u>(ms)</u>	<u>Ratio to</u> <u>"standard"</u>
No MORECOM	56	1.00
MORECOM,offset even,>0	90	1.61
MORECOM,offset odd	154	2.75

With one-word variables and arrays (INTEGER*2, LOGICAL) and four-word variables and arrays (DOUBLE PRECISION), the parity of the offset is irrelevant; it just takes more code and more time when the MORECOM option is used.

3.2.4 When the use of the MORECOM option is unavoidable, considerable time and code space can be saved by ensuring that the offsets of REAL and INTEGER*4 variables are even. A straightforward way of doing this is to list all the REAL and INTEGER*4 variables and arrays first in the COMMON statement. Offset parity in existing COMMON blocks can be checked by perusing the output of the compiler MAP option.

3.2.5 A better solution would have been to allocate two DB-primary words per common block, one pointing to the 1st word in the block and the other to the 2nd word. Thus any offset in the block would be even with respect to one of the pointers. Unfortunately this would have restricted the maximum number of common blocks to only 127 instead of 254!

3.3 \$INTEGER*4

3.3.1 As the manual says, this option forces all integer variables and arrays (other than those explicitly declared INTEGER*2) and all integer constants to be INTEGER*4. It is likely to be used in the early stages of converting a FORTRAN program from another machine.

3.3.2 Use of this option can cause gross waste of code space and data space and considerable increase in execution time.

Consider the following example:

```
REAL A(10), T
T = 0.0
DO 100 I = 1, 10
100 T = T + A(I)
```

Without the use of the option, the variable I and the constants 1 and 10 are INTEGER*2, and the loop is controlled by the efficient MTBA instruction which increments I, tests it against 10, and branches back to the start of the loop, all in one hit.

When \$INTEGER*4 is in effect, I, l and 10 are INTEGER*4, and the loop is controlled by code which is slow for t o reasons:

(a) each function of the MTBA instruction has to be simulated separately

(b) LI arithmetic is slower than SI.

3.3.3 Once a converted program has been made to run correctly, the following steps should be taken:

(a) Remove the \$INTEGER*4

(b) Insert IMPLICIT INTEGER*4 (I-N)

(c) Determine which variables and arrays do not need to be INTEGER*4 and declare them explicitly as INTEGER*2.

(d) Determine which constants need to be INTEGER*4 and append the letter "J" to them.

3.4 Character Variables

3.4.1 The following declarations are used in the discussion:

```
CHARACTER A*80, B*1(80), C*(10), D*10
EQUIVALENCE (A, B, C), (B(11), D)
INTEGER*2 I, J, N
CHARACTER S*(N), T*1, U*1
```

3.4.2 The code generated for references to character variables of constant size (e.g. A), and for references to constant substrings (e.g. A [11:10]) is generally efficient. Some special cases are:

(a) Assignment of character variables of size 1 is done by the same sort of code as is used for wider variables.

E.g. T = U is done by

```
LOAD Q + 1
LOAD Q + 2
LDI 1
MVB 3
```

instead of

```
LDB Q + 2, I
STB Q + 1, I
```

and T = " " is done by

```
LOAD Q + 1
BR P + 2
%020000
LRA P - 1
LSL 1
LDI 1
MVB PB, 3
```

instead of

```
LDI %40
STB Q + 1, I
```

- (b) When a shorter string is assigned to a longer string (e.g. C = T), the balance of the longer string is blanked by calling the procedure BLANKFILL'. If one really needs the blanking done, an alternative is:

```
C [1:1] = T
C [2:9] = "      "
```

This will run faster, but takes more code, and if one counts too few blanks, BLANKFILL' will still be called!

- (c) When the position part of the substring is not 1, code must be emitted to generate the offset from the start of the variable. Thus it is faster to use D than A[11:10]. The offset is calculated once only (using rather cunning code) in the subprogram prologue; the trade-off is the extra word taken for a pointer to the start of D.

- 3.4.3 If the size of a string is variable (e.g. S), or variable substrings are used (e.g. A[I:J]), external procedures are used not only to perform the operation required, but also for paternalistic error checking which cannot be turned off.

Where only the position part of a substring is variable (e.g. A [I:10]), it may be possible to avoid the dreaded PCALs by equating to a character array. E.g. it is better to use B(I) than A[I:1].

- 3.4.4 When reference is made to a character array of more than one dimension, the element offset is calculated as described in Section 3.1.1; then this is multiplied by the element size to get the byte offset. (The multiplication uses slow unsigned arithmetic (LMPY, DELB), because the byte offset could exceed 32K). Thus offset calculation for a character array of n dimensions is as complicated as that for an integer (say) array of n + 1 dimensions. The exception is where the character element size is 1; multiplication by 1 is avoided.

3.5 Indirect Indirection

- 3.5.1 A Fortran main program or subprogram can run more slowly than expected if it has many local variables.

All local variables and arrays must be addressed relative to the Q register, and the direct range is +1 to +127.

Space in this range is allocated according to the following priorities:

- (1) One word as a pointer to each array, character variable or variable mentioned in a DATA statement (Compilation will fail if there are more than 127 words required.)
- (2) One word for each INTEGER*2 and LOGICAL variable.
- (3) One word as a pointer for each DOUBLE PRECISION variable.
- (4) Two words for each INTEGER*4 and REAL variable.

- 3.5.2 Once the total of the above allocations exceeds 127, one location (typically Q + 1) is allocated as a pointer to an "extension area". Then the remaining variables are addressed as though the extension area were an array and they were elements of the array. The offset into this pseudo-array is shown in the output from the compiler MAP option.

The effect on the execution speed is apparent from the code generated:

Variable J's address in the MAP is $Q + 23$.

The code required for $J = 0$ is

```
ZERO, NOP; STOR Q + 23.
```

Variables K's address in the MAP is $Q + 1, I, \%11$.

The code required for $K = 0$ is

```
ZERO, NOP; LDXI \%11; STOR Q + 1, I, X.
```

3.5.3 Fortunately the problem does not seem to be compounded by parity problems with REAL and INTEGER*4 variables (as it is with MORECOM); there is no language-imposed ordering requirement (as there is with variables in COMMON) and in observed cases the compiler allocates the two-word variables first, so that they are at even offsets in the pseudo-array.

3.5.4 Unfortunately when some variables will fit in the primary area and others would not, the compiler has no way of knowing which will be used more frequently than others, so it can happen that a variable used as a DO index can end up in the secondary area.

Several things can be done by the programmer to alleviate the problem:

- (1) Split the subprogram.
- (2) Use EQUIVALENCE to equate less frequently used variables to elements of arrays (one array for each data-type).
- (3) Put some variables into COMMON. If MORECOM is required, the less frequently used variables should be put into COMMON. While this is the easiest to write, it will typically waste stack space.

3.6 \$CONTROL BOUNDS

Use this option, if you must, while debugging, but be sure to remove it for production running.

This option generates a procedure call for each subscripted array reference.

3.7 The Formatter

- 3.7.1 We have the authority of Splinter [1] saying that the implementation of the Formatter is inefficient; on top of this it should be realised that each READ or WRITE statement involves an overhead of two procedure calls, together with one procedure call for each variable, each array, and each iteration of a DO-implied list.
- 3.7.2 Unformatted I/O merely avoids the conversion to external form; it does not avoid all those procedure calls. It is often worth the effort involved in using the file system intrinsics instead of unformatted I/O.
- 3.7.3 Further details on the diseconomy of using the Formatter are given by Green [4].

3.8 \$CONTROL INIT

- 3.8.1 Use of this compiler option causes all local variables and arrays to be cleared to zero during the subprogram prologue. The code used to do this is quite efficient: one block move is done to clear variables and arrays with fixed bounds, and another is done if there are any arrays with dynamic bounds, to clear them.

- 3.8.2 It follows that if any arrays, or more than a few variables, are to be cleared at the start of a subprogram, it is much better to use \$CONTROL INIT than to write explicit statements, especially for the arrays.

3.9 Use of SPL Routines

- 3.9.1 SPL allows access to all the features of the machine, and can thus be used to perform operations which can be expressed only clumsily, if at all, in FORTRAN. As access to an SPL routine from FORTRAN necessitates a procedure call, the time saved within the SPL routine needs to be worthwhile.
- 3.9.2 As recommended in [3], a frequent choice for an excursion into SPL is usage of the MOVE and MVB instructions for initializing or assigning arrays en masse. Appendix A shows an example of how to obtain leverage from the investment in a PCAL by allowing many arrays to be cleared at once.

4. MISCELLANEOUS

4.1 Slow programs: prevention

- 4.1.1 Ensure that the best data structures and algorithms have been chosen; it is pointless to "bit-twiddle" with the X register if you are bubble-sorting a 10,000 element array.
- 4.1.2 When writing the program, bear in mind the language features which can cause a problem, and avoid them in frequently executed code.
- 4.1.3 Draw a diagram showing which procedures call which other procedures, and arrange the segmentation to minimize calls which cross segment boundaries.

4.2 Slow programs: cure

- 4.2.1 Obtain a PMAP of the program and establish the reason for each external procedure reference. This should be easy for procedure names which do not contain an apostrophe: you explicitly coded the procedure call.

- 4.2.2 However procedures whose names contain an apostrophe are likely to be called implicitly by the compiler. The Compiler Library Manual will give you an idea of what the procedure is for. The PMAP will tell you one subprogram in each segment which is calling the procedure. If you still cannot match up the procedure name with the language feature it encodes, it is possible to decompile the calling segment, find the actual references, and tie these back to the source (via the PMAP and the LOCATION output of the compiler). Then, if desired, the source can be modified to avoid the procedure call.

- 4.2.3 If the problem cannot be traced to one or more external procedure calls, several options are open:

- (a) Review the segmentation
- (b) Read your source again carefully.
- (c) Ensure debugging statements are in-operative.
- (d) Re-run the program with calls to e.g. PROCTIME inserted at salient points.

4.3 Know your machine.

4.3.1 For high-level-language programmers interested in learning more about what happens behind the scenes, a starting point is to read sections of the System Reference Manual.

This will give an overview of how the machine works at the machine code level. The Machine Instruction Set Manual should be used as a reference for particular machine instructions.

4.3.2 Specific details as to the implementation of language features can be obtained by compiling sample source programs with the MAP option (and, for FORTRAN, the LOCATION option), prepping with the PMAP option, and then decompiling the object program. The programs DECOMP (in the Contributed Library) and EMDISASM (on the Orlando swap tape) may be used for this purpose.

The INNERLIST option in the SPL compiler is often useful; however as its output is produced before code generation is complete, the result can occasionally be misleading.

4.4 Future shock

4.4.1 As stated in section 1.3, this paper is based on observation of the behaviour of the Athena versions of the SPL and FORTRAN compilers. It is hoped that future versions of the compilers will generate better code. It is likely that when an enhancement is made, the code generated by the compiler for a particular language feature will be better than that generated for the "work-around" the programmer has used in the interim.

4.4.2 It may be found useful for the programmer to set up a jobstream to compile, prepare, and decompile a program (or suite of programs) which exercise the language features and their work-arounds. This jobstream may then be run after a software update and its output compared with previous results.

REFERENCES

- [1] E. Splinter: "Optimizing FORTRAN IV/3000" in HP3000 Users Group 1977 International Meeting Proceedings.
- [2] C. Morris: "FORTRAN Optimization" in Journal of the HP General Systems Users Group, Volume 1 No. 6 March/April 1978.
- (Not quoted in this paper).
- [3] Hewlett-Packard Company: "Program Optimization", Appendix F of "FORTRAN/3000 Reference Manual", Edition 1, Update 3.
- [4] R. M. Green: "HP3000 - Optimizing Batch Jobs" in Hewlett-Packard General Systems Users Group 1981 International Meeting (Orlando, Florida) - Proceedings.

APPENDIX A: CLEARMANY PROCEDUREA.1 Purpose

This procedure will clear one or more arrays with one call.

A.2 Calling sequence

Given that n arrays are to be cleared:

The $(2i-1)$ th argument is the address of the i th array.

The $(2i)$ th argument is the number of words to be cleared in the i th array.

The $(2n+1)$ th argument is n , the number of arrays.

Limits: $1 \leq n \leq 29$

Example:

INTEGER*2 SIA(100), SIB(M)

DOUBLE PRECISION LR(10,10)

...

CALL CLEARMANY (LR,400,SIB,M,SIA(51),50,3)

Warning:

The FORTRAN compiler will complain if there are two or more calls in one subprogram and the arguments do not agree in type and number.

A.3Source

```

Procedure clearmany (n);
  integer n;
begin
  integer fence, i;
  integer pointer arg, len, block;
  intrinsic quit;
  <<start of argument checking>>
  if not (1<=n <=29) then quit(1);
  push (Q);
  fence:= tos-5-2*n;
  @arg:= @n;
  for* i:= 1 until n do begin
    @arg:= @arg-2; @len:=arg(1);
    if logical(arg) > logical(fence) then quit (2);
    if logical(@len) > logical(fence) then quit (3);
    if not (1<= len <= (fence-arg-1)) then quit (4);
  end;
  <<end of argument checking>>

```

```

@arg:=@n;
for * i:= 1 until n do begin
  @arg:=@arg-2;
  @block:= arg;
  @len:= arg (1);
  block:=0;
  move block(1):=block,(len-1);
end;
tos:= %31400 + 2 * n + 1;
assemble (XEQ 0);
end;

```

A TRY TO ESTABLISH AN OFF-LINE TIME-REPORTING & WAGE
COMBINATION SYSTEM

W. G. HSIA

W.G. HSIA
WALSIN LIHWA ELECTRIC WIRE & CABLE CORPORATION
THE WALSIN BUILDING, 219 CHUNG HSIAO E. ROAD, SEC 4
TAIPEI TAIWAN R.O.C.

Speaker: W. G. Hsia

ABSTRACT (2)

TOPIC: A TRY TO ESTABLISH AN OFF -LINE TIME-REPORTING & WAGE
COMBINATION SYSTEM

THERE IS A FURTHER CHALLENGE FOR HP-3000 USERS TO TRY TO ESTABLISH
A TIME-REPORTING & WAGE COMBINATION SYSTEM AFTER THEY HAVE SET UP
THEIR WAGE/SALARY PROCESSING SYSTEM. THIS PAPER SHOWS SOME DEVELOP-
ING IDEA OF IMPLEMENTING IT AT WALSIN LIHWA CABLE CO. LTD., ALTHOUGH
WE ARE STILL UNDER WAY, SOME EXPERIMENTAL RESULTS REVEALS THAT IT IS
A FEASIBLE WAY. THIS SYSTEM USES A LOW-COST OFF-LINE TIME-REPORT
TERMINAL (BCD CODE), A MINI-TAPE AUXILIARY RECORDER, AND SOME GATE
LOGIC FOR OPERATING SUPERVISION. A SOFTWARE DRIVER WRITTEN IN
FORTRAN LANGUAGE IS USED TO INTERFACE THE MINI-TAPE AND HP-3000
SYSTEM

WE HOPE YOU CAN GIVE US SOME NEW IDEAS AND SHARE KNOWLEDGE WITH
US THROUGH THIS MEETING.

WALSIN LIHWA ELECTRIC WIRE & CABLE CORPORATION
The WALSIN Building, 219 Chung Hsiao E. Road, Sec. 4 Taipei, Taiwan R.O.C.
Tel: 771-2121 (20 Lines) P.O. BOX 22926 Telex: 11516 WALSIN Taipei
Cable: WALSIN Taipei

NEWS TO MPE IV INTERNALS

U. JENSEN

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

U. JENSEN

HEWLETT PACKARD

Presentation Abstract

Presentation Title: A Few Well-Chosen Words Concerning A Few Chosen Ways
to do Word Processing, Some Well-Chosen, Some Not

Author(s): Wirt Atmar

Title(s): President, AICS

Address: P.O. Box 4691

University Park, NM 88003 USA

Abstract: (No more than 200 words)

Simply because you are intimately involved with HP 3000 computers,
you are also intimately involved in an environment where word-processing
is going to become an ever more pressing demand. It would seem to be no
overstatement to say that every HP 3000 is likely to possess some capa-
bility to perform some aspect of word processing in the next five years.

The major computer compatible techniques are to be reviewed and
their merits assessed and their limits discussed. Quite plainly, due to
the author's position as a producer of word processing equipment, biases,
both subtle and blatant, will inevitably interweave these few chosen
words of the presentation. The listener is encouraged to be critical.

A FEW WELL-CHOSEN WORDS CONCERNING A FEW CHOSEN WAYS

TO DO WORD PROCESSING, SOME WELL-CHOSEN, SOME NOT

WIRT ATMAR

W. ATMAR
AICS
P.O. Box 4691
UNIVERSITY PARK, NM 88003
USA

***** STILL WAITING
FOR FULL TEXT ..
(EDITOR) ***



INTRODUCTION

Motivation

The "software crisis", which has been generally recognized in the last ten years, has created the need for tools that allow programmers and users of computers to be more productive. The traditional tools (compilers, editors, file systems, etc.) are not adequate to keep pace with the growing power of computers and the expectations of those who use and pay for them.

Consequently, the past few years have seen a multitude of products introduced which claim to improve programmer productivity or, in a few instances, eliminate the need for programmers, or at least coders. Specifically, in the HP/3000 product line, Image, Query, DEL, KSAM, and V/3000 have been introduced by HP. Outside vendors have added to this list with products which, while frequently improving on the HP products, are more imitative than innovative. For example, there are several "Query like" products available from independent vendors which extend the functions of Query and remedy several of its obvious deficiencies, but do not offer a fundamentally different kind of tool.

More recently, several innovative tools have appeared on the market. Among these are two relational database management systems, Relate/3000¹ and Rel*Stor. The innovative aspect of these products is that they are based on the relational model rather than the network model of Image.

Objectives

The purpose of this paper, and the study on which it is based, is to compare these products with Image both in concept and implementation to determine the strengths and weaknesses of each. The goal is to select one of the three as the basis of further development of software tools.

The authors do not presuppose that one of these products will be clearly superior to the others or that one would be the best choice under all circumstances. However, this study should serve as the basis for a rational decision.

Scope

To do a thorough analysis of these products, one should probably use each for a year or more in a variety of applications. Since this is not feasible, the authors have elected to evaluate them on the basis of:

1. The published specifications and user manuals;
2. The mapping of a small, but demanding database onto each system;
3. Performance on the HP/3000 as indicated by carefully chosen tests.

A Comparison
of
Relational and Network
Data Base Management Systems
as Implemented on
the
HP/3000

by
Thomas R. Harbron
and
Christopher M. Funk

July 1981

Christopher M. Funk & Co., Inc.
22 North Second Street, P.O. Box 1249
Lafayette, IN 47902
(317) 423-2644

This analysis is further complicated because:

1. Both Rel*Stor and Relate/3000 are still under development with modules and features not yet implemented;
2. Their manuals are likewise under development, and not always in step with the product;
3. One product (Rel*Stor) was not made available for testing.

Therefore, the reader should be cautioned not to accept this study as the last word on these products.

BACKGROUND

Database Models

Most authors²⁻⁷ list three different models for databases. These are idealized models of how data is naturally structured and do not consider questions of implementation or efficiency. Rather, the models are based on mathematical principles.

These three models are known as the network model, the hierarchical model, and the relational model. Virtually all database systems are based on one of these three models. Moreover, an important step in designing a specific database is modeling it in one of these three forms.

Each form has its own peculiar strengths and weaknesses. These are discussed briefly below. One problem found in discussing models and database systems is that each, generally, has a unique vocabulary. This is confusing enough when considering them one-at-a-time. When three models and three systems are discussed in one paper, it is hopeless. Therefore a "generic" vocabulary will be employed here as listed below. The authors apologize to those who may find these terms imprecise or contrary to standard usage:

- Entity - an object or "thing" about which information is stored in a database.
- Attribute - a characteristic of an entity. Only attributes of an entity can be stored, not the entity itself.
- Field - the physical representation of an attribute.
- Record - the physical representation of an entity consisting of the fields that hold the attributes of that entity.
- Key - a set of attributes that distinguishes one entity from other similar entities.

File - the physical representation of a group of similar entities consisting of the records representing those entities.

Relationship - a logical connection between entities. For example, between a parent and children, or between a vendor and purchase orders to that vendor.

The adjective "logical" will be applied to the terms field, record, key, and file when discussing the corresponding parts of the models.

A "standard" database problem will be used as an example throughout the remainder of this paper. This problem is a simplified accounting system. The entities and their attributes are as follows:

Entity	Attributes
Department	Dept # - a unique number assigned to each department. Dept name - the common name of the department. Dept head - the name of the manager of the department.
Expense	Exp # - a unique number assigned to each expense type. Expense description - a description of the expense type.
Account	Account # - a unique number assigned to each account, consisting of a dept # concatenated with an expense number. Budget amount - the dollar amount budgeted for this account. YTD credit amount - the total dollar amount of all transactions credited to this account. YTD debit amount - the total dollar amount of all transactions debited to this account.
Transactions	CR Account # - the number of the account to which this transaction is credited. DB Account # - the number of the account to which this transaction is debited. Amount - the dollar amount of the transaction. Date - the date of the transaction. Reference - the account reference of the transaction.

Network Model

The network model is characterized by logical files, each of which represents an entity type. The logical files are connected by relationships that show how entities in one logical file are related to entities in other logical files.

The relationships are usually restricted to one-to-N or 1:N types. This means that exactly one entity in one logical file is related to N (zero or more) entities in another logical file. This is customarily noted by an arrow pointing from the "1" entity to the "N" entity. For example, a department entity may be related to many accounts while an account entity must be related to exactly one department.

More than one relationship may exist between two entities. For example, each transaction is related to exactly one account as a "credit account" and to exactly one account as a "debit account." This is done as two 1:N relationships from account to transaction.

A convenient way to represent a network model is by a "data structure diagram." Such a diagram is shown in Fig. 1 for the accounting problem. Note that entities are usually linked together by a shared attribute value. The name of the shared attribute is shown on the arrow in the diagram. The underlined attributes are keys.

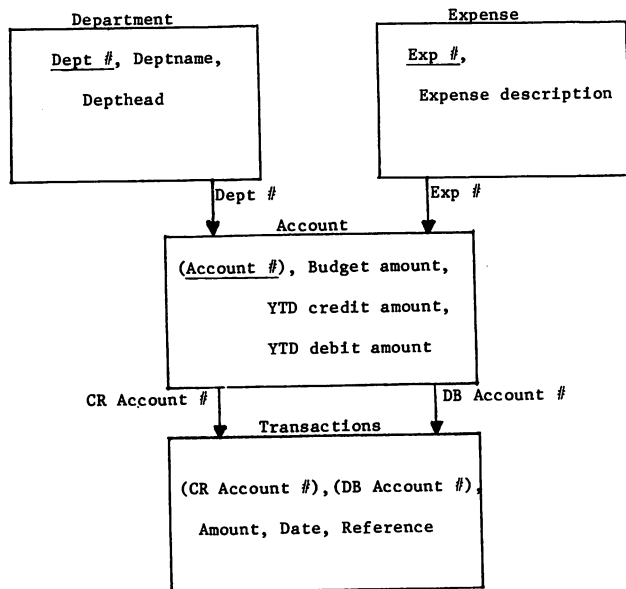


Figure 1

Notice that some of the attributes appear in parentheses. These are the same attributes that are used for the relationship linkage. Thus, it is redundant to show them as attributes; however, this is done in parentheses for logical completeness.

The network model is probably the most general of the three models. The network formed by the relationships can take on any topography and the links show the entity relationships. The other models are more restrictive.

Hierarchical Model

The hierarchical model is, structurally, a subset of the network model; i.e. any hierarchical structure can be built under the rules of the network model. The difference is that additional constraints are imposed on the hierarchical model. These have to do with the relationships between entities and are as follows:

1. There is a unique entity type called the "root" where the hierarchical network begins.
2. Each entity, except those in the root, has exactly one "parent." A parent is another entity of a different type at a higher level.
3. Each entity, except those at the lowest level of the hierarchy, may have multiple "children." A child is another entity of a different type at a lower level.

The account example does not map easily into the hierarchical model because both the "account" and "transaction" entities have multiple parents. A better example is the bill of materials problem shown in Figure 2:

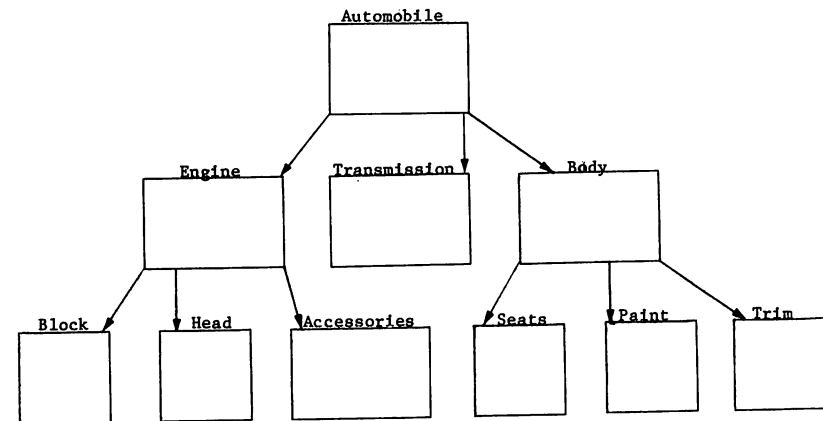


Figure 2

The hierarchical model is rightfully popular in situations where the data is naturally hierarchical. Otherwise, most of its usage seems to result from the dominance of several early database management systems based on this model. Many problems, including our elementary accounting example, would require unnatural restructuring to fit this model.

Relational Model

While the network and hierarchical models are similar to each other, the relational model is totally different from them. The database consists of multiple logical files (called relations). Each logical file has one or more keys by which the logical records may be retrieved. There are no entity relationships of any kind connecting the logical files. The entity relationships can only be determined by comparing values of attributes of different entities.⁸

A simple list of the attributes of each logical record type, with an indication of the keys is sufficient to describe a model. For example, the following describes the relational model of the accounting problem:

Logical File	Attributes
Department	<u>Dept #</u> , Deptname, Depthead
Expense	<u>Exp #</u> , Expdesc
Account	<u>Acct #</u> , Budamt, YTDCCR, YTDDB
Transaction	<u>CRAcct #</u> , <u>DBAcct #</u> , Amt, Date, Ref

Notice that transaction has two keys. Multiple keys are allowed in the relational model.

This model is unquestionably the simplest in appearance, and that is probably its greatest strength. It also has a firm mathematical foundation. Some authors⁹ regard it as the most fundamental of the three models. However, it is probably the least implemented model because of two problems.

The first is a flaw in the model - the lack of explicit entity relationships. This can lead to what are called "insertion anomalies" and "deletion anomalies." For example, if a transaction is inserted, in our accounting problem, for which no credit account exists in the account logical file, the model will accept it. Likewise, a department could be deleted, thus "orphaning" the accounts associated with that department. Thus the rules necessary to avoid these anomalies must be imposed externally to the model.

The second problem is not so much with the model as with the implementations. The model makes it easy to request operations which are logically simple, but which require considerable resources and time to execute. Thus relational systems have earned a reputation for inefficiency.

Normalization

This is one of the most important and poorly understood steps in designing a database. Normalization is essentially the process of discovering and isolating the entities represented by the data. There are three levels of normalization and the topic is discussed by most authors²⁻⁷ with varying degrees of clarity. Atre² presents an unusually lucid discussion of normalization.

Normalization is nearly always presented in mathematical terms which, unfortunately, discourages some from investigating it further. A complete discussion is beyond the scope of this paper. However, normalized data will have the following advantages over intuitively designed, or unnormalized data:

1. Numerous types of insertion and deletion anomalies will not occur. These anomalies are of the type where the insertion or deletion of one entity has an unexpected or undesirable effect on another entity.
2. All entities will be readily accessible. Functions thought of after the database is designed will not require restructuring of the database.
3. A higher degree of data independence is possible. Programs are less likely to need change as the database is changed.

Neither the models nor the database systems have any way to enforce normalization. However, failure to normalize the data will inevitably create serious problems.

Mapping

Mapping is a series of transformations on the structure of the database from its inception to the final, physical, database. There are five states in which the data is structured:

1. Initial data description
2. Normalized data description
3. The database model
4. The schema
5. The database

The mapping from the initial form to the normalized form is called "normalization" as described earlier. Normalization actually includes three separate transformations.

The mapping from the normalized form to the model is frequently accompanied by some compromises. If, for example, a hierarchical model is used,

and the data is not inherently hierarchical, an artificial constraint is placed on the structure. Likewise, it may be necessary to "unnormailize" the data to some degree to fit the model.

Mapping from the model to the schema is really two activities that are done in parallel. First, the model must be mapped to the actual database systems. Some systems will be very close to the model and present little difficulty. Others may impose either structural or efficiency constraints which cause the structure to be altered significantly from that of the model. For example, the two-level limitation of Image requires compromises from the network model.

Second, the data structure must be expressed in a form acceptable to the database system. The form is called a "Data Description Language" or DDL. All database systems have a DDL. Some have a formal syntax, such as Image, while others may be conversational, such as Relate/3000. The data structure description expressed in a DDL is called a "schema."

The final transformation, of the schema into a database, is done by the database system. In most systems it is automatic with feedback in the form of error messages, status reports, and statistics.

Implementation Considerations

The following items are factors to consider in judging the merit of a particular database system. Until the perfect system is developed, some will always be better than others on specific points. Different users will weigh these factors differently. However, all should be considered before a selection is made.

1. Mapping: What constraints are imposed when mapping from the model to the schema? Do significant changes have to be made? Are some things allowed, but not done because of performance considerations?
2. Data Manipulation Language (DML): The DML is the form in which requests are transmitted to the database system. Is the DML powerful? Is it easy to understand? Is it flexible? Can it be used from an application program? Is there a "stand-alone" mode?
3. Performance:
 - a) Run efficiency: Are efficient search algorithms used? Is response time good? Are (logically) unnecessary accesses to secondary storage required?
 - b) Storage efficiency: Is most storage space used for data? Do indexes, pointers, or other "non-data" items use up excessive space?
4. Concurrency: Has adequate thought been given to the problem of multiple users updating the database? What penalties or complications arise from shared access?

5. Restructuring: What needs to be done to change the database structure? What resources are required? What effect does restructuring have on existing applications? What must be done to initially load the database?
6. Security: How well protected is the data from unauthorized access? At what level or levels is security imposed: database, file, record, or item?
7. Integrity: Is the database prone to develop internal inconsistencies (broken chains, missing records, etc.)? Do aborts or crashes cause problems? What provisions are there for checkpointing (back-up copies) and journaling (transaction logging) and recovery?
8. Data Independence: Are application programs isolated from the physical storage considerations? Can changes be made in the physical or logical structure of the database without changing existing programs?

These implementation considerations, together with the strengths and weaknesses of the model on which it is based, will be used to judge each of the systems considered in the next section.

Three Implementations

This section of the paper will consider three database systems: Image, Relate/3000, and Rel*Stor. For each of these systems, the strengths and weaknesses of the model and the implementation considerations will be discussed. Finally vendor information will be provided.

Image/Query

Image¹⁰ is based on the network model. As such it enjoys the benefits of explicit entity relationships of the 1:N variety, and even extends the concept by allowing the N entities to be ordered by the value of an attribute of that entity.

The DDL uses a formal, but concise, syntax. The mapping is straightforward with one glaring exception: a logical file (called a "set" in Image) cannot both be on the "1" side of some entity relationships and on the "N" side of others. This limits the system, physically, to two levels.

The problem is caused by the distinction between two kinds of files: masters and details. Masters are direct access files where a record is located by hashing on a single key. The hashing algorithms are effectively implemented and work with good efficiency. Detail files are essentially sequential-chronological files. Entity relationships are implemented using pointers to form a linked list or "chain" linking all related records. Each chain starts and ends on one record in a master set. The chain links any number of records (64K maximum) in a detail set. One master record may originate up to 16 different chains. One detail record may be linked

into as many as 16 different chains. Detail records are normally accessed by following a chain from a master record. Sequential access is possible for both master and detail records.

The two-level structure causes difficulties, and requires compromises when mapping from model to schema. For example, the following data structure diagram represents the Image implementation of the accounting problem. Trapezoids are used to represent master sets while rectangles represent detail sets. Chains are represented by solid arrows while logical relationships (implemented programatically) are shown by broken arrows.

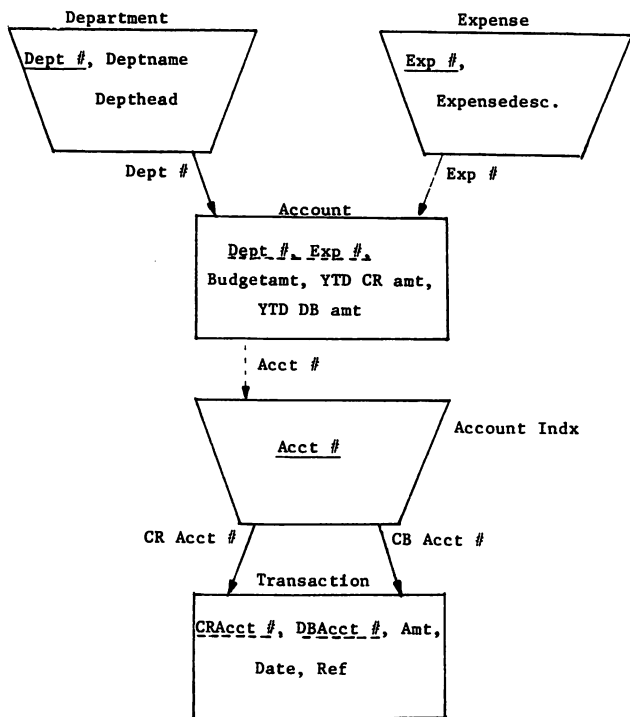


Figure 3

The broken underlines indicate that the underlined data item is a key only via the associated master. Image, however, requires that such fields be physically present in spite of the logical redundancy.

The ACCOUNT and ACCOUNTINDX sets are logically the same, but two are

required to circumvent the two-level problem. This requires redundant storage and additional access to secondary storage.

The schema for this data base is as follows:

```

BEGIN DATA BASE ACCTDB;
PASSWORDS: <<NONE>>
ITEMS:
  DEPT,      X4; <<DEPT #>>
  EXP,       X4; <<EXP #>>
  DEPTNAME,  X20; <<DEPARTMENT NAME>>
  DEPTHED,   X20; <<DEPT HEAD'S NAME>>
  EXPDESC,   X20; <<EXPENSE DESCRIPTION>>
  BUDAMT,    I2; <<BUDGET AMOUNT>>
  YTD CR,    I2; <<YEAR-TO-DATE CREDIT TOTAL>>
  YTD DB,    I2; <<YEAR-TO-DATE DEBIT TOTAL>>
  ACCT,      X8; <<ACCT # = DEPT#EXP#>>
  ACCT CR,   X8; <<CREDIT ACCT #>>
  ACCT DB,   X8; <<DEBIT ACCT #>>
  AMT,       I2; <<TRANSACTION AMOUNT>>
  DATE,      I2; <<TRANSACTION DATE>>
  REF,       X6; <<ACCOUNTING REFERENCE>>

SETS:
NAME: DEPARTMENT, MASTER;
ENTRY:  DEPT(1),
        DEPTNAME,
        DEPTHED;
CAPACITY: 23;

NAME: EXPENSE, MASTER;
ENTRY:  EXP(1),
        EXPDESC;
CAPACITY: 23;

NAME: ACCOUNT, DETAIL;
ENTRY:  DEPT (DEPARTMENT),
        EXP (EXPENSE),
        BUDAMT,
        YTD CR,
        YTD DB;
CAPACITY: 100;

NAME: ACCOUNTINDX, MASTER; <<LOGICALLY PART OF 'ACCOUNT'>>
ENTRY:  ACCT(2);
CAPACITY: 101;

NAME: TRANSACTION, DETAIL;
ENTRY:  ACCT CR (ACCOUNTINDX), <<CREDIT ACCT #>>
        ACCT DB (ACCOUNTINDX), <<DEBIT ACCT #>>
        AMT,
        DATE,
        REF;
CAPACITY: 6000;
  
```

END.

A variety of data types may be defined in the DDL. These are:

- 16 bit signed integer
- 32 bit signed integer
- 64 bit signed integer
- 16 bit unsigned integer
- 32 bit floating point
- 64 bit floating point
- Character string
- Zoned decimal
- Packed decimal

No provision is made to add user defined data types.

Image's DML consists of procedure calls which are compatible with all the standard languages. A summary of the procedures and their functions appears in Figure 4. An application program called "Query" is supplied with Image.¹¹ This program can be used interactively to access a database. It also serves as a report generator. Its usefulness is limited by its inability to look at more than one file at a time. However, it works very well otherwise and is simple enough to be used by non-programmers. Manipulations such as sorts and totals may be specified.

The run efficiency of Image is generally good. Performance problems are usually the result of design errors. For example, adding a detail record to a long ordered chain requires a sequential search of the chain. If there are 1000 detail records on the chain, 500 of them will (on the average) have to be read to determine the logical placement of the new record. Normally this would require 500 disk accesses! Thus, long ordered chains should be avoided.

Another source of performance problems can be record-level locking. Image uses dynamic locking to handle the concurrency. The locking may be done at the database level, file level, or record level. The lower the level, the greater the complexity¹² and the greater the overhead involved. The overhead at the database level is negligible; at the record level it is considerable.

Much of the time, blocking of records does not help reduce disk accesses. There is a provision to store detail records, that share a chain, in the order in which they occur on the chain. This physical ordering can only be done as part of restructuring and is not dynamically maintained. Thus there is usually a requirement for one physical access for each logical access.

Insertions and deletions require considerably more than one access. To insert a detail requires a minimum of four accesses for each chain involved as well as the write to put the record out. A deletion will usually require six accesses per chain.

The designer can consider these factors in mapping the schema and the resulting Image implementations can be as efficient as corresponding non-database applications.

PROCEDURE	FUNCTION
DBOPEN	Initiates access to a data base. Sets up user's access mode and user class number for the duration of the process.
DBLOCK	Locks one or more data entries, a data set, or an entire data base (or a combination of these) temporarily to allow the process calling the procedure to have exclusive access to the locked entities.
DBFIND	Locates the first and last entries of a data chain in preparation for access to entries in the chain.
DBGET	Reads the data items of a specified entry.
DBBEGIN	When logging, designates the beginning of a transaction and optionally writes user information to the logfile.
DBMEMO	When logging, writes user information to the logfile.
DBPUT	Add new entries to a data set.
DBUPDATE	Updates or modifies the values of data items that are not search or sort items.
DBDELETE	Deletes existing entries from a data set.
DBEND	When logging, designates the end of a transaction and optionally writes user information to the logfile.
DBUNLOCK	Releases those locks obtained with previous calls to DBLOCK.
DBCLOSE	Terminates access to a data base or a data set, or resets the pointers of a data set to their original state.
DBINFO	Provides information about the data base being accessed, such as the name and description of a data item.
DBEXPLAIN	Examines status information returned by an IMAGE procedure that has been called and prints a multi-line message on the \$STDLIST device.
DBERROR	Supplies an English language message that interprets the status information set by any callable IMAGE procedure. The message is returned to the calling program in a buffer.
DBCONTROL	Allows program operating in exclusive mode to enable or disable the "deferred update" option.

Figure 4

The storage efficiency of Image is generally good, but again there are exceptions. Each chain requires 10 bytes for chain information in each master record, and 8 bytes in each detail record. Where several chains are involved this can become significant. Where the amount of data per record is small and the number of chains is high, the total storage required can be several times that required by the data. However, this is not typical. A modest "root file" is required to contain the schema and statistical information, but this is negligible in size.

As noted above, concurrency is handled by dynamic locking at several different levels. As with any dynamic locking situation, care must be exercised to avoid lockouts or deadlocks. Experience has shown that a dozen or so interactive users can share access to a database locking at the database level without serious contention problems. More can probably be accommodated with lower levels of locking.

Restructuring of an Image database is awkward at best and nearly impossible at worst. Essentially, fields may be added to records, and existing fields may be redefined. The maximum capacity of files may be changed, and chains may be added or deleted. Sometimes, but not always, new files may be added.

Restructuring is done with the utilities DBUNLOAD and DBLOAD. DBUNLOAD dumps records from the old database to magnetic tape. The old database is then purged and the new one is created from the revised schema. DBLOAD then loads the data from the tape to the new database. All pointers and chains are built anew by DBLOAD and the process can be very slow. The new database must be very similar to the old as no structure information is carried on the tape.

Security is very good with Image. In addition to the MPE file security, Image has an internal security mechanism that is very flexible. Up to 63 user classes, with associated passwords, may be defined. For each file and/or field, it is possible to specify which classes are allowed read access and which are allowed read/write access. All other user classes have no access. Image files are "privileged files" and cannot be accessed except through Image or by a privileged mode user.

The integrity of Image is unusually high. Crashes seem to cause a problem only when caused by a catastrophic hardware failure. Even then the problem can usually be fixed by deleting and replacing the record(s) involved. At worst a DBUNLOAD/DBLOAD will repair all structural damage.

Checkpointing can be done easily by using the utilities DBSTORE/DBRESTOR. These dump the files, with pointers, to tape and from tape to disk. Since no restructuring is done, they are very fast and efficient.

Journaling may be done with the transaction logging feature of Image. A utility is available to process the logged transactions to a checkpointed version of the database to recover all processing.

The degree of data independence can vary widely depending on the application programs themselves. At the low end of the spectrum, a program can, on a DBGET, request a physical record. At the other end, it can

request the specific fields wanted and the order in which they are delivered. A useful option is that of asking for the same list of variables used on the previous access of that file. This permits a logical "view" to be defined by the initial access(es). Thereafter the program sees this same view.

Image is structured as a set of user-callable procedures and several utilities plus Query. The utilities are only needed to restructure or recover the database and are not used for routine functions. All procedures are part of the application program's process. However, an extra data segment is created for each database that each process has open. In addition, all processes using a database share an extra data segment that serves as a common buffer and locking mechanism.

Image is a well established product and is nearly error free. It is available from:

Hewlett-Packard Co.
19447 Pruneridge Avenue
Cupertino, CA 95014

Relate/3000

Relate is based on the relational model. Thus it enjoys the benefits of simplicity at the expense of losing the explicit entity relationships found in the other models. It is an unusually faithful implementation, with all standard features.

The DDL is conversational and informal. No "database" per se is defined. However, files (called relations in this model) are defined along with the name, and internal and external description of each field. These descriptions are stored in the "user label" area of each file. Thus the files are independent of one another. A database consists of those files a user has open at any given time.

Most relational systems provide for two types of relations or logical files. A "primary relation" is a permanent part of the database and is usually implemented as a physical file. A "derived relation" is one created during the run of an application and is usually not permanent. These derived relations are of two kinds: a "snapshot" is usually created by copying data from a primary relation to a new file. Thereafter it is independent of the original data. An "evolving view" is a rule that says how the derived relation is formed from the primary relations. The data remains in the primary relation.

Relate allows snapshots to be created at any time. In addition, evolving views may be created in two ways. A temporary, core resident view may be specified with the SELECT command. A permanent evolving view may be specified by the CREATE VIEW command. These are stored in separate, short files which contain the definition of the view, but no data.

A file (or relation) is created by the CREATE FILE command while keys are specified by the CREATE INDEX command. Examples of these commands are perhaps the most concise way to describe the DDL. The accounting problem will be used in these examples. Note that file names are limited to seven characters. Comments are enclosed in brackets { } but are not part of the session dialog; computer output is underlined:

```
{Create the department file}
> CREATE FILE DEPART; RECORDS=23
ENTER FIELD NAME, TYPE, LENGTH{.DECIMALS}
? DEPT, ALPHA, 4
? DEPTNAME, ALPHA, 20
? DEPTHREAD, ALPHA, 20
? //
THE "DEPART" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
```

```
{Create the expense file. Here the description for each field is contained
in the CREATE command}
> CREATE FILE EXPENSE; RECORDS=23; FIELDS=(EXP, ALPHA, 4) (EXPDESC, ALPHA, 20)
THE "EXPENSE" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
```

```
{A command may extend over multiple lines as follows}
> CREATE FILE ACCOUNT; RECORDS=1000; FIELDS=&
&> (DEPT, ALPHA, 4), &
&> (EXP, ALPHA, 4), &
&> (BUDAMT, DOUBLE, 13; COMMA=YES), &
&> (YTDCR, DOUBLE, 13; COMMA=YES), &
&> (YTDDDB, DOUBLE, 13; COMMA=YES)
THE "ACCOUNT" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
```

```
{Finally the transaction file is created}
> CREATE FILE TRANS; RECORDS=6000
ENTER FIELD NAME, TYPE, LENGTH{.DECIMALS}
? DEPTCR, ALPHA, 4
? EXPCCR, ALPHA, 4
? DEPTDB, ALPHA, 4
? EXPDB, ALPHA, 4
? AMT, DOUBLE, 13, COMMA=YES
? DATE, REAL, 8; FORMAT="MM/DD/YY"
? REF, ALPHA, 6
? //
THE "TRANS" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
```

```
{Next the keys are defined with the CREATE INDEX command. The SET PATH
command defines the "current" file for indexing}
> SET PATH DEPART
> CREATE INDEX BY DEPT; UNARY
{The "unary" specifies that keys must be unique}
> SET PATH EXP
> CREATE INDEX BY EXP; UNARY
{The following index contains one key formed by concatenating two fields}
> SET PATH ACCOUNT
> CREATE INDEX BY DEPT, EXP; UNARY
```

{The following indexes each have two keys, each of which is the concatenation of two items; neither key must be unique}

```
> SET PATH TRANS
> CREATE INDEX BY DEPTCR, EXPCCR
> CREATE INDEX BY DEPTDB, EXPDB
```

Each data file has one index file associated with it. Up to nine indexes may be defined for each data file. All indexes for one data file are stored in the one index file. The indexes are structured as "B-trees."¹⁴ The B-tree is a tree structure which neatly solves the problems of making additions and deletions to the index, and is very efficient for retrieval. Appendix B of the KSAM manual¹⁵ has a good presentation on B-tree indexes.

Eight different data types may be specified for the fields. These are:

Character String
16 bit unsigned integer
16 bit signed integer
32 bit signed integer
32 bit floating point
64 bit floating point
Packed decimal
Zoned decimal

There is currently no provision for user defined data types, but this feature is under consideration.

An external format is also specified which has several options and some nice features. It is not as flexible as the PICTURE clause of COBOL or the FORMAT statement of FORTRAN, but better than the facilities found in Query.

The DML consists of two parts: commands and procedure calls. By far the greatest power and flexibility is in the commands. The procedures provide a better interface for application programs, in some cases, and somewhat more flexibility. Both commands and procedures may be accessed from an application program. The commands can also be used with Relate running as an interactive program as in the examples above.

The diagram in Figure 5 illustrates the program structure of Relate. The program Relate is the workhorse of the system. It is all that is needed when Relate is run as an independent program. When Relate is used from an application program, the "host language interface" library procedures are called by the program. These procedures, in turn, create a son process which runs the Relate program. All calls to these procedures are passed to the son process (Relate) for execution.

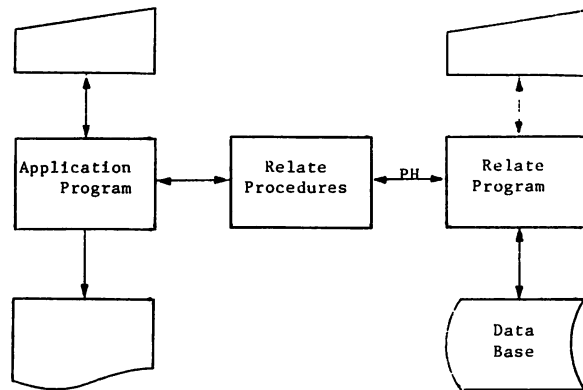


Figure 5

The following table lists the Relate commands with a brief description of the function of each. These commands may be used in the stand-alone mode or from an application program.

Command	Function
ADD	Adds a record to the current file.
ALLOW	Sets the capabilities of different users.
CHANGE*	Modifies record(s) in a file.
CLOSE	Closes files or databases.
COMPARE	Compares contents of two files and selects either matching or unmatched records.
CONSOLIDATE*	Creates a subset of the current file.
COPY*	Copies the current file to another.
CREATE FILE	Creates a Relate/3000 file.
CREATE INDEX	Creates an index for the current file.
CREATE VIEW	Creates an "evolving view" and stores its description in a file.
DELETE*	Purges selected records from current file.
DISABLE SECURITY	Releases Relate security.

Command	Function					
DISALLOW	Inverse function of ALLOW.					
ENABLE SECURITY	Turns on Relate security.					
END	Terminates Relate program.					
EXIT	Terminates Relate program.					
EXECUTE	Causes Relate commands in a file to be executed.					
HELP	Displays information about commands.					
LABEL*	Prints records in label format.					
LET*	Makes arithmetic or alphabetic assignments.					
MODIFY	Changes the field formats or descriptions for the current file.					
NOTE	The note command begins a comment line.					
OPEN	<table border="0"> <tr> <td style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;">Database</td> <td rowspan="3" style="font-size: 2em; vertical-align: middle;">}</td> <td rowspan="3">Opens the named database (Image) or file. "Path" is an alternate name for a file.</td> </tr> <tr> <td style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;">Path</td> </tr> <tr> <td style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;">File</td> </tr> </table>	Database	}	Opens the named database (Image) or file. "Path" is an alternate name for a file.	Path	File
Database	}	Opens the named database (Image) or file. "Path" is an alternate name for a file.				
Path						
File						
PRINT*	Displays selected data from current file on \$STDLIST.					
PURGE INDEX	Purges an index from the current file.					
PURGE VIEW	Purges the named view.					
RECOVER*	Restores records that have been logically, but not physically deleted.					
REDO	One line edit function for previous command.					
REORGANIZE	Physically removes logically deleted records and, optionally, changes file capacity.					
SELECT*	Creates an "evolving view" and holds it in main memory for use by subsequent command(s).					
SET INDEX	Specifies which index is to be used.					
SET PATH	Specifies which file is to be used.					
SHOW	Displays information about open files and indexes.					
SORT*	Copies selected records from the current file to another file in order specified by sort key(s).					
SUM*	Totals one or more fields in selected records.					

Command	Function
SYSTEM	Sets global parameters.
TERMINAL	Specifies terminal characteristics.
UPDATE	Merges selected files.

Those commands marked with an asterisk in the table above allow subsets of the records in the current file to be selected in either or both of two different ways. The first is by a range or ranges of values of fields that are indexed. For example:

```
> SET FILE PEOPLE
> SET INDEX NAME
> "A"/"G" PRINT
```

will print the records for all people whose names are in the range A-G.

The selection may be further qualified by a "FOR" clause. For example:

```
> "A"/"G" PRINT FOR DEPT = "SALES" AND SALARY > 250000
```

will print the records for all people whose names are in the range A-G, who work in the sales department, and whose salary is greater than 250000. Both ranges and FOR conditions may be compounded.

Again referring to the accounting problem the following dialog illustrates the power of some commands. As before, comments are in brackets, computer output is underlined.

{Open department file and add entries to it}

```
> OPEN FILE DEPART
> ADD
ENTER DEPT, DEPTNAME, DEPTH
```

```
DEPT? {Here the one field, or all three fields, may be entered. If
fields have been specified in a hierarchical form, only fields
that differ from record-to-record need be entered.}
```

```
DEPT? // {Returns control to command interpreter.}
```

{Next, a SELECT command is used to define an evolving view that combines data from two different files. The COPY command then copies this view to another file.}

```
> SELECT DEPART.DEPT,EXPENSE.EXP
```

{This will select the dept # from the department file and the exp # from the expense file. There are 8 records in the department file and 9 in the expense file. A total of 8x9=72 combinations are possible, and that many records will be copied by the next command.}

```
> COPY TO ACCOUNT
```

{We now have 72 accounts in the ACCOUNT file. We would next like to create one transaction for each combination of credit acct # and debit acct # except that the same account number may not appear in both places. This will give 72x72-72=5112 transactions. There may be more elegant

ways to do this, but the following sequence worked. It was necessary to create a temporary file similar to ACCOUNT but with different field names.}

```
> CREATE FILE TEMP; STRUCTURE=ACCOUNT
{This creates a file of the same size and with the same field definitions
as ACCOUNT.}
> SET PATH ACCOUNT
> COPY TO TEMP
{TEMP now contains the same data as ACCOUNT.}
> SET PATH TEMP
> MODIFY DEPT; NAME=DEPTDB
> MODIFY EXP; NAME=EXPDB
{The field names in TEMP have been renamed.}
> SET PATH ACCOUNT
> MODIFY DEPT; NAME=DEPTCR
> MODIFY EXP; NAME=EXPCR
{The field names in ACCOUNT have been temporarily renamed. Names of fields
in ACCOUNT and TEMP now correspond to those in TRANS. Next, an evolving
view will be defined as the product of account numbers in these two files,
excluding cases where the two account numbers are identical. These 5112
transactions will then be created with the copy command.}
> SELECT ACCOUNT.DEPTCR,ACCOUNT.EXPCR,TEMP.DEPTDB,TEMP.EXPDB WHERE&
&> ACCOUNT.DEPTCR <> TEMP.DEPTDB OR ACCOUNT.EXPCR <> TEMP.EXPDB
> COPY TO TRANS
```

Space does not permit a more complete display of the use of the commands. A rather nice demonstration package is available from CRI that shows more of the commands. A few hours "playing" with the system is also very instructive.

The programmatic interface consists of the eleven procedure calls listed in Figure 6. Notice that any of the commands may be used through the RELATE procedure. A "cursor" is a file control block. Multiple cursors may be used allowing multiple files to be processed concurrently.

The big problem with relational systems has always been run efficiency. Part of the problem comes from the apparent simplicity of the model - it is very easy to give a logically simple command that requires enormous resources. One of the authors, while experimenting with Relate, inadvertently gave a command that required $72^3 = 373,248$ logical file accesses. It required about 40 minutes to execute and, but for good blocking efficiency could have required much longer. A better way was found to do the same function in a few seconds.

The other source of low efficiency has been the indexing system. The B-tree structure has solved this nicely and this particular implementation is nearly optimally efficient. For example, a new index was created for the TRANS file of 5112 records in 78.5 seconds of CPU time. This works out to 15.4 milliseconds per record which is excellent.

Blocking factors are automatically chosen, and again, seem to be nearly optimal. Disk accesses appear to be the minimum possible in most cases tested.

Procedure	Function
RELATE	Passes a command to the RELATE/3000 data base management system.
RDBADD	Adds a new record to the file associated with the passed cursor.
RDBBIND	Binds a memory location for a return value or a substitution variable.
RDBCLOSE	Closes a cursor.
RDBDELETE	Deletes the current record from the file associated with the passed cursor.
RDBERROR	Returns information on an error condition that exists in a cursor.
RDBINFO	Returns information on the current file or status of the system.
RDBINIT	Initializes a cursor.
RDBPOINT	Positions a pointer to a specific record for reading. This call does not function on views or selections.
RDBREAD	Reads the next record from the associated cursor.
RDBUPDATE	Updates the current record on the file associated with the passed cursor.

Figure 6

In short, both storage and run efficiencies seem to be very close to the ideal. Where performance problems arise, they can likely be traced to the ease with which some very awkward operations can be requested.

Concurrency is handled by dynamic locking of files. It may be enhanced beyond the file level in subsequent releases. Even at the file level, experience with Image indicates that it should be effective.

Restructuring is certainly one of the strong points of Relate. New files are easily defined and data from one or more files can readily be copied to the new file, with undefined fields zeroed or blanked. Moreover, Relate may be used with Image, KSAM, or MPE files as well as Relate files. Many of the commands including OPEN, COPY, PRINT, and CLOSE will work with these other file types. Not only does this give the user the option of combining these other file types into a Relate database, but it also makes the task of converting from the other file types to Relate files a trivial exercise. After one has struggled with restructuring Image, Relate seems almost too good to be true!

Security is defined with the ALLOW command. Essentially this specifies which users or groups of users can execute various groups of commands. For example use of the CREATE and PURGE commands can be restricted to one user. However, these security features only are effective for users going through the Relate system. Only the MPE file security is effective for a user who goes into a Relate file through the MPE file system.

An option exists to considerably strengthen the security by making the Relate files privileged files as Image does. However, this presently involves giving PM capability to the account and to the database administrator. Other options are under study to improve the security including encryption.

The integrity is similar to Image except that journaling (logging to a tape) and recovery from the log tape are not presently implemented. They could be done through application programming.

Data independence is also similar to Image. The user may accept all fields in the record as they physically occur or specify the fields and their order.

Relate/3000 is a new product and, as with any new product of this complexity, can be expected to have some bugs in it. However, it appears to be soundly conceived, efficiently implemented, and a very effective tool. It is available for a one-time fee of \$18,500 which includes the first year's maintenance. Maintenance after the first year will be 15% of the (then) current selling price. Relate is available from:

Computer Resources, Inc.
2750 El Camino Real
Mountain View, CA 94040
(415) 941-4646

Rel*Stor

Rel*Stor is also based on the relational model and, with Relate, shares the particular strengths and weaknesses of that model. Unlike Image and Relate, which were created specifically for the HP/3000, Rel*Stor is implemented on other systems.

The DDL for Rel*Stor is slightly more formal than that for Relate, but closer to Relate's than Image's. The DEFINE command is used to create a file (relation or "table" as it is called in the Rel*Stor manual¹⁶). A database is created using the DEFINEDB command. This command specifies the name of the database and gives upper limits for the number of data files and users. Three directories are then created to store information on the database and its users, but no data files are specified or created.

Derived relations can be created as snapshots through the RETRIEVE command. The RANGE command allows the data for the snapshot to be gathered from more than one file. There is no provision for "evolving views."

The DEFINE command is functionally and syntactically similar to the CREATE command of Relate. For example, the ACCOUNT file would be created by the following:

```

DEFINE ACCOUNT
  DEPT  STRING  4,
  EXP   STRING  4,
  BUDAMT INTEGER 10,
  YTDCR INTEGER 10,
  YTDDB INTEGER 10,
  PRIME=2
  SIZE=23;

```

The PRIME=2 clause indicates that the first two fields (concatenated) constitute a unique key for each record. There is no provision for files, such as the transaction file, where no combination of fields yields a unique key. (Two transactions could be identical in all respects.) Likewise there is no provision for multiple keys. The single key may include several fields (in order of declaration starting with the first). The indexes are structured as B-trees with the advantages of ease of change and efficient retrieval.

There are only three data types specified, and data is stored in external (ASCII) form rather than internal (Binary) form. Data type appears to matter only when arithmetic operations are performed. The three types and the upper and lower limits on their "width" are:

```

Integer  4 to 12 bytes
Real     8 to 12 bytes
String   ---

```

These widths would allow both 16 and 32 bit integers, but only 32 bit floating point. No formatting capability is included.

The DML is quite rich and nearly as complex as a programming language such as Basic. This richness could be seen as Rel*Stor's strongest point. Its complexity could be a weak point.

Before proceeding to the DML it is necessary to understand the programmatic components of Rel*Stor and how they function. A program called the Relational Data Handler or RDH is the main workhorse of the system. The RDH is run as a son process of the user's process. The user's process may either be an application program or a program called the Terminal Interface Process or TIP. See Figure 7.

TIP serves as a conversational interface, text editor, and other miscellaneous functions. It operates in three different modes: control mode, edit mode, and administrative mode. The control mode allows the user to formulate queries, send them to the RDH, and see the results. The edit mode allows the user to compose, edit, modify and save queries. The administrative mode allows qualified users to define new databases, grant users access to databases and retrieve data base statistics. There are 45 TIP commands altogether as shown in Figure 8.

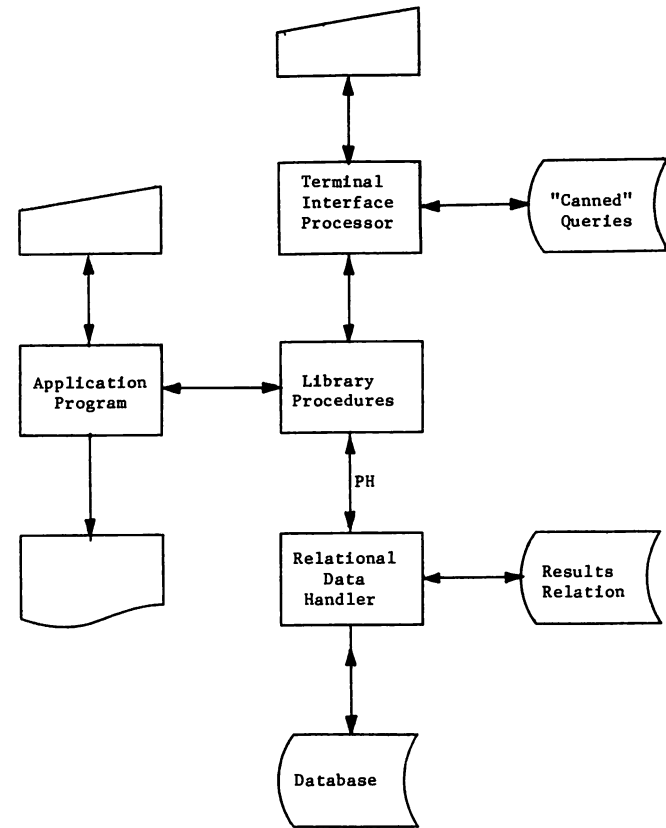


Figure 7

TIP Control Commands

<u>Name</u>	<u>Meaning</u>
APPEND	append, insert, or change data in a relation
BYE	terminate a session with a database
COMPILE	translates Buffer to code and checks syntax
CONTINUE	continue query to retrieve next page of data
DBA	switch from Control to DBA mode
EDIT	switch from Control to Edit mode
END	return control to the operating system
GO	COMPILE and RUN the Textual Buffer
HELLO	initiate a session with a database
LISTREL	list relations and their data attributes
PRINT	change automatic PRINTRR mode
PRINTL	print the number of disk accesses in your last request
PRINTRR	print the Result Relation
RUN	send the COMPILED Buffer
SAVERR	save the Result Relation, formatted for a LOAD command
SEND	send a canned query for processing
SETPAGE	set the maximum rows per page in the Result Relation
SETRR	reset the size of Result Relation file
SETWSR	reset the size of workspace relation file
SHOWME	shows the filenames and parameters of present session
STATISTICS	print the statistics associated with your last query
SYSTEM	print the system configuration parameters

Figure 8 (contd)

TIP Control Commands

<u>Name</u>	<u>Meaning</u>
TIME	print the time required to complete last query
//	terminates APPEND command
\	terminates APPEND command

TIP Edit Commands

ADD	add lines to the Buffer
DELETE	delete lines from the Buffer
END	return to control mode
JOIN	JOIN together the Buffer and a KEEPed file
KEEP	save the contents of the Buffer in a permanent file
LIST	list lines of the Buffer
PURGE	delete a file created by the KEEP command
REPLACE	replace lines of text in the Buffer
TEXT	bring a permanent file into the Buffer
//	terminate ADD or REPLACE command

TIP Administrator Commands

ABORT	set query control point to inactive
ALTUSER	change or add to the capability of a user
COOL	to be implemented
DEFINEDB	define a new data base
DEFINEI	initial installation of RELATE
END	return to control mode
LISTDBS	list data base statistics or all data base names

Figure 8 (contd)

TIP Administrator Commands

<u>Name</u>	<u>Meaning</u>
LISTUSER	list directory for one user or all users
NEWUSER	enter a new user with RETRIEVE capability
PURGEUSER	remove a user from a data base directory
REMOVEDB	remove a data base
RESTRICT	to be implemented
WARM	to be implemented

Figure 8

The programmatic interface to the RDH consists of four library procedures:

Procedure	Function
STARTRDH	Creates the RDH process
TXBFMGR	Sends a query to RDH
PROJECT	Returns one field at a time from the result of the query
GETDESC	Returns a description of the result of a query

Queries are processed in a "batch" mode by the RDH; i.e. a query is passed to the RDH, the result of the search is placed in a temporary file called the "result relation." The results may then be interrogated or even saved as a new, permanent file.

Thus, in spite of the numerous TIP commands, it is the RDH commands that constitute the true DML, and even include most of the DDL. Figure 9 lists the RDH commands and operators which may be combined in the usual ways.

The RETRIEVE command is the primary read verb and the WHERE clause the qualifier. For example, the following query would retrieve all transactions debited to department #42 for expense categories 10-29 if the amount exceeds \$100.00.

```
RANGE X = ACCOUNT
RETRIEVE X.DEPTDB, X.EXPDB, X.DEPTCR, X.EXPCR, AMT, DATE, REF
WHERE
  X.DEPTDB = 42 AND X.EXPDB >=10 AND X.EXPDB < 30
  AND AMT > 100.00
```

The LOAD command may be used to load data to a Rel*Stor file from an MPE file. It also may be used to add new records to a Rel*Stor file. There is no update verb in the RDH commands. The APPEND command in the TIP control mode will update. The only other option is to copy the entire file, modifying the necessary records. This seems to reflect a strong "batch" orientation rather than an on-line orientation.

Little can be said about performance since the product was not available for testing. The batch orientation of the DML, however, makes it likely that interactive processing, for any but a read only mode, would be awkward at best and could be very inefficient.

Storage efficiency is likewise less than optimum due to the storage of external rather than internal forms of the data.

Concurrency is not implemented at the present time. There are plans to add it in 1982. At present only one user at a time may access a database.

RDH Commands and Operators

Name	Meaning
AND	logical (Boolean) operator
AVG	COLLECT function - averages real or integer
BUT	logical (boolean) operator - same as AND
COLLECT	perform function (COUNT, SUM, MIN, MAX, AVG)
COUNT	COLLECT function - counts rows
DEFINE	create a new data base table
DELETE	delete rows from a data base table
DELETEQ	DELETE but save DELETED rows in Result Relation
LOAD	bulk load data into a table
MAX	COLLECT function - finds maximum in column
MIN	COLLECT function - finds minimum in column
NOT	logical (Boolean) operator
OR	logical (Boolean) operator
RANGE	specify table and assign relation variable(s)
REMOVE	purge a data base table
RENAME	rename a data base table
RETRIEVE	get information from a data base
SAVE	save the Result Relation as a data base table
SUM	COLLECT function - adds real or integer column
WHERE	introduce qualification clause to query
+, -, *, /	arithmetic operators - plus, minus, times, divided by
()	parenthesis - determine order of operation
:=	value of expression is assigned to variable
<, <=, =, >=, >	relational operators
#	relational operator - is not equal to
%	relational operator - is included in - string only

Figure 9

Security is defined by the NEWUSER and ALTUSER commands. These allow the administrator to control the users having access to each database, and the commands that each user may use. However, these security provisions are only effective for users going through the Rel*Stor system. Only the MPE file security is effective for a user who goes into a Rel*Stor file through the MPE file system.

Integrity is similar to Image except that there is no provision for journaling (logging transactions to tape) and recovery from the log tape. This could be implemented by application programs, however.

Data independence is also similar to Image. The user may accept all fields in the record as they occur, or specify the fields and their order.

Rel*Stor is a new product, with some features not scheduled for implementation until next year. As with any new product of some complexity, it can be expected to have some bugs. It was designed to work on a variety of computer systems and this may well cause some compromises in its design - for example the "batch" orientation of the RDH.

Rel*Stor is available for a one time fee of \$20,000 which includes delivery, installation, three copies of the documentation, maintenance/update service, and hot-line consultation for 12 months. Maintenance/update service, and hot-line consultation costs \$3000 per year after the first year. Rel*Stor is available from:

GTE Products Corporation
P.O. Box 188
Mountain View, CA 94042
(415) 966-2371

SUMMARY

Any time that complex systems are compared, it is difficult to be totally objective because the designers of the systems had different goals and viewed various features with differing importance. Likewise, users of these systems will have varying needs. Thus it is not possible to rank these systems - any one of them might be the best choice for some applications. The grading chart, shown following, is an attempt to objectively assess salient features of each system. Even here, however, each individual grade must be a subjective judgment made after considering a variety of dissimilar factors.

Bibliography

Grading

In the table below, the features of each of the three systems have been graded in ten categories. These grades are solely the judgment of the authors. The grade meanings are as follows:

- A - excellent, outstanding
- B - above average
- C - average
- D - below average but useable
- F - essentially useless or non-existent
- X - unable to determine

	<u>Image</u>	<u>Relate/3000</u>	<u>Rel*Stor</u>
1. Mapping to system	D	B	B
2. DDL and data types	B	B	C
3. DML	C	A	D
4. Run performance	B	B	X
5. Storage efficiency	C	A	D
6. Concurrency	B	C	F
7. Restructuring	D	A	B
8. Security	B	D	D
9. Integrity	A	B	B
10. Data independence	C	C	C

Future Developments

This paper is the result of one small step in a project to bring together a cohesive and effective set of program development tools. The key-stone of this project is a data dictionary which would be shared by all components. At present each system (Image, Relate/3000, Rel*Stor, V/3000, etc.) has its own data dictionary (or fragment thereof) contained within it.

The time has come for a centralized data dictionary. The dictionary would necessarily be complex enough to require a database to store it. Systems could share the information and the data conversion procedures instead of each having its own (different) version. The authors plan to do future work in this direction.

1. Newsletter of the HP General Systems Users Group, p. 24, April, 1981.
2. Atre, S., Data Base: Structured Techniques for Design, Performance, and Management. New York: John Wiley & Sons, 1980.
3. Date, C.J., An Introduction to Database Systems. Reading, Mass.: Addison-Wesley Publishing Co., 1977.
4. Kroenke, David, Database Processing. Palo Alto: Science Research Associates, Inc., 1977.
5. Tsichritzis, Dionysios C., and Lochovsky, Frederick H., Data Base Management Systems. New York: Academic Press, 1977.
6. Ullman, Jeffrey D., Principles of Database Systems. Potomac, Md.: Computer Science Press Inc., 1980.
7. Wiederhold, Gio, Database Design. New York: McGraw-Hill Book Co., 1977.
8. Date, op.cit., p. 53.
9. Atre, op.cit., p. 130.
10. Image Data Base Management System Reference Manual. Hewlett-Packard Co., Cupertino, CA., Part No. 32215-90003.
11. Query Reference Manual. Hewlett-Packard Co., Cupertino, CA., Part No. 30000-90042.
12. Image, op.cit., p. 4-17.
13. Relate/3000 Database Management System Reference Manual. Computer Resources, Inc., Mountain View, CA, July 1, 1981.
14. Comer, Douglas, "The Ubiquitous B-Tree," Computing Surveys, Vol II, No. 2, June 1979, pp 121-137.
15. KSAM/3000 Reference Manual. Hewlett-Packard Co., Cupertino, CA, Part No. 30000-90079.
16. User Reference Manual for Rel*Stor. GTE Products Corp., Mountain View, CA, March 1981.

**THE
HP 2680A
LASER PRINTING
SYSTEM
(SOFTWARE)**

Anthony G. Stieber
HP2680A Applications
BGD, Peripherals Group

THE HP 2680A LASER PRINTING SYSTEM

The HP2680A Laser Printing System consists of a medium-speed laser printer which together with its associated software, allows the development and printing of forms containing data, special character sets, company logos as well as the usual design elements making up a standard form.

The outstanding feature of this printing system is the interactive software which is used for development of all kinds of different forms, character sets, company logos and other artwork using no more than a standard graphics terminal and a digitiser linked up with an HP 3000 computer. This software has been specially designed for the use of the non-programmer - all instructions are entered through menus or special function keys on the terminal and there are various useful default values present in the menus. In addition, sensible warning and error messages are displayed to prevent the user from wasting his time with correcting mistakes.

Although the software is very easy to use, it has many built-in features allowing the user to make full use of the printing flexibility of the HP 2680A Laser Printer. It is the purpose of this presentation to describe some of these features using practical examples.

OVERVIEW

The structure of the software associated with the HP2680A Laser Printer could be represented in the form of the following equations:-

Printout = Data File + Environment File

Environment File = Forms + Character Sets + Page Layout

Forms = Lines + Boxes + Shading + Headings + Logos + Signatures

Character Sets = Characters + Logos + Signatures + Other Artwork

The following software is available for use with the HP2680A Laser Printer:-

- 1 - IFS2680 (Interactive Formatting System) is used for creating environment files.
- 2 - ID /3000 (Interactive Design Software) which includes two programmes:
 - a) IDIFORM for creating forms
 - b) IDSCHAR which is used for creating character set and logo files.
- 3 - Various intrinsics within MPE which are used to control the Laser Printer.

PRINTING

In order to control the printing out of data by means of the environment file, a modified version of the file equation can be used which allows the environment file to be specified. This file equation also specifies the target device i.e. the HP2680A printer.

eg. `.FILE PRINTOUT;DEV=EPOC;ENV=TESTENV;CCTL`

("EPOC" is the device class name for the HP2680A)

Data which is sent to this device file "PRINTOUT" will be automatically merged with the environment file "TESTENV". It is sent in this form to the spooler and then printed out using the specified character sets, forms etc.

INTERACTIVE DESIGN CONTROL

Since IDS/3000 has been designed with the non-programmer in mind, no special programming language has to be learnt in order to make full use of the features of this system.

All instructions to the system are entered by the user by means of menus and function keys. In order to save on development time, useful default values are provided by the software. These are particularly valuable when only relatively simple environments, forms and characters are to be designed. Warning and error messages prevent even an inexperienced user from making serious errors such as erasing the product of an afternoon's work!

When the user has completed and entered some menus, the system asks him to confirm his entries by repeating "enter". Some additional information may be displayed so that the user can be quite sure that his entries are correct.

IFS2680 - ENVIRONMENT DESIGN

The environment file contains all the information required to control the printing of data:-

1 - Physical page information:

Paper dimensions
Number of copies of each page to be printed

2 - Print formatting information:

Location and dimensions of printing areas on a physical page.

These printing areas are called logical pages.

Names of forms on which data is to be printed. Each form must be attached to a logical page.

The positioning of a form on a logical page may be done manually by indicating the distances from the top and left hand sides of the logical page or it may be placed automatically either in the centre of the page or in one of the four corners. If the form does not fit into the logical page, it may be scaled down automatically. However, if desired, the form may project beyond the sides of the logical page. In this case, no data may be written to the parts of the form lying outside the logical page boundary.

Vertical formatting control for each logical page

Printing direction (orientation) on a logical page

Up to thirty-two different logical pages containing two forms each may be specified. However, there is a limit of 18 different forms which can be printed on a physical page.

3 - Typeface information for printing data:

Character font name

Character font size

Character font orientation

Up to thirty-two character sets may be specified within an environment file. Each orientation and size of a character font is declared as a different character set. Logos, signatures and other artwork can be specified to be character sets and may thus be printed out as data.

INITIALISATION

If the user wishes to make use of most or all of the features of a standard environment file present in the system or of one of his own environment files, this may be specified on the Initialisation Menu. The desired features are then copied from one environment file to the one that the user is constructing.

Among the standard environment files in the system, there are several which may be used for automatic 2:1 or 4:1 reduction of data. This is especially useful for storing large amounts of computer output in archives.

MULTI-COPY FORMS

There is a special menu for specifying multi-copy forms which are intended to simulate the use of carbon paper with each copy consisting of a different form for distribution to different people. With this menu, one can indicate two forms to be printed on each copy. Up to eight different copies may be specified - each containing two different forms while the data printed on each form will be the same. It is of course possible to blank out data on certain forms by specifying black boxes within the forms.

COMPILATION

When all desired parameters have been entered into the environment file, they must be compiled so that they can be downloaded to the HP2680A in a form that can be used directly for printing. The compilation of the environment file is started by a command in the main menu.

IDSFORM - FORMS DESIGN

Forms which are used for printing data on the Laser Printer are stored in standard MPE forms files. Each forms file may contain different named forms.

The IDSFORM programme is used for interactive forms design using only an HP graphics terminal. For ease of development and modification, each form may be have a structure consisting of:

1 - Subforms

Sub-forms may be stored in a temporary "Hold" file and may be moved or copied to other locations on the same form or even to other forms.

2 - Fields which are used for printing data and for specifying headings. Fields must be contained within sub-forms.

Headings may be specified using any desired character sets in any one of the four possible orientations and in various sizes. In addition, the position of a heading within the field may be specified as well as whether it is to be justified or not. Any logo or special characters stored by the user may be specified in a heading.

3 - Subfields which may be used for printing data. They must be contained within fields.

FORM GRAPHICS

When working at the form and sub-form levels, it is possible to make use of some graphics features:






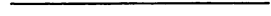
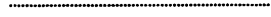

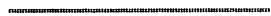
- 1 - Line drawing using 3 different line thicknesses and 8 types.
- 2 - Box drawing using 3 different outline thicknesses and 8 types.
- 3 - Box shading in 5 different shades

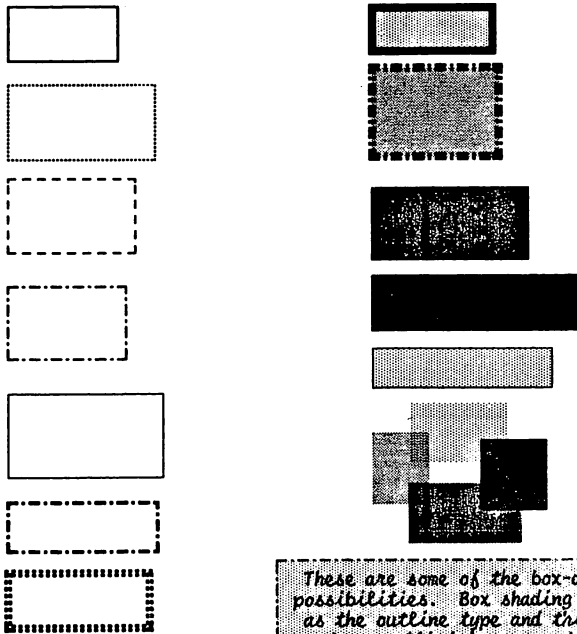
Some examples of the variations possible in drawing lines and boxes are shown opposite.

DESIGN AIDS

A grid with numbered lines can be specified by the user in order to be able to place form elements at precisely the location he requires. This grid is purely a design aid and is not printed on the final version of the form.

IDSFORM - GRAPHICS OPTIONS

	Solid single line	\	
	Dotted line		
	Dashed line		
	Dot-dash line		All combinations of these line types
	Thin solid line	>	and thicknesses are possible. They
	Normal thickness dotted line		may be specified on the graphics menu.
	Wide dashed line		
	Double dotted line		
	Double wide dashed line	/	



These are some of the box-drawing possibilities. Box shading as well as the outline type and thickness may be specified in any combination.

Printed by the HP2860A Laser Printing System

IDSCHAR - CHARACTER SET DESIGN

Characters, logos and other artwork used for printing on the Laser Printer are stored in standard MPE files. Two file types may be specified:

- 1 - Character Set Files: each of these files is used for a complete typeface ie. all sizes and orientations of each character are stored in a character set file.
- 2 - Logo Files: each of these is used to store a particular logo or piece of artwork in all of the desired sizes.

IDSCHAR is the programme used for the interactive creation of character sets and design elements by means of an HP graphics terminal and an HP digitiser or graphics tablet. The basic storage unit of a character set or logo is the character cell.

CHARACTER CELL SPECIFICATIONS

- 1 - Cell dimensions (maximum size is 255 x 255 printing dots ie. 3.6 cm or 1.4 inches square)
- 2 - If proportional spacing is to be used for printing
- 3 - The bounds used for proportional spacing
- 4 - Positioning of the cell relative to other cells when printing.

PROPORTIONAL SPACING

Character sets may be defined as being proportionally spaced ie. the printing position of a character on a line depends on the width of the previous character. For instance, in the final printout, the space taken up by an "i" will be much less than that taken up by a "w". Within a character cell, the bounds used for proportional spacing may be specified.

DESIGN AIDS

Outline Generation:

In order to simplify the task of designing characters, logos, signatures etc, an HP digitiser or graphics tablet may be used to trace the outline of a piece of artwork. Within IDSCHAR, this outline may be stored, magnified or moved within a character cell. If the user is satisfied with the outline, he may fill it in manually or automatically with printable dots.

Printed by the HP2860A Laser Printing System

Cell Grid:

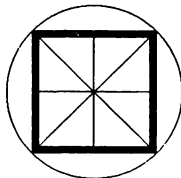
For the accurate placement of printing dots within a cell, the dot positions can be marked automatically using a two letter command.

Cell Manipulation:

When designing for instance a character set, a very useful feature of the system is the possibility to move character cells around the screen and to display different character cells at the same time.

Cell Graphics:

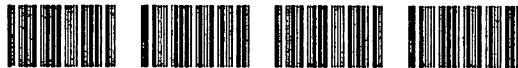
Using the function keys on the terminal, it is possible to draw lines, arcs, circles and boxes within a character cell as shown here on the right.



Examples of Character Design:



324



J. Egenmann

PROGRAMMATIC CONTROL

Under MPE, various intrinsics have been implemented to allow the user complete flexibility in controlling data printout. A programme called Translator (available in the Contributed Library) simplifies the use of these intrinsics to allow even the non-programmer access to the features of the Laser Printing System.

PHYSICAL PAGE CONTROL

A skip to a new physical page may be carried out at any time during the printing of the data using an intrinsic.

LOGICAL PAGE CONTROL

Up to thirty-two logical pages may be specified per physical page; a logical page may be either active or inactive. The data coming from a source file or programme is printed on each active logical page in sequence. When all active logical pages have been printed on, printing continues on the first active logical page of the next physical page. Logical pages may be activated or deactivated using an intrinsic.

CHARACTER SET SELECTION

Character set selection with an intrinsic is done by specifying the primary and secondary character sets. As with an ordinary printer, data is normally printed with the specified primary character set but after a shift out, the secondary character set is used for printing.

DATA DIRECTING

Data can be directed to print out on a particular named field of a specified sub-form within a selected form. The exact destination of the data may be specified using three intrinsics. One advantage of this system over traditional printing methods is that changes can be made to the form without making any changes to the applications programme producing the data - the named forms, sub-forms and fields are all the parameters that are required.

PEN CONTROL

The laser in the HP2680A can be considered to be a pen drawing an image on the paper. This pen can be moved about and positioned anywhere on the paper using intrinsics. Printing of data always starts at the current pen position.

OBTAINING PRINTING INFORMATION

Several intrinsics can be used to obtain information on the following:

- 1 - The character fonts in use
- 2 - The logical pages in use
- 3 - The current state of the print job
- 4 - The width of a given character string in dots
- 5 - Error information.

GLOSSARY

- Cell File:** An MPE file containing character cells ie. characters, company logos, signatures and other graphics elements.
- Character Cell:** Storage unit within a cell file which contains a character, logo or signature etc of a particular font, size and orientation.
- Dots:** Since the HP2680A printer prints dot patterns, cell size for instance may be specified in terms of dots (180 dots per inch or 71 dots per centimetre).
- Environment File:** An MPE file containing all the information required for the printing out of formatted data together with any desired forms.
- Forms File:** This is an MPE file containing one or more forms.
- Logical Page:** This is a printing area defined on a physical page which is used for printing data and forms.
- Multi-copy Forms:** This is a feature simulating the use of carbon paper in sending the same data on different forms to different recipients.
- Point Size:** This is the term used in the printing industry indicating type size. As a general rule, there are 70 points per inch but the exact point size of a character also depends on some other factors apart from the height of the letter.
- Proportional Spacing:** Variable spacing between letters depending on the actual width of each individual letter.

CONCLUSION

For the user who wishes only to print out a report or a memo on the Laser Printer the task is made easy by the use of default values available within the system while the user who wishes to make use of all the features of the Laser Printer is enabled to do so by the great flexibility of the software.

31st August 1981

Anthony Stieber
BGD, Peripherals Group, Boeblingen

DATA CONCENTRATORS IN FOCUS FOR MINICOMPUTER-USERS

Paper held by Peter J. Mikutta, Ing. grad.,
President
TELEMATION GmbH
Bismarckstr. 8
6232 BAD SODEN/TS.

HP-Users Group, Berlin,
Thursday, October 8, 1981

PETER J. MIKUTTA

Occasionally sales representatives of mainframe manufacturers state that asynchronous communication systems are nowadays abolished and they refer convincingly to their intelligent and synchronous TP-systems. Some users accept this information and spread it out with firm belief as their own opinion. No wonder that asynchronous communications have been more or less condemned.

However, the fact is that asynchronous communications with means of data concentrators provide the minicomputer user with significant advantages regarding error correction and transmission performance like the IBM users with an IBM 3270 or IBM 7380 system. This paper is aimed to help you to understand the advantages of using data concentrators specifically with regard to price-/performance ratio.

Before microcomputer-driven data concentrators became available, mincomputer users planning to install more than one low-speed terminal in a remote branch location had to lease a telephone line with two modems for each terminal at high cost (Fig.1a).

P.J.MIKUTTA
TELEMATION GMBH
BISMARCKSTR.8
6232 BAD SODEN/TS.

.../2

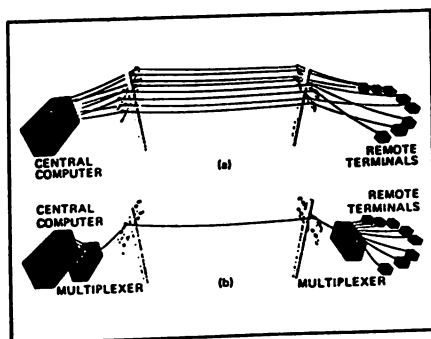


Fig. 1. Two approaches to linking multiple terminals to a central computer: (a) using a separate line for each terminal and (b) using a single multiplexed line.

Most PTT organisations provide for asynchronous modems only two alternatives:

1. a modem for 300 baud (30 characters/sec.)
2. a modem for 1200 baud (120 characters/sec.)

These low speeds do surely not encourage to operate with more than one CRT or printer - since, even with 1200 baud, to fill on CRT screen takes 16 seconds. The unsatisfying transmission speed in conjunction with the lack of error correction has carried asynchronous communications into a dead end.

Another alternative was to use time-division (TDM) or frequency division (FDM) multiplexers that enable terminals to share a telephone line (Fig. 1b). However, the lack of sophisticated error-control routines inherent in minicomputer-supported Teletype-compatible CRTs, printers, and other remote peripherals results in unacceptable line-error rates with this approach.

.../ 3

The error rate problem is especially acute with high-speed time division multiplexers that operate at the maximum rate for a single voice-grade-circuit. The best error rate quoted by any supplier, including the telephone companies, is 10^{-6} . This translates to one error every 90 seconds at 9600 bps; at 4800 bps, one error every three minutes; and at 2400 bps - the minimum line speed required to fill a CRT screen in a tolerable time frame - one error every six minutes.

This error rate is overcome when multiplexed data communication networks use mainframe processors as hosts because the terminals automatically retransmit data containing errors. As a result, mainframe terminal operators see only a slight degradation, if any at all, in terminal response when an error occurs. However, when terminals are linked to a minicomputer, erroneous data blocks cannot be retransmitted. Moreover, implementing a data communications link based on TDMs and minicomputers can be expensive. Modems needed to operate at 9600 bps can cost as much as \$ 5000 each, or more than three times the cost of a multiplexer alone. Even at slower speeds, this equipment is costly.

The need for intelligence

To implement a multiplexed multi-terminal network, mini users need "smart" multiplexers that will enable them to hang more than one terminal off a single line at each site while providing the error-control required for efficient operation. In addition, minicomputer users need to use high-speed terminals to minimize operator waiting time during interaction with the computer.

To provide these capabilities at a price mini users can afford, several US companies, such as MICOM SYSTEMS, have introduced microcomputer-driven data concentrators. These devices typically handle four or eight channels.

.../4

A multiplexed network uses two concentrators: one at the terminal end of the phone link, the other at the host computer. Both devices are linked to the telephone line via modems. In operation, the concentrators buffer data prior to transmission, enabling them to transmit variable-length data blocks depending on the loading on each terminal's channel.

In effect, the data concentrator, or statistical multiplexer, as it is often called, increases the average traffic on a high-speed line by buffering peak traffic on individual channels.

The concentration made possible by buffering data also increases throughput compared to time-division multiplexers. The reason? TDMs transmit data blocks even when a particular terminal has no data to send. Statistical multiplexers, on the other hand, assign channel capacity dynamically according to the load on a given input channel.

The load-averaging method used by statistical multiplexers makes them better suited than TDMs to the interactive mode of operation that characterizes minicomputer-based systems. In interactive applications, the loadings tend to come in bursts, rather than at the steady pace characteristic of batch or remote job entry. Hence, TDMs are often too powerful for minicomputer applications.

Smart means no error

Moreover, by buffering data, concentrators can also check data blocks received on the high-speed link and request retransmission in case of error. To implement this automatically, similar to that used in IBM's SDLC protocol. A data concentrator, for example, typically attaches a cyclic redundancy check (CRC) character to each transmitted block. The receiving concentrator then recalculates the attached CRC to check the block for errors.

.../5

With data concentration, the undetected error rate is so low (better than 1 block in 10^{12}) that data transmission is error-free for all practical purposes. The error-handling does not involve the host minicomputer at all. In fact, in most applications, the mini treats the data communications hardware as if it were simply a hardwired peripheral located in the same room.

The microcomputer based intelligence of a statistical multiplexer is also used to simplify network configuration. The Micom Micro 800, for example, is self-configuring to a large extent. All configuration parameters, including the data rates for each channel, are switch selected, with only 16 DIP switches required to configure a four-channel unit. In contrast, most TDMs have literally thousands of possible strap-option permutations, any of which may be responsible for time-consuming installation problems.

To further simplify system configuration, switch selection of configuration parameters is required only at the computer site. The host data concentrator automatically down-line loads all configuration data. Only one switch need be set in the remote concentrator to inform the unit that it is a "slave" unit.

Data concentrators do have a drawback, however. During prolonged peak transmission periods, or because lines errors have lead to excessive retransmission, "buffer overflow" may occur as data comes into the buffer faster than it can be sent out onto the line. In such a situation, data is lost, with the most active channel losing all its data first, followed by less active channels in order to buffer utilization.

To minimize data loss caused by buffer overflow, data concentrators incorporate switch-selectable options intended

.../6

to suspend data transmission temporarily. For example, the Micro800 takes advantage of the fact that most minicomputers will suspend transmission either on receipt of a special control character (XOFF), or on the dropping of the Clear-to-Send interface control signal. Transmission will resume when the system receives the XON control character or when Clear-to-Send is raised.

If buffer overflow continues and data is lost, an appropriate message is sent to the affected terminal. The Micro800 also automatically transmits a "LINK DOWN" message, when the communications link between Micro800s is down. In short, the concentrator advises terminal users of fault conditions in the communications system.

In the meantime Micom has installed more than 20.000 Micro800's, many used as terminal 'cluster controllers' in DEC, Data General, and Hewlett-Packard systems. For many of the thousands of customers already using the first generation Micro800 Data Concentrator besides the easy "do-it-yourself installation" the most remarkable feature was the "do-it-yourself troubleshooting".

Do-it-yourself Troubleshooting

Any item of data communications equipment such as the Micro800 data concentrator must be connected to a variety of equipment supplied by other vendors (modems, lines, data terminal equipment), all of which can and will malfunction from time to time. Since the Micro800 is designed for do-it-yourself installation, it also incorporates built-in test features to facilitate do-it-yourself troubleshooting.

The intelligence of the Micro800 is used, for example, to provide the response "LINK DOWN" automatically from the

.../7

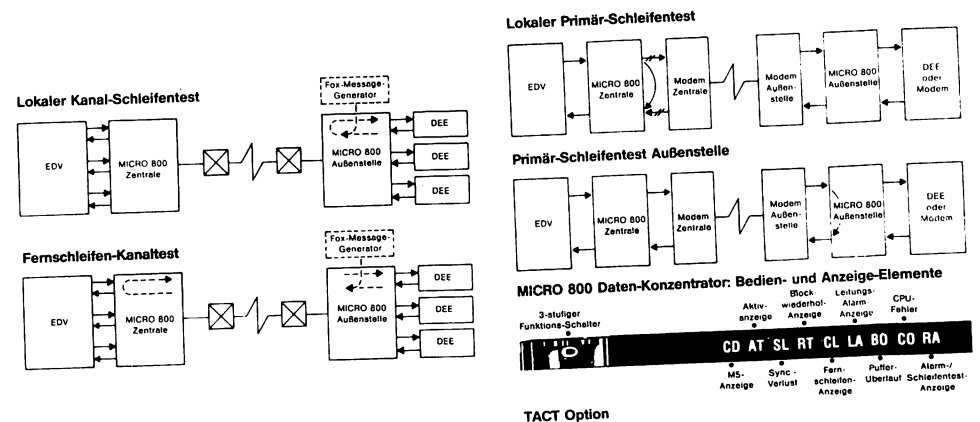
slave Micro800 if an ENQ (Control E) is entered from the terminal and the high-speed communications link between Micro800's is down. Thus, the terminal user is kept advised of fault conditions in the communications system.

Controls and Indicators

The Micro800 includes as standard a comprehensive set of status displays and a thumbwheel switch behind the Micro800 front panel. The 3-position thumbwheel switch provides for activation of two fault-isolation loopback tests and a self-test of the local Micro800 unit.

The Local Composite Loopback test position causes the composite output from the Micro800 to be looped back to itself for testing of the local concentrator.

The Remote Composite Loopback test position causes the composite interface loopback test to be performed remotely to test both the local concentrator and the transmission link to the remote concentrator.



TACT Option

The Micro800's most powerful built-in test feature is TACT, the Terminal-Activated Channel Test Option. TACT allows any terminal to check out its own operation, or the local Micro800, or the complete Micro800 system end-to-end.

TACT is activated from the terminal by depressing ENQ (Control E), followed by BREAK. This causes the local Micro800 to respond with the message "MICOM IN TACT". The terminal may now select one of four test functions. Depressing "L" (Local Test) causes the channel to enter the Local Channel Loopback mode. Upon entering this mode, the message "LOCAL TEST" is transmitted to the terminal. Thereafter all data entered from the terminal will be looped back to the terminal by the local Micro800. Depressing "R" (Remote Test), following TACT activation, causes the channel in the remote Micro800 to enter the Remote Channel Loopback mode. The message "REMOTE TEST" is transmitted to the terminal. Thereafter all data entered from the terminal will be looped back to the terminal by the remote Micro800.

Activation of the built-in "fox message generator" is achieved in the same manner as the loopback mode selection. Depressing "T" (Terminal Test), following TACT activation, causes the local Micro800 to transmit the message "TERMINAL TEST" followed by a continuous "fox" message to the terminal. Depressing "S" (System Test) following TACT activation causes the local Micro800 to place the remote Micro800 in Remote Channel Loopback mode and transmit the "fox" message continuously to the remote Micro800 where it is looped back and transmitted to the terminal. The message "SYSTEM TEST" is received by the terminal at the start of the test.

TACT is deactivated by depressing BREAK. TACT signals it has been deactivated by transmitting the message "MICOM TACT COMPLETE" to the terminal.

.../9

For Hewlett-Packard-users MICOM provides special support for hp-3000 systems. Thus special XON/XOFF characters are being recognized for buffer and flow control.

Introducing the Micro800/2

Based on the experience with their worldwide installations MICOM has now introduced the model Micro800/2, the first second generation data concentrator. Besides the features already mentioned the Micro800/2 exploits new advances in semiconductor technology to offer eight times the performance of the original Micro800. For example the eight channel-unit can handle eight (8) CRTs with 9600 bps each over one single modem line with 9600 bps. That means a concentration factor of 800 %. In addition, it offers major feature improvements such as data compression, terminal priority, terminal-initiated channel configuration, synchronous and clocked asynchronous channels, and a 'command port' to permit on-line system testing, reconfiguration, message broadcast, and performance monitoring. The Micro800/2 retains the same small size and light weight as the Micro800 to minimize logistics problems and simplify installation and replacement in the field. Like the Micro800, it is designed for "do-it-yourself installation" and ease of operation by non-technical personnel.

Command Port Feature

All standard Micro800/2 models are equipped with a Command Port which offers a wide variety of monitoring, test, and control facilities. The Command Port may be connected to a dial-up or dedicated terminal provided by the customer or directly to a computer port, and may operate at up to 1200 bps.

Message Broadcast permits a message to be transmitted from the Command Port to selected channels or to all channels, local or remote. This feature may be used, for example, to advise of an impending computer shut-down or the schedule

.../10

for system restoral.

Dynamic Channel Reconfiguration permits the data rate to be changed for a selected channel or permits activation of local echo or generation of specific delays for carriage return, line feed, and form feed, temporarily overriding the channel configuration selected by DIP switches.

Remote Busy permits busy-out of dial up modems attached to individual channels on the remote Micro800/2, facilitating centralized access control in timesharing computer systems.

Centralized Troubleshooting is available from the Command Port, including the full capabilities of TACT as well as control of local and remote composite loopbacks.

Alarm Messages with time and date of occurrence are generated automatically each time the Micro800/2 locally or remotely experiences a buffer-full or buffer-overflow condition, encounters unusually high line error rates, or loses synchronization or 'carrier' on the high-speed composite data link. Analysis of the message log helps pinpoint telephone line and modem problems.

Periodic Reports at user-selectable intervals or on demand provide statistics on data traffic, average and peak buffer memory utilization by channel, block retransmissions, and telephone line quality including outages. Analysis of these statistics shows trends in telephone line quality and provides an indication of the ability to add additional channels or increase the speed of existing channels to improve service and plan for future growth.

.../11

SYNC-Option

Furthermore, I like to mention the Sync-Option. In addition to the asynchronous terminals it allows to operate four synchronous CRTs or printers. They may either run in character-oriented protocols (BSC) or bit-oriented protocols (HDLC/SDLC).

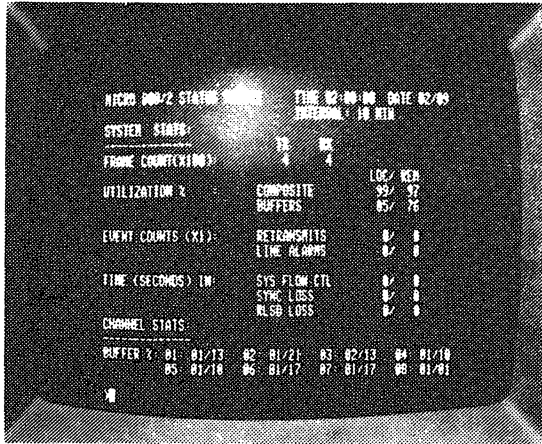
I am confident that I have been able to express to you the benefits of using data concentrators since they provide enormous savings in terms of telephone line and modem costs. As carried out in my example of a cluster-configuration with 8 terminals you will need - instead of 8 expensive telephone lines and 16 modems - one telephone line and two modems only. At the same time you can use fast synchronous modems up to 9600 bps instead of asynchronous low speed modems with maximum 1200 baud only. Automatically, you will have gained an error correction which reduces transmission errors practically to zero. Besides this, all these advantages do not load up your minicomputer system. No hard- or software-changes are required.

Nowadays, since savings are more and more becoming a must, the usage of data concentrators is the ideal and most effective tool for cost reduction. Therefore, data concentrators should be considered in any data communications concept.

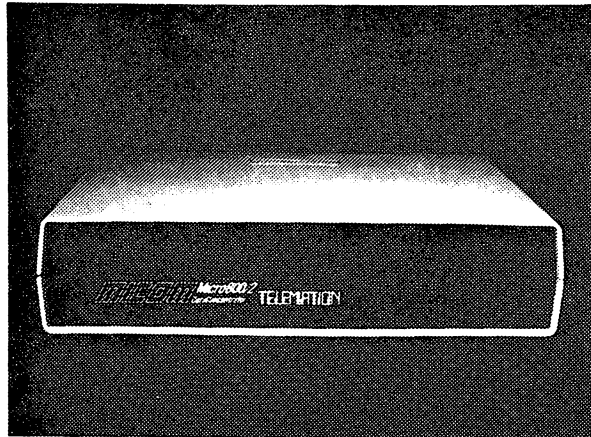
Asynchronous communications supported by data concentrators are by no means old-fashioned but state-of-art technology with progressive means of cost savings.

Due to the high compression technique telephone lines will be more efficient and transmission will become error-free.

.../12



Status Report



DISTRIBUTED PROCESSING
A HEWLETT PACKARD SOLUTION

DISTRIBUTED PROCESSING - A HEWLETT PACKARD SOLUTION

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

MATTHEW O'BRIEN

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

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

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

- localization of some computational power and program memory

M. O'BRIEN
HEWLETT PACKARD GENERAL SYSTEMS DIVISION
19410 HOMESTEAD ROAD
CUPERTINO, CALIFORNIA 95014

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

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

The mainframe computer was the first result of constructing electronic devices to perform large amounts of computation or calculation. Prior to the late 1930's and the early 1940's, rudimentary machines had been constructed to handle either calculation with numbers or some other sorting or controlling function. In order to handle problems which involved extreme efforts of mental and hand calculation, investigations were begun into constructing an electronic machine which would automate

the calculation process. Perhaps one of the most famous examples were the calculations to produce a book containing tables of artillery projectile paths under varying conditions of shell mass size, charge mass and volatility, wind conditions, atmospheric density and of course barrel elevation and azimuth. As so many variables were involved and such great accuracy was desired, it was necessary to perform many hundreds of thousands of calculations to produce a satisfactory result.

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

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

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

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

central processing unit.

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

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

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

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

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

applications in science and business.

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

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

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

needs are sufficiently limited in scope as to be adequately met by the microcomputers and small peripherals.

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

Immediate

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

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

Future

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

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

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

organization is headed.

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

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

The HP 125 combines three functional abilities within one package:

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

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

dependent I/O functions.

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

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

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

mented amount of microcomputer software to the purchaser.

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

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

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

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

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

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

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

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

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

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

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

The terminal supports remote operation and configuration by use of escape sequences. As an example, the keyboard has a 'home cursor' key which positions the cursor at the first character in video memory. An application program can also

home the cursor by transmitting the correct escape sequence to the terminal. By this means, applications running in either the CP/M CPU within the system or an application running on a mainframe can efficiently manipulate the terminal features to provide a friendly applications interface to the user.

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

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

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

In order to diagnose a difficulty in running a CP/M-based application, the HP 125 user can map the output (console out) of the CP/M CPU to be not only the CRT screen, but also datacomm port 1. To this port he has connected a modem which ties over the phone lines to another HP 125 (or terminal) on which a knowledgeable user of the application is viewing. By this means, the output of the application and keystrokes entered by the

user (CP/M operates in a full duplex mode) can be viewed for debugging. Further, were the user to map datacomm port 1 as the input for the CP/M CPU (console in), the remote viewer can also run the program and allow the direct operator to watch in order to learn the correct manner in which to run the application.

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

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

In the normal mode of operation, the cursor control and editing keys as supported by terminal firmware allow

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

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

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

In a previous section, the permanent and essential nature of the mainframe was discussed. As present users of the HP 3000 computer can probably attest, a major usage of the system involves the creation, maintenance and access to data bases which allow the smooth function-

ing of large organizations. This automation of data base with instant and accurate access has been the principle benefit of the computer to the business world.

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

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

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

For the display and analysis of data, the personal computer of today tends to fail to efficiently perform its function. The data base for most organizations is large, communal in nature, subject to frequent correction or update,

and most necessarily must be current and correct throughout the organization. Using a stand-alone personal computer, much time consuming and detailed analysis has been done only to find the raw data was incorrect due to an error in transcription or a recent update.

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

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

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

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

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

The strength of the HP 125 is its interactive ability

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

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

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

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

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

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

To summarize, it is believed that the manner in which computers are used by organizations to enter, display and analyze data is evolving towards a new distributed network of processing units. The change on the scene is due to the technological ability to produce processing units that are inexpensive, reliable and capable. The ability to place a personal computer in the hands of an individual has shown to be not only cost effective, but by being personal has involved individuals not previously utilizing computing

power directly. While personal computers have these benefits, they have not fully utilized the greater advantage of being part of the entire organizational data processing network within most organizations.

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

TERMINAL I/O

AN ENGINEERING FEEDBACK SESSION

JIM BEETEM

TERMINAL I/O
AN ENGINEERING FEEDBACK SESSION

1. SHOULD HP SUPPORT THESE FACILITIES ON FUTURE POINT-TO-POINT TERMINALS?
 - A. HALF DUPLEX MODEMS
 - B. DIFFERING INPUT/OUTPUT LINE SPEEDS
 - C. PAPER TAPE MODE (TAPEMODE)
 - D. HP 2615 (MINI BEE) TERMINAL TYPE
 - E. TERMINAL TYPE 11
 - F. HARD COPY DEVICES WITHOUT BACKSPACEABLE CURSORS ("REAL" TELETYPE ASR 33's)

2. WHICH NEW TERMINAL FACILITIES WOULD BE VALUABLE FOR EUROPEAN USERS?

PRESENTOR

JIM BEETEM
INFORMATION NETWORKS DIVISION

****STILL WAITING FOR FULL TEXT****
(EDITOR)

DAISY 3000 -
A NEW APPROACH IN TEXT PROCESSING

TIMO RAUNIO

DAISY/3000

A NEW APPROACH IN TEXT PROCESSING

Once upon a time, Mr. Gutenberg discovered typography. It was a revolution in **information distribution**.

When IBM released the electronic typewriter with type ball and later advanced it by including magnetic memory, we were talking about **word processing**.

Only a few years ago did the rush of all kind of microprocessor controlled equipment with display units and floppy discs begin. Now we can use the term **text processing**, although their different features, hardware and prices cause confusion in our minds.

However, now we are standing at the front door of **integrated information processing**, which includes all conventional data processing as well as high quality printed output, electronic archives, automatic reproduction and mailing, etc. A key to slightly open that door is **DAISY/3000**, a program which fills the gap between your existing systems and high quality formatted output with reasonable costs, and meanwhile, satisfies all of your normal text processing demands, until we have "the paperless office".

WHAT IS DAISY/3000

DAISY is a high level **document description language** with which almost any kind of document can easily be produced. It fully utilizes the intelligence of the Diablo printer and Hyfeed sheet feeder. Over 70 basic level commands are recognized by DAISY and these commands are used to define even higher level **macro commands**. To obtain high quality, even very complicated, formatted output, the user only inserts a few names of these previously defined macros within the text.

In addition to all normal text processing features, some sophisticated capabilities, such as graphics, user-written procedures, conditional execution etc., are included.

COMMAND LEVELS

Printer level commands are escape sequences which are hard to remember and difficult to use. The user does not have to worry about them with DAISY.

Basic level commands have mnemonic names and are much easier to use. They also generate all of the required escape sequences for the desired operation. However, the user seldom has to be concerned about them, unless he is the person creating macro libraries.

Macro level commands are the main tools used when entering texts to be formatted. Macro commands are named and defined by the user. Thus, their names and function can be designed according to the user's demands.

MACRO LEVEL BENEFITS

As known in the other programming languages where user defined macros are available, the macros are the key to effective and structured programming. The same also applies to text processing and especially to flexible text entry.

Some noticeable benefits of macro level commands are:

- Short commands do a lot. Thus, complex series of commands or often used phrases can be referred to by a single name.
- Flexibility. Just changing the macro library will result in a totally different output format or changing a macro definition will have its effect throughout the text.
- Mnemonic names can be used in any language.
- Less errors. A correctly designed macro does it right always and there remain only typing errors which rarely can be avoided.
- Less work. Once designed, macros can always be used.
- The user can concentrate on WHAT he is writing, NOT HOW he is writing.
- No regressive way of thinking. Each macro can cancel all unwanted states set by other macros. Thus, each macro STARTS a block of text.
- No need to adjust the text on the display. The user simply writes the text and the macros always do the proper formatting.

RESTRICTIONS

DAISY/3000 is hardware dependant. It has been written in SPL, which is known only by HP computers. Conversion to PASCAL may be possible.

The escape sequences controlling the output devices have no standards, and neither do the features of printers. Therefore, only one printer could be selected as the output device, so that all of the advanced formatting features and special effects could be realized. Thus, formatted output is available only with the DIABLO printer.

Even if the escape sequences could be converted, a display terminal cannot show all the necessary formatting. An example: A very normal situation is to have margins at columns 9 and 79. With proportional spacing, this area contains 85-100 characters per line and an 80 column display will cut the line and could not possibly show the right justified margins.

DAISY: COMMANDS

This section describes the basic level commands by classes. More detailed information can be found in the "DAISY reference manual" and in the application notes.

Command classes are:

- Define files
- Define macros
- Modes of operation
- Size of page
- Vertical movement
- Horizontal movement
- Special effects
- Automatic titles and footnotes
- Graphics
- Automatic table of contents and keyword index
- Conditional commands
- User procedures
- Hyphenation
- Miscellaneous
- End
- Edit

The commands can be located anywhere within the text and are indicated by a "command character". Some commands have numeric parameters which can be any integer expression containing variables and constants.

USER PROCEDURES

If the capabilities of DAISY are not satisfactory, the user can write his own procedures or subprograms in SPL or FORTRAN. DAISY then loads such routines from the user's SL file when desired and the procedure can return commands and/or text for DAISY to process.

This feature allows direct linkages to existing systems, data bases and registers. Also, user-written "preprocessors" for the input can easily be done to totally change the default syntax of commands.

INTERACTIVE APPLICATIONS

User-written procedures can be used to obtain interactive text processing applications. However, DAISY itself includes commands to make conversation with the user.

In many cases, it is desirable to alter some of the formatting parameters during execution. Form size and printwheel characteristics are two such parameters. Also, you may have a fixed text where only a few words should be changed from time to time. Editing of such text is unnecessary if you use interactive commands.

EDIT

There are several methods to create input files or write texts for DAISY. One way is EDIT/3000, which as known is a very powerful tool when entering or editing any kind of source texts. EDIT/3000 is available in the normal way or as a son process of DAISY through the EDIT command.

Although EDIT/3000 is quite a meritorious program, it lacks a few desirable features when entering and editing texts in human languages. Therefore, DAISY has a built-in, full-screen editor which operates in block mode and fully utilizes the local edit facilities of HP2640 series terminals. (2620 series terminals are under investigation).

DAISY IN PRACTICE

When compared to those stand-alone micro-monsters, DAISY has a totally different theory of operation. Which one is better depends on what you are doing. If you are only replacing a typewriter, DAISY may be too sophisticated. But, if you want real text processing, the macro level commands show their power. And if you are planning linkages to your existing systems or to raise your integration level, there are not too many other possibilities available.

EXAMPLES

The following four pages show some examples of DAISY applications. They are not examples of any ordinary text processing applications which any system can do. Rather, they show some of the flexibility and graphical power of DAISY.

The first example shows part of an application note which describes a convenient way to draw flowcharts. This macro library consists of about 30 defined macros, 20 of which are not visible to the user. The flowchart in the example was drawn with 12 lines of input, using 9 of these predefined macros.

The second example shows the usage of multiple printwheels. It was printed with 3 different wheels: CUBIC PS, APL and SCIENTIFIC.

The third one is an example of multicolumn printing.

The last one shows mathematical expressions with user-defined, special-symbol macros, using graphical mode.

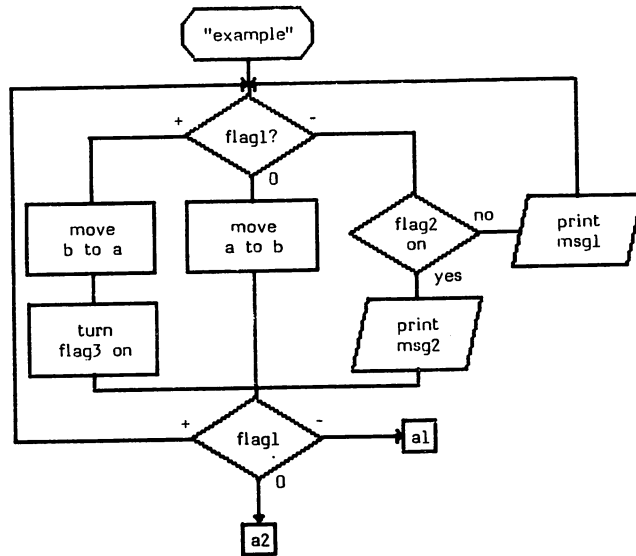
DAISY is developed in HELECON, HELSINKI SCHOOL OF ECONOMICS

Distribution: Oy PORASTO Ah
Töölöntullinkatu 8
00250 Helsinki 25
FINLAND
Telex: 125194 PSTO SF

DAISY: application note 2/81

HOW TO MAKE FLOWCHARTS WITH DAISY

Macrolibrary "FLOWCH" provides a convenient way to draw flowcharts. Example:



User input defines which symbols are used and where they are located, text within symbols and where to draw lines.

Daisy automatically calculates the form of symbols, horizontal and vertical centering of the texts, start and end points of the lines, location of arrowheads, location of "yes/no-like" labels around the "if" symbol and the scale of the picture.

2.2 Ensimmäisen kertaluvun predikaattikalkyyli

Nimikkeistö L koostuu joukosta vakio- funktio- ja predikaattisymboleja. Jokaiseen funktio ja relaatiot symboliin liittyy kokonaisluku, joka ilmaisee symbolin paikkaluvun. Täsmällisemmin,

$$L = (\text{Const}, \text{Func}, \text{Rel}, \text{ar}),$$

missä

Const -- vakiosymbolien joukko,

Func -- funktiosymbolien joukko,

Rel -- relaatiot symbolien joukko ja

$$\text{ar}: \text{Func} \cup \text{Rel} \rightarrow \mathbf{N}.$$

Kun $\text{ar}(f) = n$, $f \in \text{Func}$, sanomme, että f on n -paikkainen funktiosymboli. Vastaavasti, kun $\text{ar}(r) = n$, $r \in \text{Rel}$, sanomme, että r on n -paikkainen relaatiot symboli.

Nimikkeistön $L = (\text{Const}, \text{Func}, \text{Rel}, \text{ar})$ struktuuri on pari $M = (M, I)$, missä M on ei-tyhjä arvojoukko ja I on kuvaus joukolla $\text{Const} \cup \text{Func} \cup \text{Rel}$, siten, että

$$I(c) \in M \text{ kun } c \in \text{Const},$$

$$I(f): M^n \rightarrow M \text{ kun } f \in \text{Func}, \text{ar}(f) = n \text{ ja}$$

$$I(r) \subset M^n \text{ kun } r \in \text{Rel}, \text{ar}(r) = n$$

Käytämme usein merkintää $c^M = I(c)$, $f^M = I(f)$ ja $r^M = I(r)$.

Esimerkiksi ryhmäteoriassa käytetään nimikkeistöä

$$L_1 = (\{0\}, \{+, 0\}, \text{ar})$$

missä $\text{ar}(+) = 2$. L_1 :n struktuuri on pari $M = (M, I)$, missä $I(0) \in M$ ja $I(+): M^2 \rightarrow M$.

Lukuteoriassa taas käytetään nimikkeistöä

$$L_2 = (\{0\}, \{S, +, *\}, 0, \text{ar}),$$

missä $\text{ar}(S) = 1$ ja $\text{ar}(+) = \text{ar}(*) = 2$. Eräs tämän kielen struktuuri on $M = (M, I)$, missä $0^M = 0 \in \mathbf{N}$, $S^M(n) = n+1$, $+^M(m, n) = m+n$ ja $*^M(m, n) = m \cdot n$, ts näillä symboleilla on standardi merkityksensä.

Seminaariharjoitukset tapahtuvat pienryhmissä (yhteensä 8-15 henkilöä). Niissä käsitellään keskustelemalla joko teoreettisia tai käytännöllisiä kysymyksiä. Seminaariharjoitusten yhteydessä opiskelijat laativat usein myös esitelmiä keskustelun pohjaksi ja joissakin aineissa käsitellään seminaareissa myös heidän tekemiään tutkimussuunnitelmia. Seminaareissa on hyvä tilaisuus oppia tieteellisen keskustelun periaatteita ja tutustua oppiaineen metodiiikkaan. Ne auttavat tehokkaasti syventymään omaan aineeseen.

Ei ole liioiteltua väittää, että useimpien akateemisten aineiden opetus on nykyään niin tehokasta, että opiskelijoiden todella kannattaa seurata sitä niin paljon kuin mahdollista. Oppitunteihin, kokeellisiin töihin ja harjoituksiin osallistuminen on useimmiten tentissä onnistumisen välttämätön edellytys. Harjoitukset antavat tarpeellisen taidon selviytyä soveltavista tentteistä, ja seminaareissa saa tärkeää ohjausta tieteelliseen työhön.

1.3.4 Aktiivisuuden merkitys

Ratkaisevaa on kuitenkin, että opiskelija ei vain "istu" erilaisissa opetustilaisuuksissa. Ainoastaan osallistumalla aktiivisesti työskentelyyn hän voi saada jotakin irti opetuksesta. Sen lisäksi hänen täytyy käyttää hyväkseen jokainen tilaisuus keskusteluun opettajan kanssa ja tehdä tälle kysymyksiä. Opettajat yleensä arvostavat sitä, että opiskelijat seuraavat heidän opetustaan aktiivisesti ja kiinnostuneina.

1.4 OPINTOJEN SUUNNITTELU

1.4.1 Pitkän tähtäyksen ohjelma

Tärkeimpiä johtopäätöksiä, joita voidaan tehdä teoriaosan perusteella, on se, että yhden ja saman aineen liian keskitetty oppiminen pitkähkön ajanjakson aikana tuskin on järkevää. Sen tähden on hyvä suunnitella vaihteleva pitkän ajan ohjelma, jotta välttyttäisiin tehotonmita ja usein masentavilta oppimistasanteilta.

Tehokkaan opiskelumenetödiikan tärkeä apuväline on siis mahdollisimman yksityiskohtainen suunnitelma. Pitkän tähtäyksen suunnittelun on havaittu vaikuttavan positiivisesti opiskeluun. Jokainen saavutettu välitavoite antaa opiskelijalle tyydytyksen tunteen ja lisää opiskelutarmoa auttaen siten voittamaan seuraavan tehtävän mukanaan tuomat alkuvaikeudet.

Päämäärän tulee luonnollisesti olla realistinen: jos on asettanut itselleen liian suuret vaatimukset - toivoen siten kannustavansa omat kykynsä äärimmilleen - alkaa jonkin ajan kuluttua tuntea haluttomuutta huomattessaan, että päämäärää ei voikaan saavuttaa. Itseltään liian vähän vaatimien veltostuttaa ja opiskelija oppii vielä vähem-

män kuin oli suunnitellut. Tehokkainta on asettaa päämääränsä siten, että sen voi tavoittaa maksimaalisen ponnistelun avulla. Sen on siis oltava saavutettavissa, mutta vasta todellisen työpanoksen jälkeen. Ei riitä, jos suunnittelee "ehtivänsä niin ja niin pitkälle aineessa ensimmäisten viikkojen aikana". On vähintään yhtä tärkeää yrittää suunnitella työnsä yksityiskohdat viikko viikolta ja päivä päivältä.

1.4.2 Lue sopivasti

On niinkään hyödyllistä kehittää kiinteä työskentelytapa. Myös työskentelytapa on suunniteltava realistisesti siten, että sitä voidaan noudattaa pitkän ajanjakson kuluessa. Sen tähden on työskentelysuunnitelmaan jätettävä tilaa esim. urheilulle, ostosmatkoille, kirjallisuudelle ja sanomalehdille, järjestötoimintaan osallistumiselle, elokuvissa ja teatterissa käynnille ja muille virkistysmahdollisuuksille. Näin välttytään suunnitelman alituiselta muuttamiselta ja ennen kaikkea säästytään siltä tunteelta, että olisi "epäonnistuttu" hyvissä aikoimuksissa. Voit esimerkiksi hankkia päiväkirjan, jossa jokainen päivä on jaettu tunteihin, ja merkitä siihen suunnitelmasi. Usein on sopivinta aloittaa merkitsemällä tenttipäivät, ja sitten laskea ne tenttejä edeltävät päivät, jotka tarvitsee kertaukseen ja sen jälkeen jakaa jäljellä oleva aika niiden kurssin osien kesken, jotka täytyy suorittaa, jotta ohjelma pitäisi paikkansa.

Yliopistoissa ja korkeakouluissa tiettyyn järjestelmään sidottu opetus on muotoiltu siten, että se vaihtelee viikottain. Silloin on vielä jokaista viikkoa harkittava erikseen. On kuitenkin miltei aina mahdollista sovittaa opiskelutö kiinteisiin aikoihin ja täten luoda itselleen työskentelyrutiini, jota voi noudattaa.

1.4.3 Suunnittele yksityiskohtaisesti

Ajankäyttö ei suinkaan ole ainoa asia, joka pitää suunnitella sopivaa kaavaa noudattaen. Myös opittava aines tulee organisoida hyvin. Kurssin jokaisen osan kohdalla on suunniteltava, kuinka paljon aikaa kuluu kurssikirjojen, muun kirjallisuuden ja luentomuistinten lukemiseen, paljonko aikaa kuluu mahdollisiin harjoitustehtäviin ja esitelmien laatimiseen, ja lopuksi, kuinka paljon aikaa on varattava kertaukseen ennen loppuentä. Kertaamiseen varattu aika on usein tehokkaammin käytettyä kuin alkuperäinen oppimisaika. Edelleen voi olla hyvin edullista lukea eri kirjoja tai kurssin kohtia rinnakkain.

1.4.4 Realistinen päämäärä

Akateemisten opintojen suunnittelussa on erittäin tärkeää, ettei opiskelija yliarvioida omaa opiskelukapasiteettiaan. Suhteellisen vapaa opiskelu tarjoaa kyllä mahdollisuudet tähän. Monet opiskelijat aloittavat liian optimistisesti monta ainetta

the mean percentage mean absolute error

$$M = \frac{1}{n} \sum_{i=1}^n \left(\frac{1}{T} \sum_{t=1}^T \left| \frac{A_{it} - P_{it}}{A_{it}} \right| \right) \cdot 100 \quad \begin{matrix} t = 1, \dots, T \\ i = 1, \dots, n \end{matrix}$$

and

the mean percentage root mean square error

$$R = \frac{1}{n} \sum_{i=1}^n \left(\frac{1}{T} \frac{\sqrt{\sum_{t=1}^T (A_{it} - P_{it})^2}}{\sqrt{\sum_{t=1}^T (A_{it} - \bar{A})^2}} \right) \cdot 100 \quad \begin{matrix} t = 1, \dots, T \\ i = 1, \dots, n \end{matrix}$$

are used.¹

3. It is possible that an equation has a good statistical fit, but a poor tracking fit. This is due to the dynamic properties of the model which bear little relation to the way individual equations fit the historical data. It is for this reason that M and R tracking errors are an important criterion for the evaluation of a multi-equation model.

4. In the following comparisons of calculated vs. observed variable values will be performed between the AJKA and the ETLA models and trend forecasts:

- 1 for periods 1958-78, 1971-78 and years 1977 and 1978
- 2 for 22 common endogenous variables of AJKA and ETLA
- 3 for some of the most important variables separately.

¹ These are documented e.g. in Johnston, H.N., Klein L.R. and Shinjo K.: 'Estimation and Prediction in Dynamic Econometric Models', Chapter 2 in Sellekaerts, and in Pindyck and Rubinfeld.

Presentation Abstract

USING THE HP 3000 AS A MAINFRAMEPresentation Title: "USING THE HP 3000 AS A MAINFRAME"(Follow up to the presentation about that made in Montreux)Author(s): Carl Christian Lassen M.Sc.Title(s): DirectorAddress: DANSK OVERSØISK MOTOR INDUSTRI - DOMISDR.Ringvej 35, DK 2600 Glostrup, DANMARK

CARL CHRISTIAN LASSEN M.Sc.

Abstract: (No more than 200 words)

Running 28 different applications with more than 100 users situated all
over the country, working in the automobile import and resale trade,
and in the agricultural machine trade, forces you to design the appli-
cations with regards to the many different opinions about what is needed
and what is not.

This presentation will give you an idea about our philosophy on Systems
Development, on buing packages, programs or projects.

It will present our philosophy on Real Time versus Batch processing
and our implementation of Semi Batch tasks. (continued on next page)

C.C.LASSEN
 DANSK OVERSØISK MOTOR INDUSTRI - DOMI
 SDR.RINGVEJ 35
 DK 2600 GLOSTRUP
 DANMARK



It will present our philosophy and implied problems with the use of IMAGE, KSAM and MPE files, and how we change a system from one to another - if it seems relevant.

I will comment about our methods of squeezing more through the machines, how we find out which parts of the applications are the bottlenecks and how we reduce the resources used in an application, using among other NOBUF IO, MR, blocking to sector boundaries, reprogram to SPL etc.

I will comment about the various ways of handling terminal dialogs, and on the SL compared to stand alone programs.

I will present a method of changing the package from a set of stand alone programs to one master program handling the outer routines - database open etc - and keeping the programs as code segments. (This reduces overhead from the domain of users changing from one program to another.) It is a result of our development strategy and optimization.

(ABSTRACT)

DATA CAPTURE SYSTEMS FOR REAL-TIME MANUFACTURING MANAGEMENT

By

BRUCE TOBACK

CONTENTS: DESIGN AND IMPLEMENTATION OF A DATA CAPTURE SYSTEM
IN A MANUFACTURING ENVIRONMENT ,
HARDWARE, SOFTWARE, USER INTERFACE CONSIDERATIONS

BRUCE TOBACK

A.L.S. CORPORATION

***** ASK AUTHOR FOR FULL TEXT, WE DIDN'T FIND ONE ... (EDITOR)

H P PLUS

J. GRAHAM

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

J. GRAHAM
HENLETT PACKARD

COLD-DUMP-ANALYSIS

M.S. PAIVINEN
S. MASTRIPIERI

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

M.S. PAIVINEN
S. MASTIPIERI
HEWLETT PACKARD

SOME PROBLEMS OF SOFTWARE ENGINEERING

Wladyslaw M. Turski
Institute of Informatics
Warsaw University

WLADYSLAW M. TURSKI

PROF. W.M. TURSKI
INSTITUTE OF INFORMATICS
WARSAW UNIVERSITY
WARSAW, POLAND

The phrase "software engineering" was coined just over 13 years ago. It was considered a little provocative by its originators and was chosen explicitly for this reason. The group of conveners of the Garmisch (October 1968) and Rome (October 1969) conference sponsored by the NATO Science Committee deliberately selected a key phrase that contrasted with the then prevailing perception of software issues ("software chaos" and "software crisis" being then two phrases very much in vogue). The phrase "software engineering" hinted at law and order in an environment considered hopelessly anarchic, promised some rigour and discipline where there was none, whiffled of industrial approaches in the field where artisanship reigned unchallenged. The phrase "software engineering" evoked - no matter how nebulously - notions of standards, measures of productivity, industry-wide and commonly accepted codes of good practices. Unfortunately, the launching of a phrase seldom solves any real problems and sometimes creates new ones, primarily those of credibility.

Software engineering caught on. Conceived as an intellectual provocation by the mostly academic animators of the Garmisch-Rome conferences, the phrase was eagerly accepted by all sorts of industrial organisations. The younger set (and information processing community expands so quickly, that it constantly seems to consist more of the younger set) is now firmly convinced that software engineering is a well-established industrial discipline, opposed or at least neglected by the academia. (A year ago I witnessed a most revealing scene: at a computer-oriented gathering on the antipodes, a very articulate, but clearly not very well-read, industrial programmer addressed a professor of computing science with a question he obviously thought rethoric: when at long last, would the universities recognise the existence of software engineering, accept the inevitable and start doing something useful in this direction. The object of this tirade was the person who can justly be called the spiritus movens of the 68/69 conferences, one of the authors of the phrase "software engineering". Unlike social ones, this technological revolution turned against its fathers.

The software crisis of the 60's was very real. The quality of most computer programs was very poor, their documentation - at the very best - sketchy, the reusability of software components - practically unheard-of, software systems in use were growing patchier and patchier, software projects were notoriously behind schedule and above budget. The shortage of skilled programmers was crippling, good people were hard to get and once gotten would very often be soon bribed away. Programming languages were just about the only facet of programming sufficiently well-understood to have a comprehensive and consistent theory (even in this case the theory covered only part of the problem, i.e. syntax, leaving out the perhaps more important subject of programming language, which is semantics).

Ideally, software engineering should have cured all these and similar ills. Unfortunately these phenomena - very real ones - are merely symptoms of much deeper troubles and since most efforts directed at remedying the symptoms do not cure the sickness that causes them, even if taken jointly, all prescriptions for ameliorating particular aspects of the software crisis do not seem to appreciably improve the state of affairs. Even though the expression "software crisis" is by now nearly forgotten, almost all the complaints made fifteen years ago could be repeated today, perhaps more strongly. There is, however, one notable and very important exception: we have learned how to write correct programs given precise specifications.

I am fully aware that the ability to make correct programs given precise specifications is but a small consolation for someone faced with the full scope of issues involved in providing the software part of a computerised system, nevertheless I am going to spend some time analysing this achievement. I am going to do so for several reasons:

- (i) because it shows what it takes to solve a problem in the area of software engineering,
- (ii) because it bares some fundamental deficiencies of the common sense approach to software problems,
- (iii) because it demonstrates the intellectual techniques involved in solving a methodological problem in software construction,
- (iv) because it is the only part of software engineering that can be completely discussed in scientific terms.

Let us first consider why instead of a simpler expression "correct program" we are using a longer (and, admittedly, clumsier) one: "program correct with respect to its specification".

The only rational interpretation of the shorter term "correct program" could relate to the internal correctness of a piece of code, i.e. to its syntactic correctness. For a reasonable grammar the question of syntactic correctness can be solved quite mechanically: every acceptable compiler does it all the time. On the other hand, if the syntactic correctness of a program cannot be resolved mechanically, we are dealing with a programming language too ambiguous for any semantic interpretation, with a text too vague to be considered meaningful. (A more puritan view would be to state that syntactic unresolvability precludes semantic modelling and thus such texts are meaningless.) Since it is safe to assume that we are willing to restrict our considerations to programs that are guaranteed to be meaningful, i.e. to such programs that are guaranteed to have a nontrivial semantic model, we assume that programs we are considering are guaranteed to be syntactically correct.

As soon as we decide that we wish the term "correctness" to express more than just syntactic correctness, we must look for an external frame of reference. (Nothing surprising in it: in a scientific sense, correctness, just as truthfulness - in a slight departure from common usage - is always a dyadic relation that holds, or does not, between two entities rather than being a property enjoyed, or not, by an entity taken in isolation.) For a meaningful, purposful, useful program, the frame of reference that seems most natural for establishing the program's correctness is its specification, i.e., a statement of the program's purpose.

thus we consider a program and a specification. For some such pairs we are entitled to state that program P is correct with respect to the specification S. The verification of whether a particular pair (P,S) entitles us to make this statement depends, of course, on a great many additional considerations. (Incidentally, I am afraid that some readers are beginning to suspect that I am overly pedantic, that I indulge in an academic nitpicking; let me hasten to assure them that my concern is very practical: it is precisely because I am interested in useful programs that I am rather particular about expressing my objectives quite clearly and opt for writing a watertight warranty that the program will indeed satisfy my goals as expressed by the specifications.)

We leave aside, for the moment, the objections that one seldom is interested in actually verifying if an arbitrary program P is correct with respect to specification S, that our main interest is in producing programs that enjoy this property vis a vis given specification. After all, unless we have a better procedure, a thorough quality assurance test is a purchaser's best friend!

Perhaps the simplest form a program specification can take is a pair of statements, IN and OUT, each of which can be either true or false, depending on the environment in which it is to be evaluated. (Some sentences, tautologies, are true (or false) regardless of an environment; if we fancy it, we may introduce the statement TRUE which - by definition - is true in every environment, and the statement FALSE - false in every environment, these two statements are surprisingly useful!) Using an IN/OUT pair we express the following request: we want a program P such that if its execution starts in an environment in which IN is true then after its execution is completed we shall get an environment in which OUT will be true. This request can be written as a

formula:

$$(IN) \ P \ (OUT) \quad (x)$$

in which P is an unknown, or desired, program. Thus this formula may be considered as an equation that defines program P.

For example,

$$(integers \ x, \ y \text{ are defined}) \ P \ (integers \ x, x, m \text{ are defined} \\ \text{and } m = \max(x, y))$$

is a specification for a (small) program P that culls the maximum of two given integers.

Now, how do we proceed to verify that a given program P is correct with respect to its IN/OUT specification? There are several ways of doing it, depending on the particular fashion in which semantics of the programming language employed for coding P is formulated. If we are to proceed at all, however, the semantics of this language should be expressed in such a way as to permit calculations of the environment transformations effected by the execution of the language instructions. Knowing how the execution of each instruction transforms its inherited environment into the environment for its successor, we can ascertain if the execution of P, starting from its first executable instruction initiated in the environment satisfying IN, will lead, transformation by transformation, to an environment satisfying OUT. Thus we can establish the correctness of P with respect to IN/OUT by proving the conjecture that arises when the text of program P is substituted in (x). Note that the ability to carry out this proof depends on the ability of a rigorous definition of the programming semantics.

For example, with IN and OUT as before, and

$$P = \underline{if} \ x > y \Rightarrow m := x \\ y > x \Rightarrow m := y \ \underline{fi}$$

we get the conjecture

$$(integers \ x, y \text{ are defined}) \\ \underline{if} \ x > y \Rightarrow m := x \\ y > x \Rightarrow m := y \ \underline{fi} \\ (integers \ x, y, m \text{ are defined and } m = \max(x, y))$$

There are two varieties of thus understood correctness: partial and total. The distinction relates to the fact that some programs are known to contain endless loops, or - what is more realistic - to contain instructions which initiated in some environments may loop forever. Partial correctness amounts to saying: it is guaranteed that provided the execution of P comes to a normal end the conjecture (x) holds. Total correctness amounts to a stronger: it is guaranteed that the execution of program P will come to a normal end and that conjecture (x) holds.

The investigations into the extent of the notion of program correctness led to many useful programming techniques. Methods of composing programs in such a way that their correctness with respect to given specifications would be guaranteed by virtue of construction steps taken were developed and made quite practical. The key to the success of these developments was the appreciation of the calculability of environment transformations effected by the execution of programming language instructions with well-defined semantics. This in turn led to a considerable effort in formulating calculable semantics of programming languages, and - in due course - to certain preferences in programming languages themselves.

The intuitive approach to programming language design (wouldn't it be nice if we had such and such feature in our language) was replaced by a more somber attitude: let's have in our language only such constructs which have calculable semantics, and preferably select those whose definitions make semantic calculations easy.

In recent years, many techniques based on calculable semantics and on the principle of provable program correctness with respect to its specification emerged and found practical application. Even if the practiced version of a programming technique is not explicitly calculational (structured programming, stepwise refinement, Jackson method etc.), their origin is unmistakable and their soundness depends on the firmly established mathematical theory of program correctness.

It is often said that the formal methods of program verification and/or program derivation from specifications are applicable to small problems only, or less kindly spoken, toy problems. Two justifications are put forward in support of this thesis. The first one points out that the volume of formal manipulations needed to verify a program is usually an order of magnitude larger than the volume of the program text itself, which makes this approach impractical for large problems. The second one questions the basic premise of the method - the availability of precise specifications. Both objections are well-founded; the second one is however much more serious and will be dealt with somewhat later, in a broader context.

As far as the length of the verification proofs is concerned we should in all fairness observe that the verification of an existing program against an existing specification is a relatively infrequent event. A much more realistic approach is to use the calculable semantics for deriving the program.

The length of formal manipulations involved is still quite impressive, but in this case the effort spent on "formal manipulations" should not be considered as an addition to the cost of program development. If a programmer develops a habit of formally deriving programs from specifications, then all his activities related to program construction, indeed, the whole problem-solving process is carried out by these formal manipulations. Starting with the necessary problem analysis and derivation of auxiliary facts, through structural analysis, decomposition and linguistic interpretation (stepwise refinement), and ending with final expansion (coding) - all these steps, which one way or another must be present in program derivation, are combined into a formal derivation of a program. Seen from this point of view the length of the derivation is a measure of the effort needed to properly construct the program. As usual, the derivation may be more or less detailed, some people learn to perform in their heads longer transformations than others, but the fact remains: formal derivation of programs is not any longer than an informal one. It is, however, more explicit, provides a better documentation and is a whole lot less vulnerable to a chance mistake or oversight. In a sense, it is a pity that the published examples of formal derivations of programs - for didactic purposes - refer to very simple problems only: because it is so easy to derive the specified programs in one's head, an explicit formal protocol of the derivation seems too long and, perhaps, unnecessary.

The relationship between a specification and a program is not a function: given a specification there may be a great many different programs that satisfy it, i.e. are correct with respect to this particular specification. If the specification is too vague, i.e. if it does not capture all important requirements, a correctly constructed program may turn out to be not quite satisfactory. This raises a very

controversial issue of the extent of programmers' responsibility: does it cover a verification of the specifications? And if so, what is the frame of reference to be used in such a verification? With this problem, however, we are leaving that part of software engineering which could be conveniently called programming methodology, the only part in which solid progress over the last decade can be reported.

Contrary to a popular belief, the completeness of specifications - at least the mathematical completeness - would not necessarily be an unconditional blessing. First of all, it would be very difficult to achieve, secondly, it would almost invariably amount to overspecification in terms that matter for the ultimate use of the program.

Deriving a program to meet a given specification, a programmer is free to use his judgement, rely on his expertise, draw upon available knowledge and resources, make decisions in all issues that are left unspecified. For example, a specification may read as follows:

IN: A_1, \dots, A_N is a defined sequence of integers.

OUT: (i) B_1, \dots, B_N is a sequence of integers and
(ii) B_1, \dots, B_N is a defined permutation of
 A_1, \dots, A_N and
(iii) $i < j \Rightarrow B_i = B_j$ for all $1 < i, j < N$

This is, of course, a specification for a sorting program. The choice of the sorting algorithm is left unspecified, the programmer may explore this freedom as he wishes - provided the program he produces satisfies the given specification. If the specification was a bit "more complete" and asked, for instance, that the cost of the program execution should not exceed $kN \log N$, a large class of algorithms would be

excluded; similarly, if it was specified that the programs should not use more than 10% extra memory on top of the N cells needed for the vector A . Observe, however, that in order to express such additional constraints on the program, the specification must be formulated in a language much richer than that needed for the initial one: the extended specification must encompass notions quite alien to the problem of sorting. (Incidentally, the IN/OUT style of specifications does not lend itself very naturally to such additional specifications, but this is a relatively minor point.)

In aiming for completeness of specifications great care must be exercised that their consistency be preserved.

The inconsistency of specification is much more harmful than its incompleteness. An incomplete specification can be satisfied by many different programs, the only danger being that the one actually derived would not meet some unspecified requirements (while being in full accord with the specified ones!) An inconsistent specification cannot be met by any program! (In our example, it suffices to extend the given specification by the request that the cost of executing the program be less than kN , for fixed k and N , to make the thus extended specification inconsistent.)

Thus a modicum of incompleteness of the specification is harmless (and in practice unavoidable), whereas the inconsistency must be categorically avoided.

The ability to produce correct programs given consistent specifications, the ability gained through calculable formalisation of semantics of programming languages, has caused a marked shift of research interests away from issues of programming languages, towards the issues of specification.

There are several ways in which this relatively new research topic is likely to produce results important for practical work. (In some of these directions considerable progress has already been made, and practically significant results and techniques are available.)

The directions closest to the traditional programming activity is that of formalisation of program-objects specification (such as data types, modules, monitors etc.). In fact, the methods of specifying these objects are so closely related to programming techniques, that frequently they are considered simply as parts of programming methodology. Yet, it is worthwhile to observe that the same techniques may be applied to specification of objects not necessarily related to programs.

Consider, for example, the abstract data type specifications. In the briefest possible exposition, one could say that an abstract data type is specified by two sets of formulae:

- (i) syntactic rules,
- (ii) semantic equations.

The syntactic rules describe the morphology of objects of the type being defined and the syntax of specified operations acting on these objects; the semantic equations express - in calculable fashion - the properties of objects and operations. The publicised examples relate to well-known program objects (stacks, queues, tables, etc.) but the very same technique can be applied to specification of any objects that can be abstractly viewed as many-sorted algebras. In fact, since there is no fundamental reason why this technique could not be applied to, say, data base specification and since (as we are repeatedly told) data

bases can be used to faithfully (well, sufficiently faithfully) represent almost anything of interest in business data processing, I see no reason whatsoever why the abstract data type specification techniques could not be applied to specification of software, e.g., for management information systems. (Indeed, some experimental results in this spirit have been reported in research publications.)

Similarly, the techniques used for specifying active software components, such as modules, monitors, and classes, can probably be used for specification of simulation software. In fact, since most of the work in this direction is based on the original contributions of SIMULA 67 - a language initially intended for programming simulation computations - using these techniques for specification of simulation software would in a sense complete a full cycle of development, which is always intellectually pleasing!

Several projects are currently under way trying to combine various specification techniques into specification languages, or more precisely, into software specification languages.

The most important advantage of specification languages based on formal specification techniques would be the availability of an extensive calculable apparatus enabling the verification of software produced according to the specifications expressed in such languages. Let me once more stress the importance of such an apparatus. Unless the notation of satisfaction is formalized, it cannot be made calculable. And unless we have a calculable means of establishing whether a piece of software satisfies the specification, we are on the very shaky ground of debugging, test-case verification etc., which never leads to foolproof assurances. Recall that it was the introduction of calculable semantics of programming languages that made

possible a satisfactory interpretation of the program correctness problem, and, as a consequence, led to programming methods that guarantee the program correctness.

Another - and in a way no less important - advantage of formalized software specification languages rests in the ease with which they permit the construction of assorted aids, facilitating the process of programming by performing a host of clerical functions (cross-referencing, indexing etc.), by executing various checks (inconsistency of interfaces, use/define matches etc.) and - in some instances - by simulated "execution" of specified, but not yet fully programmed, software. Especially in large software projects such aids reduce the burden of ancillary functions on programmers and thus increase their productivity by allowing a less diluted concentration on main tasks. Again it should be stressed that the formalisation of the specification language (both of its syntax and semantics) is the crucial factor in determining how extensive a set of aids can be constructed.

Another direction of research on specifications concerns operation on specifications, such as extending a specification by additional requirements and joining two specifications into one. Such operations closely correspond to situations frequently encountered in practice; in fact, as we shall see in a moment, manipulations with specifications are about the most important tool in software engineering. In order to attach meaning to results of formally defined operations on specifications, a suitable formal view of specifications had to be established. This was accomplished by considering specifications as algebraic theories, in which case the theory of categories provided the necessary framework. Burstall's language CLEAR has been explicitly designed for describing specifications as theories and for programming operations on them.

All the so-far discussed research directions on specification issues implicitly assume that the specifications are the ultimate source of inspiration for software. This view is clearly inadequate for practical applications, where the ultimate source of inspiration is a need felt by the customer, a need usually poorly articulated and nearly always expressed in terms far removed from program-oriented terminology.

Theoretically, we could argue that the steps necessary to convert such a nebulously formulated need into a specification for a software (system) do not belong to software engineering. Personally, I do not subscribe to this view. First of all, if the pre-specification steps are excluded from the scope of software engineering we shall not have any real control over their quality, and there is very little sense in making an effort to produce high quality software hanging on low quality specifications. Secondly, it is in the link between the customer's needs and specifications that the most troublesome aspects of software engineering have their roots (as we shall see in a moment). Thirdly, an interface between the pre-specification problem analysis and the specification must be established: if we admit that the former is quite informal and the latter - formal, the interface could be extremely awkward.

One way of extending the software engineering towards the analysis of customer's needs consists in providing semi-formal tools (such as, e.g. SofTech's SADT) to be applied to the analysis of customer's requests expressed in his language. The use of such tools imposes certain discipline on the formulation of the need, mostly syntactic and structural, thereby establishing syntactic relationships between various structural entities. Roughly speaking, on successful application of such various tools we get a

counterpart of syntactic rules of algebraic specification, albeit sometimes expressed in a form much less convenient for subsequent manipulations (in the case of SADT we get a pictorial presentation of syntactic rules). The semantic equations are not so easy to obtain by semi-formal analysis.

One can point out an analogy of a sort between the semi-formal requirements of analysis and flowcharting as a means of program design. It certainly is a step forward with respect to totally informal (unstructured) analysis, but without formalisation of corresponding semantic notions we are still left without means to verifiably establish the correctness of our proceedings. It should be observed that the commercial success of semi-formal techniques in customers' problem analysis provides an empirical proof of the practical recognition of advantages extending software engineering outside the specification/program bracket.

An alternative approach to the analysis of customer's problems has been motivated by considerations of software evolution. It is a well-known fact that software systems in continuous use over extended periods of time evolve. The causes of evolution fall roughly into two categories: internal and external.

Internal causes of software evolution include two major classes:

- corrections
- improvements.

Corrections are such software changes that remove discovered errors, i.e. violations of the satisfaction relationship between an existing specification and an existing program. Theoretically, if the software is correct with respect to

its specification, no corrections are necessary. In practice, they do occur, just as errata are occasionally necessary to texts of mathematical proofs.

Improvements are such software changes that leave the satisfaction relationship between the existing specification and program intact and exploit the freedom left by the specification in order to bring about advantages that cannot be described in the linguistic system employed for the specification. A typical improvement is the replacement of an algorithm by a less complex one or by a "faster" one.

External causes of software evolution may be also classified into two groups:

- related to the change of programming environment,
- related to the change of specification.

A change of programming environment occurs e.g. when the hardware system is extended by a new component or a new hardware facility is added that extends the repertoire of programming means of expression. A more radical change of programming environment is caused by replacement of hardware, in which case (all or some) programming means of expression lose their hardware interpretation. An extreme case of programming environment change is a switch over from one programming language to another, or from one operating system to another, or from one manufacturer's hardware to another's. It should be observed that:

- (i) a change in programming environment does not necessarily involve a change in software (e.g. we may ignore an added hardware facility), in which case the situation is roughly similar to that of internal improvements,

- (ii) if the change of programming environment is forcing a change in software, such change is to be effected under the invariance of specification, excepting the somewhat ludicrous cases where the specification contains the explicit request that products of ABC company are to be used.

Thus we have isolated the only kind of software change that is caused by an as it were spontaneous change in specifications. Why should a specification change at all? Well, there are several reasons:

- (i) the original specification poorly captured the customer's needs,
- (ii) the customer changes his mind,
- (iii) the use of the system changed the customer's environment in such a way that his needs have materially changed.

All, who ever participated in a significant software development project, are well familiar with at least one of these, more often - with all.

In the professional jargon, the activity of changing the software is circumstantially called "software maintenance". It is a particularly absurd choice of terminology to speak of software maintenance when we mean software changes; unfortunately it is being done all the time! The use of this misnomer is also exceptionally harmful from the psychological point of view: it is subconsciously expected that any maintenance should be relatively inexpensive (in comparison with the original investment). Software "maintenance" being anything but inexpensive, the software

modification activities are constantly challenged as wasteful and poorly managed. It would be much better to speak of specification maintenance in the presence of internal causes of software evolution and in the case of a change in programming environment, and to call a spade a spade in case of specification changes.

Two symmetrical errors, commonly committed, contribute to the bad reputation of software evolution:

- (i) forgetting to maintain the specification when software is changed for internal reasons,
- (ii) specifying software changes without making certain that the resulting specification is consistent.

An often encountered variation of the second error consists in making software changes when the customer's needs have changed, without bothering to modify the specification at all. This practice is supported by the belief that "software models the application", hence if the application changes, the software should change accordingly. In fact, the software is related to an application through the specification, and if the verb "to model" is to be used in its technical sense, then software models its specification. Thus if we want to stabilize the software evolution, we must jealously maintain the specification-software relationship, and allow such software changes only which either verifiably preserve this relationship under invariance of the specification, or re-establish this relationship when the specification has been modified in a consistent way.

The relationship between the specification and software being thus promoted to the role of main concern of the programmer, how do we envisage the pragmatically important

link between the customer's needs and the - necessarily formal - specification? In order to achieve a pleasing symmetry and a simple, conceptually unifying treatment of both considered environments (the program environment in which a program models a specification, and the application environment) I suggest that the particular application be considered a model of the specification in the application domain. In this way, formally at least, the relationship between the particular application and the specification is exactly of the same kind as the relationship between specification and software.

The major advantages of such an arrangement come from the obligation to prove that the application satisfies the specification. This means that the application must be presented so precisely that such a proof would be possible. At the same time, it does not mean that the application must be described in programming terms. The choice of the language used for application description is left entirely to the application experts, the only requirements being that it has a calculable semantics, just as the programming language has one.

Just as there may be many programs satisfying any given specification, many applications may fit a given specification. The freedom left by the specification in the application domain is now a measure of how precisely the specification captures the customer's needs. If, with a given specification, one gets too unrestricted application models the specification has to be tightened.

Naturally, in the temporal sequence of events, the specification hardly precedes the application model. In practice it will be somehow abstracted from a description of the application, but the abstraction process (present in all system design methods) is now verifiable.

In addition to the methodological advantages, the proposed arrangement may be used as a framework in which stable evolution of software may be clearly monitored, and thus at least some calamities may be prevented. Indeed, assume that the customer feels a need to modify the software. This need must be expressible as a change in the application model (no other means of articulation is admitted, or to put it bluntly: all other means of articulation of the customer's wishes are simply dangerous and should be disregarded). Thus a modification of the application model is considered as the only possible source of the initiative for software change. We may safely assume that such modification violates the existing relationship between the specification and the application model. (Even if the specification/application relationship is not broken there may be a valid reason to change the specification - it just has been demonstrated that it is too insensitive to application domain modifications!)

The ensuing change of the specification must be effected in a formal system in which the specification is written and the modified specification must be checked for consistency. At this stage some changes may be rejected, others may cause us to think very seriously if it is really worthwhile to introduce them (if the resulting modifications of the specification are massive, the work involved in changing the software may be expected to be similarly extensive.)

When the specification is changed and thus the satisfaction relationship between the specifications and the application domain model is re-established, a crucial decision must be taken: to modify the software (because it does not model the specification any more) or to construct a new software. This unavoidable decision is in the considered arrangement somewhat less arbitrary than in other set-ups because it is predeeded by a full-scale formal modification process

performed on the specification.

Most importantly, if the specification changes are expected, the construction of software may be subject to certain rigours, making subsequent specification-directed modifications of software easier. (Modularity, separation of concerns, splitting a specification into covering "subspecifications".) A particularly promising technique consists in anticipating (at least some) changes in specification, cataloging them and providing - a priori - algorithms for changing the software so as to incorporate any of the catalogued specification changes. .

REFERENCES

(The references listed here are recommended as "further reading" on topics discussed here in this paper.)

Berg, H.K. and Giloi, W.K. (Eds.): The Use of Formal Specifications of Software. Informatik-Fachberichte 36 (1980), Springer-Verlag.

Bjorner, D. (Ed.): Abstract Software Specifications. Lecture Notes in Computer Science 86 (1980), Springer-Verlag.

Floyd, C. und Koptez, H. (Hrsg.): Software Engineering - Entwurf und Spezifikation (1981), Teubner.

Jones, C.B.: Software Development - A Rigorous Approach (1980), Prentice-Hall.

Lehman, M.N.: Programs, life cycles, and the laws of software evolution. Proc. IEEE 68 (1980), 1060.

Turski, W.M.: Software Stability. In Systems Architecture, Proc. 6th ACM European Regional Conf. (1981), London.

INTRODUCING THE HP ON-LINE PERFORMANCE TOOL
(OPT/3000)

Robert L. Mead Jr.
Member of Technical Staff
Hewlett-Packard Company
Computer Systems Division

Robin P. Rakusin
Product Manager
Hewlett-Packard Company
Computer Systems Division

Clifford A. Jager
Project Manager
Hewlett-Packard Company
Computer Systems Division

INTRODUCTION

The question of whether or not a computer system is being effectively utilized is often difficult, if not impossible, to answer. Equally difficult can be the identification of a bottleneck when the performance of a system is less than expected. These difficulties typically arise due to a lack of information on which to base a judgement or decision. Even in those situations where information is available, it is often the case that information is incomplete, or possibly inaccurate or misleading, thus forcing the analyst to make a "best guess" as to the true situation. When detailed and complete information is available, it is frequently difficult to separate the useful information from the vast amount of data provided. In this paper we describe an interactive software product designed specifically to aid in the analysis of HP 3000 computer system performance, and which addresses the problems just described.

This product, the HP On-line Performance Tool (OPT/3000), is Hewlett-Packard's first performance measurement software product, and can be used to identify performance problems or bottlenecks, to characterize the workload on an HP 3000, to collect information required for capacity planning activities, to analyze system table configurations, and in some cases, to tune the performance of individual applications. OPT/3000 provides information in 23 separate interactive displays in the following areas: CPU utilization and memory management activity, memory usage, I/O traffic, program and process activity, and system table

usage. Although each display is designed to be quickly and easily understood, the assumption is made that the user has been trained on the internal operation of MPE IV, the newest version of the HP 3000 Multiprogramming Executive operating system. OPT/3000 is designed to operate in conjunction with MPE IV and can be used on any HP 3000 Series II, Series III, Series 30, Series 33, or Series 44.

This paper presents an overview of the HP On-Line Performance Tool, and discusses some intended applications of OPT/3000. The information reported by OPT/3000 is also reviewed in-depth, as well as the techniques used to obtain the information.

OVERVIEW OF OPT/3000

The HP On-line Performance Tool is a software product that provides performance related information in an interactive environment. As mentioned earlier, OPT/3000 can generate 23 different displays containing performance related information, in addition to seven menu displays. These displays are grouped into six categories, called display contexts, each of which is associated with a different type of system resource. The six contexts are: Memory, CPU/Memory Management, I/O, Process, System Tables, and Global (a little bit of everything). Within each context, displays are available at successively greater levels of detail. This structure allows the user to progress from summary level information to more detailed information as the situation

requires. In many cases, the summary level information is sufficient.

Once a display has been generated, it is automatically updated at periodic intervals, with the length of the time interval under control of the user. A display can also be updated upon demand, simply by entering a carriage return. All commands within OPT/3000 consist of a single ASCII character, and a different set of commands are available in each display context. Certain global commands are available in all contexts. In addition, the pound sign character (#) is used as an escape character to access a set of control operation commands. These commands perform such operations as changing the current display context and suspending the updating of the current display. With this simple user interface, the generation of a different display within the current context is accomplished via a single keystroke, and the generation of a new display within a different context with a minimum of three keystrokes. Menu displays are available within each context, and list the commands available within that context.

An extensive on-line help facility is also available as an integral part of OPT/3000. With this facility, documentation explaining any command or display can be quickly displayed. In many cases, interpretation guidelines are also provided to aid in the identification of performance problems.

OPT/3000 utilizes the features of the HP 264x series of terminals to generate displays with a graphical format, where practical. The

terminal features used include the four available video enhancements (blinking, inverse video, underlining, half-bright), the line drawing character set, and the cursor addressing capabilities. OPT/3000 automatically checks to verify that an appropriate terminal is being used, and warns the user if an incompatible terminal is in use.

A hard copy of any display can be generated on the line printer (device class LP) with a single keystroke. The hard copy displays are similar in layout to the interactive displays, but some reformatting is necessary to convey the same information, due to the lack of video enhancements on a line printer (e.g. paper cannot blink).

Although the HP On-line Performance Tool is primarily designed for interactive use, it can be executed in batch mode to collect summary information about system activity. These summary reports can be used to provide data for capacity planning activities, and can be generated interactively as well. Once activated, the summary reports are generated independent of the interactively generated displays.

There is no limit on the number of copies of OPT/3000 which can be executing simultaneously. OPT/3000 obtains much of its information via a new internal measurement interface facility incorporated within MPE IV. This facility maintains a set of measurement counters accessible by multiple users. Additional information concerning the measurement interface, and the techniques used by OPT/3000 to collect information, will be discussed in a subsequent section.

APPLICATIONS OF OPT/3000

There are several anticipated uses for the HP On-line Performance Tool. Among these uses are the identification of performance problems and bottlenecks, the analysis of system table configurations, characterization of the system workload, capacity planning, and performance tuning of applications. Each of these activities utilizes some or all of the capabilities of OPT/3000. We will now briefly discuss each of these application areas before describing in some detail the information provided by OPT/3000.

The ability to quickly move between displays and the variety of information available through OPT/3000 facilitates its use in identifying performance problems and bottlenecks. In particular, it is expected that a system clearly bottlenecked by CPU, memory, or I/O will be quickly identified. OPT/3000 can also be used to determine if disc accesses are unbalanced between multiple drives. Poorly behaved application programs can also be identified, in terms of programs which use excessive numbers of files and extra data segments and those which waste stack space.

A second application area is that of system table configuration analysis. Inappropriately configured system tables can degrade system performance, either by wasting memory if the tables are unnecessarily large, or by causing processes to delay while waiting for an entry in a table that is configured too small. In the latter case, system failures

may also result if the table size is exceeded. OPT/3000 allows the user to quickly identify those tables which are not properly configured, and to determine (through utilization statistics) a more appropriate value.

The characteristics of the workload on an HP 3000 can be determined using OPT/3000. The names of all active or allocated programs on the system can be easily determined, as well as the users of each program. The CPU usage, disc I/O rate, and memory usage characteristics of an individual application can be determined if the application is running stand-alone on the system.

The summary reports which can be generated by OPT/3000 in either batch or interactive mode can be used to provide data for capacity planning activities. These reports indicate the CPU usage of the system, memory management activity, and the I/O traffic on individual discs, line printers, and magnetic tapes. The information used to generate the summary reports can also be logged to an OPT/3000 log file (disc or tape), which could be processed to provide input for generating plots. In this manner, OPT/3000 can gather trending information that can be used to determine when additional peripherals or systems are needed, or to detect changes in the day-to-day processing load.

Although OPT/3000 is oriented towards the measurement and analysis of the system as a whole, it can be of some value when tuning the performance of individual applications. In particular, OPT/3000 can provide information relating to an application's usage of files and

extra data segments, plus detailed information about the application's use of its stack. CPU usage information is also available.

INFORMATION PROVIDED BY OPT/3000

As mentioned earlier, the HP On-line Performance Tool provides information in six different display contexts: global, memory, CPU/memory management, I/O, process, and system tables. In this section we describe the types of information available in each context. In general, the information provided by OPT/3000 can be divided into two basic classes. The first class of information shows the state of some aspect of the system at the moment the display update is generated. Examples of this class of information include the current contents of main memory and the current list of active programs. The second class of information summarizes activity within the system during some interval. CPU utilization and disc I/O rates are examples of this second class of information. In most cases, this summary class information is reported for two types of intervals: the interval between the previous display update and the current display update, and the interval encompassing all update intervals since the start of OPT/3000 execution. These two intervals are herein referred to as the current interval and the overall interval, respectively. The user can clear all totals associated with the overall interval at any time in order to start a new overall interval. We will now discuss the information available in each of the display contexts.

Global Context

The global context is automatically entered upon execution of OPT/3000, and it provides summary level information concerning CPU usage, memory utilization, disc I/O rates, and process activity. The generation of summary reports is also controlled from the global context. The two displays in the global context can be used to quickly determine potential problem areas (e.g. memory bottleneck), and then more detailed displays in the other contexts used to isolate and verify the problem at hand. The global context can also be used to monitor general system activity, in order to detect fluctuations in resource usage. The CPU and disc I/O information summarizes the activity for both the current and overall intervals, whereas the memory and process information describes the situation at the time of the update.

Memory Context

The memory context consists of eight displays, and provides information related to the usage of main memory and segment sizes. Three of the displays provide information related to the current contents of memory and the remaining displays consist of histograms depicting distributions of segment sizes or free areas in memory. The highest level display concerned with the contents of memory allows the user to determine the current percentage of memory containing code segments, stacks, and extra data segments. Additionally, the user can determine the type of code and data segments in memory. For example, the user can determine the percentage of the extra data segment memory usage that is due to IMAGE/3000, KSAM/3000, the file system, or system tables. Likewise,

code segment memory usage is separated into segments originating from program files, and those from segmented libraries.

The user is also able to generate an image of the current contents of main memory, either for all of memory or for a single bank (64K words). This image indicates the type of each segment (e.g. file system data segment, stack, program file code segment), the approximate size of the segment (in either 1K or 64 word increments), and other miscellaneous information about the segment (e.g. is it locked or frozen?, is it an overlay candidate?). These images are generated by utilizing the display enhancement capabilities of the HP 264x series of terminals, and consist of a sequence of alternating white and gray rectangles (generated using inverse video and half-bright). Each rectangle represents an individual segment.

The histogram displays depict the distribution of segment sizes, in either 1K or 512-word increments. The highest level display depicts separate distributions for code, stack, and extra data segments. The remaining four histogram displays generate higher resolution histograms for each of the above three segment types, plus one for free areas in memory. The histograms are generated using the line drawing character set of the terminal, so as to provide maximum resolution.

CPU/Memory Manager Context

The CPU/memory manager context includes three displays with information related to CPU usage and memory management activity. The highest level

display provides information about both CPU and memory management activity, while the remaining two displays provide more detailed information about each of these areas. All information provided in this context is of the interval summary class, with information for both the current and overall intervals.

The CPU information provided allows the user to determine the percentage of time the CPU is in various states, as well as the rate at which processes are being allowed to execute in the CPU. The reported CPU states include CPU busy executing processes, CPU time for memory management, CPU time on background memory "garbage collection", CPU on overhead processing (e.g. handling interrupts, dispatcher time), CPU waiting for user disc I/O to complete, CPU waiting for memory management I/O to complete, and CPU idle. Information for process launches and process preemptions is reported as the number of occurrences of the event per second (i.e. as a rate). Reported rates include the current interval, the overall interval, and the maximum rate observed in a single interval since the start of the overall interval. These three rates are depicted with a one-line bar on the terminal screen, utilizing inverse video and half-bright inverse video to indicate the current and maximum rates, plus an asterisk to denote the mean rate over all intervals.

Event rate information is also reported for memory management activity in this context. The events reported include memory allocation, memory management disc I/O write, memory management disc I/O read, release code

segment from memory, and release data segment from memory. Information is also available concerning how the memory manager satisfies requests for absent segments. When a segment absence fault occurs in MPE IV, the algorithm used by the memory management routines can terminate with one of five possible outcomes, ranging from recovering the segment from the list of overlay candidates to temporarily postponing the request to avoid thrashing. OPT/3000 shows the percentage of memory allocation attempts terminating with each of the five outcomes. These percentages are shown for both the current and overall intervals, utilizing a bar with alternating white and gray areas.

I/O Context

The I/O context provides four displays regarding I/O completion rates for discs, line printers, and magnetic tapes. The highest level display indicates the I/O completion rate per second for each type of device, for both the current and overall intervals. The remaining three displays provide more detailed information about individual devices within each device category. These displays indicate the completion rate for three types of I/O operations (read, write, and control) on each individual device. This information can be used to determine if the I/O traffic is balanced between the devices on the system, or to identify times of peak activity.

Process Context

The process context includes four displays with information concerning process and program activity on the system at the time the display is

updated. The highest level display provides information about all active or allocated programs. This information includes the fully-qualified program file name, the size of the program file in words, the number of segments in the program, the number of current users of the program, and limited working set information.

Once the above display has been generated, a second level display can be used to determine more detailed information about each process sharing a program file (or for system processes or command interpreter processes). The more detailed information includes the user name and account of the user, the process number (PIN), the size of the process stack in words, the CPU time used by the process, the number of open files and extra data segments, and the job/session number.

Additional information about a specific process can then be obtained by generating a third level display. This display contains all of the information present in the second level display, plus more detailed information about how the process is utilizing its svack space (e.g. the size of the DL area, size of the global data area). Also included are the names of all open files, a list of son processes, and a list of explicitly obtained extra data segments and their sizes.

Some of the information reported in the second and third level displays could be used to circumvent the security aspects of MPE. For this reason, these two displays cannot be generated by all users of OPT/3000. The security provisions within OPT/3000 allow a user with either system

manager (SM) or operator (OP) capability to generate these two displays for any program file. Any other user can only generate the displays for a program file if they are the creator of the file, or are the account manager for the account in which the program file resides.

The remaining display in the process context provides information about the number of processes in various states. For example, the total number of processes waiting for blocked I/O, number of processes waiting for RINs, and the number of processes in the dispatch queue are reported. This information indicates the state at the time the display is updated, and no averages or totals over time are reported.

System Tables Context

The system tables context contains two displays indicating the current and maximum utilization of configurable system tables. One display provides only the current and maximum utilizations in a graphical format, using inverse video and half-bright inverse video bars. For almost all tables reported, the maximums are for the time since the last system warmstart. For the remaining tables, the maximum is that observed by OPT/3000. The second display provides more detailed information in a tabular format. This detailed information includes the configured number of entries, entry size, and maximum utilization observed by OPT/3000, as well as the current and maximum table utilizations.

MEASUREMENT TECHNIQUES

The HP On-line Performance Tool obtains the information used to generate its displays from two basic sources. The first source is the new internal measurement interface facility within MPE IV, and the second is internal MPE data structures and tables. The measurement interface provides all information related to CPU usage, memory management activity, and I/O traffic. All other information reported by OPT/3000 is obtained by examining internal MPE tables and data structures.

The measurement interface facility in MPE IV provides OPT/3000 with a formal mechanism for accessing instrumentation within MPE IV. When the facility is enabled by OPT/3000, the measurement interface obtains an extra data segment to be used as a set of counters. This segment is then locked and frozen in memory and its location stored in a global cell. As events occur, the appropriate counters within the extra data segment are incremented by code within MPE IV, and accessed in a consistent manner by OPT/3000. The CPU state time information is maintained in a similar fashion. OPT/3000 determines the activity during an interval by comparing the current sample to the previous sample, and computing the change in each counter. A count of the number of processes that have activated the interface is maintained by MPE IV. When the count falls to zero, the extra data segment is released and the instrumentation disabled. This mechanism allows multiple copies of OPT/3000 to use the same shared instrumentation. As MPE continues to evolve, both the measurement interface facility and OPT/3000 will be

modified to reflect any changes within MPE.

The overhead within MPE for maintaining the counters has been determined to be approximately 0.3 to 0.8 percent of available CPU time, depending upon the amount of activity within the system. OPT/3000 can collect data from the extra data segment, and update all of its internal totals with the change in each counter in approximately 40 milliseconds. As can be seen from this data, the measurement interface facility provides a very low overhead method for obtaining performance information.

All other information reported by OPT/3000 must be obtained by examining internal MPE data structures and tables. The information concerned with the current contents of main memory is obtained by scanning all of memory, examining each region and sub-region header (these are similar to the memory links in MPE III). The segment size histograms are produced by processing the segment tables. The information concerning program files and processes is obtained by examining the loader segment table directory, process control block table, and the process control block extension area in the stack of a process. System table utilization information is partially obtained by examining information maintained in the header portion of each table.

The overhead required to gather any of the information just mentioned varies depending upon the system configuration. In general, the CPU time required to collect the necessary information and update any display ranges from 300 to 800 milliseconds, depending upon the display.

This normally translates into a total CPU overhead for OPT/3000 ranging from 1 to 3 percent of available CPU time, depending upon the displays generated and the frequency of display updates. An update interval of 15 seconds is the default used, and results in overhead in the 1 to 2 percent range.

SUMMARY

The HP On-line Performance Tool is part of Hewlett-Packard's integrated approach towards offering HP 3000 users alternatives in performance measurement analysis. In addition to OPT/3000, the first HP 3000 software performance measurement product, a new System Performance Evaluation package and a new MPE Internals and System Performance Analysis course are being offered for HP 3000 users.

The recently introduced HP 3000 System Performance Evaluation Consulting package offers an alternative to OPT/3000 for HP 3000 users who want system performance analysis conducted by HP Performance Specialists. These Specialists have in-depth training on the internals of MPE and on the performance characteristics of the HP 3000. They also have a number of HP-supplied software tools at their disposal, such as OPT/3000, IOSTAT, and the MPE IV Data Collection Program (MPEDCP), for collecting and analyzing performance measurement information on the HP 3000.

A new MPE Internals and System Performance Analysis training class is being offered in conjunction with the HP On-line Performance Tool. The first part of the course discusses the areas of MPE IV that are necessary for understanding the performance measurement information presented in OPT/3000, in particular, the new MPE IV memory manager, the dispatcher, scheduler and I/O areas in MPE IV, and the process structures. The second part of the course reviews the inter-relationships of the performance measurement variables discussed in the first part, and presents operational guidelines for OPT/3000. In addition, case study workshops will be used to share HP Performance Specialist techniques and experiences with class participants.

ACE: OPERATORLESS JOB SCHEDULING AND PROCESSING

B. VAUGHAM

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

B. VAUGHAM
HEWLETT PACKARD

THE HAPPY TRANSITION

VERNER ANDRASSEN
HARALD HENDRIKSEN

THE HAPPY TRANSITION

by

Verner Andreassen, Data Processing Manager, City og Bergen

and

Harald Henriksen, General Manager, Aktuelldata (Norway), Sandvika

An overview of the transition from traditional mass batch
processing into user driven interaction distributed
system at City of Bergen, Norway.

V. ANDREASSEN
H. HENDRIKSEN
BERGEN, NORWAY

ORGANIZATION STRUCTURE OF THE CITY OF BERGEN

Bergen, Norway's second largest city, is situated on Norway's West Coast about 500 km north west of Oslo. Once a Hanseatic city, and the largest Nordic city in 1600, Bergen now has a population of 210.000, and covers an area of 465 km². The city administration's number of full time employee equivalents is 10.000. The total 1981 budget is \$ 400 mill.

The administration's structure is very much like the divisionalization we often see in industrial corporations with an equal number of employees, with one staff division, The City Manager Division. Within this division, the EDP department is located. In addition there are some 10 functional divisions.

To get a clear picture of the various EDP activities within the government, one has to make observations from different viewpoints. The systems have been initiated out of different needs, they are built and are working differently, and the administration's influence on the way the systems are developed varies from practically none to complete control.

After the assimilation of some of the neighbouring societies in 1972, the main objectives of the EDP department can be summed up as follows:

1. Consolidation of the joining (as of 1972) societies operating systems.
2. Initiating and implementation of a number of comprehensive systems on a large IBM mainframe, owned jointly by local administrations and private enterprise in the region (= KDV).
3. Setting up local, decentralized processing services, with data entry and information systems, partly for distributed interaction with existing centralized systems, but also for establishing, none-mainframe-dependant local solutions.

The development has been done in cooperation with the user divisions, and project groups have often been established for the various tasks.

The city decided in 1972 against setting up an internal data processing centre with the resulting centralized development and processing concept. For control, planning and approval of the EDP activities a professional organ was established located in the City Manager's staff. At this point, a committee named by and among the politicians took office. (The EDP committee.) Their mandate will be discussed later, after a look on how the consider

INFORMATION TECHNOLOGY VS. ORGANIZATION THEORY.

First, it is recognized that in relation to the development of data processing these two concepts must have a uniform and harmonic platform. It is commonly accepted that the information technology has great influence on employment and the working environment, as well as areas of responsibility and distribution of competence, but also on the quality and rapidness with which tasks are completed. The technology also defines information accessibility and availability in support of planning and managing the society's development. Traditional organization theory points out a relation between the subsystems of technology, administration and the social subsystem, where the technological system is represented by machines and in this case also software, - the social subsystem by interhuman relations and structured, as well as unstructured, ways of contact and communication. In the administrative subsystem we will find guidelines, policies, structures, plans and hierarchical command lines. In the same way as with traditional technology, the information technology will impact on the administrative and social subsystems, especially when applied for rationalization purposes. These unreflected impacts are often felt negatively. If, on the other hand, the information technology is used purposely to influence and prepare the organization to cope with new tasks, and to solve old tasks in a different and more effective way to produce detail and new management information, this impact changes radically.

The effects appear in different ways, and often several at the same time. One obviously and very little desired effect is e.g. that know-how of certain (routine) jobs is drained out of the social system and instead put into the information technology as "axioms" e.g. burnt into ROMs. This will gradually transfer employees into "operators", resulting in loss of insight. A too technological information exchange is also a risk of making people lose their feeling of belonging in a social subsystem. Reallocation of competence and information through incompetent use of technology will gradually and undeliberately restructure, and in the worst case, break down the administrative subsystem, whether the effect is extreme centralization or decentralization.

These phenomenons are some of the main reasons for the strong involvement we see throughout the world when EDP issues are discussed. They are also fundamental for the decision to have a co-ordinated development of the organization structure, people and data processing, and what is more important, the city must always master and control the information technology and the applied methods.

THE EDP MASTER PLAN.

The political controlling authority was when this situation was recognized transferred from the EDP committee to Administration Committee as a consequence of the increasing importance of technology. An excerpt of it's mandate tells that:

"The committee will have general guidelines approved for the use of EDP in the city administration, and within these consider long range plans for the city's EDP development. These plans shall be designed also with regard to their social impacts within the city. The committee will influence federal and other public plans to reflect the needs of the city".

The EDP master plan must, because of the strong relationship between organization development, personnel development and data processing, be integrated in the city overall long range planning. The master plan is the strategic plan for the EDP development the next 5 to 10 years.

It defines long range objectives, general guidelines to reach the objectives, and the span in regulation and the restructuring of activities. The resource plan will explain what resources are available or needed, in terms of personnel, technology, capital and know-how to reach the objectives. A tactical plan will state short term aims, with action plans, budgets and definition of resource available. This planning pattern has so far been prevailing in the city's overall planning procedures, and has also been true for the EDP sector.

The EDP master plan is the strategic platform for the long range activities, as well as for the single projects. The Master plan is therefore considered by the Administration Committee (the politicians), and is approved as a directive for further development prior to the planning on the division and department level.

SYSTEMS METHODOLOGY IN THE 60IES AND 70IES

Two types of systems have been prevailing: sectorial purpose system and function oriented systems. The sectorial systems have been serving the tax sector, the social security sector, the public and private property (real estate) sector, whilst the functional systems have been applied across division borders for applications as payroll, accounting and accounts payables and receivables.

The systems were designed as self-contained systems within the single functions, comprising data entry, transport/transmission to the central mainframe, storing, processing and result presentation. The functional systems have had the design objective to provide adequate solutions for the smallest as well as the largest public administration unit. Mostly these systems were developed by resources outside the city government administration. Often large project organization were employed to develop and design the systems, and to make it possible for most of the involved parties to be heard. Typically, these projects were established and completed as inter city government co-operation. The size of the systems and their complexity, has resulted in an increasing staff of specialists within the different system areas, but outside the city's administration. These systems are highly vulnerable, as operation

experience shows that their maintainability is person dependant. This city co op organization of system development have influenced the cost of development and operation, and competence has been drained out of the executing divisions into the system development organizations.

This approach in system development in the 60ies and 70ies has had the side effects of information centralization and reduced availability and accessibility for the users, centralized competence, development and maintenance with reduced user influence possibilities. Local administrations are as a consequence driven by conditions established in the EDP systems.

The system development trends of the 60ies and 70ies, have cemented the sectorial and central way federal and city governments manage.

THE COMPLEXITY VS. COST ASPECT.

Professionally, it is recognized that the more complexity you introduce into a system to make it all-comprising, the higher the cost of development and operation will be. If we consider the cost aspect related to the completeness of a system, the cost will increase as a constant function until you have an approx. 80% coverage, while it will take an exponential function to cover the last 20%. In other words, the 80-20 rule is applicable, you do 80% of a project for 20% of the total cost. There was and still is a lot of room for productivity tools and action to improve this picture.

These large systems also have certain needs to exchange data, and information integration between systems that one by one is complex, is not made easier by different software, software technology inherent in each system and the time factor for development. The systems are expensive to develop and operate, and it is necessary to have several users to make the cost of operation economically acceptable. This service bureau syndrome has gradually defined the cities' administrations as necessary for the continued growth of the IBM 3033 site mentioned earlier. Consequently, attempts to localize data processing on minis and micros are often considered as threats to the large centralized

facility, and are therefore obstructed in different ways.

The systems of the 60ies were mainly of the periodic batch type. File information was transferred from the classic ledger cards to magnetic representation. It was necessary to make frequent printouts of file information to keep an adequate level of information. The terminal solutions of the 70ies made on-line inquiries on passive registers possible. This change in information accessing made other organizing and retrieval methods necessary, involving higher cost of operations for the periodic systems. It also involved enhancements of the central computer.

These complex common city government systems produced a staff of "experts", often located outside the local administrations and with little feeling for the administration's real needs. The situation can be labelled as "Systems by the experts and for the experts". We often find the user left with complex, technically oriented manuals, without true insight in how the system functions. The expected gain in productivity and rationalization was not realized, and we see the users as suppliers of data (as contrary to information) and receivers of results. The user influence of their working environment is reduced, and in many cases the stress factors are increasing.

Another aspect of these large systems is the fact that the register information is organized and primarily has the objective to satisfy sectors within the different divisions of the administration. It is necessary, however, for effective administration to have access to more data than defined for each sectorial system. Therefore, we see many manually driven support systems connected to the large, complex systems.

As these effects of the centralized systems were getting more apparaent, it was now recognized that in order to stop the unfavourable development of cost, and to stop the drain of competence out of the administration, these large systems should be frozen, and the competence must be reestablished within the function units of the administration. Further, the units must build up their own EDP know how, enabling them to cope with their own problems.

LOCAL TECHNOLOGY.

In the mid 70ies the situation described was getting increasingly obvious within the city. Distributed data processing seemed to be a solution, as it was necessary to continue using most of the large systems for some years. It was made a policy to avoid participation in new inter city projects based on large mainframes.

The first HP-3000 was installed at the City Manager's Division in mid 1977. Successively, local computers were installed at the City treasurer, the City Transport Authority and at the City Electrical Utility. The equipment covers functions within these main application areas:

- Data entry and local storage
- Local processing of local and remotely located information (other HP-3000 x IBM)
- Data presentation (line printer, VDU and graphics)
- Statistical analysis and presentation
- System illustration models
- Programming/System Development
- Data preparation
- Data communication
- Education and training
- Integrated text processing.

The HP-3000s are linked horizontally to each other, using HP-DSN. Theoretically, any terminal user have access to all resources in the network, also including several connections to the IBM mainframe. In this concept all HP-3000 members in the network are equal.

Relatively demanding tasks are assigned to the HP-3000, such as

- setting up local, terminal based data entry and inquiry systems, working against the large (batch) periodically run inter city systems, and validation and transportation of data to and from these systems.

- establishing local interactive information systems and interaction with local micros, data network, terminal administration, data reduction and aggregation, together with statistical analysis.

A PHILOSOPHICAL VIEW OF THE 80IES.

The platform for the data processing of the 80ies is an analysis of the single data item; where it is born, generated, transported, stored, processed and where and how it is presented, and how it can be aggregated to the next information level. One of the most obvious shortcomings of the systems described earlier, was the lack of defined aggregation levels and data reduction levels. Detail transactions are of interest only up to a certain (management) level, and will represent a tremendous overhead when carried forward on all processing levels. To capitalize on this recognition, it is necessary to define each data item with reference to it's organization homestead. Definitions also have to be made for establishing, maintenance and transaction responsibility. In the same way, procedures and policies for the use of data items outside it's originating domain must be established, since all items are readily available from a technological point of view. The process of setting up a responsibility/right-to-use structure, is done over time and in line with the on-going development of organization, responsibility, delegation, decentralization and localization.

In this way, the data structure is integrated into the city's organization pattern.

The system concept can be constructed based on the same general principles.

Instead of envisioning unified systems within each of the cities' divisions, the systems are exploded into autonomous system elements for

- | | |
|------------------|-----------------------|
| - data entry | - data transportation |
| - data storing | - data processing |
| - data retrieval | - data presentation. |

A combination of the different system elements will often with little effort provide the desired solutions.

The interaction between the system elements, whether the elements are implemented on the same technology or different technologies, is based upon the standardized data items, and standardized methods for moving data between technologies. The explosion of the classic all-comprising system concept into functional elements, makes adaptations to individual and local needs easier, and the external characteristics are flexible, as the functionalized elements contain less obstacles to the practical implementations. It is also possible to standardize processing characteristics, eliminating the needs to design new processing routines for new tasks and new data items. By processing it is here meant processing to change register data into new data before conversion into information (e.g. traditional payroll batch processing). A data processing system deals with raw material, work in process and finished goods inventory.

During the 60ies and 70ies it was a recognized practice to gather all single transactions for the different system areas for a common and unified processing in a huge mass transaction system prior to the final results. The uncritical transport of single transactions to the registers of the large systems has made these systems unsuitable as management tools. Through a flexible and localized application of data entry, storing, retrieval and presentation, transaction data can be handled on a low organization level. Instead of sending the total transaction volume on to the next level in the organization, the data entry system will make aggregates to the next management level. The transactions can be stored in its system of origin for the purpose of e.g. statistical processing. On an exception basis single transactions are forwarded one or more levels up (e.g. payroll transactions). The distributed concept complies with the need for authorized information retrieval, both vertically and horizontally.

As data processing in the 60ies and 70ies focused on quantities, the focus of the 80ies will be on improved quality and productivity in retrieval, transportation, analysis and presentation of information.

RMIT STUDENT DATA BASE

N.F. RIEDL

N.F. RIEDL
E. DE GRAAUW
ROYAL MELBOURNE INST. OF TECHNOLOGY

RMIT STUDENT DATA BASE

The Student Data Base and its peripheral systems were developed and introduced by staff of the Royal Melbourne Institute of Technology. The project was supervised by Mr. N.F. Riedl, Data Base Administrator and E. de Graauw, Senior Systems Analyst.

1. DESIGN OBJECTIVES FOR THE RMIT STUDENT DATA BASE

The RMIT Student Data Base was designed to satisfy the following basic requirements.

1. The recording, in on-line mode, of the Academic progress of 12,000 students.
2. The capacity to enrol most of these students over a three week period.
3. The production of examination lists covering all intermediate and final examinations.
4. The retention of a complete academic history for each student.
5. To provide a basis for resources planning.

2. OVERVIEW OF THE RMIT STUDENT DATA BASE

2.1 Description (See Appendix I for DB Diagram)

The name RMIT Student Data Base describes both a data base containing academic and student information, as well as the systems that operate on this data base.

Day to day maintenance of data on the Student Data Base is performed using two major programs which allow on-line, real-time, processing of student

.2.

information and academic information respectively.

Additional programs are available to process bulk information, such as examination results, in batch mode and to produce a large variety of reports.

Student information held on the data base provides a complete profile of current and historical, academic and personal details, for each student who attended courses during the past years. Once a students' most recent records reach a certain age, all of his information is archived (unloaded to tape and, possibly, microfiche) and only an extract of his original record, including a microfiche reference, is kept on-line for further use.

Academic information describes course structures, including past and future courses.

Subjects are recorded as part of course years and stages and full details are available as to how subjects may be taken, who may take the subjects and study periods involved. Provision is made for sub division of subjects into units, with similar information being recorded for units as for subjects. Examination details which may be recorded for subjects and units include such items as intervals at which intermediate examinations are due and number of papers per examination.

../2

../3

.3.

2.2 Technical Aspects

The Student Data Base was set up using the HP Image data base package.

Because most of the information on the data base is subject to various conditions and relationships, special access modules were designed and written by RMIT staff, which incorporate the logic needed to take these requirements into account. As a result RMIT programmers can concentrate on processing the data without the need to get involved in basic data collection techniques. The following is an example of the technique used:-

In any semester most students are studying only for some of the subjects they enrolled for at the beginning of the academic year. Those subjects have been defined in the Student Data Base as "current" for that semester. Whether a subject is current depends on the starting year, starting semester and duration of the subject. Maximum duration was set at four semesters.

Data Base access modules have been developed which will extract, for a given student, the subjects that were current in any semester or year for a specified course.

Another example is the use of special modules to extract "current" academic information from the historical course data.

../4

R4 5

.4.

Course information is retained after courses have been phased out, because student records on file may still refer to these courses. This necessitates data base course structures which can represent the various stages of course development.

Thus users may set up details of future courses without enrolments taking place in these courses.

Or a gradual phasing in or out of course years and stages may be represented.

And, finally, courses may be phased out entirely so that no enrolments can take place, although full course details are available for reporting purposes.

All of this is controlled by a dating system which defines the total period over which a course, course year, stage, subject or unit is available as well as the periods over which different versions of the same are in use.

3. INFORMATION ON THE STUDENT DATA BASE

3.1 Student Data

Student Data is recorded under different headings to assist with the access and maintenance of information on file. The groups of data are listed below.

../5

R4 6

.5.

Personal Details - Name, addresses, etc.
Historical addresses - Recorded as a result of address changes.
Statistical details - Employment, residential and educational status.
Financial details - Annual record of fees due and paid.
Award details - Historic record of all awards presented.
Prize details - Historic records of prizes obtained by students.
Course details - One set of details for each year for each course enrolment per student, containing details such as course code, study mode, study load, enrolment year, etc.
Subject details - One set of details for each subject enrolment per student containing details such as subject number, course code, enrolment year and semester, study mode and result.
Unit details - One set of details for each unit enrolment per student containing details such as unit number, subject number, enrolment year and semester, study mode and result.

3.2 Course Data (See Appendix II)

The academic data structures in the Student Data Base permit a true representation of course structures as defined in the RMIT calendar, complete with elective structures in use in different course. This information is linked to academic data recorded for individual students, thus making possible the production of various academically oriented reports such as examination stationery.

../6

.6.

A structure of academic data groups, similar to student details, provides for economic and versatile processing facilities.

The main data groups are listed below.

Course details - These are distributed over several groups which provide historical details, basic course details such as course description and structural details which link subjects to relevant years and stages.
Subject details - These are distributed over several groups which provide historical details, basic subject details, such as name of subject, structural details, which lists units that belong to a subject, and examination details which indicate the number of examinations and timing of examinations per subject.
Unit details - These are distributed over several groups which provide information similar to that available for subjects as outlined above.

4. DATA STATUS CONCEPTS

4.1 General Description

Information on the Student Data Base is in a state of flux. From the moment a student enrolls, his enrolment details determine where he fits into the student profile. Depending on whether he is an internal or external student, doing Course A or Course Z, full time or part time mode, etc., etc., his name may or may not appear on various reports, notices, examination stationery, etc.
Every change to his status, such as a subject cancellation, receipt of results, re-enrolment, leave

../7

.7.

of absence, etc., may affect the way the next process (report, update, etc.) treats his records. It is therefore important to be aware of the major status conditions that may occur and their effects.

4.2 Student Status

This is a major indicator which determines the overall standing of a student. It takes the following values.

- Current

This is a student who is currently attending classes. The student may be enrolled in several courses at once, in which case cancellation of one course will not affect his current status.

Depending on further sub classifications this student will appear on most standard printouts.

Other status possibilities are

- Concessional (Enrolment at RMIT subsidiary to enrolment elsewhere).

- Leave of Absence

- Unsatisfactory

- Inactive

../8

.8.

- Deleted

- Archived

4.3 Enrolment Status

This status is based on how a student is currently enrolled in a course or each of several courses.

Enrolment status determines the following:-

- Year of enrolment (used to identify current and historical enrolment).
- Year or Stage of Course
- Study Mode (Internal, External, Mixed)
- Study Load (Full Time, Part Time, Single Subject).

4.4 Subject/Unit Currency per student (See Appendix III)

Subject/Unit Currency is a definition that was developed to enable identification of Subjects and Units that have reached a certain stage of completion.

The information on which this currency is based is for each Subject or Unit:

- The year of enrolment
- The starting semester
- The duration in semesters.

Using the above information it is possible to define which Subjects/Units are, or were, current at any point in time. In practice it was found that two

../9

.9.

types of currency stood out as a basis for further refinement.

Thus, to extract all Subjects and Units that a student is (or was) enrolled in any year, we use: Year Currency "A Subject or Unit is current in any given year if the study period for the Subject or Unit takes in at least one semester in that year".

For the purpose of selecting Subjects and Units that may be due for examination in a particular semester, we use:

Semester Currency "A Subject or Unit is current in a given semester if part or all of the study period for the Subject or Unit coincides with that Semester".

To further determine whether an examination is due for a particular Subject or Unit enrolment, the examination details include an item called EXAM-SEMESTERS, which specifies the number of semesters required to be completed in the Subject or Unit before the examination is due. This permits users of the system to define intermediate examinations. A number of formulae have been developed which are used to determine enrolment currency, exam currency, re-enrolment currency, etc., using the above criteria.

4.5 Currency of Academic Data (See Appendix II)

Due to the historical nature of the academic data on file, it is necessary to ascertain that student details are related to academic data for corresponding periods.

../10

.10.

The following definitions of currency apply.

- Academic Definitions of Current Course

This is a Course which includes the year specified as 'current' in its period of currency.

- Academic Definitions of Current Subject

This is a Subject which includes the year specified as 'current' in its period of currency.

- Academic Definition of Current Unit

This is a Unit which includes the year specified as 'current' in its period of currency.

Note that in the above definitions current year may be any year specified for this purpose.

4.6 Example of Student Data Currency

The last page in this chapter contains a typical academic student data structure, representing schematically the type of structure that exists on the Student Data Base.

From this structure the following information may be extracted.

- a. This student has attempted three courses.
- b. He is currently (1980) studying subjects in Courses B and C.
- c. This year (1980) he is enrolled for Subjects CBS2 (Unitized Subject)

CBS3

CCS5

CCS6

CCS7

../11

.11.

and units CBS2U1
 CBS2U2
 CBS2U3
 CCS7U1
 CCS7U2
 CCS7U3
 CCS7U4

b. This semester (Semester 2, 1980) he is engaged
 in the following Subjects and Units (Note: 1/2
 means starting semester is 1 and duration is 2
 semester):

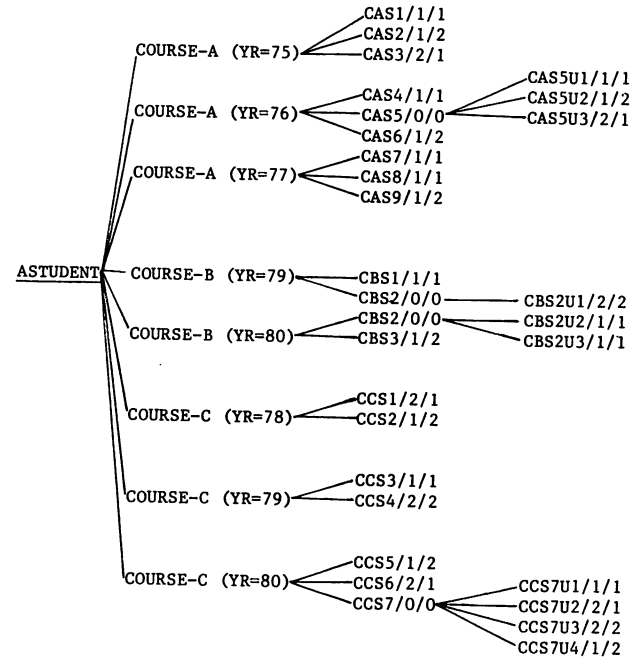
Subjects CBS3/1/2
 CCS4/2/2
 CCS5/1/2
 CCS6/2/1
 CCS7/0/0 (Unitized Subject)
 Units CCS7U2/2/1
 CCS7U3/2/2
 CCS7U4/1/2

Note in the last example that if CCS3/1/1 had
 been recorded as CCS3/2/3, this subject would
 have been current in semester 2, 1980.

.12.

TYPICAL ACADEMIC STUDENT DATA STRUCTURE

SETS: MSTUDENT DSTUDENT-COURSE DSTUDENT-SUBJECT DSTUDENT-UNIT



../13

../12

5. ENROLMENTS

5.1 Enrolment Data Base (See Appendix IV)

The enrolment system reduces input data handling to a minimum.

This is achieved through the use of an enrolment data base. The enrolment data base contains copies of data which was printed on enrolment forms for new and returning students. For new students the enrolment information consists of data transferred from applications processed by the Admissions System and academic details extracted from the Student Data Base.

For returning students the enrolment information is taken from their records on the Student Data Base.

The data in the enrolment data base is used to speed up on-line enrolments as explained in a later chapter. (5.4)

Students who are not shown on any of the enrolment files may be enrolled by overriding the normal data checks. This procedure permits the enrolment of students who enrol before the addition data has been processed or who do not enrol via the Admissions system.

../14

5.2 Enrolment Forms

Details printed on enrolment forms include Subjects and Units that new students are expected to study in the first year or stage of their course.

Preprinting of subjects and Units can be valuable for the following reasons:

Reduction in Keying - if the preprinted subject and unit details are correct, they need not be keyed again.

Early Enrolment Statistics - Any saving in processing time will speed up the production of enrolment statistics.

Fewer Errors - If less data is transcribed manually, fewer errors will occur.

For returning students the selection of Subjects and Units for preprinting poses serious problems at RMIT, as a result of loosely defined course structures, insufficient knowledge of Subjects and Units completed at printing time and the large number of students who "straddle" course years and stages.

Although the system is capable of preprinting most Subjects and Units for returning students, due to the above problems this facility is currently not used.

../15

.15.

The only Subjects and Units that are preprinted for returning students are those that they have started in the preceding year and have not yet completed in terms of the total duration of these Subjects and Units.

5.3 Data Priorities

The enrolment process is designed to satisfy the following requirements:

While enrolments are in progress, the Academic departments and Planning Branch need information on total numbers of students enrolled to-date in various categories.

The enrolment details required for this are treated as high priority data, to be processed on-line. (PART I)

The remainder is processed in batch mode (PART II).

Note, that PART I details may be batched if necessary.

5.4 On-Line

A specially designed enrolment form is in use which clearly identifies the on-line and batch input data areas.

When a student hands in a completed enrolment form, the keyboard operator enters the Student Number shown on the form, or, if the student is a new student, the Application Number.

../16

R4 17

.16.

Depending on the type of number submitted, the system will access either the admission data or the returning student data on the enrolment data base and display all of the information that was printed earlier on the enrolment form.

The operator modifies the details shown on the screen as required and adds any missing PART I details from the form. This tends to involve mainly Subject information.

The system then checks the data and reports on errors found. When the data has been corrected to a satisfactory extent, and provided the data is not rejected entirely, due to serious errors, the operator indicates that the data is to be processed.

5.5 NORMAL and FAST Modes

To provide for greater flexibility in the on-line process, the system allows operators to specify "FAST MODE" in case of peak loads.

In NORMAL mode the system updates the Student Data Base directly with data received from the key board operator.

In FAST mode the data submitted by the operator is stored in a holding file and used to update the Data Base in batch mode at a later stage, when demands on the computer are less.

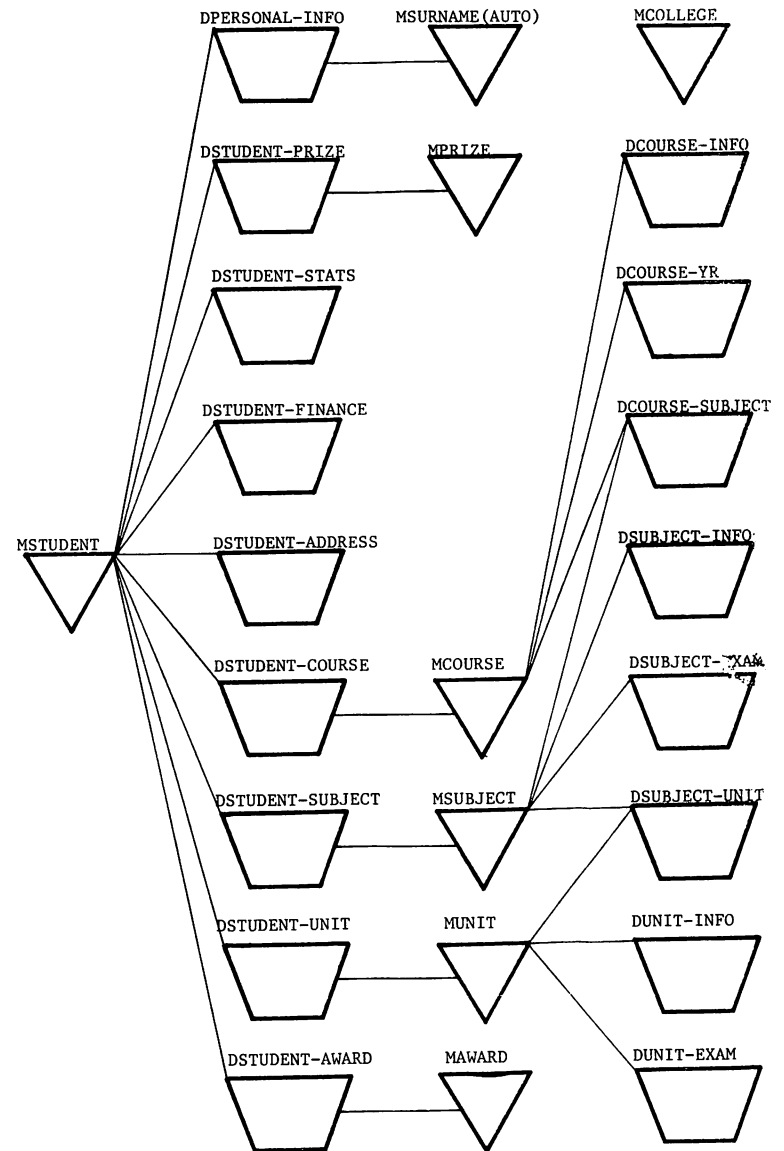
../17

R4 18

When a number of operators are keying enrolment information simultaneously, any number of them may be using one or other of the input modes described above.

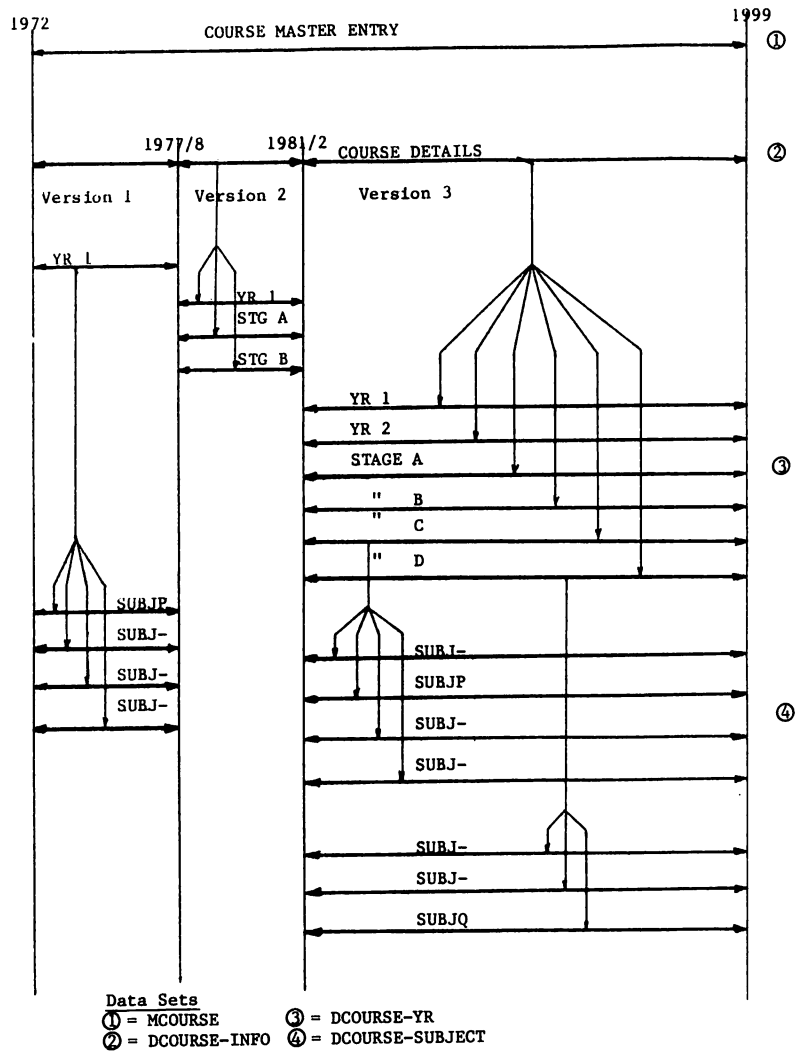
RMIT STUDENT DATA BASE DIAGRAM

APPENDIX I

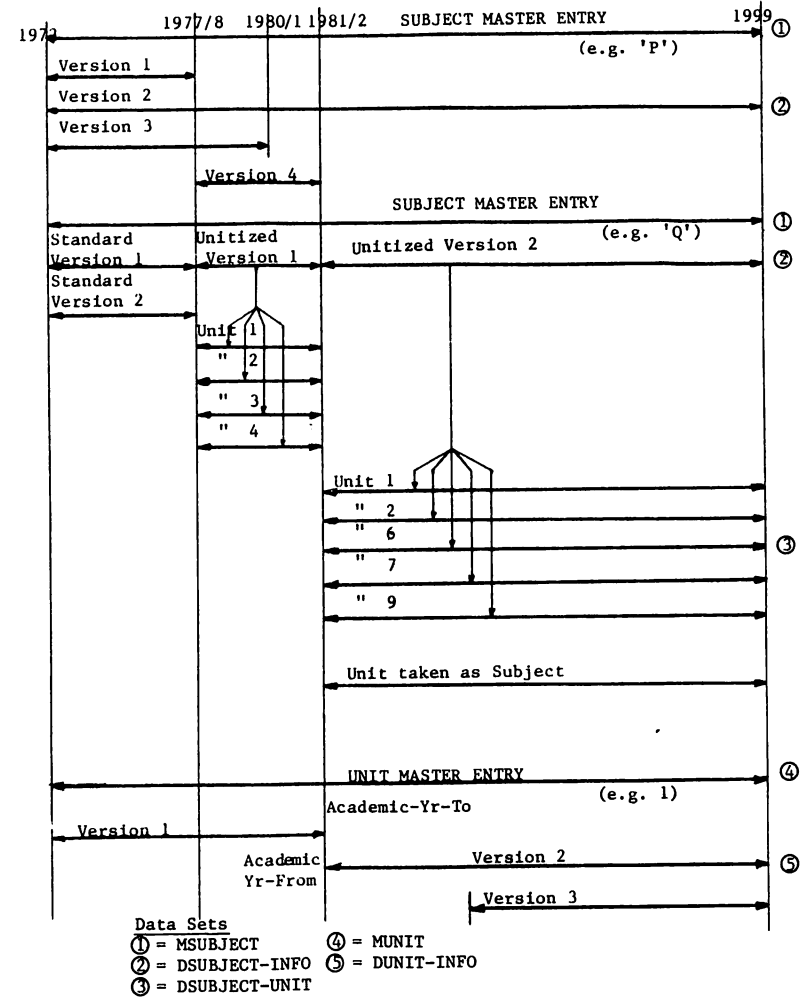


19/8/81
E.de G.

STUDENT DATA BASE COURSE STRUCTURES

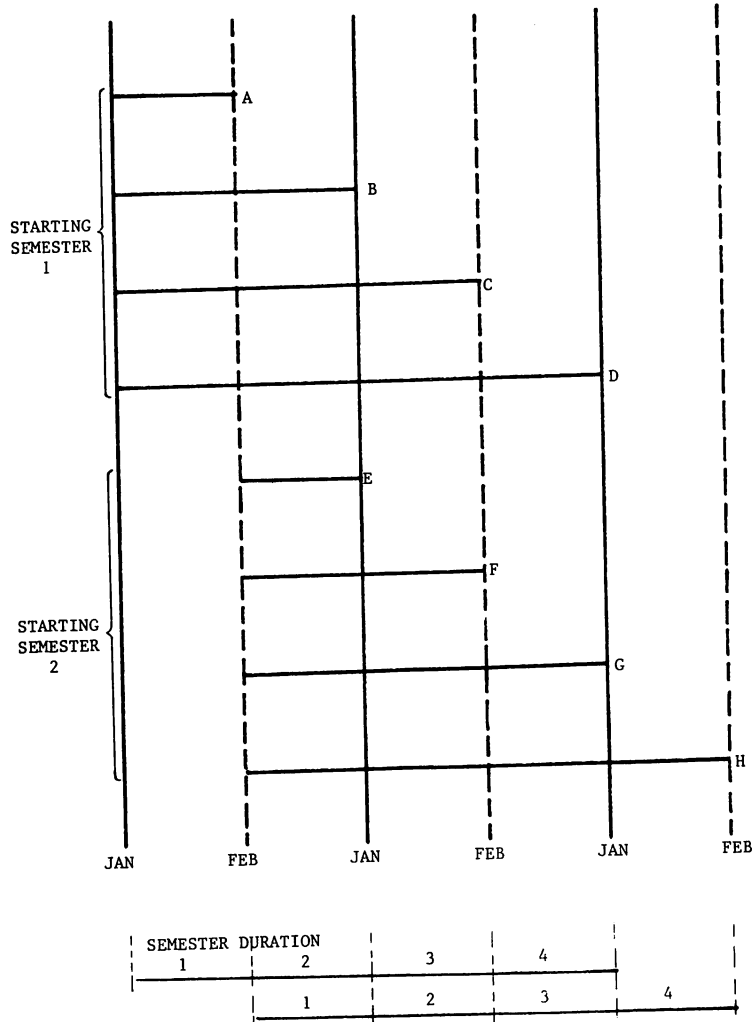


STUDENT DATA BASE SUBJECT STRUCTURES



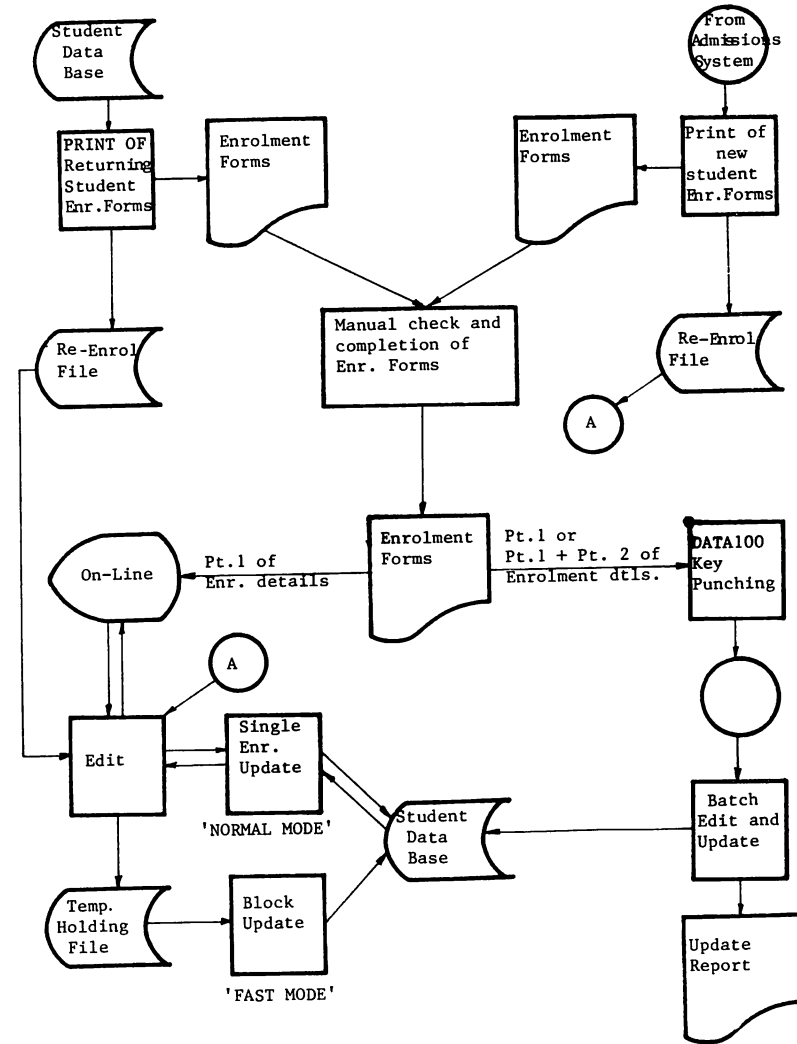
APPENDIX III

SUBJECT CURRENCY RANGES



RMIT ENROLMENT FLOW DIAGRAM

APPENDIX IV



AN INTRODUCTION TO CCITT RECOMMENDATION X.21

Bill Baddeley
Hewlett Packard
Commercial Systems Pinewood

BILL BADDELEY

This paper is an introduction to CCITT Recommendation X.21. The recommendation specifies a new data communications interface which you will probably encounter soon if you haven't already.

At the present time, most data communications among remote terminals and computer systems are carried over the public switched telephone network or over leased telephone circuits. The telephone network has evolved over many years and is an excellent medium for voice transmission. When people started connecting computers and terminals over long distances, the telephone network provided a readily available, though not an optimal, connection medium for digital signalling. Computer systems and terminal equipment are typically connected to the public switched telephone network or leased telephone lines through an interface which is referred to by the label RS-232 or V.24. This interface type has been around for quite some time and is commonly available on modem and terminal equipment.

An organization devoted to international cooperation in telecommunications, the CCITT, is an international advisory body which deals with telephone and telegraph communications. The CCITT issues recommendations for services and implementation of services. There is a series of CCITT recommendations (the "V-series" recommendations) for the connection of data terminal equipment or DTE (the category DTE includes terminals and computer systems) to the telephone network through a modem. The connection point to the telephone network is called the data circuit terminating equipment, or DCE. The standard connection between terminals or systems and the telephone network is described in CCITT recommendation V.24 and the EIA (the U.S. Electronic Industry Association) RS-232. The 25-pin connector which you may have seen on your terminal cable or computer system is the standard connector for this interface. The V.24 or RS232 connection carries signals between the terminal or computer and the modem. These signals include the transmitted and received data signals, timing information and control signals which constitute a dialog between the modem and the computer concerning the state of the connection. For leased telephone circuits, the V.24/RS-232 connection carries all of the information needed between the modem and the system or terminal. The V.24 recommendation specifies the function of each of the data and control circuits in the interface, and the dialog between the terminal/system and the modem/DCE. For switched telephone connections, some PTTs offer automatic calling units. CCITT

B. BADDELEY
HEWLETT PACKARD
COMMERCIAL SYSTEMS PINWOOD
308-314 KINGS ROAD, READING
BERKSHIRE, RG1 4ES

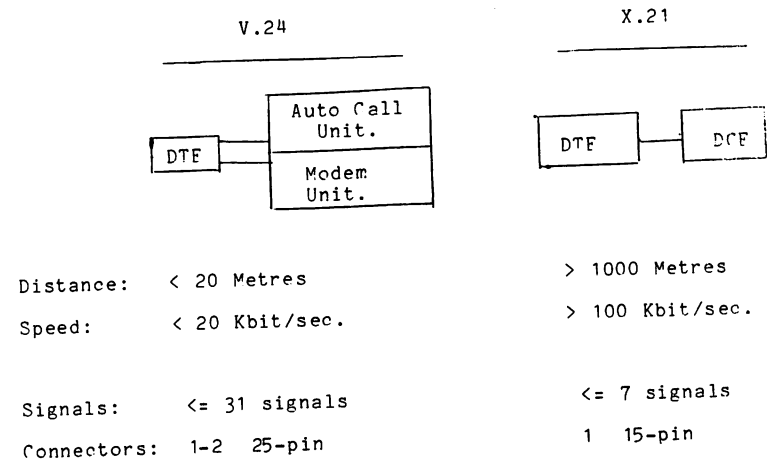
recommendation V.24 also describes the standard interface between a system or terminal and the automatic calling unit (the FIA specification is RS-366). This interface uses the same 25-pin connector as the V.24 modem interface with redefined signal lines. The V.24 recommendation specifies the dialog between the system/terminal and the automatic calling unit for establishing connections.

The telephone network is being replaced for some classes of data communications traffic by public data networks. There is a lot of information in the computer press these days about the new public data networks and the new international standards for these data networks. Depending on where you are located, you may have heard of either X.25 or X.21 or both. The CCITT has issued a series of recommendations (the X series) which deal with data communications networks. The X series of recommendations deal with services on networks which are specifically designed for data communications. There are quite a few public data networks now in operation, and more services are planned for introduction within the next few years.

These networks can be broken into two principal categories according to the type of service which they offer. One type is Circuit-Switched, where the connection between two systems or terminals is equivalent to a hardwired connection once it has been established through the network. Examples of circuit switched networks are the Nordic Public Data Network in Denmark, Finland, Norway and Sweden and the DATEX-L network in the Federal Republic of Germany. The other type of network is Packet-Switched. Packet switched networks support multiple virtual connections through the network over a single DTE/DCF connection.

The CCITT X.21 recommendation describes an interface between Data Terminal Equipment (terminals and systems) and Data Circuit-terminating Equipment for synchronous operation on public data networks. This interface replaces the two-connector, multiple signal interface of V.24 with a simpler set of signals, a smaller connector, and improved electrical characteristics (fig. 1).

Comparison of CCITT Recommendations V.24 and X.21



X.21 Call Progress Signals

The X.21 interface is better able to meet the requirements of current systems for data communications than is V.24 in terms of speed and distance between the system and the network port interface. The speeds of service in public data networks under the CCITT X recommendations are easily met by the X.21 interface, the fastest being 48 kbits/sec. For this reason, X.21 is gradually replacing the V.24 interface.

The CCITT X.25 recommendation, which covers packet-switched data networks, specifies the X.21 interface as the electrical and mechanical interface between the DTE and DCE. The X.25 recommendation will be covered in more detail later in this session.

In addition to the electrical and logical definition of the X.21 interface, the CCITT recommendation describes logical procedures for the operation of the interface in a circuit-switched network and in leased-circuit applications.

The circuit-switched network procedures are broken into four phases: the quiescent phase, when no connection exists between the local station and any remote station; the call establishment phase, when either the DTE starts a call or the DCE signals an incoming call; the data transfer phase, and; the call clearing phase. The circuit-switched network procedures are very much like the procedures one uses with the switched telephone network.

During the quiescent phase, the telephone is on-hook. In the X.21 case, the system/terminal and the network signal their respective states of readiness to establish a connection through the network.

The call establishment phase, in the case of the telephone conversation, begins when one lifts the receiver and dials a number or when the telephone rings, signalling an incoming call. In the X.21 network, the terminal/system signals to the DCE that it is preparing to issue selection signals; the DCE responds by signalling "proceed to select", and then the DTE issues a selection signal sequence specifying a remote station and/or network services. An incoming call is announced by the DCE to the DTE/system/terminal. All cases of call collision (incoming and outgoing calls at the same time) are resolved in favor of the outgoing call. If one dials a busy station or mis-dial a number, you receive in response a tone or tone sequence, or perhaps a recorded message. In the case of an X.21 network, the calling DTE receives call progress signals (figure) which indicate why a call is delayed or why it has failed. These call progress signals are useful in determining a reasonable next action. In addition to call progress signals, the X.21 recommendation describes optional facilities for transferring information such as called line identification to the calling DTE and calling line identification to the called DTE as part of the call setup procedure.

<u>Group</u>	<u>Code</u>	<u>Meaning</u>
0	01	Terminal Called
	02	Redirected Call
	03	Connect When Free
2	20	No Connection
	21	Number Busy
	22	Selection Signals Procedure Error
	23	Selection Signals Transmission Error
4,5	41	Access Barred
	42	Changed Number
	43	Not Obtainable
	44	Out Of Order
	45	Controlled Not Ready
	46	Uncontrolled Not Ready
	47	DCE Power Off
	48	Invalid Facility Request
	49	Network Fault in Local Loop
	51	Call Information Service
52	Incompatible User Class of Service	
6	61	Network Congestion (short term)
7	71	Long-term Network Congestion
8	81	Registration/Cancellation Confirmed
	82	Redirection Activated
	83	Redirection Deactivated

Group 0 signals are delay conditions without call clearing
Group 2 signals indicate short-term conditions
Group 4 and 5 signals indicate long-term conditions
Group 6 signals are short-term network related conditions
Group 7 signals are long-term network related conditions
Group 8 signals are confirmation signals (with call clearing)

Once the call setup is complete, the connection between the calling and the called DTEs is transparent. The network transfers the states of each station's transmit data line bit-for-bit exactly as it appears at the DCF interface. The data phase continues until one of the DTEs signals a clear request. This is analogous to the conversation part of a telephone call.

Call clearing is signalled by either end and transferred to the opposite end. Following call clearing, the DTE and DCF re-enter the quiescent phase.

The circuit switched public data network has several advantages over the public telephone network, having been designed expressly for data communications. One clear advantage is automatic call establishment (no operator dialling).

The X-series recommendations include recommendations for a uniform international node numbering scheme which is analogous to the international telephone numbering scheme. For example, each Nordic network connection has a 6-digit network number, which is unique within the country. International calls include an international prefix, a network/nation code, and a network node identification number. Within the Nordic Public Data Network, calls are possible among the nations of Denmark, Finland, Norway and Sweden.

Another feature of the new public data networks is fast call establishment and high reliability. For example, the Nordic Public Data Network specifications state that all calls will be set up within 2 seconds, 99% will be set up within 0.5 seconds and 90% of all calls will be set up within 0.1 seconds. Similarly all call clearing operations will take under 0.2 seconds, with 90% under 0.05 seconds. This is clearly an improvement over the performance of the switched telephone network.

The X.21 call progress signals, described previously, provide a clear indication of the status of a given call attempt along with information about the probable success of a retrv.

Additional facilities allow the subscriber to simplify the call process by using short-form addresses for commonly called remote nodes. Access restrictions can be placed on a given node by specifying optional facilities to bar incoming or outgoing calls. A group number facility allows several ports to be accessed from the network by a common node address; a central computer can be equipped with several ports which, in addition to their unique addresses are also accessed via a common national/network number. This facility is also referred to as "multiple lines at the same address". The Closed User Group facility allows the creation of private networks within the larger public data network. If a company has several systems at different geographical locations, and has no need to connect them to systems outside of the particular set, then the specification of closed user group

membership for each of them eliminates access from computers outside the network. The facility can also be used in such a way as to restrict the connections from any node in the group to only other group members. It is also possible for a particular node to belong to several closed user groups, one of which is the default or preferential one. In order to switch from the preferential to an alternate closed user group, the selection signal sequence is prefixed with a facility request code specifying which alternate group is to be used for the call setup, followed by the address of the node within that group.

A call queueing facility allows incoming calls to be held in a first in first out queue with a specified number of positions. The caller receives a call progress signal indicating that the call is queued at the remote end. The call redirection facility allows one node to temporarily transfer its address to another node; for the period during which this facility is activated, all calls for the node are redirected automatically to the alternate node. A call progress signal informs callers that the call has been redirected. The Calling and Called Line identification facilities provide for verification and monitoring of connections made through the network. A node specifying the Charge Transfer facility is charged for all incoming calls (which are normally charged to the caller). The charge advice facility allows a node to be informed, following disconnection, of the charges for a call.

Circuit-switched X.21 networks are currently offered in Denmark, Finland, Norway, Sweden, F.R. Germany, and Japan. Several other European nations have X.21 networks in their future plans. Further information about the specification and the network services is available from the implementing PTTs, and from the CCITT (Union Internationale Des Telecommunications, Place des Nations, CH-1211 GENEVE 20, SUISSE).

OPERATOR/CONSOLE INTERFACE
AN ENGINEERING FEEDBACK SESSION

PRESENTOR
MICHAEL PAIVINEN
COMPUTER SYSTEMS DIVISION

OPERATOR/CONSOLE INTERFACE

AN ENGINEERING FEEDBACK SESSION

MICHAEL PAIVINEN

IN MPE III VERSION B.01.00, THE CONSOLE INTERFACE WAS REDESIGNED TO PROVIDE A DISTRIBUTED CONSOLE FACILITY BY INTRODUCING THE ASSOCIATE, ALLOW, AND CONSOLE COMMANDS. IN ORDER TO PROVIDE DIRECTION FOR THE FURTHER DEVELOPMENT OF THE OPERATOR INTERFACE, THE AUTHOR WILL BE SOLICITING CUSTOMER INPUT ON THE FOLLOWING QUESTIONS:

1. WHO IS THE "TYPICAL" OPERATOR? WHAT IS HIS/HER COMPUTER SCIENCE BACKGROUND? SOPHISTICATION? WHAT SET OF SYSTEM CAPABILITIES ARE GIVEN TO THE OPERATOR?
2. WHAT FUNCTIONS DO THE OPERATORS PERFORM ROUTINELY? WHAT ARE THEIR ADDITIONAL RESPONSIBILITIES? WHAT SYSTEM OPERATIONS ARE NOT IN THE HANDS OF THE OPERATOR? WHY?
3. ARE CUSTOMERS USING THE DISTRIBUTE CONSOLE FACILITY? IF SO, HOW IS IT BEING USED? IF NOT, WHAT ARE THE PROBLEMS?
4. WHAT FEATURES OF THE INTERFACE HELP OPERATORS CONTROL THE SYSTEM? HOW COULD THE INTERFACE BE IMPROVED TO PROVIDE BETTER CONTROL?
5. HOW WOULD CUSTOMERS LIKE TO SEE THE CONSOLE INTERFACE EVOLVE? MORE POWERFUL OPERATOR CAPABILITIES? AUTOMATION OF SYSTEM FUNCTIONS TO REQUIRE NO OPERATOR INTERVENTION? OTHER SUGGESTIONS?

M. PAIVINEN
HEWLETT PACKARD

HP 3000 SECURITY/RISK MANAGEMENT

C.W. LAZAR

C.W. LAZAR
SYSTEMS INFORMATION AND TECHNOLOGY
ARCO TRANSPORTATION COMPANY
LOS ANGELES

C.W. Lazar
Systems information and technology
Arco transportation company
Los Angeles

HP 3000 Security/risk management

Purpose

The purpose of security/risk management is to maintain operations in planned mode to prevent as far as practicable unauthorized access to (discovery or modification) of program and data files to prevent, mitigate or recover from inside or outside dysfunctions.

Environment

Organizations that can afford an HP 3000 and the support staff generally are significant businesses. Edp costs vary between 1.5 pct and 5 pct of total costs and business related systems may handle 30 pct to 50 pct of company revenues.

The following system illustrate the point.

Payroll	30	pct	plus
materials purchasing	30	"	"
accounts payable	30	"	"
materials inventory	10	"	"
general ledger	100	"	"

Hence a 20 Dollar a year business may run 6 to 20 million through its HP 3000

my company runs approximately 70 million through and we may double that in 12 months.

Most HP 3000 installations represent a department's first or most ambitious step into electronic data processing away from manual or service center operations. a large portion of system managers have had little or no system management responsibilities.

This presentation is aimed at them and their concerned auditors.

2 - 1

2. The fundamental security deviation rule decision about security measures should be based on cost versus worth. An organization shouldn't spend more to avoid

an increase than the libel or expected cost of the incident.

In mathematical terms

de(pic1) is greater than(de(p1'ci') plus demci)

where

de equals discounted expected value

pi " " probability of incident i

c1 " " cost of incident i

mci " " cost of mitigation measures for incident i

No absolutes

The cedision rule is not a new or original

concept. It is a game theory rule that

emphasizes that there are no absolutes.

That measures short of suicide can't eliminate

undesired incidents: they can only reduce their

probability on their cost.

Consider a fire in the computer room. It can

be caused by a dropped cigarette or an electrical

short or an overheated cooling fan or arson or a

fire in the next room or one probagated through

the plenum or false floor. Rules can ban smoking

but not electrical shorts.

2 - 3

Paperless computer rooms can reduce the source

of fuel and halon systems can reduce the source

of oxygen. But how often is the computer room

the first source of fire in a building. How

many computer rooms share buildings with chemical

closets used by cleaning personnel or oily rags

used by engineers?

How many computer centers are built on bed rock

with fire proof walls and no common air conditioning

equipment?

How well will these fire retardent measures combate

and externally sourced fire and which more probable?

2 - 4

The probability of a computer room fire is very

low on the order less than 0.1 pct per year. The

cost of an automatic halon system is 4000 to

10,000 for a 10' x 15' room. It would imply

that the cost of the fire totally suppressed

should be

4000/.001

or

4,000,000

if back-up tapes are stored off-site and there

is a back-up computer access agreement, then it

is unlikely that the cost of a total hardware

loss fire would equal 4,000,000.

It follows that an expensive automatic halon

system may be a waste of stockholders' money

for a business data processing machine. This

of course may not be true for a real time

process control computer or an airline reservation

system.

This is a reasonable example of applying the

decision rule. It of course doesn't leave the

auditors with a sanguine feeling, that I placated

by installing a 60 handcarried halon system.

3. Quantification of security costs

Decisions about security need to be couched in

reasonable estimates of costs of security systems

and probabilities of undesired incidents.

3.1 Security systems costs.

Security systems can be divided into two arbitrary

classes

probability reducers

cost mitigators

The following are examples of

probability reducers, their objectives

and ball park costs.

item	objective	cost range	comment
computer room locks	reduce unauthor- ized access	150- 1000	cheap looks can be jimmied, with credit card
door locks	reduce unauthor- ized access to files reduce probabil- ity of theft	100- 300 one time	beveled latches can be jimmied with credit card

item	objective	cost range	comment				
sentries	reduce unauthorized access to files	10,000- 20,000 per shift per year	cheap sentry can become thief can become lazy	no smoking	reduce fire probability	1 to?	same port. some smokers will violate rule. Some may quit.
passwords	reduce unauthorized access to files and programs	1.00- 5.00 per password per change	effective-ness is inversely proportioned to age and number of cognocenti	manual fire extinguisher	put out fire	50- 200	Good for limited fire. Doesn't work without operator.
item	objective	cost range	comment	tape back-up system	recover lost files	15 to 50 per tape per day plus re-entry cost at 1/2 a day per person	Typically half a week day will be lost and will have manually re-entered back-up tapes
item	objective	cost range	comment	item	objective	cost range	comment
			zero after three days. Low cost and low effectiveness passwords are stored in clear text in stream jobs that are not lock worded and in image schema that are not lock worded. Multi user passwords obviate accountability	remote tape storage	protect back-up tapes from local dysfunction	2- 10 per month per tape	Should be stored remotely. Need protection system. Should be tested episodically
				hardware insurance	recover costs of disaster	2 to 5 pct of most hardware 1.5 to 3 times costs the expected cost of the disaster	large companies self-insured. Read policies carefully.
terminal locks	keep unauthorized users from accessing system	50- 400	comment	Internal	violation of privacy manipulations fraud ghost vendors and employees theft or hardware, information		

External

fires

earthquakes

bombings

power failures

toxic spills

phone system failures

TERMINALS STRATEGY AND NEW PRODUCTS

R. FRANKLIN

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

R. FRANKLIN
HEWLETT PACKARD

Presentation Abstract

Presentation Title: RAPID/3000, new from HP: Relational Access, Prototyping and
Interactive Development.

Author(s): Jutta Kerne

Title(s): Product Manager, HP Information Network Division

Address: 19420 Homestead Road Cupertino, CA 95104

RAPID 3000, NEW FROM HP:

RELATIONAL ACCES, PROTOTYPING AND INTERACTIVE DEVELOPMENT

JUTTA KERNKE

Abstract: (No more than 200 words)

RAPID/3000: A family of productivity tools for the application programmer,
analyst, data base administrator and end-user. It provides a total solution
to 80% of transaction requirements. RAPID/3000 provides: Relational ad-hoc
inquiry and comprehensive, customized reports to the end-user and decision-
maker - - more information easier and quicker! Simplifies design and increases
productivity in development, testing and implementation - - more applications
faster! Reduces time spent in debugging, maintaining and documenting - -
more time away from tedious routines! A relational dictionary/directory
separates the user-world from the system environment - - better control for
more information resources for all users. RAPID/3000 - - - to get more
information to more management faster and
easier!

J. KERNKE
HP INFORMATION NETWORK DIVISION
19420 HOMESTEAD ROAD
CUPPERTINO, CA 95104

TERMINAL I/O CONTROLLER FOR HP 3000 SYSTEMS

J. BEETEM

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

J. BEETEM
HEWLETT PACKARD

FAST EDITING AND PROGRAM DEVELOPMENT

USING A FULL SCREEN EDITOR

J. VAN DAMME

J. VAN DAMME
SYDES
A. GOSSETLAAN 30A
B 1720 GROOT-BIJGAARDEN
BELGIUM

FAST EDITING
AND
PROGRAM DEVELOPMENT
USING A
FULL SCREEN EDITOR

1. INTRODUCTION.

In most computer systems, text handling is one of the main tasks. Program development, which is a form of text handling, can take a considerable amount of CPU resources. The choice of a good text editor can result in a large increase of productivity.

We will now look at the advantages and disadvantages of several kinds of text editors.

2. THE SEQUENTIAL EDITOR.

The first text editors considered three sequential files. The first file contains the old text to be modified; a second file contains commands to allow the deletion, replacement and addition of full lines. The result is a new sequential file, containing the corrected text.

An example of this editor type is QUERY's 'ALTER' command to modify a procedure.

Typical to these editors is that the line numbers, given in the commands, must be strictly ascending.

Only few of the sequential editors have string handling capabilities.

3. THE INTERACTIF EDITOR.

A major enhancement to the sequential editor was the introduction of interactif capabilities. In general these include

- the possibility to jump back and forth in the text file.
- string handling capabilities (e.g. FIND "string")

Interactif editors act on one single line at a time. So, if the user wants to see the context of the changes he is introducing, parts of the text have to be listed after entering some edit commands.

Most interactif editors work on a copy of the original text (a work file). The operation of copying the original text into the work file and vice versa, takes a considerable amount of time and system resources. On the other hand, working directly on the original text, is an hazardous task to undertake since a mere typing error could destroy large parts of the text.

Regularly taking a copy of the text solves the problem. An alternate solution is to copy the file, at the start of each edit run, and save the work file as the new text at the end. This saves the time of a 'KEEP' operation.

4. THE FULL PAGE EDITOR.

At the time, when more intelligent terminals became available a large part of the editing task could be performed by the terminal instead of the computer.

The editor sends a full page of text to the the screen. This text is edited using the edit features of the terminal (e.g. 'INSERT LINE', 'DELETE CHAR').

After editing of the page is finished, the terminal sends the full page back to the computer and this one sends a new page to the terminal.

The advantages of this method are obviously:

- a good oversight of the context of the changes performed.
- a very natural way of editing (it looks like editing on a sheet of paper).
- very powerfull if the terminal used has good editing capabilities.

The disadvantages are:

- Sending a page back and forth takes a considerable amount of time if the terminal speed is low.
 - e.g. eight seconds to send a 24 lines by 80 characters page at 2400 baud.
 This transmission is performed twice, even if only a single character is to be modified.
- A sophisticated and hence expensive terminal is needed.
- The amount of text that can be added to a page is dependant on the amount of local store in the terminal.

5. THE FULL SCREEN EDITOR.

To overcome the disadvantages of the full page editor, the editor itself can emulate the sophisticated edit features of an expensive terminal. The user then edits the whole file at a time instead of only one page. The editor accepts one edit command (this can be a single key stroke) and performs the edit on the text file and on the screen.

A typical example of this editor type is the editor available on the HP 300 system.

The advantages of such a system are:

- all the advantages of a full page editor.
- considerable savings of time.
- The only special features needed on the terminal are
 - . cursor positioning
 - . character and line insertion and deletion

All other features can be emulated by the editor.

This allows to implement a full screen editor on a low cost terminal.

6. ADDITIONAL FEATURES.

Many additional features are added to text editors. Some of them are:

- The capability of compiling directly from the work file.
- Disc space saving by record compression.
- Handling of MACRO's.
- Automatic generation of program code.
- Word processing capabilities.
- Loop constructs.
- Conditional editing.

FULL SCREEN EDITING

7. FSEDIT.

FSEDIT is a full screen editor for the HP3000 computer systems.

It will run on any terminal from the HP264x and HP262x families, including the low cost HP2621.

Its main features are:

- full screen editing as described above.
- single key-stroke commands.
- direct compile, prepare and run from the work file.
- word processing capabilities.
- COBOL source generation.
- macro handling.
- powerfull command set.

These and other features make FSEDIT an easy to use tool, that allows a high increase in productivity of programmers and other users.

FSEDIT is developed by

SYDES N.V.

TEL 02/4662813

A. GOSSETLAAN 30A

TELEX 63435

B 1720 GROOT BIJGAARDEN

BELGIUM.

Demonstrations of FSEDIT are given at the SYDES N.V. booth.

RELATIONAL DATABASE-CONCEPT,
CONSEQUENCES FOR ORGANIZATION AND MANAGEMENT-STRUCTURES

UWE HINRICHS

UWE HINRICHS
SAUER-INFORMATIC
KROKAMP 35
2350 NEUMÜNSTER
WEST-GERMANY

Summary of: RELATIONAL DATABASE-CONCEPT, CONSEQUENCES FOR ORGANIZATION
AND MANAGEMENT-STRUCTURES

1. What is a relational database

If you look at EDP-concepts, you will find in almost all cases an hierarchically structured file organization. Data sets were organized in different stand-alone files, which were accessed through SEARCH-ITEMS. Most of the FILES were designed for a single application. The structure of this file-system (i.e. the PATHES to the DATA-ENTRIES) has been defined before implementing the data sets (i.e. in ISAM, HISAM, KSAM files etc.).

EXAMPLE : find "city of Berlin"

if ZIP-CODE is the search-item to all cities, you could find the city of Berlin only, if you knew the zip-code.

Only with SORT/MERGE you could reassemble your file, so that it would satisfy a different QUERY.

If changes would become more complex, you would have to reorganize your file-system for this particular application. that means building up new files with new PATHES and reflecting these changes within the program/system.

With the appearance of database systems this became much easier. But still you needed a strong, hierarchical structure of organization at the beginning. The advantage was that your DATA didn't fit for just one application but for your application system in total.

EXAMPLE : a whole application system ---▷ COMPANY
a single application ---▷ SALES
MATERIALS MANAGEMENT
FINANCE
etc.

The understanding of a database system was originally a mass storage file system, in which the data sets were application independent. Still you had to organize the paths within the database in the beginning, so that you had a regulated structure, which couldn't be modified without any problem.

EXAMPLE : implemented references between : SALES ---▷ FINANCE
postulate references between : SALES ---▷ MATERIALS MANAGEMENT

To fulfill this postulate, you needed a modification of the database structure or you could use a QUERY-SYSTEM. But a disadvantage of QUERY systems is the time used to search items in a large database through PATHES, which have not been implemented (SCHEMA).

There are different types of such traditional DATABASE-SYSTEMS; for Example : HIERARCHICAL and NETWORK DATABASES. Both TYPES OF DATABASE SYSTEMS have the same features of a traditional file management organization :

- a regulated structure
- modifying and adding of references and programs are very difficult/time consuming
- cycle of lifetime about 5 years, because the requirements are changing (trouble shouting)
- must of top/down approach
- batch approach
- price/performance for modifying and adding

The relational database system, now, has a completely different approach. The structure of TRADITIONAL FILE-SYSTEM/DATABASE-SYSTEM has to be defined first, before you start solving your detail problem. With RELATIONAL DATABASE SYSTEMS (RDBS) the structure is variable : you still might define temporary relations like you could with traditional QUERY SYSTEMS. However, the most important feature of a RDBS is the capability of learning relations (PATHES).

The capability of modifying, deleting and adding new relations at any time is a feature of this system.

EXAMPLE : available relation --> SALES - FINANCE
 postulate relation --> SALES - PRODUCTION PLANNING

You can define this relation easily with the new variable path information.

The advantage of relational database systems are :

- a variable structure without hierarchical levels
- easy to modify, delete and add relations and programs
- bottom up approach
- lifetime of the systems without a limit, because the system is flexible to adapt new requirements immediately
- full dialogue approach
- high quality of price/performance
- high data quality
- high automation level

2. Organizational Aspects

With a traditional computer organization you need a general concept before you can start programming and building the FILE-/DATABASE-SYSTEM in the field (i.e. manufacturing firms, administrations etc.).

The main chapters of this concept are :

- analysis of given structures
- definition of future requirements
- definition of implementation steps
- description of analysis
- implementation

The general concept - also the analysis - is divided into two parts.

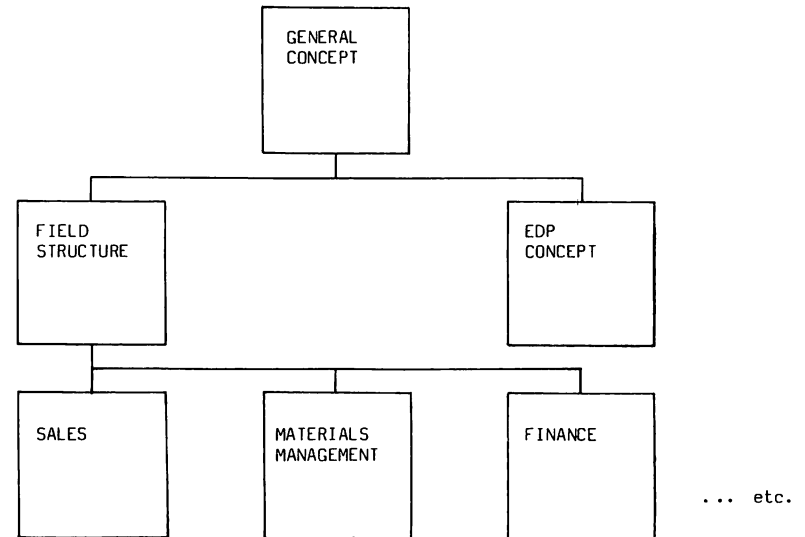
first part : organizational concept of the analysed field related to field structure and needs

second part : organizational concept of EDP related to Hard- and Software-Systems (Software-tools).

These steps are time consuming with the additional disadvantage that the result is always just a snapshot of an existent system, possibly reflecting some future requirements.

With a traditional concept there is a strong interdependence between the first and the second part and therefore it is important to analyze the complete system in order to get an optimal, general concept which covers future requirements, too. Before starting to implement the application system with a traditional file/or database system it is necessary to have the organizational concept completed. The traditional TOP/DOWN APPROACH is usually used for this type of concept.

EXAMPLE : Organization Analysis



You must modify the whole structure and analyse the whole system again, if you modify or add relations between elements of the defined concept.

The great advantage of a relational database system is the variability in organization structure. The steps of system analysis are flexible, because you don't need a general concept that is structured in a first and second part like the general concept.

The main steps of realization are here :

- description of given structure with future requirements
- implementation

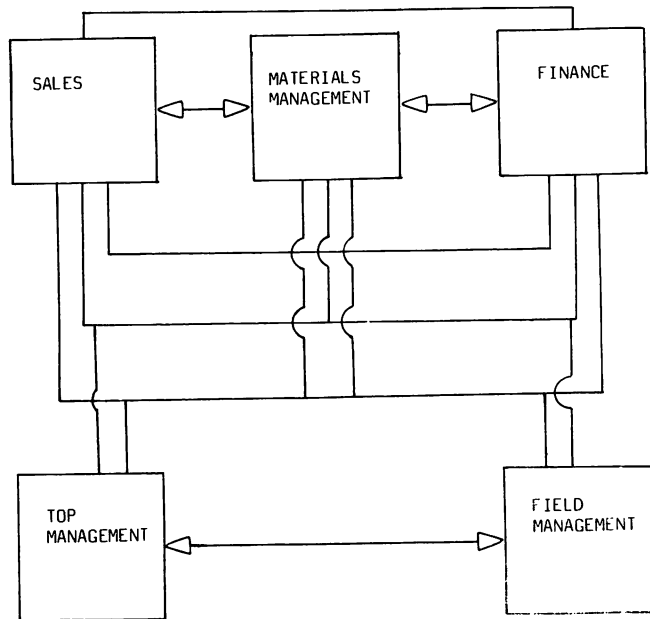
If new requirements arise you will implement them immediately without reflection on previous written Software. The problem is now restricted to generation of new field structure elements.

The system isn't a snapshot any longer, but a living system. Using this technique will now allow you to use the BOTTOM UP APPROACH for your organizational work. This is in fact the most efficient way to proceed because the level of quality of your DECISION-DATA is depending on the level of quality of your BASIS DATA.

The features of this organization method are :

- horizontal structured information --> relations between the field elements
- vertical structured information --> Management Information System
- high level automation
- all kinds of information available immediately

EXAMPLE : Relational Concept



PROGRAMMING FOR DEVICE INDEPENDENCE

PROGRAMMING FOR DEVICE INDEPENDENCE

John Hulme
Applied Cybernetics, Inc.
Los Gatos, California
(408) 356-7296
224 Camino Del Cerro

JOHN HULME

J. HULME
APPLIED CYBERNETICS, INC.
LOS GATOS, CALIFORNIA
224 CAMINO DEL CERRO

INTRODUCTION

The purpose of this presentation is to discuss techniques and facilities which:

- 1) isolate the programmer from specific hardware considerations
- 2) provide for data and device independence
- 3) allow the programmer to deal with a logical rather than a physical view of data and devices
- 4) allow computer resources to be reconfigured, replaced, rearranged, reorganized, restructured or otherwise optimized either automatically by system utilities or explicitly by a system manager or database administrator, without the need to rewrite programs.

The evolutionary development of these techniques will be reviewed from a historical perspective, and the specific principles identified will be applied to the problem of producing formatted screen applications which will run on any type of CRT.

WHAT IS A COMPUTER?

As you already know, a computer consists of one or more electronic and/or electromechanical devices, each capable of executing a limited set of explicit commands. For each type of device some means is provided to allow the device to receive electrical impulses indicating the sequence of commands it is to execute. In addition to commands, most of these devices can receive electrical impulses representing bits of information (commonly called data) which the device is to process in some way. Nearly all of these devices also produce electrical impulses as output, which may in turn be received as commands and/or data by other devices in the system.

Nowadays, most devices also have some form of "memory" or storage media where commands or other data can be recorded, either temporarily or semi-permanently, and a means by which that data can later be received in the form of electrical impulses.

The tangible, visible, material components which these devices are physically made up of is generally called computer hardware. Any systematic set of instructions describing a useful sequence of commands for the computer to execute can be called computer software. As we will see later, software can be further subdivided into system software, which is essentially an extension of the capabilities of the hardware, and application programs, which instruct the computer how to solve specific problems, handle day-to-day applications,

and produce specific results.

Originally it was necessary for a computer operator to directly input the precise sequence of electrical signals by setting a series of switches and turning on the current. This process was repeated over and over until the desired sequence of instructions had been executed.

By comparison with today's methods of operating computers, those earlier methods can truly be called archaic. Yet the progressive advancement of computer systems from that day to this, however spectacular, is nothing more than a step-by-step development of hardware and software building blocks, an evolutionary process occurring almost entirely during the past 25 years.

ENGINEERING AND AUTOMATION

I think we mostly take for granted the tremendous computing power that is at our fingertips today. How many of us, before running a program on the computer, sit down and think about the details of hardware and software that make it all possible? For that matter, who stops to figure out where the electrical power is coming from before turning on a light or using a household appliance? Before driving a car or riding in an airplane, who stops to analyze how it is put together?

Probably none of us do, and that is exactly what the design engineers intended. You see, it is the function of product engineering to build products which people will buy

and use, which usually means building products which are easy to use. The fact that we don't have to think about how something works is a measure of how simple it is to use.

Wherever a process can be automated and incorporated into the product, there is that much less that the consumer has to do himself. Instead of cranking the engine of a car, we just turn a key. Instead of walking up 30 flights of stairs, we just push a button in the elevator.

It's not that we are interested in being lazy. We are interested in labor-saving devices because we can no longer afford to waste the time; we have to meet deadlines; we want to be more efficient; we want to cut costs; we want to increase productivity. We also want to reduce the chance for human error. By automating a complicated process, we produce consistent results, and when those results are thoroughly debugged, error is virtually eliminated. We can rely on those consistent results, which sometimes have to be executed with split second timing and absolute accuracy. Without reliable results there might be significant economic loss or danger to life and limb. Imagine trying to fly modern aircraft without automated procedures.

Automation also facilitates standardization, which allows interchangeability of individual components. This leads to functional specialization of components, which in turn leads to specialization of personnel, with the attendant savings in training and maintenance costs. And because the engineering problem only has to be solved once, with the

benefits to be realized every time the device is used, more time can profitably be spent coming up with the optimum design.

BUILDING BLOCKS

In my opinion, the overwhelming advantage of automating a complicated process is that the process can thereafter be treated as a single unit, a "black box" if you will, in constructing solutions to even more complicated processes.

Later, someone could devise a better version of the black box, and as long as the functional parameters remain the same, the component could be integrated into the total system at any time in place of the original without destroying the integrity of any other components.

It is this "building-block" approach which has permitted such remarkable progress in the development of computer hardware and software. As we review the evolution of these hardware/software building blocks, keep in mind that the chronological sequence of these developments undoubtedly varied from vendor to vendor as a function of how each perceived the market demand and how their respective engineering efforts progressed.

ONE STEP AT A TIME

Even before the advent of electronic computers, various mechanical and electro-mechanical devices had been produced, some utilizing punched card input. Besides providing an

effective means of input, punched cards and paper tape represent a rudimentary storage medium. Incorporating paper tape and card readers into early computer systems not only allowed the user to input programs and data more quickly, more easily, and more accurately (compared with flipping switches manually), but on top of that it allowed him to enter the same programs and data time after time with hardly more effort than entering it once.

The next useful development was the "stored program" concept. Instead of re-entering the program with each new set of data, the program could be read in once, stored in memory, and used over and over.

This concept is an essential feature of all real computers, but it would have been practically worthless except for one other essential feature of computers known as internal logic. We take these two features so much for granted that it's hard to imagine a computer without them. In fact, without internal logic, computers really wouldn't be much good for anything, since they would only be able to execute a program in sequential order beginning with the first instruction and ending with the nth. Internal logic is based on special hardware commands which provide the ability first of all to test for various conditions and secondly to specify which command will be executed next, depending on the results of the test. In modern computer languages, internal logic is manifest in such constructs as IF statements, GO TO statements, FOR loops, and subroutine calls.

But at the stage we are discussing there were no modern programming languages, just the language of electrical signals. These came to be represented as numbers (even letters and other symbols were given a numeric equivalent) and programs consisted of a long list of numbers.

Suppose, for example, that the numbers 17, 11, and 14 represented hardware commands for reading a number, adding another number to it, and storing the result, respectively, and suppose further that variables A through Z were stored in memory locations 1 through 26. Then the program steps to accomplish the statement "give Z a value equal to the sum of X and Y" might be expressed as the following series of numbers, which we will call machine instructions:

17, 24, 11, 25, 14, 26

In essence, the programmer was expected to learn the language of the computer.

A slight improvement was realized when someone thought to devise a meaningful mnemonic for each hardware command and to have the programmer write programs using the easier-to-remember mnemonics, as follows:

READ, 24, ADD, 25, STORE, 26

or perhaps even

READ, X, ADD, Y, STORE, Z.

After the programmer had described the logic in this way, any program could be readily converted to the numeric form by a competent secretary. But since the conversion was relatively straightforward, it would be automated, saving the

secretary some very boring work. A special computer program was written, known as a translator. The mnemonic form, or source program as it was known, was submitted as input data to the translator, which substituted for each mnemonic the equivalent hardware command or memory location, thus producing machine instructions, also known as object code. Translators required two phases of execution, or two passes, one to process the source program and a second to execute the resulting object code. Once the program functioned properly, of course, it could be executed repeatedly without the translation phase.

It would have been possible for the hardware engineers to keep designing more and more complicated hardware commands, and to some extent this has been done, either by combining existing circuitry or by designing new circuits to implement some new elemental command. Each new machine produced in this way would thus be more powerful than the last, but it would have been economically prohibitive to continue this type of development for very long and the resulting machines would have been too large to be practical anyway.

Engineers quickly recognized that instead of creating a more powerful command by combining the circuitry of existing commands, the equivalent result could be achieved by combining the appropriate collection of commands in a miniature program. This mini-program could then be repeated as needed within an application program in place of the more complex command. Or better yet, it could be kept at a fixed location in memory and

be accessed as a subroutine just the same as if it were actually a part of each program.

Another approach was to use an interpreter, a special purpose computer program similar to a translator. The interpreter would accept a source program in much the same way as the translator did, but instead of converting the whole thing to an object program, it would cause each hardware command to be executed as soon as it had been decoded.

Besides requiring only one pass, interpreters had the added advantage of only having to decode the commands that were actually used, though this might also be a disadvantage, since a command used more than once would also have to be decoded more than once.

The chief benefit of an interpreter lay in its ability to accept mnemonics for commands more complex than those actually available in the hardware, and to simulate the execution of those complex commands through the use of subroutines. In this way, new commands could be implemented without any hardware modifications merely by including the appropriate subroutines in the interpreter. This step marked the beginning of system software.

In addition, source programs for nearly any computer could be interpreted on nearly any other computer, as long as someone had taken the time to write the necessary interpreter. Interpreters could even be written for fictional computers or computers that had been designed but not yet manufactured. This technique, though generally regarded as very inefficient,

provided the first means of making a program transportable from one computer to another incompatible computer.

It is possible, of course, to apply this technique to translators as well, allowing a given mnemonic to represent a whole series of commands or a subroutine call rather than a single hardware instruction. Such mnemonics, sometimes called macros, gave users the impression that the hardware contained a much broader repertoire of commands than was actually the case.

Implementing a new feature in software is theoretically equivalent to implementing the same function in hardware. The choice is strictly an economic one and as conditions change so might the choices. One factor is the universality or frequency with which the feature is likely to be used. Putting it in hardware generally provides more efficient execution, but putting it in the software is considerably easier and provides much greater flexibility.

The practice of restricting hardware implementation to the bare essentials also facilitated hardware standardization and compatibility, which was crucial to the commercial user who wanted to minimize the impact on all his programs if he should find it necessary to convert to a machine with greater capacity. Beginning with the IBM 360 series in 1964 "families" of compatible hardware emerged, including the RCA Spectra 70 series, NCR Century series, and Honeywell 200 series, among others.

Each family of machines had its own operating system, software monitor, or executive system overseeing the operation of every other program running on the machine. In some systems, concurrent users were allowed, utilizing such techniques as memory partitioning, time-sharing, multi-threading, and memory-swapping. Some form of job control language was devised for each operating system to allow the person submitting the jobs to communicate with the monitor about the jobs to be executed.

Introducing families of hardware did not solve the problem of compatibility between one vendor and the next, however, a problem which could only be solved by developing programming languages which were truly independent of any particular piece of hardware.

Since the inventors of these so-called higher-level languages were not bound by any hardware constraints, an effort was made to make the languages as natural as possible. FORTRAN imitated the language of mathematical formulas, while ALGOL claimed to be the ideal language for describing algorithmic logic; COBOL provided an English-like syntax, and so on.

Instead of having to learn the computer's language, a programmer could now deal with computers that understood his language. Actually, it was not the hardware which could understand his language, but a more sophisticated type of translator-interpreter known as a compiler.

To the degree that a particular language enjoyed enough popular support to convince multiple vendors to implement it, programs written in that language could be transported among

those machines for which the corresponding compiler was available.

The term compiler may have been coined to indicate that program units were collected from various sources besides the source program itself, and were compiled into a single functioning module. Subroutines to perform a complex calculation such as a square root, for example, might be inserted by the compiler whenever one or more square root operations had been specified in the body of the source program.

Embedding subroutines in the object code was not the only solution, however. It became more and more common to have the generated object programs merely "CALL" on subroutines which were external to the object program, having been pre-compiled and stored in vendor-supplied "subroutine libraries". This concept was later extended to allow users a means of placing their own separately-compiled modules in the library and accessing them wherever needed in a program.

I should mention that an important objective of any higher level language should be to enable a user to describe the problem he is solving as clearly and concisely as possible. Although the emphasis is ostensibly on making the program easy to write, being able to understand the program once it has been written may be an even greater benefit, particularly when program maintenance is likely to be performed by someone other than the original author.

It is well-known that program maintenance occupies a great deal of the available time in the typical data

processing shop. Some studies estimate the figure at over 50% and increasing. In order to be responsive to changing user requirements, it is essential to develop methods which facilitate rapid and even frequent program changes without jeopardizing the integrity of the system, and without tying up the whole DP staff.

To avoid having to re-debug the logic every time a change is made, it is often possible to use data-driven or table-driven programming techniques. The portion of the program which is likely to change, and which does not really affect the overall procedural logic of the program, is built into tables or special data files. These are accessed by the procedural code to determine the effective instructions to execute.

The most common example in the United States, and perhaps in other countries as well, is probably the table of income tax rates, which changes by law now at least once a year. The algorithm to compute the taxes changes very rarely, if at all, so it does not have to be debugged each time the tables change. In simple cases like this, non-programmer clerks might safely be permitted to revise the table entries.

In more sophisticated applications, tables of data called logic tables may more directly determine the logic flow within a program. The program becomes a kind of interpreter, and elements in the logic table may be regarded as instructions in some esoteric machine language. Such programs are generally more difficult to thoroughly debug, but once debugged

provide solutions to a broad class of problems without ever having to revise the procedural portion of the program.

Sometimes, logic-controlling information is neither compiled into the program nor stored in tables, but is provided to the program when it is first initiated or even during the course of execution, in the form of run-time parameters or user responses. The program has to be pre-programmed to handle every valid parameter, of course, and to gracefully reject the invalid ones, but this method is useful for cutting down the number of separate programs that have to be written, debugged, and maintained. For example, why write eight slightly different inventory print programs, if a single program could handle eight separate formats through the use of run-time options?

Incidentally, program recompilations need not always cause alarm. Through the proper use of COPY code, programs can be modified, recompiled, and produce the new results without the original source program ever having to be revised. This is made possible by a facility which allows the source program to contain references to named program elements stored in a COPY library instead of having those elements actually duplicated within the program. A COPY statement is in effect a kind of macro which the compiler expands at the time it reads in the source program.

For example, if a record description or a table of values appears in one program, it is likely to appear in other programs as well. It is faster, easier, safer, and more

concise to say "COPY RECORD-A." or "COPY TABLEXYZ." than to re-enter the same information again and again. And if for some reason the record layout or table of values should have to be changed, merely change it in the COPY library, not in every program.

By changing the contents of a COPY member in this way and subsequently recompiling selected programs in which the member is referenced, those programs can be updated without any need to modify the source. If procedure code is involved, the new COPY code only need be debugged and retested once rather than revalidating all the individual programs.

Where blocks of procedural code appearing in many programs can be isolated and separately compiled, however, this would probably be better than using COPY code. For one thing, the separate modules would not have to be recompiled every time the procedural code was revised.

BITE-SIZE PIECES

Breaking a complex problem into manageable independent pieces and dealing with them as separate problems is a valuable strategy in any problem-solving situation. Such a strategy has added benefits in a programming environment:

1. Smaller modules are typically easier to understand, debug, and optimize.
2. Smaller modules are usually easier to rewrite or replace if necessary.
3. Independent functions which are useful to one application are often useful to another application; using an existing module for additional applications cuts down on programming, debugging, and compilation time.

4. Allowing applications to share a module reduces memory requirements.
5. Having only one copy of a module ensures that the module can be replaced with a new version from time to time without having to worry that an undiscovered copy of an older version might still be lurking around somewhere in the system.

The fact that a routine only has to be coded once usually more than compensates for the extra effort that may have to go into generalizing the routine. The more often it's used, the more time you can afford to spend improving it.

SYSTEM SOFTWARE

Functions which are so general as to be of value to every user of the computer, such as i/o routines, sort utilities, file systems, and a whole host of other utilities, are usually included in the system software supplied by the hardware vendor. Just what facilities are provided, how sophisticated those facilities are, and whether the vendor charges anything extra for them, is a matter of perceived user need and marketing strategy. Sometimes vendors choose to provide text editors and other development tools, and sometimes they don't. Sometimes they provide a very powerful data base management system, sometime only rudimentary file access commands. And so on.

When hardware vendors fail to provide some needed piece of software, it may be worthwhile for the user to write it himself. If the need is general enough, software vendors may rush in to fill the void; or perhaps user pressure will eventually convince hardware vendors to implement it themselves.

In this way, many alternative products may become available, and the user will have to evaluate which approach he wishes to take advantage of, based on such factors as cost, efficiency, other performance criteria, flexibility of operation, compatibility with existing software, and the comparative benefits of using each product.

PRINCIPLES OF GOOD SYSTEM DESIGN

In case you may need to design your own supporting software, or evaluate some that is commercially available, let's summarize the techniques which will permit you to achieve the greatest degree of data, program, and device independence. I have already given illustrations of most of the following principles:

1. Modularity--Conceptually break everything up into the smallest modules you feel comfortable dealing with.
2. Factoring--Whenever a functional unit appears in more than one location, investigate whether it is feasible to "factor it out" as a separate module (this is analogous to rewriting $A*B+A*C+A*D$ as $A*(B+C+D)$ in math).
3. Critical Sections--Refrain from separating modules which are intricately interconnected or subdividing existing modules which are logically intact.
4. Independence--Strive to make every module self-contained and independent of every external factor except as represented by predefined parameters.
5. Interfacing--Keep to a minimum the amount of communication required between modules; provide a consistent method of passing parameters; make the interface sufficiently general to allow for later extensions.

6. Isolation--Isolate all but the lowest-level modules from all hardware considerations and physical data characteristics.
7. Testing--Test each individual module by itself as soon as it is completed and as it is integrated with other modules.
8. Generalization--Produce modules which solve the problem in a general way instead of dealing with specific cases. Be careful, however, not to over-generalize. Trying to make a new technology fit the mold of an existing one may seem like the best modular approach, and the easiest to implement, but the very features for which the new technology has been introduced must not become lost in the process.

 EXAMPLE--When CRT's were first attached to computers they were treated as teletypes, a class of i/o devices incompatible with two of the CRT's most useful features: cursor-addressing and the ability to type over existing characters. Putting the CRT in block-mode and treating it as a fixed-length file represents the opposite extreme: the interactive capabilities are suppressed and the CRT becomes little more than a batch input device, a super-card-reader in effect.
9. Standardization--Develop a set of sound programming standards including structured programming methods, and insist that each module be coded in strict compliance with those standards.
10. Evaluation--Once the functional characteristics have been achieved, use available performance measurement methods to determine the areas which most need to be further optimized.
11. Piecewise Refinement--Continue to make improvements, one module at a time, concentrating on those with the largest potential for improving system performance, user acceptance, and/or functional capabilities.
12. Binding--For greater flexibility and independence, postpone binding of variables; for greater efficiency of execution, do the opposite; pre-bind constants at the earliest possible stage.

BINDING

As the name suggests, "binding" is the process of tying together all the various elements which make up an executing program. Binding occurs in several different stages ultimately making procedures and data accessible to one another.

For example, the various statements in an application program are bound together in an object module when the source program is compiled. Similarly, the various data items comprising an IMAGE data base become bound into a fixed structure when the root file is created. A third case of binding involves the passing of parameters between separately compiled modules.

Remember that at the hardware level, where everything is actually accomplished, individual instructions refer to data elements and to other instructions by their location in memory. The "address" of these elements must either be built into the object code at the time a program is compiled, be placed there sometime prior to execution, or be provided during execution. Likewise, information governing the flow of logic can be built into the program originally, placed in a file which the program accesses, passed as a parameter when the program is initiated, or provided through user interaction during execution.

Binding sets in concrete a particular choice of options to the exclusion of all other alternatives. Delayed binding therefore provides more flexibility, while early binding

provides greater efficiency. Binding during execution time can be especially powerful but at the same time potentially critical to system performance. In general, variables should be bound as early as possible unless you specifically plan to take advantage of leaving them unbound, in which case you should delay binding as long as it proves beneficial and can still be afforded. Incidentally, on the HP 3000, address resolution between separately-compiled modules will occur during program preparation (PREP) except for routines in the segmented library, which will be resolved in connection with program initiation. If your program pauses initially each time you run it, this run-time binding is the probable cause.

A SPECIFIC APPLICATION

About five years ago, we were faced with the problem of developing a system of about 300 on-line application programs for a client with no previous computer experience. Their objective was to completely automate all record-keeping, paper-flow, analysis, and decision making, from sales and engineering to inventory and manufacturing to payroll and accounting. The client had ordered an HP 3000 with 256K bytes of memory and had already purchased about 20 Lear-Sigler ADM-1 CRT's. About 12 terminals were to be in use during normal business hours for continuous interactive data entry; the remaining eight terminals were primarily intended for inquiry and remote reporting.

Up-to-date information had to be on-line at all times using formatted screens at every work station. Operator satisfaction was also a high priority, with two- to five-second response time considered intolerable.

DISCUSSION QUESTIONS

Based on the "principles of good system design" summarized earlier, what recommendations would you have made to the development team?

At the time, HP's Data Entry Language (DEL) seemed to be the only formatted screen handler available on the HP 3000. Consultation with DEL users convinced us it was rather awkward to use and exhibited very poor response time. Also it did not support non-HP character-mode terminals.

We elected to write a simple character-mode terminal interface, which was soon expanded to provide internal editing of data fields, and later enhanced to handle background forms. We presently market this product under the name TERMINAL/3000. You've probably heard of it.

The compact SPL routines reside in the system SL and are shared by all programs. The subroutine which interfaces directly with the terminals is table-driven to ensure device-independence. By implementing additional tables of escape sequences, we have added support for more than a dozen different types of terminals besides the original ADM-1's.

If we were faced with a similar task today, would your recommendations be any different?

After completing most of the project, we did what should have been done much earlier: we implemented a CRT forms editor and COBOL program generator which together automate the process of writing formatted-screen data entry programs utilizing TERMINAL/3000. We call this approach "results-oriented systems development"; the package is called ADEPT/3000. Programs which previously took a week to develop can now be produced in only half a day.

Since we were using computers to eliminate monotonous tasks and improve productivity for our clients, it was only natural that we should consider using computers to

reduce monotony and increase productivity in our own business, the business of writing application programs. If you write application programs or manage people who do, you also may wish to take advantage of this approach.

What features of VIEW/3000 would have made it unsuitable for this particular situation?

- not available five years ago
- HP 2640 series of terminals only
- block-mode only (not interactive field-by-field)
- requires huge buffers (not enough memory available)
- response time and overall system performance inadequate

From what you know of TERMINAL/3000 and ADEPT/3000, how do these products enable a programmer to conform to the principles of good system design?

TERMINAL/3000 itself: modular, well-factored, single critical section, device-independent, independent of external formats, simple 1-parameter interface, table-driven hardware isolation, well-tested, generalized, optimized for efficiency, run-time binding of cursor-positioning and edit characteristics.

ADEPT/3000: produces COBOL source programs that are modular, well-segmented, device-independent, and contain pre-debugged logic conforming to user-tailored programming standards; built-in interfaces to TERMINAL/3000 and IMAGE/3000 (or KSAM/3000) isolate the programs from hardware considerations and provide device and data independence.

BIBLIOGRAPHY

- Boyes, Rodney L., Introduction to Electronic Computing: A Management Approach (New York: John Wiley and Sons, Inc., 1971).
- Hellerman, Herbert, Digital Computer System Principles (New York: McGraw-Hill Book Co., Inc., 1967).
- Knuth, Donald E., The Art of Computer Programming (Reading, Mass.: Addison-Wesley Publishing Company, 1968).
- Swallow, Kenneth P., Elements of Computer Programming (New York: Holt, Rinhart and Winston, Inc., 1965).
- Weiss, Eric A. (ed.), Computer Usage Fundamentals (New York: McGraw-Hill Book Co., Inc., 1969).

HOW TO GET MORE FROM YOUR CORE MEMORY

PIERRE SENANT

HOW TO GET MORE FROM YOUR CORE MEMORY

OR

CFS/3000

A CORE RESIDENT FILE SYSTEM

Pierre SenantA LITTLE HISTORY ...

A data processing system ordinarily uses three main types of memory, which have each a different speed level. Besides, the byte cost of these types of memory varies with their access speed. Here are the three types of memory in decreasing cost order :

1. Core memory. The latest technology uses integrated circuits and the access time is calculated in micro-seconds. The purpose of this memory is to contain program segments during the time they are executed, as well as a part of the data to be handled. The life time of these elements in memory is between a few milli-seconds, and many hours.

P. SENANT
COGELOG
ETUDE ET RÉALISATION DE SYSTÈMES INFORMATIQUES
AVENUE DE LA BALTIQUE, Z.A. DE COURTABOEUF
91940 LES ULIS, FRANCE

2.

2. Auxiliary memory : Magnetic discs are normally used. The time to access information is calculated in milli-seconds. The purpose of this kind of memory is to save programs and data to be used during a given period, which can be from a few minutes to many months.
3. Archival memory : This can be done through many devices, but the most common is the magnetic tape. Since the access to data is performed serially, the access time to data can be from a few seconds to many minutes. This low-cost type of memory is used as back-up memory, and for saving all data currently unused.

When we consider the evolution of these different types of memory during the last past years, we observe a fall in prices for all types, but a particular drop for the core memory.

This drop has considerably changed the physionomy of data processing systems during the last years. The HP-3000 system remained in the swim. When it came out in 1972, the maximum memory size supported was 128 K bytes. The model of 1981 can support up to 4000 K bytes.

In addition, this trend will probably be confirmed in the next years, and an HP-3000 with 20.000 K memory will be common soon. The advantages of the increase of the core memory are evident :

3.

- . The amount of code that can be put in memory at a given time is larger. This implies a fall in data segment swapping, which dramatically reduces the disc overhead and increases the throughput of the system.
- . The amount of data handled in one time can theoretically be increased, by using large file system buffers. However this technique gives disappointing results, especially in random access.

Most data processing applications are now using more and more interactive mode, instead of batch mode. Improving batches is important, but improving on-line programs is vital. Batches can run in off-peak hours, and most of the time the jobs can be done. But in interactive applications, each second of response time lost must be multiplied by the number of users, and employer's time is getting expensive.

So, the benefit of increasing core memory comes almost exclusively from improving the programs flow. Suppose we could increase indefinitely the core memory size, we would reach a critical point where adding a memory module would not affect the response time, because all program segments are already in core.

Another important factor bears heavily on the response time disc accesses due to transactions of application programs. An on-line program using a data base system (IMAGE 3000 for example) may be very greedy in disc accesses. This overhead

is independant of the number of program segments in memory and it becomes the actual bottleneck. This makes unprofitable an increase of core memory.

When a system has reached this level of evolution, one possible way to reduce the overhead is to suppress some disc accesses. At first sight, this load seems to be incompressible. If we assume the files and the data bases have been correctly organized, and application programs have been written with shrewdness, what can we do ?

However, when we examine all kinds of files involved in an application, we are surprised by their diversity, in the size, in the organization, and in the usage.

In fact, most of them are small enough, and are so frequently accessed that we would do better to make them core-resident.

NOW THE PRESENT : CFS/3000

The idea to make some files core-resident might not be very original, but it certainly remained a dream until to day.

Now the dream becomes reality with CFS/3000.

In my opinion, a real in-core memory file system must follow the following principles.

- . It must be program independant.
 - Allowing programming people to decide if a file must be core-resident or disc resident would be catastrophic.
- . A core resident file must be accessible from all processes. The duplication of data is not acceptable.
- . It must be reliable. Data integrity must not be affected, as well as the process independancy.
- . It must respect all security and privacy provisions of the file system, as well as those of subsystems (IMAGE, KSAM).
- . It must be fully controlled by the System Manager, who must know at any time.
 - the name and the sizes of core-resident files
 - the total memory size used
 - access frequency of any data-set.

In addition, the System Manager can decide to put in or out of core-memory any file at any time without disturbance in operation.

The conception of CFS/3000 has been founded upon these principles.

HOW TO USE CFS/3000

This product is intended to optimize both batch programs and on-line programs. In all cases, a good knowledge of application programs and system management will be required.

- . In a batch environment, the execution time can be dramatically reduced. It will be strongly recommended to run one program at a time, in order to devote the maximum core-memory to data files.
- . In an interactive environment, the tuning must be more accurate.

You will have to use the utility program IOSTAT2 to determine how busy your discs are, and what they are doing (swapping or accessing data files).

If the swapping is low, you may use CFS/3000. You will select the file to make core-resident by considering their impact on the response time.

With CFS/3000 you will be able to make trials without stopping the daily operation.

As a first step, you may declare core-resident some small files frequently accessed like.

- . Tables of parameters
- . Automatic master data sets (IMAGE)
- . KSAM key-files

In a second step, you will be able to design your future applications, by taking into account this new possibility.

Opening Address

DECENTRALIZED PROCESSING - NEW HORIZONS FOR SYSTEMS'DESIGNERS

NORMAN MIDGLEY
D. A. SAUNDERS

1. Introduce who you are: D.A. Saunders, B.Sc., M.I.C.S. OR ...
2. Thank audience for attending, as this is the last session and we realise a number of them would like to wind up.
3. Introduce subject matter:
 - a) We are going to talk about a Financial & Management Accounting System which utilises the most up to date System Software facilities available.
 - b) Computer Manufacturers invest substantial amounts of money in the Research & Development of Systems Software - however the development of application Software to match the System Software in capability is mostly out of the reach of the user due to its high cost.
 - c) This normally would have led to a totally unbalanced state in the development of advanced end-users software. However, imaginative systems' designers with the background and backing have clutched the challenge so that now there are some application systems matching the system software capability.
 - d) These systems in general are either produced by a Software House so that costs can be spread among many users or by extremely large concerns that can justify the expenditure.

NORMAN MIDGLEY
SALES REP.
EUROCO SOFTWARE SYSTEME
AN DER ALSTER 1
2000 HAMBURG 1
W - GERMANY

History

In 1971 when EUROSPAN started, the COMPUTER installations were in the main 'Main Frame' oriented. We approached our design of EUROSPAN by looking at the following:

- a) STRUCTURE OF FILES
- b) ACCOUNTING PRINCIPLES
- c) HISTORICAL COMPUTER DESIGN OF ACCOUNTING SYSTEMS

In structuring our files we recognised that the system should be capable of handling a multi division or multi company type structure. Indeed our first customer was a service bureau and needed such a structure. Systems Analysts traditionally designed systems which broke the accountants normal conventions which he was used to (i.e. non double-entry bookkeeping and seperate ledger systems whereas he was used to 3 in 1 type systems). This led to an alienation of accountants to computers so we brought the computer back to the liking of the accountant by designing an integrated double system for him. This design was not done by computer people but by accountants with the help of computer people. We were fortunate in having a qualified accountant who also was an expert in the computer field.

From 1971 the system evolved from a basic accounting system to a more complex one include such facilities as automatic creditor payments and cost accounting. This changing design brought us up to 1977 when we realised the significance of the emerging mini computer market place. Which market has brought computing power to companies heretofore unable to afford it.

We decided to completely rewrite EUROSPAN to allow for Online facilities whilst still retaining it's much vaunted and accepted structure. In addition auditors by now were aware of the possibilities and dangers of computerised accounts and through consultation with them we included new further enhanced security measures into EUROSPAN.

Theoretical Frame Work

There were many implications in designing and producing a truly Online-System. Many of you, no doubt, have been presented with semi-online (not interactive) type systems that effectively only replaced the punched card and we felt that our EUROSPAN should definitely not be confused with such systems and should be the tool of the non computer trained accountant.

The implications of such a system needed consideration in the following areas:

- Storage capabilities
- Hardware and
- Software Portability
- Integration of other applications
- End of Year Problems
- User Friendliness
- Speed
- Security in Multiprocessing

At that stage also computer manufacturers were introducing and 'Singing about' the new buzz word 'Decentralisation' (naturally this was influenced largely by an ever-aware end-user demand owing to their increased awareness of computer power) and we took the initiative of designing our system to cater for the communications problems that usually exist.

B I L D

In recognising communications we also took account of multi-international requirements by solving those at the same time. The implications of this are not technical but conceptual and are as follows:

- a) Currency
- b) Language
- c) Corporate Reporting

Methods and Techniques

In consideration of the implications of online, multi-international systems we adopted the following solution taking into account a rigid standards system development by one of our sister companies.

1.) Storage capabilities

In a 'multi type' situation large duplication of data can exist so we set out in adapting EUROSPAN to the end-user to perform a clear analysis of each Group's/Companies'/Divisions' responsibilities to cut down on the unnecessary holding of duplicate data by the use of a common data base and subsidiary ones. We integrated our ledgers into a simple data base.

2.) u.

3.) Hardware/Software

Nowadays functionality depends largely on the joint facilities provided both in hardware and software. We did not want a system which tied us exclusively to a particular manufacturer or operating system as we don't determine that manufacturer's future. We are confident that many of you present are in the same frame of mind. So we wrote the following facilities ourselves and used an industry standard language of COBOL.

- Menu Processor
- Screen Handler
- Logging and Recovery Routines
- File Handler
- Security Routines

These enabled us to retain a large degree of Manufacturer Independence.

4.) Integration of Job Functions/Other Applications

Through our menu processor we have the ability to switch from function to function without having to go back through the menu itself which gives the flexibility and ease of use demanded by the discerning end-user. In addition we adopted a parameter/table driver system to enable the end-user to define their own requirements (e.g. functions, languages, help routines, reporting formats) without the painstaking job of changing programs for minor requirements.

5.) End of Year Problems

Most systems we have come across impose fixed booking periods on the end-user and at the end of year before starting a new one cumbersome procedures must be carried out in balancing/finalising the old year before proceeding with the new. This costs a company real money in delayed revenue and has put an unacceptable strain on the data processing divisions within companies.

Through parameters, the EUROSPAN user can determine exactly the periods he requires and can post details to previous periods until he finally decides to close them off, whilst always having an up-to-date balance on all accounts for a period of two years. He can also retain his history for ad-indefinitum depending on his disc-storage.

6.) User Friendliness

It is enough to say that through the use of our menu system and breakdown of functions in the familiar way the user is used to using, he relates totally to the system.

7.) Speed

Speed can be a real 'bug bear' in any real-time system especially with sophisticated processing requirements. We incorporated the following procedures to overcome this:

- a) Logical grouping of fields in Data Base Design which afforded us less system overhead and additional logical data item space for individual requirements.

- 6 -

- Furthermore grouping of data items eliminates the need to concatenate data items and hence reduces system overhead.
- b) By freeing space on the data base other applications such as order processing, fixed assets accounting, stock/warehouse etc. control can be incorporated under the one schema.
 - c) Our menu processor enables the user to move between online functions without necessarily going through the menu each time, carrying across all relevant data. This makes a very efficient and fast usage of the system possible.
 - d) We have developed screen handling routines with prompt and help facility on each field. Online processing is character mode, i.e. fields are validated upon acceptance and redundant screen input by the user is avoided. Field by field processing has proven to be overall faster and more efficient, especially in a very large user environment.
 - e) Our file handling routines were also designed for fast input and retrieval of data. The database design and own logging/recovery procedures that we implemented reduce the overall system overhead.
 - f) Finally our concept of defining up to 99 different transaction types within the bookings-posting functions, with user-defined parameters, reduces dramatically the necessity to input recurrent data within logically similar types of transactions (e.g. posting invoices, cash etc.).
The speed of entry and validation is thus very high within all areas of postings, as default or predefined values need not be entered anymore by the user.

- 7 -

- 7 -

8.) Multiprocessing

One of the major headaches of software designers that multiprocessing has brought about, is security and recovery of data. As already mentioned, we have developed our own logging techniques.

The terminal number, user number, date, time, and type of transaction as well as the logical end of cycle for transaction cycles are part of the log information. One is thus in the position of determining the last transaction per terminal as well as any logically incomplete transactions. Recovery may be then effected, either from the previous data security, or merely by eliminating any incomplete transaction cycles from the database, i.e. working backwards.

9.) Communications

Communications software and hardware are now readily available. Postal links are constantly improving through postal radio developments and satellite communication.

We have found HP's DS software & H/W a very easy to use and reliable tool, within our distributed network applications.

Our philosophy in this area is simple. Provide the facility for immediate retrieval of relevant data through communications, but always ensure that every unit within a network may continue to process its own data independantly, at any time. We must subsequently provide subsets of the central database to remote locations and update these as required daily. The relevant information as to which data is required where, is held on file so that again, no redundant transfer of data needs to take place. Also a set of reconciliation procedures had to be developed to ensure the integrity of all remote databases. When communications are across borders, there are also time differences to take into account. Here the priority is to enable data exchange at the most efficient and also economic time, while ensuring maximum data security.

- 8 -

10. International

Factors are most important in any international system are:

- Currency
- Language
- Different accounting periods and fiscal years
- Different balance sheet formats
- Different reporting conventions
- Different legal considerations to be considered

We have given everyone of these points considerable attention, as the constantly growing list of our multinational users proves.

We have kept language constants separate from the program code.

Flexible currency conversion routines have been developed to enable the user to define the rules according to the procedures he is used to, and the legal regulations in the respective country.

Decimal points are not available if the foreign currency does not require them (increased speed of entry).

EUROSPAN offers multinational concerns the tools to maintain accounts in local and remote company format, currency and language, as well as providing each company with a number of different reporting, budgeting, consolidation and grouping options.

In addition to this our report generator WHAT-IF integrated with EUROSPAN gives the user an additional tool for retrieving data and producing new reports in any desired format, as and when necessary.

Result

Sophisticated application software for sophisticated system software is now made possible at a price that end-users' budgets can allow, as the development costs have been spread over a large number of users. Installation and trial time is also cut short by the fact that EUROSPAN is a proven system.

If you have time please come and see us at stand 1.

NEW DIRECTIONS IN INVESTMENT MANAGEMENT

F. HELSOM

R. STECK

**** WE DIDN'T RECEIVE THE PAPERS YET (EDITOR) ****

F. HELSOM

R. STECK

LINCOLN NAT. INVESTMENT MANAGEMENT CORP.

High Speed Digital Image processing using a Picture-Scanning technique on Incremental Plotters.

ABSTRACT:

A method for high speed digital image processing, using incremental plotters is described. This project involves the use of computer graphics to electronically scan a picture and convert the resultant analogue signals to digital signals, subsequently to be processed on a high resolution TV monitor. The picture scanning attachment consists of a high resolution optical reflective sensor and associated amplification stages. This is in the form of a small probe which replaces the pen and the pen holder in an incremental plotter, and thus allows pictures to be scanned and stored as digital output for subsequent processing by a digital computer. The processing of this digital output could be on a high resolution TV monitor or a printer, using the Grey Scale contrast technique. Relative merits of this scanning technique are discussed. A special mention is made of a possible interface with the Hewlett-Packard Laser Printer software, where the digitising process could be speeded up many fold.

Ramesh Panchal
Hewlett Packard Ltd
King Street Lane
Winnersh
Wokingham
Berks
RG11 5AR

INTRODUCTION:

Digital image processing technology has reached a stage in its development where a considerable impact has been made in most technological fields. In addition to commercial and military applications, its impact on the medical technology has been far reaching e.g. its use in brain and body scanners, where the resultant digital images are processed on a high resolution TV monitor.

A digital image memory and processor offers a wide range of capabilities. It can digitise and store image data provide scan conversion, improve signal-to-noise ratio (SNR) via frame integration or averaging, detect and accumulate image differences, enhance very low contrast images and provide an interface between video and digital technologies. A well designed system will also have the flexibility to be interfaced with a variety of image sensors, computers, displays and recording devices as well as provide for software expansion for image analysis applications. Figure 1 is a block diagram showing a state-of-art digital image memory and processor with such a range of capabilities.

As it can be envisaged from above, there are various techniques one can employ for digital image processing. This paper examines a very simple and inexpensive picture scanning device in conjunction with an incremental plotter to digitise a picture and process the image on a high resolution TV monitor. The study was a direct result of a requirement for an inexpensive method for inputting shaded images to a digital computer. One of the projects involved the use of these shaded images in comparing the relative merits of Fourier, Walsh and Haar transforms in data compression and noise reduction of images. The mathematics involved is rather complex and beyond the scope of this paper and hence has been omitted for simplicity.

The entire study was carried out in the department of Computer Science at Brunel University.

The outcome of this study offers a good, inexpensive practical avenue for digital image processing, and a possible application with the Hewlett Packard laser printer is mentioned.

THE SENSOR AND PLOTTER:

The picture scanning device, assembled at the Computer Science laboratory of Brunel University, consisted of the HEDS-1000 high resolution optical reflective sensor (Hewlett Packard 1979), mounted at the end of a small plastic tube which replaced the pen and the pen holder on the plotter. Mounted within the tube was the associated circuitry (see figure 2) comprising a current feedback amplifier utilising the sensor's internal transistor, thus providing current gain and bias point stability. Further gain was provided by an operational amplifier with adjustable output voltage level, which allowed for optimum contrast to be selected for any given shaded picture. The output from the sensor was converted by an ADC (Computer Technology 0000a) which was connected to a Modular One computer for storage as a data file.

The incremental plotter used (Computer Technology, 0000b) was a small roller type, rather than the drum or flat-bed construction which was the only one available at the time of this study.

Apart from the low cost aspect of this simple design approach, another favourable feature was that it enabled the employment of standard Calcomp digital plotter software in driving the modified pen carriage during picture scanning.

DIGITISED OUTPUT:

After image processing, the output can be programmatically presented to a line printer or a matrix printer using overprinting techniques, (See figure 3) with an output voltage swing from the sensor of 8v for a high contrast picture, and a noise level of less than 20mv peak-to-peak, a considerable number of grey levels can be achieved. In order to test the sensor, the digital output was initially processed on a matrix printer (Centronix 702) for which sixteen levels of grey were chosen. This produced a rather poor quality of overprinted images but proved to be sufficient to test the sensor. The subsequent image processing was carried out on a high resolution TV monitor.

PROBLEMS ENCOUNTERED:

Apart from the poor quality of overprinted images obtained from the matrix printer, a major problem area was due to picture wrinkling. As the plotter was a roller type, and the pictures were fastened on the plotter paper, there was a tendency for wrinkling to occur which caused image defocusing. A 0.5mm high ridge caused a 50% drop in reflected photo current. This was overcome, for test purposes by using small mint postage stamps (figure 4). (The problem would be non-existent on Hewlett-Packard flat-bed plotter series where the paper is electrostatically held absolutely flat on the plotter.)

CONCLUSION:

The sensor is sufficiently sensitive to detect variations in whiteness (or blackness) quite unnoticed by the naked eye. A high resolution is achievable which also enables line following and other complex scanning routines to be programmed.

APPLICATIONS AND FURTHER WORK:

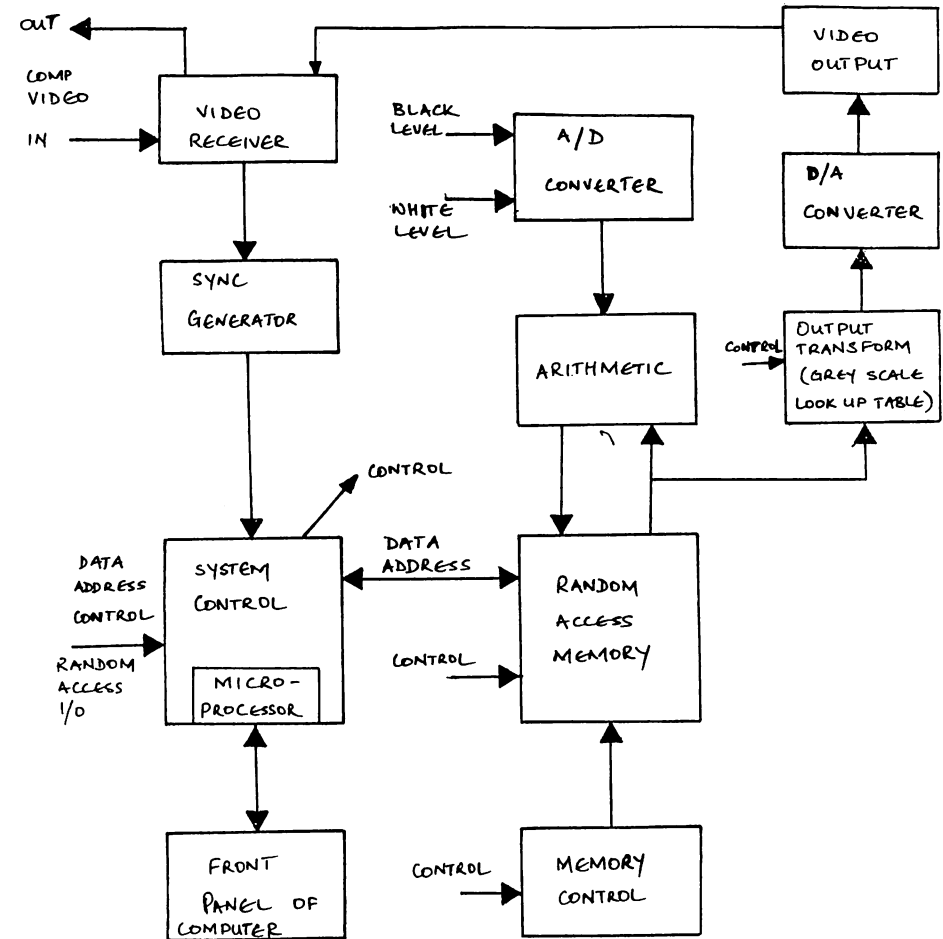
This technique provides a relatively inexpensive method for digital image processing. As a result of a matrix availability on the Hewlett Packard laser printer - 2680 software, a special character set with varying levels of darkness can be generated. Using IDSCCHAR, a character designing program provided with the 2680 software, a special set of characters, with each character cell showing a varying shade ranging from total white to total black can be generated. As there are numerous levels of grey achievable by the sensor, these can be associated with the special character set on the laser printer and a TV monitor like resolution is achievable, although further work in this area is necessary.

Similarly, the sensor can be used using the primary colour filters and data stored for each colour, which can be subsequently integrated on a colour monitor. It is also possible to transmit these digital data files across a communications network and process the image remotely.

REFERENCES:

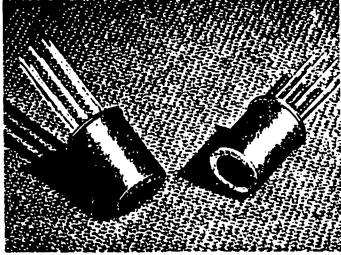
1. MENGERS Paul et al, Research & Development OCT 1977
2. PAGE Ivor & WOOD Robert, The Computer Journal, 24 1981
3. RYAN Daniel, Computer Aided Graphics & Design, MDI New York 1979.
4. HEWLETT PACKARD (1979) - High resolution optical reflective sensor. Data sheet - Hewlett Packard Components.
5. COMPUTER TECHNOLOGY (0000a) 1.722 Analogue input module, outline specification sheet - Computer Technology Ltd.
6. COMPUTER TECHNOLOGY (0000b) 1.43 Incremental Plotter - Modular One interim user handbook, Computer Technology Ltd.

Figure 1.



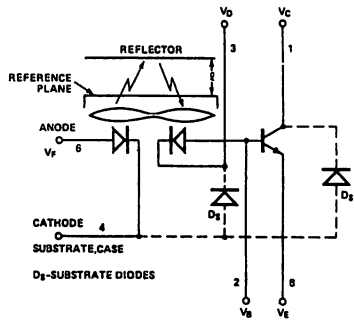
A block diagram shows relationship of elements comprising digital image memory/processor with random access memory.

Figure 2.

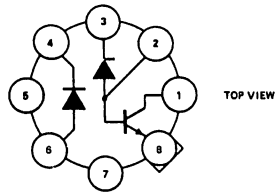


High resolution optical reflective sensor shown enlarged.

SCHEMATIC DIAGRAM



CONNECTION DIAGRAM



PIN	FUNCTION
1	TRANSISTOR COLLECTOR
2	TRANSISTOR BASE, PHOTODIODE ANODE
3	PHOTODIODE CATHODE
4	LED CATHODE, SUBSTRATE, CASE
5	NC
6	LED ANODE
7	NC
8	TRANSISTOR EMITTER

Figure 3.

```

PROGRAM FPRINT
J. *****
C....THIS PROGRAM READS A FILE AND PRINTS OUT A
C....PICTURE ON THE CENTRONIX 702. THE FILE FORMAT IS :-
C.... The first char. in the file is the letter N,
C.... followed by the size of picture matrix (1-128).
C.... The rest of the file is a string of greyscale
C.... values in the range 0-15, numbers outside
C.... this range will be scaled and truncated into it.
C.... These are read in free format, after the
C.... last number there should be another 'N' for
C.... a new picture or an 'E' to terminate the plot.
C.... The ch. set used is read from a datafile this
C.... file contains a string of ch. pairs which are
C.... overprinted to set the required pattern.
C.... Unit assignment is :-
C....      2 Console
C....      3 Datafile
C....      4 Tty line for centronix. e.g. /TTB
C....      5 Char. set file, currently .PCCHSET
C....      DIMENSION LINE1(128),LINE2(128),IGREY1(16),
+IGREY2(16),IPICT(128),VALUES(128)
C.... READ CH SET FROM UNIT 5
DO 15 I=1,16
  READ (5,1015)ICH
1015 FORMAT(A2)
C.... leftshift 1 and rightshift 9 to get top 7 bits
! LDA ICH
! SFT 49
! SFT 41
! STA ICH1
C.... left 9 right 9 for bottom 7 bits
! LDA ICH
! SFT 57
! SFT 41
! STA ICH2
IGREY1 (I) = ICH1
IGREY2 (I) = ICH2
15 CONTINUE
LN=78
LE=69
C....READ SINGLE CHAR FROM DATAFILE
10 READ (3,1010)ICH
1010 FORMAT(A1)
C....left 1 & right 9 to convert to 7 bit value
! LDA ICH
! SFT 49
! SFT 41
! STA ICH
C....CHECK FOR N OR E
IF (ICH.EQ.LE) GOTO 9999
IF (ICH.EQ.LN) GOTO 20
GOTO 9991
20 CONTINUE
C....READ SIZE OF SQUARE FROM DATAFILE
READ (3,1020)ISQR
IF (ISQR.LE.0) GOTO 9995
IF (ISQR.GT.128) GOTO 9995
  
```

```

C....READ SCALING FACTOR FROM TERMINAL
WRITE(2,25)
25  FORMAT(' TYPE SCALING FACTOR? '/')
   READ(2,1020) FACTOR
   DO 200 J=1,ISQR
   READ (3,1020)(VALUES(I),I=1,ISQR)
1020 FORMAT()
   DO 100 I=1,ISQR
   IPICT (I) = VALUES(I)*FACTOR
90  IF (IPICT(I).LT.0) IPICT(I)=0
   IF (IPICT(I).GT.15) IPICT(I)=15
   IGREY=IPICT(I)+1
100  LINE1 (I) = IGREY1 (IGREY)
   LINE2 (I) = IGREY2 (IGREY)
   WRITE (4,1050)(LINE1(I),I=1,ISQR)
   CALL CR
1050 WRITE (4,1050)(LINE2(I),I=1,ISQR)
   FORMAT (1H+,128A2)
   CALL CR
   CALL LF
200  CONTINUE
   WRITE (4,1055)
1055 FORMAT (1H1)
   GOTO 10
9991 WRITE (2,2010)
2010 FORMAT (1X,'UNKNOWN CHARACTER'/)
   GOTO 9999
9995 WRITE (2,2020)ISQR
2020 FORMAT (1X,'MATRIX SIZE OF ',I6,' IS OUT OF RANGE'/)
9999 STOP
   END

```

```

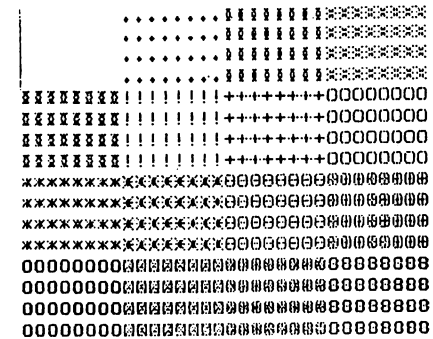
SUBROUTINE CR
NULL = 0
ICR = 13
WRITE (4,1000)ICR
100  DO 100 I1=1,13
   WRITE (4,1000)NULL
1000 FORMAT (1H+,A2)
   RETURN
   END

```

```

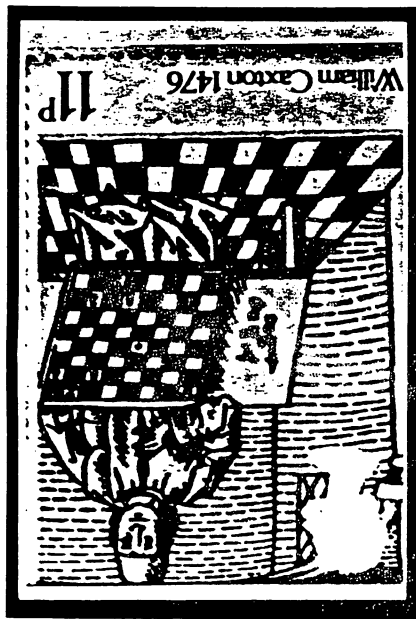
SUBROUTINE LF
NULL = 0
ILF = 10
WRITE (4,1000)ILF
100  DO 100 I2=1,13
   WRITE (4,1000)NULL
1000 FORMAT (1H+,A2)
   RETURN
   END

```

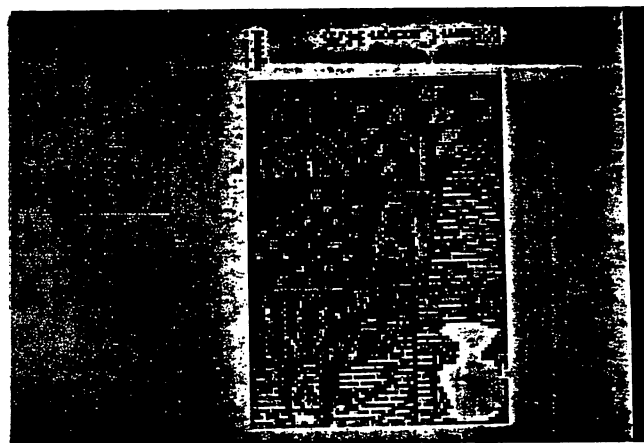


16 Grey Scale values used for digital image numbers

Figure 4.



B. High resolution sample



A. Low resolution sample

Understanding Hewlett-Packard, a view from the inside

by

Jan Stambaugh

Computer Systems Division
19447 Pruneridge Avenue
Cupertino, CA 95014

Four months ago, I resigned as chairman of the HP3000 International Users Group's board of directors to accept a position as manager of the quality improvement program for Hewlett-Packard's Computer Systems Division. In the past four months, I have learned a great deal about the Hewlett-Packard company, a complex yet simple organization which focuses on quality and excellence in the design of its products and services. It has been both amusing and disturbing for me to realize how little I ever really understood about how things work at Hewlett-Packard and why.

This paper is designed to help customers gain an awareness of Hewlett-Packard, its structure, its management, and its goals and objectives.

HP BCG BASIC

STEVE NG

ENGINEERING SECTION MANAGER

HEWLETT PACKARD GENERAL SYSTEMS DIVISION

SEPTEMBER 21, 1981

ABSTRACT

This paper describes the strategy for developing and maintaining a competitive, compatible BASIC language and compatible BASIC interface to tools (e.g., database, data entry, reports, graphics) for the Business Computer Group families (125, 250 and 3000). BCG BASIC project objectives, features, technology used and conversion aids will be discussed.

BACKGROUND

It has long been a goal at HP to have a single HP standard for BASIC to provide a growth path through the desktop computers to the HP1000 and the commercial machines (HP125, HP250 and HP3000). However, the implementors on single-user single-language machines and those on multi-user multi-language machines differ philosophically with respect to the inclusion of operating system functions at language level. As a result, we have more than a dozen BASICs at HP, most of them are incompatible with one another.

Our near-term goal is to narrow the number of BASICs at HP to two: BCG BASIC and TCG BASIC. In August this year, the General Systems Division has been given the charter of the BCG BASIC. GSD has the total responsibility for developing and maintaining a competitive, compatible BASIC language and compatible BASIC interface to tools (e.g., database, data entry, reports, graphics) for the Business Computer Group families (125, 250, 3000 and future BCG computers).

OBJECTIVES

Compatibility across all BCG machines is the primary objective. It is difficult to write an application program without using any tools such as database, data entry and report writers.

Even if we achieve 100% language compatibility but don't have a common interface to the application tools, then it will be extremely difficult to transport programs from one system to the other. Thus for the BCG BASIC project, we set the following objectives:

- o To provide a compatible BASIC language for the HP3000 and future BCG products.
- o To provide an upgrade path for current users of BASIC/250, BASIC/125 and BASIC/3000.
- o To be a user-friendly system in the 250 tradition.
- o To provide uniform interfaces to application tools on all target systems.
- o To provide conversion aids for current 125, 250 and 3000 BASIC users ensuring at least 80% automatic conversion.

FEATURES

The new ANSI BASIC standard, expected to be adopted in 1982, describes a very full and powerful BASIC language. It provides the user with capabilities and features previously lacking in the language and necessary for large applications. HP needs a compatible BASIC across all its product lines, and conforming to the BASIC ANSI standard is the best way to accomplish that.

BCG BASIC will include all of the features specified in the ANSI BASIC Level I standard proposed by ANSI-ECMA. In addition, many of the better features of the HP250 and HP125 will be implemented.

Highlights of the BCG BASIC include:

- o Identifier names up to 31 characters long
- o Named subprograms
- o Sophisticated exception handling
- o Integrated file system
- o Arrays up to six dimensions
- o Commercial formatter
- o 250-like programmer interface
- o Built-in application tools for database and Forms Management
- o Alphanumeric labels
- o IF-THEN-ELSE, WHILE-LOOPS AND CASE constructs
- o Templates for reading files written by other languages
- o Enhanced string handling
- o Support for non-English character sets and collating sequences

TECHNOLOGY USED

The BCG BASIC will use our Portable Compiler Writing System (PCWS) which is an integrated compiler system for a set of Programming languages and machine architecture and has been under development at HP for the last two years. The PCWS consists of a Common Intermediate Data Structure (CIDS), which is source language and architecture independent, and a Code Generator for each machine. With PCWS, it permits the production of M different languages for N different machines with only (M+N) rather than the traditional (MxN) programs.

Other advantages of the PCWS approach include the following:

1. Provides the user with uniformity across architectures for a given language and across languages for a given architecture.
2. Reduce development and maintenance costs.
3. Increase reliability and efficiency.
4. A global optimizer can be produced to work on the CIDS and hence all compiled languages.

In order to provide the friendly, interactive environment of the HP250, and to eliminate the current problem of having inconsistencies between the compiler and interpreter, BCG BASIC will be a hybrid interpreter/compiler implementation. Run-time performance should be somewhere between the current HP3000 interpreter and compiler. Special terminal drivers will be implemented for BCG BASIC to provide the 250 "personality".

CONVERSION AIDS

Utility programs will be developed which will attempt to automatically convert 125, 250 and 3000 BASIC programs and data files to BCG BASIC. A report will be generated detailing the changes made to the program. If the converter has problems or cannot translate certain constructs, the user will be asked for more information and/or the statement in question will be flagged for manual conversion.

Our goal is to have at least 80% automatic conversion for current 125, 250 and 3000 BASIC programs. This means that the users' conversion effort will be no more than 20% of the effort required to rewrite the application.

MPE IV

by

Mike Paivinen
Computer Systems Division
Hewlett-Packard Company
19447 Pruneridge Avenue
Cupertino, CA 95014

This presentation will discuss the MPE changes in the kernel policy at a conceptual level. A general overview of the new feature set will be given. This presentation is basically aimed at those people who are just becoming familiar with MPE IV and will not be an in-depth technical presentation.

JOB PROCESSING FUNCTION

Job processing for batch jobs can be broken down into several tasks:

1. SCHEDULING. This consists of determining when to run each job that is on the Master Job list. The Master Job list is the set of all batch jobs whose processing needs to be scheduled (reasons for scheduling will be discussed). Along with the scheduling of jobs, interdependencies among jobs must be taken into consideration. Certain jobs' processing will be "dependent" upon the successful completion of other jobs. If the job stream that processes Labor vouchers aborts, we would not want to begin the processing of our Labor Variance job stream until we successfully finish the Labor Vouchers. Scheduling also includes identifying jobs that require "extra considerations" such as tape mounts or private volume mounts.

2. PROCESSING. This includes 2 interrelated tasks:

a. Streaming the jobs on the schedule. For the next job to be processed, a check must be made for the successful completion of all jobs this job is dependent on, check to make sure all extra considerations have been taken care of (private volumes, tapes, etc.), and then stream the job.

b. Monitoring the processing of each job. Processing must be monitored so that when a job finishes or aborts,

processing continues with either a restart of the aborted job or the streaming of the next available job.

3. REPORTING. After all processing is completed, the results must be recorded onto the master schedule and reported to the system users. In Data Processing it is necessary to inform the users as to what was accomplished and to inform the systems group of what problems must be fixed.

AUTOMATION OF THE JOB PROCESSING FUNCTION: ACE

We have found that almost all of the job processing function can be automated. ACE is our applications subsystem that handles ALL of our schedule processing for our Accounting department. ACE is an online job-scheduling and processing application. The application controls simultaneous processing queues of jobs in a multi-machine network. A single master schedule controls the queues and the machines on which execution will occur. ACE is operatorless in its execution and requires NO changes in existing application programs or JCL to implement. The subsystem has two online programs, SCHEDULER and PROCESSOR, that are used to create and execute the job schedules.

Taking the job processing function described above it will now be shown how ACE is set up and executed to process jobs. Figure 2.1 shows an overview of the implementation of the ACE subsystem.

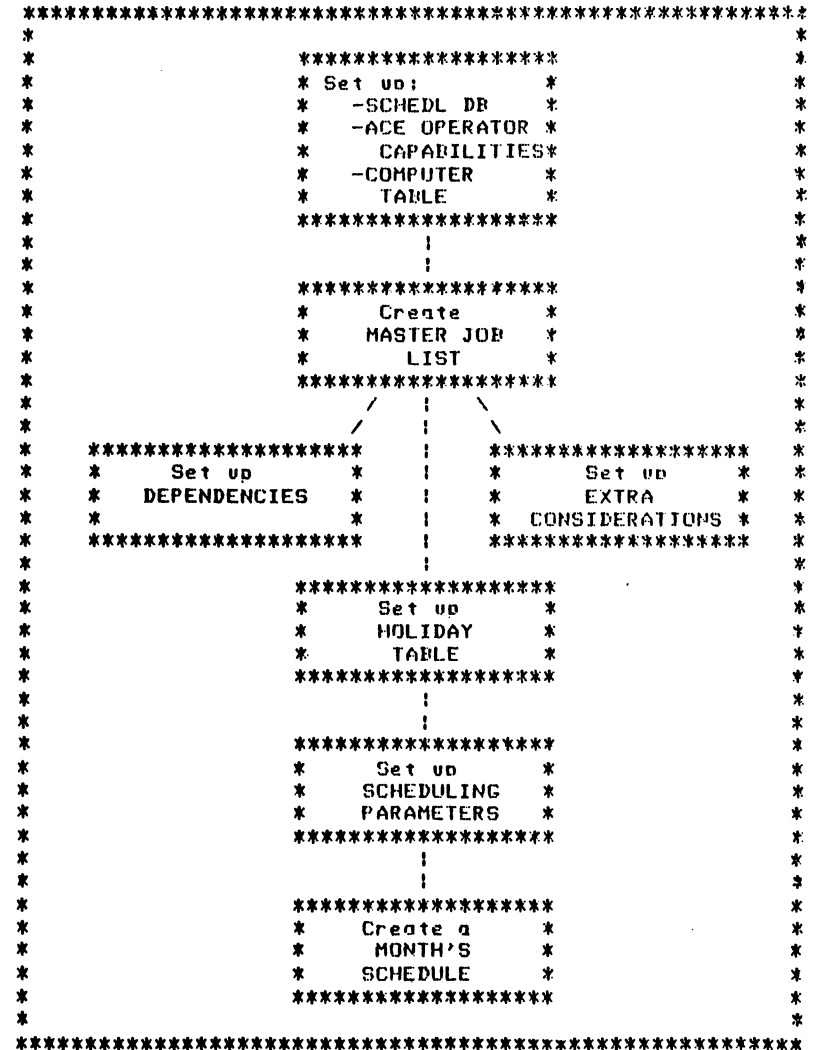


Figure 2.1 Implementing ACE

ACE: THE MASTER JOB LIST

ACE begins with the creation of a MASTER JOB LIST. This list contains all the jobs (on all the machines) that ACE will control. To create the list only two pieces of information are necessary, (1) the fully-qualified name of the file in which the JCL resides, which will be called the "JOB-NAME", and (2) the computer on which the JCL usually resides. Optional information includes "DEPENDENCIES", "EXTRA-CONSIDERATIONS", START- TIME-STOP-TIME WINDOW, DEFAULT-QUEUE, AND DEFAULT- SEQUENCE NUMBER.

Dependencies are the job-names of those jobs upon whose successful completion the current job depends. Extra considerations are dependencies that ACE can't handle. The application can check to see if the proper private volume is currently mounted, but it cannot physically mount the private volume itself. So the extra consideration is the reminder to the person setting up the daily schedule that "PRIVATE VOLUME AUDIT01 IS NEEDED FOR THIS JOB". Extra considerations are also commonly used to remind of needed tape mounts. The start-time-stop-time window allows a scheduled job to be run only between two specified times, ie. the "window". If the window cannot be satisfied, the job is not streamed. The default queue and default sequence number allow a one-time specification of these items when their values are fairly static.

ACE: CREATING MONTHLY JOB SCHEDULES

Once the Master Job List has been specified the monthly job schedules can be created. To aid in the scheduling of jobs a set of "scheduling parameters" are defined within the system and can be used to create a schedule.

In our Accounting department we process Labor Vouchers every Wednesday night. We process our first General Ledger to microfiche on the 3rd working-day of every month. We process some jobs every other week on Monday, some jobs every day of the week and some jobs only twice a year. What we have found is that most of our jobs have a distinct pattern to their schedule. We've defined a set of parameters that describe these patterns to aid us in our monthly scheduling. These parameters allow for scheduling a job to be executed by the day of the week, the workday of the month or the particular month of the year. Using combinations of parameters any pattern can be described. There are also parameters that allow for the exclusion of days, weeks, or months.

Once the parameters have been specified for each job, creating a specified month's schedule consists of "applying" these parameters to the month's calendar. The result is the actual schedule of the date each job will run during that month. ACE maintains a table of holidays (inputted by the user) and will not schedule jobs on those specific dates. Once satisfied with the schedule for the month, and when appropriate (eg. the beginning of the month), the schedule is activated for use.

ACE: DAILY JOB PROCESSING

Daily processing begins by entering SCHEDULER and requesting the day's schedule. All jobs scheduled for this day, and all jobs from previous schedules that either aborted or were "deferred" will be presented. Only when a job finishes successfully or is cancelled is it removed from the current schedule. Sequence numbers and queue numbers must be assigned to each job on the schedule. Dependencies are reviewed and are "deactivated" or additional added if desired. For jobs requiring private volumes or tapes, the private volume name or the logical device number on which the tape resides can be entered into the system. (PROCESSOR will then check to make sure the private volume is mounted before streaming a job and in the case of tapes will reply to the first tape request). Each job that appears on the schedule can either (1) be left scheduled to run this day, (2) deferred for future processing, or (3) cancelled from the schedule. Deferred jobs will continue to appear each following day until scheduled or cancelled. Cancelled jobs will not appear again until the next date they are scheduled to run. Jobs that were not scheduled may be added to the schedule at this time as long as they exist on the Master Job List. For all jobs that are left on the schedule, MPE passwords are entered and the schedule is ready to process.

The program PROCESSOR (Figure 3.1) executes the schedule. PROCESSOR is an online program that can be run from any terminal on the system. The program is executed in one of two modes:

- (1) AUTOMATIC RESUME mode in which PROCESSOR will execute the entire schedule without need of any user interaction. This is the pure "operatorless" mode.
- (2) MANUAL RESUME mode in which PROCESSOR will execute the entire schedule by stopping after each job is streamed and waiting for the user to let it stream its next job. This mode is extremely useful when many jobs are dependent on a single job. One can let the first job run and then monitor the job to completion before letting PROCESSOR pick the next job. Should the key job abort the user could fix the problem and restart the job so it finishes successfully. This way the jobs depending on its completion would be streamed instead of being deferred because their dependencies couldn't be met.

The program allows the user to switch between automatic resume and manual resume at any time by using control-Y. The user could monitor that first key job using manual resume and upon its successful completion leave PROCESSOR in automatic-resume and go home.

PROCESSOR picks the next job to stream according to a straightforward set of rules.

- (1) Only one job from a particular queue can be in processing at any time. Until the outcome of the current job is resolved, all following jobs in that queue wait.
- (2) Each queue is processed in the order of the sequence numbers assigned to each job. No job will be considered

for processing until all jobs ahead of it have been processed (this means either finished successfully, aborted or deferred because dependencies weren't met)

(3) PROCESSOR will continually loop sequentially through the queues checking to see if the next job in each queue is ready to stream or must be deferred because of dependency aborts. PROCESSOR can have as many jobs processing as there are queues active for the schedule (it can't be more because of rule (1) and it could be less because dependencies can cross queues forcing one queue to wait until another's job finishes).

```
*****
* MAIN MENU *
*****
```

```
*****
*                               *
*****
*                               *
*****
*AUTOMATIC *LIST SCHEDULE* *CHANGE SCHEDULE* *RUN SYS UTIL*
RESUME      *LIST SCHEDULE* *SUSPEND A QUEUE *STREAM A JOB
*MANUAL     ONLINE/OFFLINE *DELETE A QUEUE *RUN TOP
RESUME      *LIST A QUEUE   *ADD, DELETE, OR *RUN SPOOK
            ONLINE/OFFLINE *MODIFY A JOB  *EXECUTE AN
            *LIST A JOB    MPE COMMAND
            ONLINE/OFFLINE
            *'SHOW' COMMANDS
            SHOWJOB JOB=@J
            SHOWJOB
            SHOWOUT JOB=@
            SHOWOUT JOB=@J
            SHOWOUT JOB=@;READY,N
```

Figure 3.1 PROCESSOR.

First, there are a whole set of "LIST" commands for reviewing scheduled and running jobs. Jobs, queues, or the entire schedule can be listed either online or offline. Figures 3.2 and 3.3 show the format of the schedule lists.

Second, the schedule can be altered from within PROCESSOR. Queues can be deferred, cancelled, or suspended. Jobs can be added, modified or deleted. When modifying jobs, dependencies can be added or deactivated.

Third, PROCESSOR uses process handling to allow the user to stream a job, run TDP,PUB,SYS, run SPOOK,PUB,SYS, or execute other MPE commands. This is to allow for restarting aborted jobs from within PROCESSOR. SPOOK is used to read spoolfiles, TDP is an editor, and the stream command not only streams the job but also sets up the job streamed as an ACE job.

Besides being able to specify which mode to run the schedule in, there are other tasks which can be performed from PROCESSOR.

QUEUE	PRIORITY*	STATUS	COMPUTER	CODE	JOB-NAME	DEPENDENCIES/ EXTRA CONSIDERATIONS
01*10*01	R				JOB11.GROUP.ACCOUNT	**JOBXX.GROUP.ACCOUNT **JOBY.Y.GROUP.ACCOUNT JOBZZ.GROUP.ACCOUNT
01*20*02	S				JOB12.GROUP.ACCOUNT	**JOB11.GROUP.ACCOUNT NEEDS TAPE MOUNT FOR LAST STEP

Figure 3.2 Current Schedule List

```

JOB11.GROUP.ACCOUNT      STATUS-CODE:      SCHEDULE-DATE
                           QUEUE-NUMBER:  __      DEFAULT-QUEUE-NUMBER:  __
                           PRIORITY:  __      DEFAULT-PRIORITY:  __
                           COMPUTER-NUMBER:  __      DEFAULT-COMPUTER-NUMBER:  __
                           START-TIME:  _____      START-TIME-WINDOW:  _____
                           STOP-TIME:  _____      STOP-TIME-WINDOW:  _____
                           AVG-RUN-TIME:  _____

                           1           2           3
                           12345678901234567890123456789012
                           -----
                           THIS MONTH'S SCHEDULE:  0000R0AR0000DR010000010000010001
  
```

DEPENDENCIES: JOBXX.GROUP.ACCOUNT
 JOBY.Y.GROUP.ACCOUNT
 JOBZZ.GROUP.ACCOUNT

EXTRA CONSIDERATIONS:

JOB11.GROUP.ACCOUNT INFORMATION:

Figure 3.3 Current Schedule Job Information.

When PROCESSOR finishes processing the production schedule the last job streamed generates a report summarizing the results of the processing. Aborted jobs, deferred jobs, cancelled jobs, and successful jobs are all reported in this report. We then post this report for the users.

An Introduction to CCITT Recommendation X.21

Bill Baddeley
Hewlett Packard
Commercial Systems Pinewood

This paper is an introduction to CCITT Recommendation X.21. The recommendation specifies a new data communications interface which you will probably encounter soon if you haven't already.

At the present time, most data communications among remote terminals and computer systems are carried over the public switched telephone network or over leased telephone circuits. The telephone network has evolved over many years and is an excellent medium for voice transmission. When people started connecting computers and terminals over long distances, the telephone network provided a readily available, though not an optimal, connection medium for digital signalling. Computer systems and terminal equipment are typically connected to the public switched telephone network or leased telephone lines through an interface which is referred to by the label RS-232 or V.24. This interface type has been around for quite some time and is commonly available on modem and terminal equipment.

An organization devoted to international cooperation in telecommunications, the CCITT, is an international advisory body which deals with telephone and telegraph communications. The CCITT issues recommendations for services and implementation of services. There is a series of CCITT recommendations (the "V-series" recommendations) for the connection of data terminal equipment or DTE (the category DTE includes terminals and computer systems) to the telephone network through a modem. The connection point to the telephone network is called the data circuit terminating equipment, or DCE. The standard connection between terminals or systems and the telephone network is described in CCITT recommendation V.24 and the EIA (the U.S. Electronic Industry Association) RS-232. The 25-pin connector which you may have seen on your terminal cable or computer system is the standard connector for this interface. The V.24 or RS232 connection carries signals between the terminal or computer and the modem. These signals include the transmitted and received data signals, timing information and control signals which constitute a dialog between the modem and the computer concerning the state of the connection. For leased telephone circuits, the V.24/RS-232 connection carries all of the information needed between the modem and the system or terminal. The V.24 recommendation specifies the function of each of the data and control circuits in the interface, and the dialog between the terminal/system and the modem/DCE. For switched telephone connections, some PTTs offer automatic calling units. CCITT

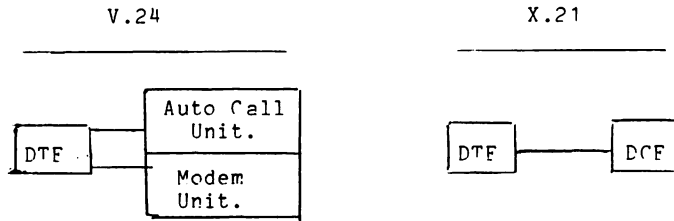
recommendation V.24 also describes the standard interface between a system or terminal and the automatic calling unit (the EIA specification is RS-366). This interface uses the same 25-pin connector as the V.24 modem interface with redefined signal lines. The V.24 recommendation specifies the dialog between the system/terminal and the automatic calling unit for establishing connections.

The telephone network is being replaced for some classes of data communications traffic by public data networks. There is a lot of information in the computer press these days about the new public data networks and the new international standards for these data networks. Depending on where you are located, you may have heard of either X.25 or X.21 or both. The CCITT has issued a series of recommendations (the X series) which deal with data communications networks. The X series of recommendations deal with services on networks which are specifically designed for data communications. There are quite a few public data networks now in operation, and more services are planned for introduction within the next few years.

These networks can be broken into two principal categories according to the type of service which they offer. One type is Circuit-Switched, where the connection between two systems or terminals is equivalent to a hardwired connection once it has been established through the network. Examples of circuit switched networks are the Nordic Public Data Network in Denmark, Finland, Norway and Sweden and the DATEX-L network in the Federal Republic of Germany. The other type of network is Packet-Switched. Packet switched networks support multiple virtual connections through the network over a single DTE/DCE connection.

The CCITT X.21 recommendation describes an interface between Data Terminal Equipment (terminals and systems) and Data Circuit-terminating Equipment for synchronous operation on public data networks. This interface replaces the two-connector, multiple signal interface of V.24 with a simpler set of signals, a smaller connector, and improved electrical characteristics (fig. 1).

Comparison of CCITT Recommendations V.24 and X.21



Distance:	< 20 Metres	> 1000 Metres
Speed:	< 20 Kbit/sec.	> 100 Kbit/sec.
Signals:	<= 31 signals	<= 7 signals
Connectors:	1-2 25-pin	1 15-pin

The X.21 interface is better able to meet the requirements of current systems for data communications than is V.24 in terms of speed and distance between the system and the network port interface. The speeds of service in public data networks under the CCITT X recommendations are easily met by the X.21 interface, the fastest being 48 kbits/sec. For this reason, X.21 is gradually replacing the V.24 interface.

The CCITT X.25 recommendation, which covers packet-switched data networks, specifies the X.21 interface as the electrical and mechanical interface between the DTE and DCF. The X.25 recommendation will be covered in more detail later in this session.

In addition to the electrical and logical definition of the X.21 interface, the CCITT recommendation describes logical procedures for the operation of the interface in a circuit-switched network and in leased-circuit applications.

The circuit-switched network procedures are broken into four phases: the quiescent phase, when no connection exists between the local station and any remote station; the call establishment phase, when either the DTE starts a call or the DCF signals an incoming call; the data transfer phase, and; the call clearing phase. The circuit-switched network procedures are very much like the procedures one uses with the switched telephone network.

During the quiescent phase, the telephone is on-hook. In the X.21 case, the system/terminal and the network signal their respective states of readiness to establish a connection through the network.

The call establishment phase, in the case of the telephone conversation, begins when one lifts the receiver and dials a number or when the telephone rings, signalling an incoming call. In the X.21 network, the terminal/system signals to the DCF that it is preparing to issue selection signals; the DCF responds by signalling "proceed to select", and then the DTE issues a selection signal sequence specifying a remote station and/or network services. An incoming call is announced by the DCF to the DTE/system/terminal. All cases of call collision (incoming and outgoing calls at the same time) are resolved in favor of the outgoing call. If one dials a busy station or mis-dial a number, you receive in response a tone or tone sequence, or perhaps a recorded message. In the case of an X.21 network, the calling DTE receives call progress signals (figure) which indicate why a call is delayed or why it has failed. These call progress signals are useful in determining a reasonable next action. In addition to call progress signals, the X.21 recommendation describes optional facilities for transferring information such as called line identification to the calling DTE and calling line identification to the called DTE as part of the call setup procedure.

X.21 Call Progress Signals

<u>Group</u>	<u>Code</u>	<u>Meaning</u>
0	01	Terminal Called
	02	Redirected Call
	03	Connect When Free
2	20	No Connection
	21	Number Busy
	22	Selection Signals Procedure Error
	23	Selection Signals Transmission Error
4,5	41	Access Barred
	42	Changed Number
	43	Not Obtainable
	44	Out Of Order
	45	Controlled Not Ready
	46	Uncontrolled Not Ready
	47	DCF Power Off
	48	Invalid Facility Request
	49	Network Fault in Local Loop
	51	Call Information Service
52	Incompatible User Class of Service	
6	61	Network Congestion (short term)
7	71	Long-term Network Congestion
8	81	Registration/Cancellation Confirmed
	82	Redirection Activated
	83	Redirection Deactivated

Group 0 signals are delay conditions without call clearing
Group 2 signals indicate short-term conditions
Group 4 and 5 signals indicate long-term conditions
Group 6 signals are short-term network related conditions
Group 7 signals are long-term network related conditions
Group 8 signals are confirmation signals (with call clearing)

Once the call setup is complete, the connection between the calling and the called DTFs is transparent. The network transfers the states of each station's transmit data line bit-for-bit exactly as it appears at the DCF interface. The data phase continues until one of the DTFs signals a clear request. This is analogous to the conversation part of a telephone call.

Call clearing is signalled by either end and transferred to the opposite end. Following call clearing, the DTF and DCF re-enter the quiescent phase.

The circuit switched public data network has several advantages over the public telephone network, having been designed expressly for data communications. One clear advantage is automatic call establishment (no operator dialling).

The X-series recommendations include recommendations for a uniform international node numbering scheme which is analogous to the international telephone numbering scheme. For example, each Nordic network connection has a 6-digit network number, which is unique within the country. International calls include an international prefix, a network/nation code, and a network node identification number. Within the Nordic Public Data Network, calls are possible among the nations of Denmark, Finland, Norway and Sweden.

Another feature of the new public data networks is fast call establishment and high reliability. For example, the Nordic Public Data Network specifications state that all calls will be set up within 2 seconds, 99% will be set up within 0.5 seconds and 90% of all calls will be set up within 0.1 seconds. Similarly all call clearing operations will take under 0.2 seconds, with 90% under 0.05 seconds. This is clearly an improvement over the performance of the switched telephone network.

The X.21 call progress signals, described previously, provide a clear indication of the status of a given call attempt along with information about the probable success of a retrv.

Additional facilities allow the subscriber to simplify the call process by using short-form addresses for commonly called remote nodes. Access restrictions can be placed on a given node by specifying optional facilities to bar incoming or outgoing calls. A group number facility allows several ports to be accessed from the network by a common node address; a central computer can be equipped with several ports which, in addition to their unique addresses are also accessed via a common national/network number. This facility is also referred to as "multiple lines at the same address". The Closed User Group facility allows the creation of private networks within the larger public data network. If a company has several systems at different geographical locations, and has no need to connect them to systems outside of the particular set, then the specification of closed user group

membership for each of them eliminates access from computers outside the network. The facility can also be used in such a way as to restrict the connections from any node in the group to only other group members. It is also possible for a particular node to belong to several closed user groups, one of which is the default or preferential one. In order to switch from the preferential to an alternate closed user group, the selection signal sequence is prefixed with a facility request code specifying which alternate group is to be used for the call setup, followed by the address of the node within that group.

A call queueing facility allows incoming calls to be held in a first in first out queue with a specified number of positions. The caller receives a call progress signal indicating that the call is queued at the remote end. The call redirection facility allows one node to temporarily transfer its address to another node; for the period during which this facility is activated, all calls for the node are redirected automatically to the alternate node. A call progress signal informs callers that the call has been redirected. The Calling and Called Line identification facilities provide for verification and monitoring of connections made through the network. A node specifying the Charge Transfer facility is charged for all incoming calls (which are normally charged to the caller). The charge advice facility allows a node to be informed, following disconnection, of the charges for a call.

Circuit-switched X.21 networks are currently offered in Denmark, Finland, Norway, Sweden, F.R. Germany, and Japan. Several other European nations have X.21 networks in their future plans. Further information about the specification and the network services is available from the implementing PTTs, and from the CCITT (Union Internationale Des Telecommunications, Place des Nations, CH-1211 GENÈVE 20, SUISSE).

Design Considerations for Support X.25
Communicator Networks

by

Shnider Youssef-Digaleh
Hewlett-Packard Company
Information Networks Division
19447 Pruneridge Avenue
Cupertino, CA 95014

by
Richard Franklin
Grenoble Terminals Division
Hewlett Packard Company

Over the past decade Data Communications has been revolutionized by a radically new technology called Packet Switching. Ccitt Standards have been defined and adopted in a timely fashion as a solution to the urgent need for an agreed upon protocol. This has the advantage that Ccitt Standards are widely applicable and are not limited to a specific manufacturer or vendor defined protocol.

Hewlett-Packard has initiated a major commitment to the use of internationally accepted standards protocols as the fundamental basis for its Hewlett-Packard distributed System Network Communication Architecture.

Since the first HP terminal 2640A was introduced in November 1974, HP has constantly been making head lines thanks to its terminals features, reliability and contributions.

With 11 terminals, HP has a very wide range of terminals available today to fit any user needs in the five application areas onto which HP is concentrating.

- Data entry
- Program preparation
- Text preparation
- Data analysis
- Cad/Cam

Data analysis, mainly thanks to the HP3000 success, is by far the application where our terminals have been sold.

Today we introduce a new terminal, the 2382A office display terminal which offers features such as:

- 9" display
- 2640/2622A compatibility
- 2 full pages of memory
- 8 screen labeled softkeys

With its compact design, this unique device can fit on top of any desk.

The 2624B is also new in this area. It offers all the 2624A features plus multipoint, local forms cache, printer pass through and record mode.

For the ever growing data analysis application, HP presents today the 2623A graphics terminal with its block mode capability, 2 full pages of memory, user definable soft keys, national character sets, its 512 x 390 dots resolution, and DSG/3000 support, the 2623A provides a low cost graphics solution.

But not only HP works on designing new terminals but also on providing new software tools for existing products. For example 4 new software packages are now available on the 2647A.

INTRODUCTION

Interactive Mainframe Facility (IMF), previously known as Interactive Mainframe Link, has been enhanced with new features and improvements on existing features. The purpose of this paper is to introduce DSN/IMF to new users as well as inform current users of the improvements and new features.

OVERVIEW

IMF is a product that enables an HP 3000 Computer System to emulate a remote IBM 3270 series Cluster Control Unit. In previous releases, IMF only supported Binary Synchronous Communications (BSC). Now IMF also supports 3271 Synchronous Data Link Control (SDLC) as an SNA physical unit type one. Through IMF, users on an HP 3000 can send data to and receive data from applications on a remote host. A user can access a remote host application either programmatically through IMF Intrinsic or interactively through IMF's Pass Thru Facility.

It is possible to use IMF for data entry and retrieval to a remote host application. These host programs issue write commands which send screens of data to remote terminals and read commands which receive data from remote terminals. The host regulates data transmission, therefore IMF sends and receives data when the host instructs it to do so.

In order to establish communications with a host system, the HP 3000 uses an interface called an Intelligent Network Processor. The INP handles the line protocol and performs many 3270 controller functions. The INP software communicates to the HP 3000 software through HP's internal Communication Subsystem (CS). IMF and the INP together appear to the host like a 3270 Control Unit.

There are four major components in the IMF product. They are the INP, the IMF Intrinsic, Pass Through Mode, and the IMF Manager Program. To understand the product, a user needs to know what these components do and how they interact. These components and their interaction are briefly described below.

INTERACTIVE HP-3000 TO IBM HOST COMMUNICATIONS

Cynthia L. Smyth
Member of Technical Staff

Data Communications Laboratory
Information Networks Division

Hewlett-Packard Corporation
19447 Pruneridge Avenue
Cupertino, California, 95014 USA

IMF provides Intrinsics which users can call from COBOL, FORTRAN, SPL, or BASIC. In most cases, the intrinsics allow a user to tailor his HP 3000 program to fit an existing IBM application. The Intrinsics are easy to use, and they perform functions which include transmitting data, receiving data, reading a screen of data, and locating the 3270 field attribute bytes in a screen of data. An HP 3000 application using the IMF intrinsics appears as an interactive session to an IBM Host.

IMF also provides a Pass Thru Facility, previously called the Inquiry and Development Facility, which allows most HP terminals and printers to interact with a host application without additional programming on the HP 3000. When using the Pass Thru Facility, HP terminals and printers function as 3270 terminals and printers.

The IMF communications line is controlled by its Manager Program. From within this program, a user with OP (System Supervisor) capability can issue commands such as START and STOP to control the IMF subsystem. The Console Operator may also issue commands to control the subsystem.

IMF requires a configuration file for each IMF communications line in operation. This file contains information about the line such as the logical device number of the INP, the control unit number, and a list of the devices configured on the host system. The configuration file also contains a list of users that are allowed to use the communications line. The IMF manager may override the allowed list of users in the configuration file.

ENHANCEMENTS

As mentioned in the overview, IMF now gives the user the option of using either 3271 Synchronous Data Link Control (SDLC) as an SNA physical unit type one or Binary Synchronous Communications (BSC). There are many advantages in using SDLC protocol. SDLC provides better data security because it does more handshaking than BSC protocol. Although BSC lines can be supported in an SNA network, IMF support of SDLC eliminates the need and overhead to maintain BSC lines. SDLC protocol allows different types of SNA devices and systems to use the same multidrop line. Multidrop lines can use higher speeds with SDLC protocol than BSC protocol.

IMF now supports IBM's Conversational Monitor System (CMS). This is an important enhancement since CMS and IBM's Virtual Machine 370 (VM) are widely used.

The first release of IMF had a restriction on the size of a data block that could be received from the host application. This restriction has been removed in the second release for both protocols. A segmentation scheme has been implemented in the BSC driver to segment the incoming host data into 256 byte blocks for processing, and the SDLC protocol sends 256 bytes of data in one frame.

IMF's design has changed to solve two character code problems encountered in the first version. First, the ASCII to EBCDIC translation function has been moved from the INP to the 3000 and now uses MPE's CTRANSLATE Intrinsic. Therefore, users who have special character translations will see those translations in their IMF applications as in all other Data Communications products. Second, eight-bit character codes like Katakana will be easily supportable in the new version. Eight-bit character code support is now possible because IMF no longer uses the high order bit to denote a 3270 field attribute byte.

The receive timer for the RECV3270 Intrinsic has been expanded to cover the TRAN3270 Intrinsic so that a user may detect that the host has stopped communicating without needing to use no-wait I/O.

The IMF monitor now identifies the communications line number in its messages. This is essential for installations where more than one IMF line is in use at once.

The IMF Manager's DISPLAY ALL command has been improved. A message has been added to indicate whether the host is communicating or not. Also, if CS Trace is running the name of the currently opened trace file is displayed.

The default name on a CSTRACE file name has been changed to IMFTRXXX.PUB.SYS where XXX is the ldev of the pseudo-device.

When the Pass Thru Facility is terminated on a 2640B terminal, it will print a reminder message on the screen to unlatch the block mode key.

A new Intrinsic, PRINT3270, has been added to IMF. This Intrinsic will enable the user to get a hardcopy snapshot of his screen image. The screen can be printed in two different formats. One format shows attribute characters and nulls and is very useful for debug purposes. The other format prints the screen as it would be viewed on a terminal and thus provides some

INTERACTIVE HP 3000 TO IBM HOST COMMUNICATIONS

of the features of a local copy. A user interface to this Intrinsic will allow it to be called from the Pass Thru Facility.

There has been a change to the IMF philosophy when the host stops communicating. Basically, the IMF Subsystem will remain active if the host stops communicating and will not have to be restarted when the host starts communicating again. Pass Thru terminals, however, will exit the Pass Thru Facility and return to MPE. Users might have to re-establish their sessions with the host since IMF cannot maintain them on the host side. During the time that the host is down, any TRANS3270 or RECV3270 intrinsic calls will receive an error message stating that the host is not communicating. It is important to remember that IMF cannot distinguish when a host subsystem or application terminates if the access method continues to send control sequences through the phone line.

A data stream mode has been added to IMF to provide a means of file transfer between the host and the 3000. This mode will allow an application program to obtain the untranslated data stream from the host. This mode is only allowed when using SDLC protocol. Two new intrinsics, READSTREAM and WRITESTREAM, have been added for using this mode. The data stream is kept in the same format as the host sent it in and no screen image is produced. Hence, a user in data stream mode is not allowed to use any of the IMF Intrinsics which access an internal screen image. Likewise, a user who is not in data stream mode may not access the READSTREAM and WRITESTREAM Intrinsics as the data stream is not maintained.

CONCLUSION

For HP 3000 users who need to access data on a remote IBM mainframe, IMF is an excellent solution. In addition to 3270 emulation, IMF enables users to write HP 3000 applications that can communicate with host applications. The new functions and enhanced features provide increased functionality and extend IMF's useability in SNA host environments.

BUSINESS GRAPHICS

AN EFFECTIVE MEANS OF IMPROVING MANAGERIAL PRODUCTIVITY

Christopher Kocher
Product Manager
Information Networks Division
Hewlett-Packard Company
Cupertino, California

- I. Introduction
- II. A Historical Perspective of Office Productivity
- III. What is Management?
- IV. How can Graphics Help Managers Spend Time More Effectively
- V. How Does Graphics Help Managers Make Better Decisions?
- VI. Conclusion
- VII. Implications for Data Processing Managers
- VIII. References

I. INTRODUCTION

The market for business computer graphics products is expected to grow at 59% per year over the next five years. In part, this growth is a result of the increasing interest in managerial productivity and the realization that computer generated graphics is a tool that can be employed today to improve the effectiveness of management decision making.

Functional managers and professionals in finance, marketing, manufacturing, personnel and accounting are seeking better ways of defining and analyzing problems as well as communicating their findings to other decision makers. These managers are looking to their data processing departments for advice and recommendations on how to develop computer graphics systems. Data Processing managers will play a very important role in improving managerial productivity in the future.

This paper will describe some of the historical trends in Office Productivity, and then explore the role of management in today's organizations. Emphasis will be placed on how graphics can help managers in the decision making process. The conclusion also offers a brief discussion of the implications and opportunities that these trends have for data processing professionals.

II. A HISTORICAL PERSPECTIVE OF OFFICE PRODUCTIVITY

In this century enormous strides have been made in increasing the productivity of industry and agriculture. Technological innovations have greatly improved not only output per capita, but also the standard of living. Until relatively recently however, very little attention has been given to office productivity. This lack of concern is very apparent in the low capital investments and low productivity improvements made in the office in the last decade for the average worker in the United States.

	<u>Capital Investment</u>	<u>Productivity Gains</u> [1]
Office Worker	\$ 2,000	4%
Industrial Worker	\$25,000	90%
Farm Worker	\$35,000	185%

At the same time, it is interesting to note some of the other trends that are taking place in business organizations and specifically in the office:

- o In the United States, over 240 billion pages of computer printout was created in 1980, up 25% over 1979.[2]
- o Office workers are becoming a larger percentage of the total labor force as the U.S. becomes more of a service

oriented economy from 22% in 1980 to 30% in 1990.[3]

- o Electronics and memory costs are dropping between 20% and 40% per year, while labor costs are rising at 7%. [1]
- o The total cost of producing a letter in the United States is estimated at \$6.63.[4]

These facts all bear witness to what has commonly become called "The Information Explosion". As Alvin Toffler, the well known futurist points out in his book, The Third Wave, we have tried to automate our offices the way we automate factories in a routinized manner that increases the quantity of output but not necessarily the quality. This approach has not yielded commensurate productivity gains. More data is not always better.

The advent of electronic word processors is evidence of the economic incentives to automate the office. It is clear that there is vast room for improving the manner in which we handle documents. Time savings may be quantified and document creation costs reduced. What is not so measurable however, is management productivity. How does one measure a manager's productivity? The number of decisions made? The quantity of memos written? The length of meetings attended? Clearly these are not very good measures. And yet, when examining office productivity one needs to put secretarial and clerical vs. managerial and professional productivity in perspective.

- o 75% of the \$640 billion spent on direct labor costs in U.S. offices goes towards management and professional salaries and benefits.[5]
- o 80% of one U.S. insurance company's office costs went toward management and professional salaries.[1]

One needs to ask, why we are not looking more closely at improving managerial productivity. In fact, this process has begun.

The result has been a high degree of interest in tools to make managers more efficient and more effective: electronic mail, teleconferencing, decision support tools and graphics. Before exploring managerial productivity further one needs to define exactly what is meant by management and what managers do in the decision making process.

III. WHAT IS MANAGEMENT?

Managing means many things to many people. In fact, there are probably as many definitions of management as there are management styles. In any case, several key ingredients invariably appear in a description of management:

- o Planning, organizing and controlling
 - key elements in management.
- o "Getting things done through other people"
 - clearly communicating goals, providing incentives to motivate employees and delegating responsibility
- o Making Decisions
 - one of the main purposes of managers is to take relevant information and determine the best course of action or allocation of resources to maximize or minimize occurrences that conform to their organizations objectives.

From these definitions of management, one observation is very clear. If managers and business professionals primary function is decision making, then information is their most important resource. Note the distinction between data and information. There is an excessive amount of untimely and irrelevant data available to most managers today which explains the reason so many computer printouts go unread. Information on the other hand may be a summary of only the most important elements that affect a managers sphere of influence.

What type of information do managers need?

- o Relevant information
 - unrelated data will only be time-consuming and distracting
- o Timely information
 - obsolete data in todays competitive environment is usually worthless information
- o Accurate information
 - unreliable data may mislead a decision maker; however, spending excessive time accumulating and verifying it may make it untimely.
- o Summarized information
 - although some levels of decision makers need details, most managers need to see the "big picture". Managers can ill afford to spend time reading about an operation or division that only affects 1 or 2% of their sales, profits, costs, etc.
- o Well formatted information
 - nothing is worse than pertinent information that is buried in long listings of hundreds or even thousands of variables. Key facts, trends and relationships should stand

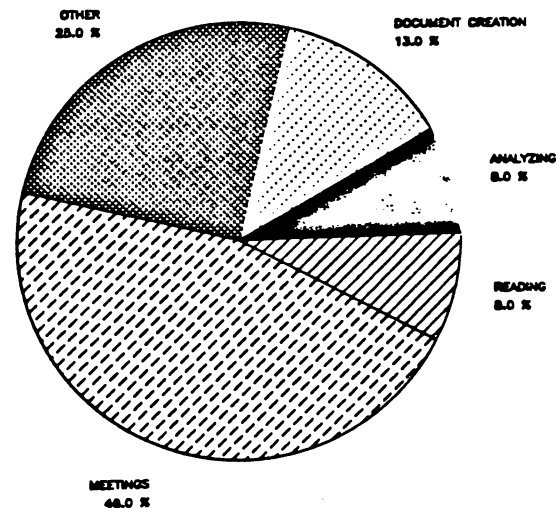
out so that managers can quickly grasp their meaning and proceed in their analysis.

With these information needs in mind, it is very interesting to observe how managers really do spend their time on a day to day basis.

IV. HOW CAN GRAPHICS HELP MANAGERS SPEND TIME MORE EFFECTIVELY?

Many theories have been postulated and estimates made of how managers spend their time. There have even been a number of professional studies. One of the most comprehensive surveys was conducted by recording the activities of 300 managers and professionals in 15 organizations every 20 minutes throughout their working day. [6] The results were very revealing:

MANAGERIAL ACTIVITY PROFILE



This vividly illustrates the role of meetings in the management process. Being able to convey information succinctly as well as understanding the presentations of others is essential for success. Any significant contribution to managerial productivity must address this communication intensive joint decision-making process.

Meetings

Electronic mail may reduce the need for some meetings and teleconferencing may reduce the travel time and expense of others, but graphics can make meetings more effective. Using visual aids such as charts and graphs:

- o Ideas can be conveyed more rapidly using colors and textures in a visual format -- reducing meeting time. It is interesting to note that 35% of managers thought meetings were too long and 34% thought they were unproductive[6]
- o Information can be displayed in a more interesting format -- keeping the attention of participants
- o Concepts are more easily retained by listeners because humans are more accustomed to storing visual images. This improves the results of the meeting. For a physiological explanation of why graphs and visual images are more easily retained by the human brain see [7].

The most relevant data can be presented in a summarized format -- focusing attention on the key points of discussion.

Document Creation

Document creation which consumes 13% of the average manager's time consists of composing, creating, editing, designing, and drawing documents. This activity has enjoyed some productivity improvements with the advent of less expensive dictation equipment and word processing. Graphics can also make a contribution in this activity by providing managers with the tools to more effectively communicate their ideas and influence other decision makers. In realization of this fact, the Harvard Business School recently created a new course on the use of business graphs.

Reading

Reading which accounts for 8% of managers and professionals time would benefit greatly if document creators used graphics since readers would be able to more quickly grasp key ideas. In fact, the analysis of [7] indicates that the Human brain is able to absorb between 48 and 72 million words per minute in visual images vs about 600 to 1200 words per minute for the average reader.

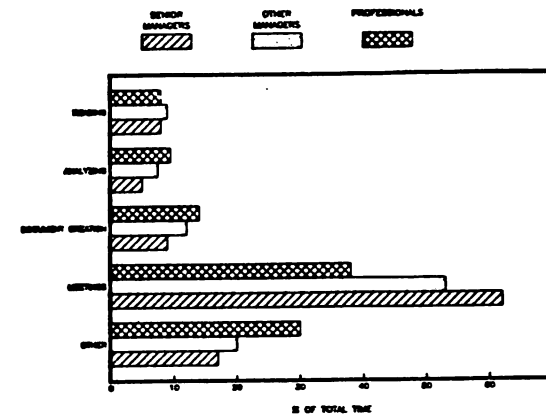
Aside from these activities, professionals and managers spend about 25% of their time on less productive activities that should frequently be performed by support personnel: filing, copying, seeking information, seeing people, scheduling, organizing work,

waiting, etc. Electronic filing, electronic mail, and intelligent copiers will help to improve productivity in this area in the future. Traditional business display graphics are not very relevant to these activities although some software packages will provide capabilities to create organizational charts, Gant charts, and Program Evaluation and Review Techniue charts, (PERT). Some professionals will use these specialized tools, quite frequently but support personnel will probably prepare these charts because of their routinized nature.

The last activity which takes up an average of 8% of a managers time is analysis. This is probably the one activity that people most closely associate with managing and decision making. Before discussing the very important role of graphics in analysis and the decision making process one last point needs to be made.

Management styles are different just as managers' jobs vary from one organization to another. The amount of time spent on different activities varies from one manager to the next. In the chart below, it is apparent that professionals and lower level managers spend more time on less productive tasks probably because they do not have supporting staffs. It is also clear that they spend more time doing analysis and document preparation than upper level managers. At the same time, senior managers spend considerably more time in meetings.

Figure 2



From these observations a picture begins to emerge of lower level managers collecting and analyzing data, generating solutions and communicating these solutions to upper level managers who review them. It also brings up the question of the type of data that is desired by different managers.

Typically lower level managers deal with data internal to the organization (orders, shipments, inventories, etc.) while senior level managers deal with both internal and external data (inflation rates, interest rates, industry growth, etc.). These lower level managers and professionals may need graphs produced from data resident on an existing data base while senior managers may want to use their own data.

In fact, Computerworld magazine estimated that "top managers spend less than 2% or 3% of their time dealing with computer printouts, terminals, or intelligent stations".[8]

This has important ramifications for the decision making process as we will see in the next section.

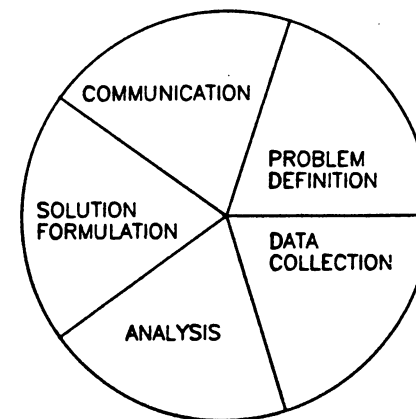
V. HOW DOES GRAPHICS HELP MANAGERS MAKE BETTER DECISIONS?

With this basic understanding of how managers spend their time and the ways graphics can make them more productive, it is instructive to look even more closely at how an individual manager might go through the decision making process. Specifically, I will use Hewlett-Packard's Decision Support Graphics/3000 as an example of a powerful computer graphics package that can help managers make better decisions.

Although there are a multitude of ways to make decisions ranging from the basic scientific method to sophisticated operations research models, there are usually several steps that are common to all techniques. The schematic below illustrates some of the main steps in any decision making process.

Figure 3

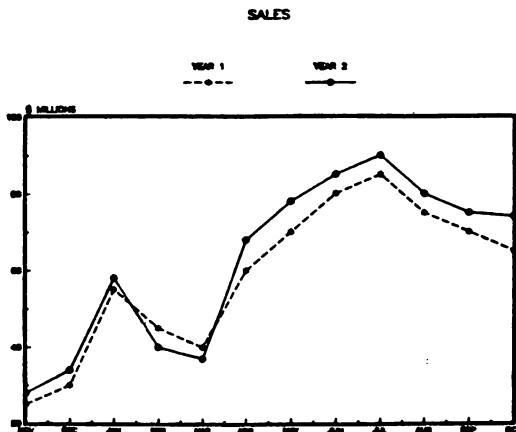
THE DECISION MAKING PROCESS



Problem Definition: Before launching into any type of analysis, the decision maker needs a clear understanding of the problem at hand, perhaps even further defining it. In some cases, objectives need to be set and assumptions stated. In others, the decision maker may need to question the basic premise of whether a problem even exists or what the decision criteria are.

A good example of how a graph can help define a problem is shown below. A manager looking at this sales data might be very concerned about the decline in sales. However, by displaying last years data next to this years, it is apparent that there may not even be a problem, rather, sales are seasonal and one can expect a decline during certain months.

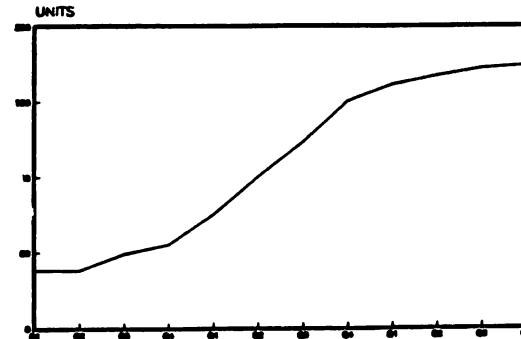
Figure 4



The next example shows sales of a product over several years. By looking at a column of numbers, it might be obvious that sales have been growing. In the graph below however, something else emerges: sales grew at a rapid pace the first three years but have almost leveled off in the last year.

Figure 5

SALES GROWTH



Data Collection: Once the problem is defined, the decision maker can proceed to gather data that is relevant to the decision. This data may reside in file cabinets, in reports, with other people or hopefully, in a computer data base where it can be quickly retrieved.

DSG/3000 was specifically designed with data retrieval in mind. If data originates from a managers own information base (external to the organization), this information can be entered quickly by filling in a menu. If the data is in an IMAGE data base, Query may be used to retrieve the data for a graph. Other tabular MPE files may also be used. Below is an example showing how data can be entered in the DSG/3000 data prompt menu.

Figure 6

120	139	155	FEB	1001 TS
145	155	199	MAR	1002 TS
166	172	215	APR	1003 TS
188	193	240	MAY	1004 TS
193	205	245	JUN	1005 TS
175	210	238	JUL	1006 TS
183	205	265	AUG	1007 TS
199	225	275	SEP	1008 TS
210	238	278	OCT	1009 TS
118	255	290	NOV	1010 TS
29	29	29	DEC	1011 TS

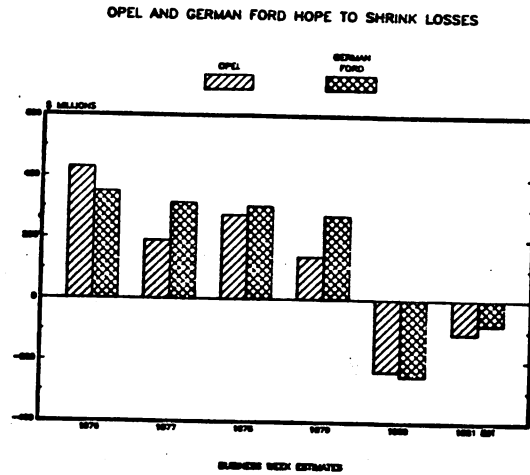
Thus, DSG/3000 is ideal for periodic reporting where the same chart is used month after month since it can be updated automatically with the most recent data. At the same time, however, customized graphs can be created with manually entered data for special presentations. This flexibility is important to consider when designing a graphics system. Since senior managers often deal with external data that is not resident in the organization's information systems.

Analysis: With the appropriate data in hand, the decision maker can proceed to determine what is relevant, how accurate it is, and then begin analysis. Frequently, this is the longest step in the process because the data must be prepared and formatted so it can be manipulated. Most business managers and professionals then perform some combination of the following analyses:

- o extrapolating trends over time
- o studying relationships between variables
- o making comparisons between entities
- o looking for exceptions and variances

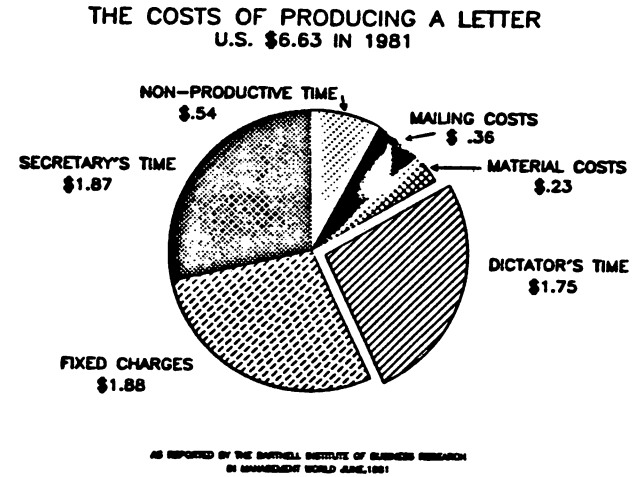
These four types of analyses lend themselves to visual portrayal. Trends as we saw in the example of sales over time are usually displayed in a line chart or a series of bars in a bar chart as shown below.

Figure 7



Relationships can also be displayed in line charts like the previous example where we saw the relationship of sales in the different periods. Clustered bar charts like the example above vividly show the relationships of two variables over time. Pie charts like the one below are ideal for comparing the relative sizes of individual parts of a whole.

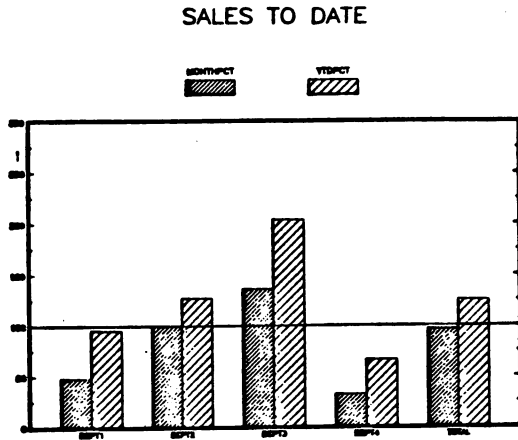
Figure 8



Graphics are also very valuable in showing exceptions and variances. This is an essential analysis for those managers who manage by objective or manage by exception. Charts see frequent applications in budgeting where forecasts are compared with actual results. The chart below shows how different departments within one division have performed on both a monthly and yearly basis. The bars depict what % of the forecast each department has achieved. At a glance, it is clear that departments two and three have reached or exceeded their goals. Department one has had a poor month, meeting only 50% of its goal but is still almost on target for the year. Department four, on the other

hand is considerably below expectations both for the month and the year to date.

Figure 9

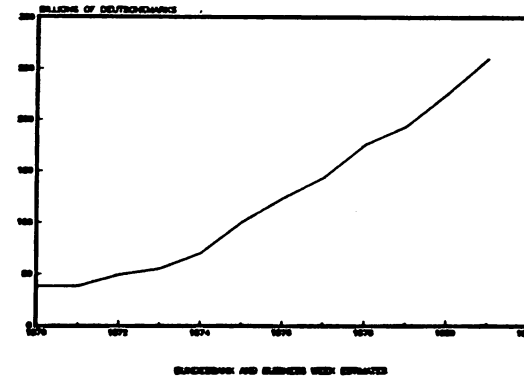


Solution Formulation: After completing analyses, the decision maker may have enough understanding of the problem to generate several alternative solutions. If so, the decision maker can choose the alternative that best fits the decision criteria or, if the problem is not fully understood the decision maker can redefine the problem, collect more data, and do more analyses.

In generating alternative solutions, a manager may choose to manipulate data in several ways. To accommodate this activity, DSG/3000 has built in transformation capabilities: cumulations to give totals up through a given time period; moving averages to smooth out irregularities in trends; logarithmic scaling to display variables which have values very far apart; and several others. Manipulations of the data can be performed within DSG/3000 by simply filling in one menu -- there is no need to exit or write special programs. Below is an example of some cumulated data:

Figure 10

GERMANY'S CUMULATIVE DEBT



Communication: Once the best solution has been determined, the answer can be communicated to those who need to act upon it. Quite often this step is overlooked in the decision-making process and yet it is probably one of the most important. How often do senior managers redefine problems, ask for more data or request additional analyses? These activities often set the whole process in motion again which explains the extremely iterative nature of the decision making process. This points out the necessity of being able to easily update and create graphs manually, as well as using them for periodic reporting.

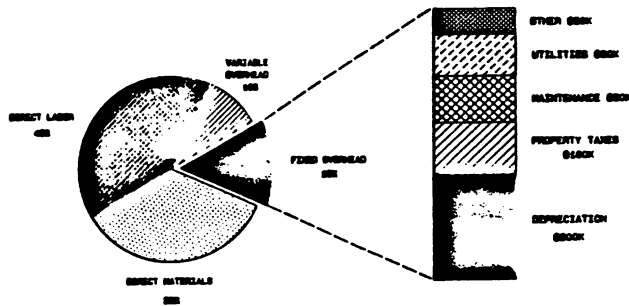
In fact, very few managers make decisions by themselves, most often they must seek approval from higher levels. This in part explains why managers spend 46% of their time in meetings and top level managers spend only 2 - 3% of their time dealing with computers.

As we discussed earlier, business graphics are very valuable in meetings and presentations. They can reduce the amount of time spent in meetings or make the time spent more productive by rapidly conveying the most relevant information in a concise and interesting format that will maintain the listeners interest as well as improving their retention of key points. DSG/3000 provides presentation quality graphs and slides appropriate for reports and meetings. In fact all the graphs I have shown you today were created with DSG/3000. One major consumer package goods company in the United States found they were able to reduce the cost of producing presentation slides by 50% using DSG/3000.

The ability to store the data and chart information separately allows users to update graphs on a periodic basis without recreating all of the chart specifications. Arrows, text, and lines as well as color selection can be employed to highlight the most important information. The ability to put multiple graphs on one page is also very valuable since it allows you to show relationships between various elements of your data. The two examples below demonstrate the amount of information content that can be placed in a graph.

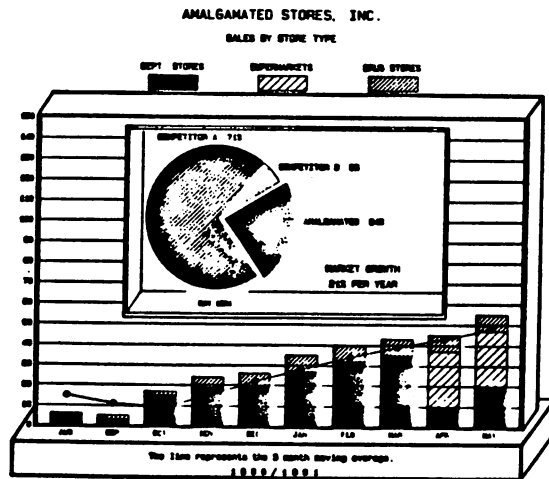
Figure 11

AIR CONDITIONER PLANT #8 DAYTON, OHIO
PRODUCTION COST ANALYSIS



SOURCE: AMALGAMATED STORES, INC. 1981 ANNUAL REPORT

Figure 12



VI. CONCLUSION

The trend toward office automation will continue at an accelerating pace as word processing and data processing systems are integrated over the next decade. In addition to productivity improvements for secretaries and clerical personnel, more sophisticated tools will be made available for managers and professionals. Some of the technologies that will come into widespread use are electronic mail, teleconferencing, personal support tools for scheduling and calculating as well as sophisticated decision support tools.

Graphics are already playing a major role in improving managerial productivity today because of their applicability in the decision making process. In the future, charts will become an integral part of business reporting because of their exceptional ability to communicate trends, relationships, comparisons and variances. Lets quickly review why this is true.

Graphics is an especially useful tool to managers because of:

Information needs --

Graphics can provide concise summaries of the most relevant information in a visually appealing format that is easily understood by others.

Daily Activities --

Graphics can assist managers in communicating to others in meetings (46% of their time) and in creating documents (13% of time). The ability to understand material when reading (8% of time) and do sophisticated analysis (8% of time) is vastly enhanced with the use of graphics.

Decision Making Process

Graphics can help all steps of decision making by focusing attention on the most important facts during problem definition. Computer graphics with data base access expedites the collection of data in a highly accurate manner. Graphics is probably most valuable in analysis because it quickly shows trends, relationships, comparisons, and variances and exceptions. Combined with some simple statistical tools, graphics can be invaluable in exploring alternative solutions and visually displaying "what if" analyses. And last, but probably most important, it can communicate information to others very effectively.

involved in making many interrelated decisions. Graphics can be a valuable resource in this iterative process.

Thus, managers with graphics at their fingertips, despite their differing management styles and job descriptions will have access to more relevant data, in a more concise fashion, in an easy to comprehend format, that will let managers spend more time doing what they do best -- making decisions.

How can Graphics Help Management?

Planning, organizing and controlling-- These key ingredients of management are each a series of interrelated decisions. Planning for example asks the questions who will do what, where, at what cost, etc.? Budgeting is an excellent example of planning in which many people may be

VII. IMPLICATIONS FOR DATA PROCESSING MANAGERS

Today senior managers are looking for better more timely information. They are looking to their management information systems personnel to improve their decision making capabilities. This means providing better information rather than just more data. It also means providing it in a more succinct, highly comprehensible format like graphics that can be easily understood by business managers.

In the future, those managers and professionals who use graphics will be able to more effectively manage their businesses. And those computer systems specialists who implement graphics on their systems will be able to provide higher quality information, not just data to their end user decision makers.

In fact, one observer pointed out that the role of DP directors in the future will change "from technician to management scientist. [3] But, it is up to you, the experts on computers and information systems, the professionals who are abreast of the latest technology to show managers throughout your organizations how they can be more productive using the tools and services you provide. They will not come to you because in most cases, they are successful now and they are not aware of many of the tools you have at your disposal. You need to be the agent of change and take the lead in improving your organizations management productivity.

VIII. REFERENCES

- [1] Fronk, Robert L.
"Driving Forces for Office Automation"
Arthur D. Little, Inc.
- [2] Friend, David
"The Promise of Information Graphics"
Computer Pictures Corporation, Boston, Mass.
- [3] Kline, Frank R.
"Management of Information Technology in the 80s"
Drexel Burnham Lambert, Inc., New York, N.Y., 6/80
- [4] "Briefs" on Dartnell Institute of Business Research Report
Management World
Administrative Management Society, Willow Grove, PA, 6/81
- [5] "Graphics in the Executive Workstation"
Computer Graphics for Management, Vol. I, No. 3
- [6] "Booz, Allen Study of Managerial/
Professional Productivity"
Booz, Allen & Hamilton, Inc., New York, N.Y., 6/80
- [7] Patterson, Marvin
"Graphics Representation of Numeric Data: A Versatile Key
to Better Decision Making"
Hewlett-Packard Conference, Detroit, Michigan, October,
1978
- [8] "Managers vs. Office Automation"
Planet News, No. 41
Infomedia Corporation, San Bruno, California, 5/81

```

***** h *****
***** h *****
***** h *****
***** hhh ppp ***** H E W L F T Y
***** n h p p *****
***** h h p p *****
***** h h p ppp ***** P A C K A R D
***** p *****
***** o *****
***** p *****

```

COMPUTER SYSTEMS DIVISION
Accounting Systems Group
19447 Pruneridge Ave.
Cupertino, Ca. 95014
(408)-725-8111 X4257

Bill Vaughan, Accounting Systems Supervisor
=====

Increased Reliability at a Lower Cost

ABSTRACT:

This paper will discuss various techniques utilized to increase the reliability of application software and to simplify the operations management of the HP3000. Topics presented will include:

- MONITOR
 - Complete system security, application control and friendly user interface in a single online program
- A GENERAL PURPOSE AUTOMATED CONTROL APPLICATION
 - How to insure that what one program writes to a file is the same as what the next program reads
- INTELLIGENT DATABASE CAPACITY CHECKS
 - How to prevent databases from hitting capacity, thereby avoiding time-consuming recovery and clean-up
- PRIVATE VOLUMES
 - How Private Volumes are meant to be used to
 - (1) better utilize disc drives
 - (2) insure data integrity and security
 - (3) do system backup
- APPLICATION TESTING
 - A sound testing strategy that pays off in the long run
- IMAGE LOGGING
 - Showing its benefits both online and batch
- MISCELLANEOUS
 - Key file recovery after catastrophic crashes
 - Building files to avoid run-time aborts
 - Unique approaches to JCL, UDC's and MPE capability maintenance
 - Utilizing "INFO=" for passing parameters to COBOL programs

Background

The Accounting Systems Group of CSY reports to the CSY Controller and handles all accounting data processing for CSY. Our role within the Accounting Department is to support and develop computerized accounting systems.

In addition to support we have become heavily involved and dedicated to:

- (1) Testing new HP products - both hardware and software. This includes not only doing pre-release testing for functionality and reliability but also utilizing these products to develop our distributed environment.
- (2) Fully utilizing HP software and hardware to implement a "distributed" data processing environment, i.e. one in which the computing power is where the people and problems are. This includes addressing the problems of system security and operatorless-computers.

We currently have our applications spread across two HP3000 systems, a SERIES 44 and a SERIES 33, with a total of about 1000 Mb of disc storage (four of our disc drives are Private Volumes). One 2619A does the printing for both machines (we use DS/3000 to copy spoolfiles from the Series 33 to the Series 44). We have one HP125 microcomputer in the department and are currently evaluating an HP3000 to be put in Accounts Payable for dedicated processing. Our systems group of 12 professionals supports an accounting department of 40 people.

A GENERAL PURPOSE AUTOMATED CONTROL APPLICATION

One of the tasks a systems administrator has to do is to verify control totals for batch jobs. Most of our batch jobs consist of multiple programs passing information through the use of sequential disc files and for certain jobs it is not enough to successfully reach EOJ to say that processing was indeed successful. We would have the programs within the job print out control reports to STDLIST and the system administrator would manually verify that the control totals matched.

We have designed and implemented an automated control logging procedure that (1) is standard for all application subsystems, (2) eliminates manual calculations thereby eliminating human error, (3) stores and reports the information being logged, (4) directs the systems administrator's attention to variances in the control totals, and (5) is simple and straight forward to implement.

This system, called Control Logs, is driven off a database that maintains the totals and the parameters that define how the system should handle the totals. All of the code to be inserted into each program is kept in our copplib.

The following example will show how Control Logs works. Let program "A" write out data to a file named "D". Then program "B" will use file D as an input file to do further processing. In program A, each time a record is written to D two totals are kept (in accounting we usually keep record count and a dollar total) and at the end of program A the control log subroutine is called. This takes the totals

puts them into the database, and then prints a standardized control report to STDLIST. Then program B comes along and as it reads file D it keeps the same two totals that A kept. At the end of its processing it also calls the control logs subroutine but it will compare its totals to those in the database for file D. If they do not match an error report is printed showing the differences and the subroutine aborts the program.

There are several functions that can be accomplished with Control Logs:

- (1) Replace - this is used by the program creating a file as was program A in the example above. The totals taken are put into the database and no further processing is done
- (2) Compare - this takes the totals being passed by the program and compares them to the totals presently in the database for that particular file. If they do not match the program is aborted and an error message is printed
- (3) Update - this takes the totals being passed by the program and adds them to the totals already in the database and the resulting new totals are then put in the database.
- (4) Compare and Update - this does the compare function first with the first two totals passed and then does an Update using two optional totals that are used for this function and function (5)

(5) Compare and Replace - this does the Compare function first with the first two totals passed and then does an Update using the second two totals.

We now currently have all of our batch processing using Control logs for every file that is passed between two or more programs. Generally, it has not been inconsistent data that has led to Control Logs mismatches, rather it has been that when we fix bugs in programs we have inadvertently introduced other bugs that affect these files.

INTELLIGENT DATABASE/FILE CAPACITY CHECKS

Recovering a database after a dataset has hit capacity is one of the more painful recovery processes. Using IMAGE LOGGING helps because you can recover right up to the process that filled the database but you still have to retrieve the database from your last backup and run the log file back in. It is also difficult to always keep right on top of the amount of free space each dataset within each database has. That 30% or so free space you like to maintain can disappear alarmingly quickly and when it does you are faced with what we refer to as a "capacity abort".

Along with databases, data files can be used in a similar manner where free space is maintained and information is "appended" to the file on a regular basis using "ACC=APPEND" on file equations.

What we have developed is a simple method of "up-front" capacity checking within programs so that processing can be terminated before any data is out to the database or file in the case where there will not be enough space available.

We do almost all of our programming in COBOL, and with COBOL II MPE INTRINSICS can be called directly so that none of what we are going to do requires any fancy subroutines or coding. By using "GETINFO" MODE202 calls for datasets and "FGETINFO" intrinsic calls for files, all current-count and capacity information can be obtained, from there it is only a matter of determining if what you have to put in will fit.

PRIVATE VOLUMES

In moving towards an operatorless environment Private Volumes have played a key role in several ways.

We now use private volumes for almost all of the processing that used to go to tape. An HP7925 disc pack can hold about 3 1600bpi tapes worth of information. Using the MPE command "UMOUNT ON,AUTO" with a private no one has to to "REPLY" when the pack is needed. All the advantages of disc access are available as well as the ease of removability and storage that tapes have.

We now use private volumes for all of our partial dumps (system backup). We use the volumes as "SERIAL DISCS" and so we backup to disc the same way we used to backup to tape.

We have found that using private volumes for backup is faster because of the lack of multiple tape mounts and that disc packs do not suffer from the parity errors that are frequent on aging tapes. In day-to-day use we have found that major system crashes that call for reloads seldom destroy the files on our private volumes so that only system-domain drives need to be reloaded. This has cut our reload time to less than half what it was before.

We have created a private volume called "SPACE" that has one group on it also called SPACE. This pack is mounted whenever we need to do extremely large sorts. What we do is to "point" our sort files to SPACE and do all of the sorting on this completely empty HP7925 pack (that's nearly 500,000 sectors of sorting space!).

APPLICATION TESTING

Being a systems group that does application maintenance as well as development we do alot of testing on existing systems as bugs are fixed and programs are enhanced. In the past we always tested with a subset of our "live" data in a group made just for testing and there were no rules or even guidelines on testing. What happened was that:

- (1) Test data was easily destroyed as one person would purge files or alter data that another had set up. This usually didn't happen while both were testing. The first was done but when he came back a month later to use the data to test a new change in the program he found his prior data (which he knew and understood) gone or altered. This created a situation where test data integrity was nonexistent and much time and effort were wasted always having to recreate data.
- (2) Because the test data wasn't actually designed it very seldom really "tested" the programs it flowed through. To be meaningful each different type of transaction must be included in the data and especially ones that fully exercise the area of the program that has been changed. Production data also generally contains a high volume of only a few types of these transactions so that when data is just being copied from the live data files it usually wasn't of much worth except to see if the program could run from beginning to end without aborting!

(3) Because the test data wasn't actually designed it was very seldom that the programmer REALLY understood the interrelationships within the data and the program. Very often the programmer was simply putting in code that he was told to put in and wasn't ever understanding the problem to be solved.

This list could go on and on. Our solution to the problem is what we call "TESTBASE".

TESTBASE is a group within our FINANCE account that is designed to be a complete, self-supporting environment for the testing of our Accounting software. By "complete" it is meant that using the procedures we've outlined, TESTBASE can be enhanced to fully test any "live" scenario desired. By "self-supporting" it is meant that closure exists within TESTBASE'S set of data.

Listed below is the set of guidelines that we have set up for TESTBASE. The way TESTBASE works in practice is that we keep the group TESTBASE as a "clean" copy and for each person we have a group for them to do their testing in. When we create this group we give them a copy of all the necessary files that they will need from TESTBASE and we make sure that when testing is finished that they go back and add to TESTBASE the new test data that they have developed. We have found that for TESTBASE to work requires a serious commitment and effort from management but that the testing time saved, the increased thoroughness and quality of testing, and the knowledge gained by the programmers makes TESTBASE one of the best investments we've made.

TESTBASE guidelines:

- (1) TESTBASE should have complete closure. Any data needed for jobs, validation, etc. will be kept within TESTBASE. If needed, TESTBASE could be removed to another machine along with necessary program files and all testing could be accomplished.
- (2) Naming conventions should be independent from actual naming conventions. Most production naming conventions (eg. Part-Numbers) are not the result of predefined naming schemes designed to minimize start-up costs and overhead involved in user understanding. TESTBASE names will try to be as simple and orderly as is possible.
- (3) All test data should be independent of actual production data values. In this way data can be designed to provide specific information to the testor.
- (4) TESTBASE should be recoverable. This means that at any time the programmer may retreat back to time 0 and begin again with exactly the same scenario or environment he started with. Also, with not being tied to the production environment testbase or any part of it means TESTBASE can be stored at any point in time during testing and then at any time be recovered to that point for restart.
- (5) TESTBASE should be dynamic in that whenever it is used, time should be taken to enhance the original testbase so that (a) it does not become outdated and (b) so that testbase grows with new databases and files being added to increase the range of systems that can be tested in the future without having to "reinvent the wheel" with each new user.

IMAGE LOGGING

In our environment we are using Image Logging for virtually all of our databases. All of our logging is done to disc and we've realized some unexpected benefits from having the logging processes.

First, we've seen no problems with logging to disc. When we build our log files we obtain the disc address of the files so that in a serious crash we can pull the files off to tape using SADUTIL (see MISCELLANEOUS section on Key file recovery).

Second, one of the biggest benefits from logging comes not from when the system crashes but when an application aborts and we need to recover the databases involved. It is nice being able to recover a database right up to the beginning of processing of an application and not lose previous processing to that database.

Third, the incremental processing time involved with logging is unnoticeable and implementation of logging requires no program changes. A common practice we've seen is to put DBSTORE'S at the beginning of job streams for recoverability. That definitely adds processing time!

MISCELLANEOUS TOPICS

Key file recovery after catastrophic crashes

There used to be a time when catastrophic system crashes meant a total reload of the system from the last backup. For Accounting this meant losing all the processing that had occurred from backup to the crash. However, with a little planning and a system utility called SADUTIL (see MPE Systems Utilities reference manual - Part No. 30000-90044) files on the inoperable system can be copied to tape and thereby recovered.

Planning needs to be done because if the system directory is destroyed in the crash, the only way to get the file is to know the logical device it resides on and its starting disc address. This information can be obtained from the MPE STORE command using the SHOW parameter or can be obtained by doing a LISTF within LISTDIR2.PUB.SYS.

In our environment we use both, depending on the specific file. For our Image Logging files we put LISTDIR2 in the job stream that builds the log files. This way the address is printed right on the STDLIST and filed with the backup listings. For the few strategic files that we need to keep track of their whereabouts on the system, we've put LISTDIR2 into the job streams that create the files. If the system crashes, we get the address of the file from the last STDLIST for that job and use SADUTIL to get our file back!

Building files to avoid run-time aborts

When building files within job streams the "DISC=" Parameter of the BUILD command can be used to allocate the entire amount of disc space needed. This is accomplished by setting the initial allocation equal to the number of extents (remember that DISC=[numrec][,numextents][,initialloc]) so that DISC=10000,32,32 would allocate the entire space for 10,000 records or fail due to lack of disc space. This way lack of disc space will abort the job stream outside of, and before the program that would've used the file.

How not to lose JCL

In the past we had alot of problems with the STDLIST that is printed for every batch job. First, it seems as though STDLIST's page eject the line printer about every other line. This always destroyed any attempt to keep paper piling properly and jammed the printer on a regular basis. Secondly, user's don't understand the importance of STDLIST's so they got thrown out, filed with reports, inadvertantly left attached to somebody's output (and therefore thrown out),etc. It always seemed to be the case that the STDLIST was missing for that critical job that aborted and trying to fix the job became a difficult task.

We have implemented a simple solution that has solved our STDLIST problems. Any job that has a STDLIST worth saving has had "OUTCLASS=LP,5" added to its job card. This defers the STDLIST so it doesn't print (our OUTFENCE is normally 7) until a systems person

prints it off. We print off the STDLIST's once a day and file the them by the day we print them off (this avoids having to separate them). Now our printer jams alot less often and we always know exactly where the STDLIST's are!

MPE IV'S "INFO=" PARAMETER

MPE IV has a new parameter, called "INFO", for the RUN command. Using INFO alphanumeric information can be passed to application programs. Before INFO the only run-time parameter for passing information was "PARM", and it handled only numeric information.

We have written an SPL subroutine that retrieves the alphanumeric string and the length of this string to a COBOL II program. A simple COBOL example and the subroutine listing follow:

WORKING-STORAGE SECTION.

```
01 INFO          PIC X(80).
01 INFO-LENGTH  PIC S9(04) COMP.
```

CALL "GETINFO" USING INFO, INFO-LENGTH.

From MPE:

```
;RUN PROGRAM;INFO="HELLO-THERE"
```

```
00000 0 $CONTROL SUBPROGRAM,MAP,ADR,SEGMENT=GETINFO
00000 0 $TITLE "FMS06155 - GETINFO"
00000 0
00000 0 << THIS PROGRAM WILL RETURN AN 80 CHARACTER STRING >>
00000 0 << AND THE LENGTH OF THIS STRING >>
00000 0 << TO A COBOL PROGRAM THAT WAS RUN WITH THE "INFO=" >>
00000 0 << OPTION. THE ADDRESS OF THE STRING IS STORED IN >>
00000 0 << Q-5 AND THE LENGTH IS STORED IN Q-6 AT RUN TIME. >>
00000 0
00000 0 BEGIN
00000 1
00000 1 PROCEDURE GETINFO(INFO.LEN);
00000 1 ARRAY INFO;
00000 1 INTEGER LEN;
00000 1
00000 1 BEGIN
00000 2 LOGICAL QSTART=0;
00000 2 Q +000
00000 2 INTEGER DELTA,X;
00000 2 Q +001
00000 2 Q +002
00000 2 BYTE POINTER PINFO;
00000 2 Q +003
00000 2 POINTER P'LEN,PQ,PREG;
00000 2 Q +004
00000 2 Q +005
00000 2 Q +006
00000 2 POINTER W'PINFO;
00000 2 Q +007
00000 2 LOGICAL VAR;
00000 2 Q +010
00000 2
00000 2 X := @QSTART; << SET POINTER PQ TO CURRENT >>
00000 2 @PQ := X; << VALUE OF Q-REGISTER >>
00000 2
00000 2 AGAIN:
00000 2 DELTA := PQ; << FIND OUT HOW MUCH TO CHANGE Q >>
00000 2 @PREG := @PQ(-2); << SET POINTER TO VALUE OF >>
00000 2 << P-REGISTER IN STACK MARKER >>
00000 2
00000 2 @PINFO := @PQ - 5; << SET POINTER TO WHERE >>
00000 2 << RUN INFO ADDRESS WILL BE >>
00000 2
00000 2 @P'LEN := @PQ - 6; << RETURN LENGTH OF >>
00000 2 <<STRING FROM Q(I)-6 >>
00000 2
00000 2 LEN := P'LEN;
00000 2
00000 2 IF PREG = 0 << END OF STACK CHAIN. >>
00000 2 << CAN STOP NOW >>
00000 2 THEN GOTO STOP'THIS;
```

```

00025 2
00025 2 X := @PQ;
00027 2 @PQ := X - DELTA;          << DECREMENT Q-PTR >>
                                     << TO PREVIOUS   >>
00032 2                                << VALUE AND START AGAIN >>
00032 2 GOTO AGAIN;
00033 2
00033 2
00033 2 STOP'THIS:
00033 2 @W'PINFO := @PINFO;        << SET UP THE INFO >>
                                     << POINTER   >>
00035 2 VAR := W'PINFO;
00037 2 @W'PINFO := VAR;
00041 2 @W'PINFO := @W'PINFO / 2;
00044 2 MOVE INFO := "
";
00100 2 MOVE INFO := W'PINFO,(LEN); <<RETURN INFO >>
                                     << STRING >>
00104 2
00104 2 END;

```

IDENTIFIER	CLASS	TYPE	ADDRESS
AGAIN	LABEL		PB+005
DELTA	SIMP. VAR.	INTEGER	Q +001
INFO	ARRAY (R)	LOGICAL	Q -005
LEN	SIMP. VAR.(R)	INTEGER	Q -004
P'LEN	POINTER	LOGICAL	Q +004
PINFO	POINTER	BYTE	Q +003
PQ	POINTER	LOGICAL	Q +005
PREG	POINTER	LOGICAL	Q +006
QSTART	SIMP. VAR.	LOGICAL	Q +000
STOP'THIS	LABEL		PB+033
VAR	SIMP. VAR.	LOGICAL	Q +010
W'PINFO	POINTER	LOGICAL	Q +007
X	SIMP. VAR.	INTEGER	Q +002

00000 1 END.

IDENTIFIER	CLASS	TYPE	ADDRESS
GETINFO	PROCEDURE		

```

PRIMARY DB STORAGE=X000; SECONDARY DB STORAGE=X00000
NO. ERRORS=0000; NO. WARNINGS=0000
PROCESSOR TIME=0:00:01; ELAPSED TIME=0:00:07

```