WORKSHOP ON SOFTWARE IN HIGH-ENERGY PHYSICS
(Where do we go from here?)

CERN, Geneva, Switzerland
4–6 October 1982
Propriété littéraire et scientifique réservée pour tous les pays du monde. Ce document ne peut être reproduit ou traduit en tout ou en partie sans l'autorisation écrite du Directeur général du CERN, titulaire du droit d'auteur. Dans les cas appropriés, et s'il s'agit d'utiliser le document à des fins non commerciales, cette autorisation sera volontiers accordée.
Le CERN ne revendique pas la propriété des inventions brevetables et dessins ou modèles susceptibles de dépôt qui pourraient être décrits dans le présent document; ceux-ci peuvent être librement utilisés par les instituts de recherche, les industriels et autres intéressés. Cependant, le CERN se réserve le droit de s'opposer à toute revendication qu'un usager pourrait faire de la propriété scientifique ou industrielle de toute invention et tout dessin ou modèle décrits dans le présent document.

Copyright CERN, Genève, 1982

Literary and scientific copyrights reserved in all countries of the world. This report, or any part of it, may not be reprinted or translated without written permission of the copyright holder, the Director-General of CERN. However, permission will be freely granted for appropriate non-commercial use. If any patentable invention or registrable design is described in the report, CERN makes no claim to property rights in it but offers it for the free use of research institutions, manufacturers and others. CERN, however, may oppose any attempt by a user to claim any proprietary or patent rights in such inventions or designs as may be described in the present document.
ABSTRACT

A Workshop on 'Software in High-Energy Physics', attended by some 130 people, took place between 4-6 October 1982 at CERN, Geneva, under the joint sponsorship of CERN (European Organization for Nuclear Research) and ECFA (European Committee for Future Accelerators). Sixteen invited speakers from computer science, from software- and hardware-producing industry, and from other various fields of science, presented their views on the question chosen as the Workshop's subtitle: 'Where do we go from here?'.

These proceedings contain most of the invited papers and the discussion contributions, as well as the transcript of a closing panel discussion.
INTRODUCTION

This Workshop is an attempt to solve a problem of communication arising from history.

High-energy physicists were amongst the very early users of large computer systems. Together with the first representatives of what later became computer science, they discovered in the 1960's the shortcomings of the then available systems and techniques. They developed or proposed solutions which the market did not offer, and created for themselves a computer environment in which the analysis programs they needed could be developed, tested, and exchanged as required. This environment is found, in a similar form, in all computer centres used for high-energy physics; it is dominated by large amounts of data and non-trivial evolutive analysis programs, all shipped around between different computer centres, and the solution adopted is that of FORTRAN programs and extensions by subroutine packages and libraries.

The need for frequent adaptation to changing requirements, the necessity for true portability, and the fact that the task of writing and maintaining these programs is carried out mostly by physicists who are without either training or specific interest in computer science, make this environment remarkably resistant to changes.

Today, of course, computer science has established itself, a market of commercially available software products is building up, the classical interface between user and computer is being dramatically reviewed, and high-energy physicists have become a comparatively small group of customers amongst many others. All this has made it seem a necessary exercise to assess whether or not the existing approach to software is about to fall behind the state of the art, and to look for areas in which experience, help, or guidance could be brought to high-energy physics from other fields. Some questions for which we hoped to find the answers are the following:

- How much of our software can be properly engineered rather than put together in an artistic approach? and by which methods? Can software quality be measured?
- Which methods allow us to optimize the necessary breaking of large problems into modular parts?
- How closely is the writing of software linked to the computing environment? Is it beneficial to follow the permanent change with the ensuing agonizingly frequent re-appraisal? How easy is it to narrow down the multitude of choices to a reasonable number of solutions?
- Do there exist computer users with similar problems, whose solutions could serve as a model for high-energy physics? Does the commercial market offer some solutions?
In consequence, a number of prominent and experienced speakers were invited to present to physicists, most of them from European high-energy physics laboratories, their personal views on these questions: views of computer scientists, views of other computer-intensive branches of science, presentations of future hardware with impact on software writing.

Here is the place to thank all invited speakers for their contributions, and for the remarkable understanding they showed towards the problems with which they were confronted. Thanks are also due to all participants who contributed to the lively discussions.

R.K. Bock, for the Organizing Committee
P. Zanella, for the CERN Management

* * *

THE PROGRAMME AND ORGANIZING COMMITTEE

<table>
<thead>
<tr>
<th>J. Allaby, CERN</th>
<th>V. Blobel, DESY</th>
</tr>
</thead>
<tbody>
<tr>
<td>R. Bock, CERN</td>
<td>R. Dohinson, CERN</td>
</tr>
<tr>
<td>R. Fruhwirth, Vienna</td>
<td>I. Hughes, Glasgow</td>
</tr>
<tr>
<td>Ch. Jones, CERN</td>
<td>E. Lillestol, Bergen</td>
</tr>
<tr>
<td>L. Pape, CERN</td>
<td>Ch. Petit-Jean-Genaz - Redman, CERN</td>
</tr>
<tr>
<td>I. Willers, CERN</td>
<td></td>
</tr>
</tbody>
</table>
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTRODUCTION</strong></td>
<td>v</td>
</tr>
<tr>
<td><strong>PROGRAMMING</strong></td>
<td></td>
</tr>
<tr>
<td>M.A. Jackson</td>
<td>1</td>
</tr>
<tr>
<td><strong>PHYSICS vs. COMPUTER SCIENCE</strong></td>
<td></td>
</tr>
<tr>
<td>R. Pike</td>
<td>13</td>
</tr>
<tr>
<td><strong>FORTRAN IS ALIVE AND WELL!</strong></td>
<td></td>
</tr>
<tr>
<td>M. Metcalf</td>
<td>19</td>
</tr>
<tr>
<td><strong>ADA AND ITS IMPACT ON THE SCIENTIFIC USER</strong></td>
<td></td>
</tr>
<tr>
<td>R. Marty</td>
<td>26</td>
</tr>
<tr>
<td><strong>SOME SOFTWARE TOOLS FOR SCIENTIFIC PROGRAMMING</strong></td>
<td></td>
</tr>
<tr>
<td>R.P. Worden</td>
<td>41</td>
</tr>
<tr>
<td><strong>DATABASE APPLICATIONS IN HIGH ENERGY PHYSICS</strong></td>
<td></td>
</tr>
<tr>
<td>K.G. Jeffery</td>
<td>55</td>
</tr>
<tr>
<td><strong>STORAGE SYSTEMS AND TECHNOLOGY: STATUS AND FUTURE DIRECTIONS</strong></td>
<td></td>
</tr>
<tr>
<td>M.D. Canon</td>
<td>66</td>
</tr>
<tr>
<td><strong>SCIENTIFIC COMPUTING IN METEOROLOGY</strong></td>
<td></td>
</tr>
<tr>
<td>D.M. Burridge</td>
<td>67</td>
</tr>
<tr>
<td><strong>SOFTWARE IN THE OIL INDUSTRY</strong></td>
<td></td>
</tr>
<tr>
<td>M.C. Turnill</td>
<td>68</td>
</tr>
<tr>
<td><strong>SOFTWARE PROBLEMS IN MAGNETIC FUSION RESEARCH</strong></td>
<td></td>
</tr>
<tr>
<td>R. Gruber</td>
<td>88</td>
</tr>
<tr>
<td><strong>DATA PROCESSING AT THE EUROPEAN SPACE AGENCY</strong></td>
<td></td>
</tr>
<tr>
<td>C. Mazza</td>
<td>101</td>
</tr>
<tr>
<td><strong>HIGHLIGHTS OF A NEW SYSTEM FOR DATA ANALYSIS</strong></td>
<td></td>
</tr>
<tr>
<td>P. Svensson</td>
<td>119</td>
</tr>
<tr>
<td><strong>THE GRAPHICS FUTURE IN SCIENTIFIC APPLICATIONS</strong></td>
<td></td>
</tr>
<tr>
<td>- Trends and developments in computer graphics -</td>
<td></td>
</tr>
<tr>
<td>G. Ederle</td>
<td>147</td>
</tr>
<tr>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>OLYMP - EVALUATION OF MULTI-DIMENSIONAL DATA ON VAX 11/780</td>
<td>164</td>
</tr>
<tr>
<td>R. Egloff</td>
<td></td>
</tr>
<tr>
<td>COMPUTER AIDED DESIGN AROUND THE A300 AIRBUS</td>
<td>180</td>
</tr>
<tr>
<td>J.J. Audy</td>
<td></td>
</tr>
<tr>
<td>PERSONAL WORKSTATIONS</td>
<td>194</td>
</tr>
<tr>
<td>I.M. Willers</td>
<td></td>
</tr>
<tr>
<td>HMINET 2 - A LOCAL X.25 NETWORK CONNECTED TO THE GERMAN PUBLIC DATA</td>
<td>201</td>
</tr>
<tr>
<td>NETWORK</td>
<td></td>
</tr>
<tr>
<td>B. Butsoher</td>
<td></td>
</tr>
<tr>
<td>LOCAL AREA NETWORKS</td>
<td>215</td>
</tr>
<tr>
<td>P. Zafiropoulos</td>
<td></td>
</tr>
<tr>
<td>THE PROVISION OF WIDE-AREA NETWORKING FACILITIES</td>
<td>240</td>
</tr>
<tr>
<td>B.J. Charles</td>
<td></td>
</tr>
<tr>
<td>THE DESY COMPUTER CENTRE: OBJECTIVES TODAY AND TOMORROW</td>
<td>248</td>
</tr>
<tr>
<td>P.E. Kuhlmann</td>
<td></td>
</tr>
<tr>
<td>PANEL DISCUSSION</td>
<td>250</td>
</tr>
<tr>
<td>Ch. Jones and I. Willers</td>
<td></td>
</tr>
<tr>
<td>LIST OF PARTICIPANTS</td>
<td>256</td>
</tr>
</tbody>
</table>
PROGRAMMING

M A Jackson

Michael Jackson Systems Limited
101 Hamilton Terrace
London N W 8 England

ABSTRACT

The programmer's task is often taken to be the construction of algorithms, expressed in hierarchical structures of procedures; this view underlies the majority of traditional programming languages, such as Fortran. A different view is appropriate to a wide class of problem, perhaps including some problems in High Energy Physics. The programmer's task is regarded as having three main stages: first, an explicit model is constructed of the reality with which the program is concerned; second, this model is elaborated to produce the required program outputs; third, the resulting program is transformed to run efficiently in the execution environment. The first two stages deal in network structures of sequential processes; only the third is concerned with procedure hierarchies.

1. INTRODUCTION

Conventionally, the written version of a conference paper should omit those introductory remarks of a personal nature - such as compliments and thanks to the Program and Organising Committee - that are appropriate to the oral presentation. But not in this case. In addition to thanking the Committee for inviting me to this Workshop, I must firmly disclaim any knowledge of High Energy Physics and of the programming of problems in that discipline. My unease at this ignorance is somewhat mitigated at finding myself placed so early in the programme of the workshop; clearly, the Committee are anxious to ensure that my paper should be presented in this state of virgin ignorance, before I can have the advantage of hearing other papers and discussions that might impart some fragments of knowledge. It is my hope, and presumably theirs also, that ideas derived from other programming contexts may prove to have some relevance to the problems in High Energy Physics which are the subject of our meeting. With that hope, I will proceed.

2. THE DIFFICULTY OF PROGRAMMING

Programming is not an easy task. This became clear quite early in the history of computing, and has become even clearer as programs become more ambitious in their specifications and in the applications that they serve. Embedded systems and process control systems must be very reliable; computer operating systems are inevitably very complex, and must be efficient; compilers for modern programming languages such as Ada are also complex, and must also be efficient; data processing systems are extremely complex - although their complexity sometimes passes unnoticed because they tend to be loosely
textured - and must be capable of constant change as the needs of their users change. All this is well known.

The problem of programming is exacerbated by the material of which programs are built. A bare computer provides a repertoire of simple operations on small data objects: operations to add integers or floating-point numbers; operations to read or write records from or to the external devices, such as disks, tape drives, and line printers; operations to transfer execution control to specified addresses in the stored program. Some means is evidently needed for introducing larger structures into the programmer's mental view of the program. A large program may contain 10,000, or 100,000, or even 1,000,000 machine operations; some organising principle is necessary to deal with a whole composed of so many parts.

The first such principle was established early in the history of programming. Wilkes \(^{(14)}\) writes:

"From the very first, I had seen the establishment of a library of subroutines as being of prime importance. Not only would the availability of such a library to draw on save the programmer effort, but library subroutines could ... enable the programmer to work at a level above that of a raw binary computer. The importance of a library of tested subroutines took on a new aspect once practical experience had begun to show how difficult it was to write correct programs. Finally, there was the invention by David J Wheeler of the closed subroutine, which made possible the development of a coherent system of programming based on nested subroutines."

These two ideas dominated discussion of programming methods for many years, and are still dominant in some circles. The subroutine, or procedure, provides a means of raising the level of the elementary objects that the programmer must deal with. Instead of dealing only with scalar addition and subtraction, he can deal also with multiplication and division (which were often provided by subroutine in early machines), and with exponentiation; he can also deal directly with operations on larger data objects, such as vectors and matrices. Further, one subroutine may contain calls of other subroutines, allowing the whole program to be structured as a hierarchy of subroutines: the top level of the hierarchy may express the whole problem solution in a very small compass, no matter how large the entire program may be.

The other major structuring principle to emerge early was concerned with control flow. The bare machine provided jump instructions, by which control could be passed from any point in the program to any other. It soon became clear that some discipline was needed here also, and broad agreement was reached that control flow constructs could fruitfully be limited to three: sequence, or concatenation; iteration, or repetition; and selection, or choice among alternatives.

Fortran, COBOL, and Algol 60 could be said to have exploited these two principles of structuring: nested subroutines and disciplined control flow.
The exploitation, at least in the cases of Fortran and COBOL, was grossly imperfect: both languages have serious anomalies in their control flow constructs, and both have serious limitations on the power of their subroutine calls. But the trend was clear, and all three languages represented a vital step forward in simplifying the programming task.

3. PROGRAM DESIGN WITH SUBROUTINES

In the middle 1960s it was widely agreed that programs should be constructed as hierarchies of subroutines. A design question immediately arose: how should the programmer choose and develop the appropriate hierarchy for each particular problem? How could the subroutine hierarchy be used as a tool in proceeding from the statement of the problem to the finished program? It seemed that the programmer might proceed in either of two directions, or possibly in some combination of them: either from the top of the hierarchy downwards, or from the bottom up.

The top-down approach, in the academically more respectable form known as stepwise refinement, was convincingly advocated in Dijkstra's influential paper Notes on Structured Programming\(^5\). The essential idea is that the programmer begins by stating the program in an extremely simple form that uses instructions that are not available in the existing repertoire: this is the top level of the hierarchy. Each of these instructions must then be refined into instructions at the next level, and the development proceeds in this way until the lowest level is reached, where the instructions used are all in the available repertoire. Dijkstra would not, I think, advocate this approach today, but it is widely advocated and used, especially among some groups of people concerned with data processing.

The bottom-up approach has, generally, received less attention. Here, the essential idea is to raise the level of the available instruction repertoire, starting with the level of the programming language to be used. As each level of subroutine is defined, it can be used in the next higher level, until eventually the complete program can be stated in terms of a few high-level instructions. Although this bottom-up approach, in the primitive form described here, has not found much favour, it is clearly discernible in much of the recent and current work on abstract data types\(^6\).

Neither top-down nor bottom-up design is easy to practise. Both may be criticised for demanding too much foresight from the practitioner. Indeed, it is not unfair to say that both are applicable only to problems whose solutions are already known: the programmer first achieves, by hidden intuition, an outline conception of the whole solution; a top-down or bottom-up approach can then be used to describe this solution in detail. The truth of this criticism can be seen by considering the case of the top-down programmer confronted by several alternative top levels or first refinement steps. How is the choice to be made among the alternatives? Only by exercising a degree of foresight, by recognising that this alternative will lead to greater difficulty
than that alternative, at lower levels of the development.

A deeper criticism of both approaches is that they assume a hierarchical structure. The critic who castigates top-down will inevitably be confronted with the reply 'so you prefer bottom-up?': the assumption is that structure is hierarchical, and that the only plausible ways of developing structure are to explore the hierarchy in one of the two obvious orders. The justification for this assumption is partly historical: if subroutines are the sole medium of structure, then structures must necessarily be hierarchical. But it is also related to the Von Neumann machine architecture. A single, bare, Von Neumann processor can execute a program only if it is constructed as a single sequential process, at least as seen by the machine. A hierarchy of closed subroutines naturally forms a single sequential process from the machine's point of view. If the structure of the program as executed must be the same as the structure of the program as designed, it follows that the designer should naturally think of the program as a hierarchy. We will return later in this paper to the relationship between the designed and executed structures.

4. THE PROGRAM AND THE REALITY

Any useful program is concerned to compute about some external reality. A payroll system is concerned with the organisation's employees and the work they do; a program to control a chemical plant is concerned with the vessels and pipes and valves of the plant, and their behaviour. A compiler is concerned with the source language program and the object program that is to be constructed.

In every case, there are good reasons why the structure of the program, in some wide sense, should correspond closely to the structure of the reality with which it is concerned. One reason is that there will then be only one structure to be considered by the programmer: the structure of the problem and the structure of the solution are identical. Another reason is well known in data processing: the need for program maintenance – that is, for changing the program to reflect changed problem requirements. A common source of user dissatisfaction in data processing is the difficulty of program maintenance. The user requests a change that in terms of the problem seems to be small, simple, and local; the programmer examines the program, and finds that in program terms the change is large, complex, and diffuse. This is prima facie evidence that the structure of the program is radically different from the structure of the problem. The resulting high cost of difficulty of adapting data processing systems and program to the constantly changing needs of the user organisation is the theme of countless complaints.

I believe that program development, and system development also, must begin with an explicit stage in which the developer constructs a formal model of the reality with which the program is concerned. Only in this way can the program be given an appropriate structure. It is important to recognise that any program or system will inevitably embody a model of reality: the only
question is whether the model is to be explicitly defined or not. The point may be illustrated by a very simple program. This program was shown at a conference on structured programming as an example of approved top-down design; but in fact it shows clearly the inadequacies of that approach.

The problem is to merge two arrays, A and B. Each array element contains an integer key and an alphanumeric value, arranged in ascending index order by integer key. Since the number of elements in each array may vary, there is a special terminating element in each array, having a distinct high key. An example of a pair of arrays is:

A: 1a 2r 2f 4q 5t XX -- -- -- --
B: 1w 1f 2k 3t 3c 5y 6z XX -- -- --

XX is the special terminating element. As shown in the example, there may be duplicate occurrences of a key between the two arrays and also within one array. The rules for merging are that the result array, C, must contain no duplicates: where duplicates exist, the C element must be taken from A in preference to B, and an element with a lower index in preference to one with a higher. The program presented was essentially as follows, at the second or third step in development:

P: initialise indices;
    do while (more to come)
        do while (next C element should be from A)
            if (not duplicate in C)
                transfer A element;
            endif
            increase A index;
        enddo
        do while (next C element should be from B)
            if (not duplicate in C)
                transfer B element;
            endif
            increase B index;
        enddo
    enddo
    transfer XX marker element to C;
end P

With suitable further refinement, it can be shown that this program will produce the required array C:

C: 1a 2r 3t 4q 5t 6z XX -- -- --

However, our concern is with the structure of the program: the fact that it works is not evidence of correct structure. The gross structure of the program is:
P: initialise;
do while (..)
   AGROUP: do while (..)
      AELEMENT;
   enddo
   BGROUP: do while (..)
      BELEMENT;
   enddo
endo
terminate
end P

AGROUP and BGROUP respectively transfer a group of elements from A and B to C. We may ask ourselves what defines an AGROUP or a BGROUP? For the example arrays given, we can see that the groups are these:

1st AGROUP: 1a            1st BGROUP: 1w 1f
2nd AGROUP: 2r 2f          2nd BGROUP: 2k 3t 3c
3rd AGROUP: 4q 5t          3rd BGROUP: 5y 6z

We may say, therefore, that the program imposes a structure on the arrays which groups their elements as shown above. As soon as this is made explicit, it is clear that the structure is wrong: no conceivable view of the problem would be based on these groupings. The penalty paid is that the program would be very difficult to change if, for example, it were required to diagnose the occurrence of duplicate elements, either within one array or between the two arrays. We may also observe that if the elements were placed in the high, rather than in the low, index positions of the three arrays, the grouping of elements would be different from that shown above. Essentially, the structure imposed on the data is an implicit by-product of the algorithm chosen. I am advocating that it should be an explicit product of the first development stage.

5. ONE REALITY AND ANOTHER

In the array example, we have taken the reality to be the data on which the program operates directly. The input to the program is a pair of arrays, and the output is another array. The nature of the problem invites a treatment of the arrays as sequential data objects, each accessed in ascending order of index.

Sequential data objects are found in a wide variety of problems. More properly, we may say that objects are found which can usefully be treated as sequential. Obvious examples include a source program input to a compiler, a stream of input messages at a keyboard terminal, a tape file, and a sequentially accessed disk file. From the programmer's point of view, some of these are inescapably sequential, such as a stream of terminal input messages; others can conveniently be treated as sequential, even if there is no
inescapable sequential constraint on their accessing.

Sequentiality is a central aspect of most of those problems whose solutions merit the name 'system' rather than 'program'. In particular, most systems for data processing, for process controls and switching, and for embedded applications, are concerned with a reality in which time sequencing is a central feature. For a payroll system, the employee joins the company before starting work, goes on holiday before returning to work from holiday, clocks on at the beginning of each day and subsequently clocks off at the end, and so on. For a sales system, the customer places the order, then it is allocated, then it is delivered, then it is billed, and so on. Throughout, we are concerned with time-sequenced actions.

One might draw a distinction between systems and programs based on the nature of the reality with which they are concerned. One might say that a system is concerned with the 'real world' of its users, while a program is concerned only with a reality expressed in terms of computer input and output data. In the array merging problem, we did not ask ourselves what 'real world' the arrays described: we took the arrays themselves to be the reality. Sometimes the two realities - that of the user and that of the computer - are related in an interesting way, illustrated by the following problem. (The problem is an adaptation of a problem discussed by Henderson in a paper on structured program design\(^{(9)}\), and further discussed in other authors' writings \(^{(12, 7, 11)}\).) For our present purposes a brief outline of the problem and its solution is enough.

A file of telegrams has been punched into paper tape. The paper tape is arranged in blocks, each block being read into main storage as a variable-length character string, terminated by a special EOB character. Each telegram is terminated by a special word 'ZZZZ', and the whole file is terminated by a telegram consisting only of this special word. The program is to produce an analysis of the telegrams, showing such information as the number of words in each telegram. The difficulty of the program arises from the fact that a telegram may begin or end anywhere in a tape block; one block may contain, for example, the last words of one telegram, another complete telegram, and the first words of a third. Words are never split between blocks.

Following my recommendation that development must begin with an explicit model of reality, we see that the structures of the input and output are, broadly:
The diagrammatic notation means, for example, that the PAPER TAPE consists of some number of BLOCKs, one following another, each BLOCK consists of some number of WORDs, one following another, and so on. (The structures shown are highly simplified, for brevity of exposition.)

We now wish to construct the program so that its structure corresponds to both of these structures, but we cannot do so. Because the boundaries of BLOCKs and the boundaries of TELEGRAMs are not synchronised, we cannot construct a single sequential process reflecting both of the data structures: there is a conflict, or clash, between the data structures. The standard resolution of this 'boundary clash' is to construct the program as two sequential processes, not one. With obvious notation:

WORD FILE is an intermediate sequential stream of records, each of which is a WORD. This solution is dictated by the fact that WORD is the 'highest common factor' between the clashing components BLOCK and TELEGRAM. Process P1 dissects the PAPER TAPE into WORDS, and process P2 builds TELEGRAMs from the WORD FILE. It can be readily demonstrated that the 'structure clash' has thus been removed, and that each of the processes P1 and P2 is trivially easy to design and build. Communication between P1 and P2 is limited to their execution of 'write WORD FILE' and 'read WORD FILE' operations respectively.

We have derived our decomposition of the problem into a pair of sequential processes, communicating by writing and reading a sequential data stream, from a consideration of the structures, or grammars, of the problem input and output data streams. But we might have arrived at the same point by considering the reality lying behind the programming problem. Simplifying a little, we may say that there are two independent actors in the real world: a telegraph
clerk, who receives the texts of successive telegrams at a window in the
telegraph office; and a paper-tape punch operator, who punches the texts
into paper tape. These two actors are independent, except that they are
constrained by the rule that the punch operator must punch the words of the
telegrams, in their correct order, into the paper tape. No synchronisation
of the two actors is specified, beyond the inevitable restriction that the
punch operator cannot run ahead of the clerk.

Broadly, this reality is modelled in the two-process solution. The
process P1 models (albeit in reverse) the behaviour of the punch operator,
while process P2 models the behaviour of the clerk. The WORD FILE data
stream is considered to be an unbounded buffer, modelling the freedom of the
punch operator to wait an arbitrary length of time before punching each word,
after the clerk has made it available.

6. PROCESS NETWORKS

Where time-sequencing is a central aspect of the real world, it is
attractive to develop programs and systems in terms of sequential processes.
Within one sequential process text we can capture the total orderings by time;
in data stream communication among processes we can capture partial orderings.
I believe that it is this property of sequential process networks that under-
lies the increasing trend away from procedure hierarchy design and towards
process network design (1, 6, 10, 13).

The freedom to deal in partial orderings brings with it a new responsi-
bility for the developer. Some means must be found of scheduling several
sequential processes on a single sequential machine, of implementing a
scheduling algorithm that is chosen more or less consciously, and adapted
more or less exactly to the needs of the particular problem. The solution
to this scheduling problem may lie anywhere on a wide spectrum. At one end,
we may use a machine equipped with a general-purpose operating system capable
of running many concurrent processes and of providing message-passing services
including buffering in queues. At the other end, we may manipulate or trans-
form the program or system so that it becomes, for execution purposes, a
single sequential process in which all scheduling of the original processes
has been fully bound.

An example of the second approach, applied to the telegrams analysis
problem, would be the use of coroutine communication between the processes
P1 and P2. A 'master program' initiates execution of the system by 'calling'
P1; when P1 has produced a record of WORD FILE, it 'resumes' P2, which in
turn 'resumes' P1 when it has consumed that record and is ready to consume
the next. Eventually, when P2 has consumed all the records of WORD FILE, it
'detaches'; control then passes back to the 'master program', and execution
of the system is complete (3). This approach binds the scheduling of P1 and
P2 fully, when the system is built: the algorithm is essentially that each
process is suspended and activated once for each record of WORD FILE. A similar algorithm may be implemented by transforming P1 into a subroutine ('produce the next record of WORD FILE') called by P2. Such a transformation can be readily systematised \(^{(12)}\) and has been mechanised for COBOL programs in a precompiler available from the author's company. In the context in which the transformation is mechanised, the processes P1 and P2 are designed using 'write WORD FILE' and 'read WORD FILE' operations; the required transformation is specified merely by stating how the WORD FILE, the medium of communication, is to be implemented. Evidently, such a transformation can be applied to more elaborate systems of processes, and other more powerful transformations can also be used to bind process scheduling. In general, a process network can be implemented by (a) transforming the processes themselves, and (b) constructing a special-purpose scheduling process which activates and suspends the processes of the network and provides buffering as needed for communication data streams.

Just as increasing attention is being paid to process networks, so there is increasing work on program transformations of various kinds \(^{(2, 4)}\). These transformations are not, to this author's knowledge, presented as a means of binding process scheduling, but many of them are well suited to this purpose.

7. UNDERSTANDING PROGRAMS

Any development method, whether for systems or for programs, can be seen as a way of structuring and ordering the development decisions that must be taken. The burden of this paper is that for a large class of problem we can recognise three distinct groups of decisions:

- decisions about the reality which furnishes the subject matter of the computation;
- decisions about the program or system outputs containing information about the reality;
- decisions about the scheduling of processes which have been specified in the first two decision groups.

Although all development is necessarily iterative, if only because of human fallibility, we aim to make our decisions in the order given above. A weaker aim, recommended where the first aim is for any reason not adopted, is to understand which group any decision belongs to, and hence to understand more fully what is happening in any actual development work.

I would like to end by offering another example for the reader to ponder in the light of this stated aim. The problem is the well-known problem of printing the first 1000 prime numbers; a procedure-oriented solution is discussed by Dijkstra in Notes on Structured Programming \(^{(5)}\), and a network oriented solution is discussed by Kahn and McQueen \(^{(13)}\). Here no serious solution is attempted: instead some first thoughts are offered, intended to stimulate the reader to consider the problem and the decisions that might lead to a reasonable solution.
We begin by identifying our real world, which is that of the natural numbers. The natural numbers are totally ordered: 0, 1, 2, ... We may therefore construct a process PNN which generates all the natural numbers in order. Each natural number, other than 0, has a totally ordered set of multiples: for examples, the multiples of 3 are 3, 6, 9, ... For each natural number i, other than 0, we may construct a process PMi which generates all its multiples in order, by execution of addition operations. The processes PMi can be created by PNN itself, which writes to each process PMi a data stream containing only the single record whose value is the number i. We now have the network

![Diagram of processes PNN, i, PMi, Mi]

in which the double bar on the arrow entering PMi indicates that there are an unknown number of instances of PMi connected to PNN. Each data stream Mi contains the ordered multiples of the number i.

We now turn our attention to the required output, expecting that our model of the real world (in which the natural numbers and their multiples are modelled) will be sufficient to provide a list of the first 1000 primes. What is a prime? A prime is a natural number except itself and 1. The problem is therefore to identify and list natural numbers satisfying that specification. These are the numbers p which appear only in M1 and in Mp. We can easily add an output process PP which collates the Mi streams and prints the required primes:

![Diagram of processes PNN, i, PMi, Mi, PP, list]

PP can collate the indeterminate number of Mi streams by relying on the fact that if j is a multiple of k then j ≥ k: for a record k in M1, only streams M1, M2, ..., Mk need be considered.

The defects of this solution are many, and most of them are self-evident. The reader, in considering and removing them, is invited to examine each decision and determine whether it is a change to our model of the real world, a change to the function specification, or a change to the scheduling of the system's processes.

* * *

**REFERENCES**


11) W M McKeeman, Respecifying the Telegram Problem, University of Toronto CSRG Paper, November 1976.
13) G Kahn & D McQueen, Coroutines and Networks of Parallel Processes, IRIA Rapport de Recherche No 202, November 1976 (in English).
Physics vs. Computer Science

Robert Pike
Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

With computers becoming more frequently used in theoretical and experimental physics, physicists can no longer afford to be ignorant of the basic techniques and results of computer science. Computing principles belong in a physicist's tool box, along with experimental methods and applied mathematics, and the easiest way to educate physicists in computing is to provide, as part of the undergraduate curriculum, a computing course designed specifically for physicists. As well, the working physicist should interact with computer scientists, giving them challenging problems in return for their expertise.

1. Computing as a Part of Physics

Computing has become an integral part of the practice of physics. Computers are now used throughout physics, from experimental control to plotting graphs, from simulations to typesetting papers, yet many physicists are not adequately educated in their use. As with mathematics, proper use of computing requires some knowledge of the techniques of the field and how they can be brought to bear on a problem. Although from a physicist's point of view there is less to know about computing than mathematics, the importance of computing in physics is increasing dramatically and physicists well-versed in the methods of computer science can use computers to advance science more effectively.

By computing I mean the manipulation of data in a practical sense — the engineering of problem solving on a computer — rather than formal computer science. Physicists face many practical computing problems in their research, problems that the practice of computer science addresses. The general feeling among physicists is that computing is a form of math which can be picked up just by trying it — basically, that it is an easy subject for the physicist. I am not going to dispute that assessment, but will argue that a modest investment of time in the study of the basics of computing will pay off richly in time saved and, sometimes, quality of research. Effective use of computers requires a fair amount of expertise, and much of that expertise is difficult to acquire without expert help. The situation is analogous to learning applied mathematics, say integral calculus: to become adept at calculus, considerable practice is required, but no one is expected to derive the basic results of calculus, discover all the tricks such as integration by parts, and then memorize the integral tables. Similarly, good computer programmers have plenty of experience, but they haven't invented for themselves all the techniques: they have books which are basically algorithm catalogues, and have learned, from other programmers, computer scientists, books and courses, enough about the theory to choose suitable algorithms or programs for a problem.

There is much to know about computing. Some of the areas where computing know-how can help the physicist are:

Experimental control
Running an experiment with various detectors and instruments operating simultaneously is similar to keeping a time-sharing operating system running.

Data Reduction
Interactive statistical packages and computer graphics enable a scientist to understand the implications of an experiment much sooner than line printer graphics on a batch computer.

Discrete simulations
Many physical problems involve more than solutions to differential equations. Group theory, non-rectangular lattices and other components of modern problems can be approached more easily using ideas from graph theory and other computing fields far removed from the typical FORTRAN program.
Numerical simulations

Many numerical problems are now considered solved, in the sense that commercial software packages are available that give accurate solutions efficiently.

Symbolic manipulation

Using computers to solve algebraic problems is more of a computing task than a mathematical one, and using an algebraic manipulation system requires a thorough understanding of the programming art, as well as mathematics.

The techniques of computing belong in a physicist’s toolkit, alongside the techniques of instrumentation and applied mathematics. A physicist need not be an expert computer scientist, just as he or she need not be a research mathematician, but well-informed use of computers can be more effective than the ad hoc use that pervades the physics community. Also, knowing what computers can do and how they do it demystifies them. Any result generated by a computer should be regarded with healthy skepticism — the output from an erroneous program does not describe a new physical effect — but a computer can also solve problems that are unmanageable by other means. It is important to be able to criticize constructively research based on computer methods.

Nowadays, many experiments take place largely inside a computer: experimental high-energy physics would be unthinkable without computers to design the apparatus, control the experiment, detect interesting events and reduce the data. Many theoretical ideas are explored by using computer simulations or symbolic manipulation programs to test hypotheses, perform laborious calculations or create worlds inaccessible to experimentalists. But despite the importance of computing to experimental and theoretical methods, the details of the computing procedures used in a problem are rarely published in the literature. Experimental methods are presented in the literature thoroughly enough to permit others to reproduce the experiment, or at least assess its validity, but computing methods are usually dismissed with a phrase such as “computer simulations of the model showed that…. ” The computing techniques deserve attention in the literature as part of the experimental methods. When a result depends on a computer model or an innovative program, the validity of the result can be assessed only when the programming techniques are presented well enough to critically evaluate them, if not reproduce them. Just as a paper would mention that second-order perturbation theory was used to derive a result, or state which spectrum analyzer generated the plots in a figure, it should name the numerical algorithm or commercial subroutine used to integrate a differential equation, since the accuracy of a result generated by numerical integration depends on the algorithm used. It is not necessary to reproduce the program, just to provide a reference so that the critical reader can assess the suitability of the methods used.

On a related topic, with computers controlling and interpreting so much in modern experiments, the computer methods and programs in an experiment should be designed along with the apparatus and data reduction techniques, not left for a first-year graduate student after the experiment comes on-line. There are important issues in the design of large experiments that depend on understanding of a computer’s capabilities for proper solution: questions such as what the interface from the experiment to the controlling processor should be, which data should be collected for later analysis, or how much processing should be done immediately and how much deferred for later analysis. Without some experience in real-time programming, an experimentalist might be surprised that the on-line processing power is insufficient to keep up with the data flowing from the apparatus; for example, the processor may be too busy servicing I/O interrupts keeping the experiment running to perform first-order event rejection.

2. The Importance of Data Structures and Algorithms

A great deal has been written about the suitability of various languages, particularly (in the physics community) FORTRAN, to modern programming problems, so I won’t spend much time on the subject. Nonetheless, there are a few points that cannot be overemphasized. FORTRAN is a language of the 1950’s. Although FORTRAN was an impressive accomplishment for its time, in the 25 years since its creation, great progress has been made not just in programming languages, but in understanding models of computation and how we use computers. FORTRAN reflects none of that progress, remaining alive primarily because it is the lingua franca of scientific computing, and the vast bulk of existing physics software is written in it. FORTRAN’s position is reminiscent of that of Latin in the 18th century, supported more by tradition than merit.

Perhaps the greatest weakness in FORTRAN is the lack of facilities for structuring data. The notion of a data structure is best conveyed by example: In a program simulating an ideal gas, each particle in the gas
might be described by a position 3-vector, a velocity 3-vector, a mass and some identification of its composition. In FORTRAN, these variables would have to be stored in separate arrays, and a particular particle identified by an index into the arrays. In more modern languages, the variables of each particle can be grouped together in a single structure and treated as a unit. The following data declaration (the language is C) should make the idea clear:

```
struct Particle {
    float pos[3];
    float vel[3];
    float mass;
    char comp[10];
};
```

The Particle structure contains all the information about a particle in a single place, and the different components of the data have names with mnemonic significance.

It is convenient to have a particle’s information localized. For instance, if the program is simulating a chemical process, particles may change in type and number as compounds form or dissociate, and maintaining an array of particles may be difficult in practice. With all the data for a particle in one place — in a Particle structure — the storage for a particle may be allocated dynamically by the language’s storage allocator, and returned to the pool of available storage when the particle “dies.” Then, a particle is referred to by its address stored in a pointer to the data, and the pointer can be conveniently used to pass all the information about the particle to other parts of the program. Such techniques are difficult to arrange in FORTRAN, which has neither dynamic storage allocation nor pointers.

Using a different data structure for a problem, perhaps a graph instead of an array, can sometimes permit use of a more efficient algorithm. A good algorithm can mean the difference between an unsolvable problem and an tractable one. In percolation, the obvious algorithms take time proportional to \(n^2\), where \(n\) is the number of occupied sites in a cluster. Algorithms based on sets and graphs solve the same problem in time proportional to \(n \log n\). Using an optimal algorithm on a cluster with a million sites can reduce a calculation from 50 years of computer time to a few hours (in percolation, clusters must be large to be inside the critical region, the area of physical interest).

The point here is not that the modern ideas make possible computations that were impossible in FORTRAN, but that problems can become easier to solve, and the programs easier to write, understand and debug, when expressed in a better language. Many physicists see computers only as what FORTRAN provides to them, but computers can be much friendlier and more flexible. And even if the programming environment supports only FORTRAN, knowledge of what other languages do can dramatically simplify the programming process. For example, many algorithms are most easily expressed recursively, but FORTRAN does not permit recursive subroutines. But if FORTRAN is the only language supported on the computer, there exist tools for converting recursive algorithms into iterative ones, and knowing how to exploit recursion in a problem may lead to a quicker, simpler solution, even if the final program is, of necessity, iterative.

Not all programs are written to solve physics problems directly: many are tools for the programming process. Examples, besides the obvious ones such as compilers and text editors, are libraries of useful numerical subroutines, sorting programs, graph plotting packages and interactive languages for restricted problems such as statistical analysis. Some programming environments make tool-building and tool-using easy. This applies not just to languages, but computing environments: subroutine libraries, operating systems and computer centers. It is often much easier to throw together a few small programs that can collectively perform a task than to write one from scratch. If the problem can be broken into small pieces, some of the subproblems may be solved by available programs, and it will be unnecessary to write a new, large program for each problem. Knowing this, a programmer can make better use of the machine by writing small programs with clean interfaces to other programs — tools — and combining them quickly, perhaps writing a new tool or two, when a new problem arises. Many of the basic tools should already exist in the environment: a programmer should *never* have to write a matrix subroutine or a sorting program. But often the required tool does not exist, and the programmer must create it. If designed carefully, it will fit smoothly into the programmer’s tool kit and be useful on other problems.
As a specific example, the Unix\textsuperscript{\textdagger} time-sharing system provides a simple mechanism, called a pipe, for combining programs interactively. Programs typically read some data from a single input, process it, and write the result on its output file. The pipe notion is simply a mechanism to connect the output of one program to the input of another. The syntax is simple:

\[ A \mid B \mid C \mid D \]
causes A's output to be processed through B, B's output through C and C's through D. A could be a program that reads a data tape from an experiment and extracts events, putting them into a simple format on its output. B could be a program that recognizes a certain class of event, C might sort the events based on some property, and D could produce a graph of the resulting data. If the right tools are available, a program (a pipeline like this is really a little program) can be put together in a few seconds, to try out some hypothesis. Although each program must read and write data, so there is inherently some loss of efficiency, programmer time is a resource scarcer than computer time; small tools are easier to write and can to combine quickly, leaving most of the work for the machine.

3. Education

Just as physicists learn mathematics, they must learn some computer science if they are to make proper use of the machines. It is certainly not good enough to take a first-year university course in FORTRAN programming; a physicist must learn some of the basics of computing at a level above that provided by a single, ancient, language. The following is a proposed curriculum for one or two computing courses for physicists:

Machine models
What a computer can do; the notion of an instruction; recursion and iteration; basic complexity theory. Writing good programs requires some understanding of what computers are, how they perform computation, and how long the computation will take.

Algorithms and Data structures
Basic algorithms; sorting; graph theory; abstract data types. The simplest solution to a problem may be very different from a DO loop, and the data to describe parameters in the problem may be represented poorly by integers or arrays. Some understanding of the basic algorithms and data structures of computer science can make simple solutions easier to find.

Programming languages and parsing
FORTRAN viewed of a language, rather than the language; Pascal or some other language with data structures; parsing techniques. Learning two or more languages gives a much better understanding of what a computer can do, and how to make it do it. As with the wave and matrix formulations of quantum mechanics, some problems are easier to handle in one language than another. The FORTRAN part of the course should be handled as necessary background, teaching no: only how to use FORTRAN but how to avoid it when possible, or at least overcome its inadequacies. Physicists sometimes write special programs that read some input language, and knowledge of parsing theory may help in designing a language that can be easily handled by both the physicist and the computer.

Numerical Analysis
Basic error analysis; what numerical problems computers can solve; where to get subroutines to do the job. It is much more important to give guidelines to finding a solution than to develop the skills to write libraries of numerical analysis subroutines. Physicists must be aware of what problems are solvable by current numerical software, what commercial subroutines are available, and how to express their problems in a form suitable for solution.

Operating systems and real-time programming
The I/O architecture of computers; interrupts; real-time processing; multiprogramming. Modern experiments require computer control, and a physicist needs at least the basic notions of how a computer can control a machine. Interrupts allow a processor to service multiple I/O ports conveniently, but a program which spends much time servicing interrupts may be unable to keep up with the incoming data. Operating systems face the same sorts of problems, and the techniques developed to solve them — processes and inter-process communication, buffering, etc. — are directly applicable to real-time control of experiments.

\textsuperscript{\textdagger} Unix is a trademark of Bell Laboratories
Graphics

Bitmap and vector displays; interactive graphics; data display; 3-d graphics. Physics describes the interrelationships between the variables in a system, and graphs are used constantly to present and explore relationships. Computers are good at drawing graphs on paper, but they can also be used interactively to explore dynamically the properties of a function or the parameters of a system in pictorial form. As with numerical methods, familiarity with commercial software is more important than being able to create new graphics packages.

These subjects should be taught in courses specifically designed for physicists, not computer science students.

4. Communication

Along with education goes communication: talking with experts in computing. Computer scientists have solved many problems that arise in physics, and physicists should waste no time solving them again. On the other hand, physics is full of interesting computational problems that computer scientists would find challenging and enriching. More interaction between the disciplines can only help both sides.

Numerical analysts, in particular, are always looking for new problems to solve, and know where to find solutions to old problems. But to get the most out of interacting with a numerical analyst, a physicist must not try to second-guess the solution. A numerical analyst is better, from experience at least, at deciding how to best code a problem for computer solution, and a physicist should spend little time trying to rearrange the problem for simpler solution on a computer. For example, a physicist might try to convert a moving boundary value problem into a more difficult equation with a stationary boundary, unaware that second order moving boundary value problems are basically a solved discipline of numerical analysis.

In summary, computer science has a great deal to offer the practicing physicist, theoretical or experimental, and physicists would do well to add computing techniques to their repertoire of problem-solving methods. The result can be more effective use of computers, and more time free for solving the real problems of physics.

5. Acknowledgements

The ideas in this paper can hardly be considered my invention; I have just been asked to write them down. Stimulating conversations with Stu Feldman, Norm Schryer and Stephen Wolfram helped me get the ideas into a cogent form, and Al Aho, Charlie Harris, Brian Kernighan, and Doug McIlroy made invaluable comments about early manuscripts.

* * *

QUESTIONS

MR. M. METCALF CERN

- Q ---→ The proposed curriculum was excellent, but unfortunately the contents would change. Therefore the course would have to be repeated at regular intervals.

- A ---→ It is often enough to get physicists started on the right road, then they would be able to look after themselves.
MR. R. BOCK CERN

- COMMENT ---> Certainly most present in the audience would agree with the proposal to educate physicists in 'computer thinking' and to improve the communication between physics and computer science. However, education happens before physicists become high energy physicists and come to CERN. Also, the speaker is probably preaching to the converted: for every physicist attending the workshop there might be five others who wouldn't accept anybody's help.

MR. D. WILLIAMS CERN

- COMMENT ---> The abstract mentions design of experiments and data reduction. Could the speaker say a few words on the subjects?

- A ---> The speaker replied with a story of a complicated cosmic ray detector which had to be simulated after it was launched into the sky, but did not go into details about the subject.
FORTRAN IS ALIVE AND WELL!

M. Metcalf
CERN, Geneva, Switzerland

1. A LITTLE HISTORY

In 1982 it is perhaps better to examine the state of health of FORTRAN, before making the bold assertion which has been assigned to my talk. In the course of such a review, we can try to extract, at the same time, the answers to three questions:
- Why did physicists start to use FORTRAN?
- Why do physicists continue to use FORTRAN?
- Will physicists always use FORTRAN?

FORTRAN was invented by a team led by John Backus, and was introduced by IBM for its 704 computer in 1957. It was both an innovatory and a revolutionary stride. Innovatory, firstly because it was the first high-level language ever to be devised; secondly, because the use of a high-level language implies the existence of a compiler, and it was, in those days, by no means obvious how to design such a program, as no theories of compiler techniques existed, and they had to be developed by the team itself.

The revolutionary aspects were also twofold: scientific programmers were released from the burden of tedious programming in octal code or assembler language, the first step in a series increasing programmer productivity; but more importantly any scientist was able to approach and use a computer, without needing either a specialist as an intermediary or to learn a difficult low-level language.

The mainstream version at that time was known as FORTRAN II, and by 1964 there were 43 different compilers running on 16 systems. This wide availability of FORTRAN meant that high-energy physicists were also able to switch from low-level languages, and in the early 1960's FORTRAN became the main programming language at CERN, in a dialect known as CERN FORTRAN, a common set among the various compilers available. The diversification of dialects led to ASA (later ANSI, the American National Standards Institute) issuing the first standard in 1966, after four years' work. This standard was the first produced for any programming language, and was at the time the longest standard ever produced by ASA.

It is interesting to consider the advances brought by the introduction of FORTRAN through the eyes of someone writing in 1969. In her history of programming languages, Jean Sammet listed them as:
- use of available hardware;
- the possibility, via the EQUIVALENCE statement, for a programmer to control storage allocation;
- the non-dependence of blanks in the syntax;
- the ease of learning;
- the stress on optimization.

The first point is evident—the language was designed to run on the IBM 704. The second point is no longer obvious today, in the era of cheap semi-conductor storage, but in the time of small core memories it was indeed a boon for programmers to be able to overlay storage areas directly. The last three points remain equally valid now but, as we shall see, the blank will reassert its significance in a future FORTRAN standard, as a separator between syntactic items.

An interesting post script to the early history of FORTRAN has been provided by Backus in his 1977 Turing Award Lecture. In the published paper, he describes how he considers that all programming languages (with some exception made for APL and LISP) have followed a misguided development path, determined by their close correspondence to the von Neumann computer architecture. These languages are large, clumsy and not capable of proper verification. He proposes a new class of programming systems, functional programs, which are capable of algebraic manipulation and proof, and it will be interesting to see whether his ideas have any impact on future computer and language design.
2. FORTRAN'S PRESENT STATUS

Following the publication of the 1966 standard, a new proliferation of dialects and extensions began to appear. These, and the more obvious flaws in the language, led to a new standardization effort which resulted, in 1978, in the definition of what is now known as FORTRAN 77. This is the only version of FORTRAN available on many computer systems, and has been adopted as the CERN standard for new programs as of this Summer.

In principle, the transition from FORTRAN 66 to FORTRAN 77 is relatively simple, as backwards compatibility for standard conforming programs is guaranteed by the standard, apart from a number of minor details, and two major ones: the elimination of the extended range DO-loop, and the replacement of Hollerith constants and data by the CHARACTER data type. This latter point has led to some significant problems for some types of code used in high-energy physics programs, particularly as the Hollerith data are often used in argument lists, for example as histogram titles, and are also often mixed via an EQUIVALENCE statement with numeric data, for example in I/O buffers.

3. THE MAIN NEW FEATURES OF FORTRAN 77

The new features of FORTRAN 77 are described in many books on the subject, and are also summarized in Ref. 4. Here I list only those points I consider especially useful for scientific applications:

i) the ability to declare arrays with up to seven dimensions, each with an optional lower bound, and the ability to use any integer expression as an array subscript expression;
ii) the introduction of the block-IF construct (IF...THEN...ELSE);
iii) the extension of the DO-construct to accept expressions as control parameters, to allow redefinition within the range of the loop of any variable in a control parameter expression without affecting the control of the loop, and the concept of the zero-trip loop;
iv) the introduction of symbolic constants via the PARAMETER statement;
v) the introduction of implied DO-loops in DATA statements;
vii) the introduction of the alternate RETURN;
vi) the introduction of the CHARACTER data type with its associated operators and function:

viii) the extensions to the I/O specification (now occupying 45 pages in the standard) to include direct access files, internal files, execution-time format specification, list directed I/O, file control and enquiry, and some new edit descriptors.

4. THE PORTABILITY OF FORTRAN 77

Program portability is vital for all those working in a dispersed scientific computing community, and since the old dialects have now once again been unified, there should be a higher degree of portability with the new standard than with the old, particularly in view of the new CHARACTER data type and the introduction of direct access files. However, already new extensions are creeping into some compilers, and this trend is to be much regretted, and the use of the extensions avoided, as they will not be generally available, nor necessarily included in a future standard.

Detailed guidance on this subject is to be found in Ref. 5, but the basic rule is, as always, to stick to the standard, using compiler diagnostics and manuals, where they are provided, to give the relevant information. Reading the standard is in itself a worthwhile activity, even if it is somewhat hard going.

5. FUTURE FORTRAN—FORTRAN 8x

The ANSI committee responsible for FORTRAN standardization, X3J3, is currently engaged in a major redesign of the language. Their work and the current status of the revision is described in some detail in Ref. 5. Here, once again, only a few of the more significant points are listed:

i) CORE + MODULE design, allowing flexible growth of the language;
ii) introduction of free form source, with longer names and in-line comments;
iii) entity-oriented declarations;
iv) data structures;
v) new style of DO-loops;
vi) basic and advanced array processing facilities, especially useful on vector processors;
vii) precision specification;

viii) enhanced procedure CALL's
ix) BIT data type;
x) environmental enquiry;
xi) recursion;
xii) dynamic storage allocation (of local arrays);
xiii) CASE control construct;
xiv) compile-time facilities.

6. THE PAST, PRESENT AND FUTURE

Since FORTRAN was the first high-level language to appear, it is clear why physicists began to use it. The reasons why they kept to FORTRAN, rather than follow the ALGOL path, have been outlined in Ref. 7. More relevant is the actual use of PASCAL in areas where FORTRAN would previously have been the obvious choice. In 1976 an attempt to encourage the use of PASCAL in high-energy physics was made\(^9\), but has remained unsuccessful. The reasons for this may perhaps be given as:

- the smallness of the language (e.g. no exponentiation);
- its primitive I/O capability;
- its lack of a mechanism to override strong typing;
- its lack of extended precision;
- its lack of a means to initialize variables (cf. BLOCK DATA);
- its lack of complex arithmetic;
- its lack of a means to pass variably dimensioned arrays through argument lists (except as an option);
- its lack of an interface to libraries or other languages (except in extensions);
- its lack of separate compilation (except in extensions);
- the generally poor optimization of its compilers.

We may thus conclude that whilst it is ideal as a teaching language, and for smaller-scale applications, it remains generally unsuited for large-scale scientific programming over many sites and mainframes.

When discussing the future of FORTRAN, we cannot ignore the emergence of a new language, ADA, which has been developed by the American Department of Defense for use in embedded systems, but which has become to be a general purpose language, containing a strong numerical capability. It will not be long before compilers for ADA become available, and we can expect to see strong competition between the two languages at the end of this decade.

The ADA language will be described by another speaker (Marty, these proceedings), but I want to give here a short list of its main features, as listed by Barnes\(^9\).

FORTRAN was the first language to introduce what is known as expression abstraction, namely the ability to write directly a statement such as \( X = A + C(J) \) without having to be concerned about the allocation of registers, etc. ALGOL introduced the concept of control abstraction, whereby it is possible to write a statement such as if \( X = Y \) then \( A := B \) else \( P := Q \), without having to worry about GO TO's and statement labels. ADA, drawing on the experience of PASCAL and SIMULA, contains a still higher level of abstraction known as data abstraction, which implies a separation of the representation of the data from the abstract operations performed upon them. This is achieved by using enumeration data types, and by hiding the details of data as 'private data' types inside 'packages'. Other features of ADA are:

- its readability;
- its strong typing;
- its provision of facilities for large-scale programming;
- its exception handling;
- its tasking features (the reason for its design);
- its generic units.
Coming back to the title I have been given, I think that nobody can deny that FORTRAN is alive. If we read the software catalogues of programs for use in scientific and engineering applications issued by some computer vendors, we find that, for instance, the 1980 DEC catalogue contains, out of 331 programs, 245 in FORTRAN (74%), 27 in BASIC, 21 in Assembler, 2 in PASCAL and 34 in other languages. The larger 1982 catalogue contains 13 programs in PASCAL, showing some growth in an area completely dominated by FORTRAN. Looking at the similar IBM catalogue, we find in a collection of 175 programs about 150 written in FORTRAN (87%), 8 in Assembler, 5 in COBOL, 12 in other languages and none in PASCAL. We see that in an established scientific computing environment involving commercially available programs, FORTRAN is very much the principal language.

The statement that FORTRAN is well can be deduced from a comparison with some other modern languages. In a recent comparison of PASCAL and C, Feuer and Gehani list eight attributes which they consider desirable for a language used for numerical computation. To this list I have added ease of learning and efficiency of object code, in the following table. In each box, I give a tick for a feature which is fully defined in the given language, enclosed in parentheses where there is a possible hardware dependence, and a question mark where a feature is not fully defined, is unclear or is not defined but nevertheless possible by a more or less easy mean.

Of course, the weights which should be given to each of the attributes varies in a subjective way according to one's own applications, and the assignment of ticks, question marks and blanks is also an exercise whose solution will vary from programmer to programmer. Nevertheless, it is clear that FORTRAN, both present and future, comes out very well by this comparison, which ignores other features which are less relevant to scientific computing.

<table>
<thead>
<tr>
<th></th>
<th>PASCAL</th>
<th>C</th>
<th>F77</th>
<th>F8x</th>
<th>ADA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended precision arithmetic</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Overflow and underflow detection</td>
<td></td>
<td></td>
<td>(✓)</td>
<td>(✓)</td>
<td></td>
</tr>
<tr>
<td>Array operations</td>
<td></td>
<td>?</td>
<td>?</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Complex arithmetic</td>
<td></td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Large no. of maths. functions</td>
<td></td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Binary I/O</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Routine names as parameters</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Lower bounds for arrays</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ease of learning</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>Efficiency of object code</td>
<td></td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
</tbody>
</table>

7. FINAL STATEMENTS

I conclude these brief remarks by making three statements which summarize my own opinion:

i) In the past, FORTRAN was usually the only high-level language generally available, and was therefore used not only for those applications for which it was intended, but also for many for which it was not wholly appropriate. This has given it a worse reputation than it deserves.

ii) FORTRAN is today the only high-level general purpose language generally available for large-scale scientific and numerical applications.

iii) FORTRAN is likely to remain into the next century as, at the very least, a special purpose scientific and numerical language for large-scale, computing intensive applications, and strengthened especially by its array processing capabilities, will be one of a small range of widely used languages in general use.
REFERENCES

8. I. Willers (Editor), High-Level Languages at CERN, CERN report DD/MCM/76/60 (1976).
10. DEC, Engineering Systems and Software Referal Catalog (Digital Equipment Corp. Marlborough, MA, 1982).
11. IBM, Engineering and Scientific Applications Programs available from non-IBM Sources (IBM, White Plains, N.Y., 1981).

* * *

QUESTIONS

MR. R. PIKE - BELL TELEPHONE LAB. MURRAY HILL, N.J.

- Q --> a) FORTRAN 8x seems to be all features and no design.
  b) It is unclear how subroutines discover the type (e.g. array or scalar) of their arguments.

- A --> a) This is due to the selection I have made, and the fact that X3J3 has worked until now on the features, but not yet on making them coherent and regular. This work is now beginning.
  b) Where information about arguments is required, an INTERFACE block will be written by the programmer or a dope vector provided by the compiler.

MR. R. CAILLIAU CERN

- Q --> I want to add some recent information about ADA: Matrix and complex arithmetic is not in the language, but it can be defined in a package. This makes it possible to implement it either as a set of operator routines (in ADA) or to generate the appropriate code for a machine with matrix arithmetic instructions. The user would always see the same interface.
  Also, ADA may have a big definition, but it is not such a big language, FORTRAN 8x is definitely of the same size.
  As to the criteria in your evaluation of languages, I would like to see "ease of mastering" rather than "ease of learning", i.e. how easy is it for me to remember all its details once I am reasonably fluent in a language, irrespective of how easy it was to start.
Lastly, the tasking features of ADA are about the best ones I have seen, and they will influence the way we think about programming very much. Thus I would like to see tasking as one of ADA's most important features.

- A ---> Your points are taken. About matrix arithmetic: this is true, but the advantage of FORTRAN 8x is that it is defined there in a standard fashion.

MR. R. BOCK CERN

- Q ---> When saying FORTRAN 77 is easy to learn and ADA is not, do you not refer unconsciously to FORTRAN-experienced users?

- A ---> I believe ADA, by its size, will be harder to learn, even for a newcomer. There are estimates of six months to re-train a programmer to use ADA.

MR. M. TURNILL BNOC GLASGOW

- Q ---> The driving force for the introduction of new language standards must be improved productivity. What is the perceived or expected gain for the introduction of FORTRAN 77?

- A ---> We have too little experience to judge that yet.

MR. W. MITAROFF INST. F. HOCHENERGIEPHYSIK VIENNA

- COMMENT ---> My experience is 30%. Programs look more readable as it is easier to express problems.

- REMARK ---> I want to stress the importance of producing efficient object code. We are told that, as hardware becomes cheaper, this is no longer important - but bitter experience has shown me that the mainframe hardware to which I have access tends to be always a speed factor of 2-3 behind what I would need. If, by using another language, the execution time would increase by another factor of (say) =3, this would be catastrophic for the performance of our experimental analysis programs. Only FORTRAN has, as one of its primary design goals, to produce efficient object code!

- Q ---> a) what about "freezing" preliminary "desired" features of 8x, to be implemented as (optional) extensions to 77? (I am thinking of better DO-loop structures, and array operations).

b) will a "Dynamic Memory Management" (like "GEM") become part of the 8x standard, or at least become an "Application Module"?

- A ---> It is not possible under ANSI procedures for individual items to be standardized before others. There is no plan to introduce dynamic memory management into the standard.
MR. R. WORDEN LOGICA LTD LONDON

• Q ——> You say that the array handling features of FORTRAN 8x are now well defined, and that manufactures of vector processors are likely to implement them in advance of the full standard. As they are useful not only for efficiency of execution but also as a programmer convenience, would it be possible to provide a pre-processor to FORTRAN 77 so that people could start using them now?

• A ——> It could be done, but would be difficult as a full syntax analysis would be required.

MR. S. O'NEALE CERN

• Q ——> (privately) A large number of physicists write fairly small analysis routines (which are often interfaced to a large package) which are in the throw-away category. Many of these physicists are interested in improved debugging aids in batch and interactive running. Would you comment on the thoughts of manufactures and committees on providing or extending such facilities in FORTRAN 77, 8x, etc.

• A ——> The standards committee is not considering this issue at all. Better facilities will come from manufactures in response to competition and to user pressure. It is up to users to use both these levers to improve their working FORTRAN environment. The excellent debugging facilities of the Siemens/Fujitsu compiler are to be compared with those of IBM's VS FORTRAN compiler and, of course, the VAX interactive debugger is a delight to use. Wield your purchasing power!
ADA AND ITS IMPACT ON THE SCIENTIFIC USER

R. Marty,
Institut für Informatik, Universität Zürich, Switzerland.

ABSTRACT

The Ada programming language is the result of a collective effort to design a common language for programming real-time systems. The design of Ada was initiated by the United States Department of Defense (DoD) in 1975. Ada combines facilities found in most classical languages like Fortran, PL/I, Pascal, and Basic together with many features formerly found only in experimental languages. It is argued that these features make Ada a very decent tool not only for writing real-time programs but also for the development of software in the scientific sector.

1 INTRODUCTION

The DoD is the largest software consumer on earth. As early as in 1970, it became apparent to DoD officials that a new standardized programming language should be defined and implemented to break the sharply rising trend of the exorbitant software costs. The requirements for such a standard language were worked out by DoD's High Order Language Working Group. 17 proposals for the new language were received and four of these were selected to compete for the final choice. The "Green" language, submitted by CII Honeywell Bull of France, made the race and was accepted as the new standard language for embedded computer systems programming. By this time, "Green" was renamed "Ada" in honour of the Countess of Lovelace, daughter of Lord Byron, who was Babbage's assistant and the world's first programmer.

The key features of Ada are:

- **data abstraction** (specification of abstract properties of data and not of physical representations, new types may be defined together with the operations on them)
- **strong typing** (every data object is bound to an abstract type, type incompatibilities are detected)
- **block structure** (hierarchical levels of scope for named objects)
- **exception handling** (mainly for user controlled error recovery)
- **tasking** (creation, deletion, and synchronization of parallel processes, interprocess communication)
- **generic units** (program units can be parametrized over appropriate data types, especially useful for library routines)
- **programming in-the-large** (encapsulation mechanisms, type-safe separate compilation, overloading of operators and subroutines, library mechanisms are part of the language definition)

The history of Ada's development clearly shows that Ada is a language which is to be used in programming embedded systems applications, i.e., for real-time programs. It is the intent of this paper to argue that Ada is also a very decent language for scientific programming, an area now largely dominated by the antiquated programming language FORTRAN, a language that fails to meet several paramount aspects of modern software engineering.

For the layout of this paper, measuring Ada against aspects of modern software engineering has given preference over an enumeration of Ada's features. Ada will not be discussed in its full scope here. We are only trying to give a first insight into the power of Ada. A

2 DATA ABSTRACTION

Programming a computer is always based on some mental model of computation residing in the programmer's brain. It is the programmer's task to map this abstract model of computation into the real model of computation given by the programming language used. The wider the gap between the abstract model and the programming language, the more cumbersome the programmer's work gets, and the more difficult will it be for a third person to remap the program into his or her own model of computation. The costs of maintaining software are significantly influenced by the difficulty of this process. A programming language should match an abstract model of computation as closely as possible.

A particularly well structured model of computation emerged from the Algol school and found its implementation in e.g. Algol 60, Algol 68, PL/I, Pascal, and Ada. Since the Algol model of computation found wide acceptance in education and practice and was the core of the bulk of algorithmic programming languages introduced in the last 15 years (including Ada), it seems reasonable to adapt it for scientific programming as well. Consequently, the programming language we choose should match this model as closely as possible, a requirement that is not met by FORTRAN, COBOL, or BASIC.

A program describes actions (or algorithms) to be performed on data. This leads to a discussion of Ada's abstraction mechanisms under the two headings "Data Abstraction" and "Algorithmic Abstraction". The former will follow right here, the latter in Chapter 3.

2.1 Numeric Types

The guiding principle of describing data in Ada is that a data description should define the abstract properties of the data object without any reference to a specific representation in hardware. As a consequence, the introduction of a new type for representing integer values includes an indication of the range of values to be represented by objects of this integer type:

```
type DAY is range 1..31;
type TEMP is range -50..+50;
type LIRE is range -99999999..+99999999;
```

(It is just a notational convention to write keywords in small letters and identifiers in capitals). Objects of such a type may then be created by an object declaration:

```
TODAY : DAY;          -- uninitialized variable
AMOUNT : LIRE := 0;   -- initialized variable
LIMIT : constant TEMP := 27;    -- constant
```

An important advantage of explicitly defining the desired range is the independence from specific internal representations of integer values, i.e. from the hardware range of integers. This means that a language violation is detected immediately upon an integer value falling outside the declared range and not only upon producing a hardware overflow/underflow
condition. Giving explicit ranges of integer values also allows the Ada implementation to choose whatever internal representation for the integer object as long as the language semantics hold. The type LIKELY above could for example be represented as a double word or in floating-point format. It is very important, however, that the programmer is and should be unaware of such implementation considerations.

The fixed-point types and the floating-point types are described in the same spirit of data abstraction as discussed for integers:

```ada
type HOURS is delta 0.01 range -9999.99..9999.99;
type WEIGHT is delta 0.005 range 0..1000;
type MASS is digits 7 range 0.0..1.0E10;
type COEFFICIENT is digits 10 range -1.0..+1.0;
```

Besides the range, the accuracy of fixed-point types is specified by an absolute value, called the "delta" of the fixed-point type (the difference between two adjacent values of this type). The accuracy of floating-point types is given by indicating the minimum number of decimal digits for the decimal mantissa.

2.2 Enumeration Types

A very common use of integers is for designating elements of a set of nonnumeric values. Ada's so-called "enumeration type" is a natural construct for this purpose:

```ada
type WEEKDAY is (MON,TUE,WED,THU,FRI,SAT,SUN);
type MARITAL_STATUS is (SINGLE,MARRIED,DIVORCED);
type DEPT is (FOOD,HOUSEHOLD,STATIONERY,TOOLS);
```

The identifiers specified for an enumerated type are the ordered values of this type and are used as constants in expressions of this type:

```ada
TODAY : WEEKDAY;
:
if TODAY=SUN then ...
TODAY := MON;
```

The advantage of using enumerated types over encoding the corresponding information by integer values lies in the better documentation of the purpose of such a data object and in the enforced type safety.

2.3 Other Scalar Types

Scalar types are types whose values have no components. In addition to the scalar types introduced above, Ada defines a scalar type for logical data objects (data objects that can assume the values TRUE and FALSE) and one for character data objects. It is further possible to define subtypes of scalar types:

```ada
subtype WORKING_DAY is WEEKDAY range MON..FRI;
subtype ALPHA is CHARACTER range 'a'..'z';
```
2.4 Composite Types

A composite type is a type that is made up of components of other types. Ada defines the "array" and the "record" as composite types. The index type(s) of an array can be any discrete scalar type, e.g.:

```ada
  type HOURS_WORKED is array (WEEKDAY) of HOURS;
  type MATRIX is array (1..50,1..50) of range 0..1000;
  type SOLD is array (DEPT,WEEKDAY) of FRANCS;
```

Ada "records" are similar to COBOL and PL/I data structures:

```ada
  type TIME is record
    HH : INTEGER range 00..23;
    MM : INTEGER range 00..59;
  end record;

  type FLIGHT is record
    AIRLINE : STRING(2);
    FLIGHT : INTEGER range 0001..9999;
    ARRIVAL : TIME;
    LANDED : BOOLEAN;
  end record;
```

It is also possible to overlay the same storage space with different variants of records.

2.5 Access Types

Ada's access types correspond to pointers in PL/I and Pascal, a concept unimplemented in FORTRAN. In contrast to PL/I pointers but in accordance with Pascal, an Ada access type is bound to a specific type to prevent the pitfalls of unbound pointers:

```ada
  type LINK is access FLIGHT;
```

Access types permit the building of dynamic data structures of almost arbitrary shape and complexity (linked lists, trees, graphs, etc.).

3 ALGORITHMIC ABSTRACTION

The chapter on data abstraction dealt with the abstract model of computation given by Ada as viewed from the description of data objects. This chapter introduces Ada's way of describing actions, i.e. the algorithmic abstraction as implemented in Ada.

3.1 Description of Expressions

An expression is a construct to create a new data object through application of operations on existing data objects. Examples for Ada expressions are:

```ada
  AMOUNT / 100 * INTEREST RATE / 360 * NO_OF_DAYS
  (DEBIT-CREDIT)*FACTOR > x LOAN
  MONTH not in JUN..SEP and TODAY in WORKING_DAY
  FIRST_NAME & " " & LAST_NAME
```

Ada defines the following operators:

- arithmetic: +, -, *, /, mod (modulo division), rem (remainder), ** (exponentiation)
- relational: =, /= (not equal), <, >, <=, >=
3.2 Flow of Control

The usual flow of control in an Ada program unit is sequentially from top to bottom. An **if** statement selects for execution one or none of a number of sequences of statements, depending on the truth value of one or more corresponding conditions:

```ada
if A<0 then
  ...
else
  elsif A>100 then
    ...
  elsif A>10 then
    ...
  else
    ...
end if;
end if;
```

A second statement for specifying conditional execution of sequences of statements is the **case** statement, roughly corresponding to PL/I's **SELECT**:

```ada
case TODAY is
  when MON => ...
  when TUE..FRI => ...
  when others => ...
end case;
```

Repetition of a sequence of statements is described by a **loop** statement in one of the following forms:

```ada
loop
  ...
end loop;
```

```ada
while A<0 loop
  ...
end loop;
```

```ada
for I in 1..100 loop
  ...
end loop;
```

The leftmost form specifies an endless loop. Naming of loops is also possible. Any loop can be terminated by the execution of an **exit** statement.

Ada's statements used to describe the flow of control within a program unit are actually equivalent to many graphic notions (Jackson Diagrams, Nassi-Shneiderman Charts, etc.) used to design and document programs written in languages with poor control statements. Using such diagrammatical aids for designing and describing Ada programs is unnecessary since the programming language Ada itself can very well be used for this purpose during all stages of program development, from global layout right down to the final form of the program.

3.3 Subprograms

A subprogram is an executable program unit that is invoked by a subprogram call. There are two forms of subprograms: procedures and functions. A procedure call is a statement, a func-
tion call returns a value.

In Ada every formal parameter of a procedure or function has a mode:

- **in** parameters act as inputs to the subprogram
- **out** parameters act as outputs from the subprogram
- **inout** parameters act as variables whose values may be updated during execution of the subprogram

Formal parameter specifications may also include default values which are used when the corresponding formal parameter is not present in the call:

```ada
procedure PRINT_HEADER (PAGES : in INTEGER := 1;
                        HEADER : in LINE;
                        CENTER : in BOOLEAN := FALSE);
```

All Ada subprograms are reentrant and can be called recursively.

A suitable subprogram concept is a very important cornerstone in modern software engineering. It allows programming on different levels of abstraction by hiding internal mechanisms and only interacting with the environment via a well-defined interface. This leads to simpler and better understandable programs.

### 3.4 Parallel Processes

Many readers will probably wonder why we talk about parallel processes in a paper that concentrates on scientific software. We argue that parallel processes should be used as an elementary means of algorithmic abstraction in programming. To illustrate this, let us look at a practical problem that has its roots in business-oriented programming but serves our purposes very well:

A publisher keeps a mailing file to produce labels for a periodical, one label per subscriber. Every record holds the subscriber’s address and the number of copies he gets. Our program has to print mailing labels, four abreast, to be processed by a labelling machine (e.g. Cheshire). The periodicals must be bundled up as prescribed by the postal authorities:

- If there are 5 or more copies for a single ZIP code (or postal code as it is called in many countries) they have to be collected in one or more bundles addressed to this ZIP code.
- If there are 4 or less copies for a single ZIP code they have to be collected in one or more "compound bundles" addressed to the associated regional distribution office (addressed by, say, the first two digits of the ZIP code).
- No bundle may consist of more than 18 copies. If there are more remaining to be bundled up, collect 15 into one bundle and start a new bundle with the rest.
- Every bundle must be topped by a cover sheet with the full ZIP code or the regional distribution office code if it is a compound bundle. Our program has to produce these cover sheets also. To indicate the end of a bundle to the labelling machine, the program has to augment the last address label per bundle with some optically recognizable sign at a given position.

The data flow in our program will be:
None of the five modules A to E can be called complex. Modules B and C have to implement a lookahead to take the correct decision, and module E has to suspend printing of the addresses until it has four ready to print. These problems, however, can be easily solved with some temporary storage of addresses. The addresses to be collected into the compound bundle may be written into a workfile by module B. Upon control break on the first two digits of the ZIP code, this workfile is reread and its addresses passed on to module C with the ZIP code of the regional distribution office to which the compound bundle should be mailed.

The most natural way of programming the data flow of Fig. 1 is to create a task (or process) per module, all tasks running in parallel. The tasks communicate with each other to exchange data (addresses in our example). The creation, deletion, and synchronization of tasks as well as intertask communication is implemented by Ada on a very high level of abstraction. Ada's task concept is certainly no more complex to understand and use than other concepts in high level programming languages.

As an example of an Ada task, let us look at the implementation of module C in the data flow of Fig. 1:
task MODULE_C is
  entry NEXT_ADDR (A : in ADDRESS);
end;

task body MODULE_C is
  BUFFER : array (0..4) of ADDRESS; -- buffer for 5 addresses
begin
  for I in 1..4 loop FILL_BUFFER -- slots 1 to 4
    accept NEXT_ADDR (A : in ADDRESS) do
      BUFFER(I) := A;
    end;
  end loop FILL_BUFFER;
  while BUFFER(1).ZIP/=9999 loop
    for I in 1..18 loop BUNDLE_UP
      MODULE_E.PRINT_ADDR(BUFFER(1));
      BUFFER(0..3) := BUFFER(1..4); -- shift left buffer
      accept NEXT_ADDR (A : in ADDRESS) do
        BUFFER(4) := A;
      end;
      exit when BUFFER(0).ZIP/=BUFFER(1).ZIP;
      exit when I=15 and BUFFER(4).ZIP=BUFFER(1).ZIP;
    end loop BUNDLE_UP;
    MODULE_E.PRINT_STOP MARK;
    MODULE_D.PRINT_COVER(BUFFER(0).ZIP);
  end loop;
  MODULE_D.DONE; -- signal end to module D
  MODULE_E.DONE; -- signal end to module E
end MODULE_C;

The type ADDRESS is assumed to be a record containing an address to be printed. MODULE_C task expects its caller to deliver 4 pseudo addresses with the ZIP code 9999 to signal the end.

As we see, a call to an entry of another task has the same form as a procedure call (e.g. "MODULE_D.PRINT_COVER(BUFFER(0).ZIP)"). An accept statement in a task waits for the corresponding task entry to be called by another task. When called, the statements embraced by "do" and "end" are executed. Waiting for any one of a number of entries to be called is also possible. For a complete explanation of the tasking mechanisms in Ada see reference [2].

Of course, the above problem may be programmed with procedures only, without using tasks. Every procedure would have to remember its internal state in global variables before returning control. The program could even be written without procedures, namely by inverting the data flow into one single program module. Such a program, however, would certainly no longer reflect the structure of the data flow as shown in Fig. 1. Its complexity will be much greater than the sum of the complexities of the single modules A to E.

4 DIVIDE AND CONQUER

The principle "divide and conquer" is attributed to the Old Romans and kept its importance for the management of all but the simplest systems of any kind. Its message is that a global system should be broken down into subsystems, these in turn again into smaller units, and so on. This division of a system into several subsystems and the associated definition of functional dependencies and intersystem communication is supposed to result in a system that is better manageable than a monolithic system.
Software is nothing else but some kind of a system, albeit not one that is physically perceptible. Hence, "divide and conquer" is also an important guiding principle for software development. It means that a program should be designed as a hierarchical composition of subsystems (or modules) which are easy to program and to maintain. Ada supports the method of "divide and conquer" in software production in many ways. Some of them have already been mentioned in the previous chapters. In addition we shall discuss Ada's concept of block structure, packages, and separate compilation here.

4.1 Block Structure

In FORTRAN, a given identifier is known throughout a program unit (main program or subroutine) but is hidden from other program units (except for COMMON identifiers). Such a uniform, global scope of names implies that all statements of a program unit operate on the same name space, i.e. an identifier X always refers to the same object. It is not possible to encapsulate certain data objects in a subunit while still having access to global data objects (information hiding). This is a severe drawback to writing independent program parts in the sense of "divide and conquer". One has to carefully avoid malicious side effects of functionally independent program parts. Such side effects mainly occur because of modifications to a variable that is expected to remain invariant over the execution of such a program part. The danger of introducing malicious side effects is greatest during program modifications by someone who has not written the original program.

A block structured programming language such as PL/I, Pascal, or Ada eliminates this problem by introducing hierarchically nested scopes of names:

```
program block
  block A
    block B
  block C
```

The programmer may declare named objects within every block. A name used in a block refers to the locally declared object with this name. If no such object exists, the name refers to the accordingly named object in the smallest surrounding block. It is never possible to refer to an object declared inside a block from outside this block, but only from the block itself and from all blocks nested directly or indirectly within this block.

Ada associates a separate scope of names with every subprogram, task, and package (plus some other constructs not discussed here). The programmer may also open a scope anywhere in a statement part as follows:
Here, the identifier TEMP is introduced as a temporary storage used in swapping the contents of NEXT and NEW which are assumed to be of the type FLIGHT. The scope of TEMP extends to the "end" closing the block.

Block structure is one of the most important attributes of a modern programming language. It supports the method of "divide and conquer" by hiding information to the outside world of a block and by providing a separate name space for every block.

4.2 Packages

In its simplest form, a package represents a pool of common data and type declarations:

```plaintext
package WORK_DATA is
  type DAY is (MON,TUE,WED,THU,FRI,SAT,SUN);
  type HOURS is delta 0.01 range 0.00 .. 24.0;
  type TIME_TABLE is array (DAY) of HOURS;
  WORKED : TIME_TABLE;
  NORMAL_HOURS : constant TIME_TABLE := (MON..THU => 8.25,
                                          FRI => 7.0, SAT|SUN => 0.00);
end WORK_DATA;
```

This package only has a so called visible part. The visible information is given as a sequence of declarations. This information, however, is not directly visible to other parts of the program or, in Ada terminology, to other program units. One way of referring an identifier declared in a package is by specifying the package name and the identifier in the so called dot notation, e.g.

```plaintext
WORK_DATA.WORKED
```

Thus, we might have the assignment

```plaintext
WORK_DATA.WORKED(MON) := 5.25;
```

The second form of packages also includes subprograms (procedures, functions) to process the data declared in the package. Assume that we want to include to the package WORK_DATA a function to calculate the overtime and a procedure to update the array WORKED. The visible part of the package would then look like this:
package WORK_DATA is
  type DAY is (MON, TUE, WED, THU, FRI, SAT, SUN);
  type HOURS is delta 0.01 range 0.00 .. 24.0;
  type TIME_TABLE is array (DAY) of HOURS;
  WORKED : TIME_TABLE;
  NORMAL_HOURS : constant TIME_TABLE := (MON..THU => 8.25,
                                       FRI => 7.0, SAT|SUN => 0.00);
  function OVERTIME (WHEN : in DAY) return HOURS;
  procedure UPDATE (WHEN : in DAY; TIME : in HOURS);
end WORK_DATA;

This package is obviously incomplete since it lacks the implementation of the two subprograms OVERTIME and UPDATE. They are defined in a separate so called package body:

package body WORK_DATA is
  -- internal declarations
  -- internal subprograms
  function OVERTIME (WHEN : in DAY) return HOURS;
  :
  end OVERTIME;

  procedure UPDATE (WHEN : in DAY; TIME : in HOURS);
  :
  end UPDATE;
begin
  -- statements initializing the package
end WORK_DATA;

A package body essentially contains the implementation of the subprograms declared in the visible part of the package. Writing these subprograms may in turn require the use of local declarations (variables, constants, types) and local subprograms. They are part of the package body as well, but are not visible outside the package itself, since they do not appear in the visible part. The use of local variables may require the execution of statements to initialize them. These statements are executed when the package comes into existence at run time.

Ada also provides mechanisms to restrict operations on objects declared in the visible part of a package, such that only subprograms of the package itself may operate on these objects. Thus, Ada packages can implement the concept of "abstract data types".

Ada packages are an excellent tool to develop software by the method of "divide and conquer". Unlike simple "include" or "copy" clauses known from other languages, they offer all features necessary to encapsulate data (types, constants, variables) and subprograms providing access to the data. The subdivision of a package into a part visible to the outside world and an invisible part is a very effective safeguard against misuses of the package (intentional and unintentional ones). In some way, the visible part of a package represents a contract between the writer of a package and its user, where the package body is the implementation of the contract.

4.3 Separate Compilation

Ada packages and subprograms may be compiled separately and placed in a library. The speci-
fication of a package (the visible part) and the package body (the implementation of the package) are treated as distinct units and are kept separately by the Ada library mechanism.

A program unit that is compiled may refer to a precompiled package or subprogram. Since not only the implementation of the packages and subroutines but also their interfaces (the visible part of the packages, respectively the definitions of the subprograms) are kept in the library, the Ada compiler can ensure a correct usage of separately compiled subroutines. In other words, Ada supports type-safe separate compilation.

5 PORTABILITY

According to Poole and Waite [3], "Portability is a measure of the ease with which a program can be transferred from one environment to another: if the effort required to move the program is much less than that required to implement it initially, then we say that it is highly portable".

We can expect Ada programs to be more portable than programs written in any other language that is used widely today. Several facts contribute to this property of Ada programs:

- Ada will be the first widely used programming language for which a complete language definition existed before its implementation began. The language definition acts as a coercive standard for all implementations.

- If an implementor calls his language implementation "Ada", it is neither allowed to be a subset of Ada as defined in the Ada Reference Manual, nor to be a superset of it. The latter restriction is a very important rule enhancing program portability: If a certain implementation of a standardized language includes extensions to the standard, the programmers will probably make use of these goodies. The undesirable result are programs written in a "standard language" that are not portable to other implementations of the same "standard language".

- There will be validation software available to check an Ada implementation to be in conformance with the language definition. Any Ada implementation will be checked out thoroughly by the validation software. It may use the name Ada only after it succeeds the validation procedure.

- The tremendous pressure behind Ada (from the world's largest software consumer, the DoD) together with the applicability of Ada in various fields of programming will result in Ada implementations on almost any computer system from micro computers to mainframes.

Even a completely standardized language such as Ada, however, does not guarantee an absolute portability from one environment to another since the interface to certain system software components is very likely to differ between the two. If Ada packages are used to map application oriented interfaces to the actual system interfaces, a high degree of portability can be established. To port the entire software system, these packages would have to be adapted to the new situation while there would be no changes to application programs.

6 CONCLUSION

Ada is a tool for scientific programming which has important advantages over FORTRAN. Its superiority is manifest in all three aspects of software engineering discussed in this paper: in data abstraction issues, in the way a suitable algorithmic abstraction is implemented, and in supporting a software production by the method of "divide and conquer".
A software system written in Ada will not necessarily live up to high expectations, if old-fashioned project management principles and outdated implementation methods are used. It is not difficult to program Ada in a FORTRAN style. There is also a danger of setting up project guidelines which do not take advantage of Ada's features or even hinder their use.

Combined with an adequate project management and modern software engineering principles, however, Ada is a very promising landmark on the way to cheaper software.

REFERENCES


QUESTIONS

MR. J.T. CARROLL CERN

- Q --> As a consequence of data abstraction and algorithm abstraction the computer hardware is completely hidden from the user. Is this not in conflict with writing efficient programs.

- A --> The execution efficiency is of a lesser concern than development efficiency.

MR. R. PIKE BELL TELEPHONE LAB. MURRAY HILL, N.J.

- COMMENT --> The most important thing for an efficient program is the right algorithm.

MR. R. CAILLIAU CERN

- Q --> Tell something about the support environment of ADA.

- A --> The DoD also specifies a programming environment, but the speaker does not want to spend time on this subject.

MR. R. CAILLIAU CERN

- COMMENT --> The support environment is 50% of ADA and the way to cheaper software.
MR. J. GAMBLE CERN

- Q ---› Arguments supporting strong typing have been brought forward in several presentations and that in examples types like FRANC, LIRA etc. have been recommended. But then one could not write:
  Variable of type FRANC = Variable of type LIRA*exchange rate

- A ---› It could be done in ADA, but would be flagged by the compiler.

MR. H. GROTE CERN

- Q ---› Will ADA programs running on different types of computers with different floating point representations and precision give exactly the same results? (i.e. will they be really fully portable).

- A ---› For a floating point number declared with n digits, full precision will be guaranteed up to the n-th digit.

MR. G. ENDERLE KERNFORSCHUNGSZENTRUM, KARLSRUHE

- Q ---› Are there plans to standardize certain ADA packages? If every implementation created a different package for complex number operations, no portability would exist?

- A ---› Some packages would have to be standardized.

MR. G. ENDERLE KERNFORSCHUNGSZENTRUM, KARLSRUHE

- Q ---› How portable are the packages themselves?

- A ---› Portability will, in first place, be given above the package level. The packages may have to be adapted to different systems and therefore need to be designed very carefully.

MR. J. ZOLL CERN

- Q ---› Is a standard format foreseen to ensure the portability of data?

- A ---› This is not a language issue.
MR. J. ZOLL CERN

- COMMENT ---> The data may be hidden so carefully from the user that he is not able to find them any more.

- A ---> The writer of an I/O package will know the hardware representation of the data.

H. RENSHALL CERN

- COMMENT ---> The mass exchange of data will be made very difficult by the many possible data types allowed by ADA.

MR. D. WILLIAMS CERN

- COMMENT ---> The HEP community wants concrete typing, not abstract typing.

- A ---> Only a small group should be aware of the internal number representation.

MR. J.G. JEFFERY RUTHERFORD APPLETON LABORATORY

- COMMENT ---> Use either data definitions or subroutine definitions to handle I/O of applications.

- A ---> This would not be safe and would allow break of the abstraction concepts.

MR. W. MITAROFF OST. AKADEMIE D. WISSENSCHAFTEN, VIENNA

- COMMENT ---> There have been serious objections recently to ADA by Dijkstra, and the ACM did not recommend ADA to become an ANSI standard, on the grounds that ADA is supposed to be not well defined.

- A ---> ADA has some ill-defined parts, but the speaker expresses his doubts whether it is possible to define a considerably simpler language suited for software development in the large.

MR. D. WILLIAMS CERN

- Q ---> When will the statement 'ADA is a better scientific programming language than PASCAL or FORTRAN' become true?

- A ---> It will be in about 2 years from now.
SOME SOFTWARE TOOLS FOR SCIENTIFIC PROGRAMMING

R.P. Worden

ABSTRACT
A number of advanced software tools are described which have been used by Logica or its clients for scientific or technical software development. These are: RAPPORT, a Relational Database Management System; A Fortran Program Analyser, which is designed to answer those questions about large Fortran Programs which are not easily answered by examining listings; A Test Coverage Monitor, which measures how well the code and branches in a Fortran program have been exercised by a set of test runs; The UNIX operating system and the tools available with it. These tools will be described with examples of their use in practice.

1. INTRODUCTION

This talk is not a general review of the field of software tools in scientific programming, but is a description of a few specific tools which I have come in contact with - and in some cases helped to develop - while working at Logica. Logica, whom I joined in 1975, is one of Britain's leading software houses with a strong reputation in advanced scientific and technical applications. Prior to that I did research in High Energy Physics, in the grey areas between theory and experiment, with two spells at CERN. So I can fairly claim to appreciate two different points of view - that of the professional scientist whose main interest in the computer is to get results out fast, and that of the professional software engineer who cares about software quality, structure and so on. They are two very different perspectives on the same problem. During both my visits to CERN, I actually spent most of my time programming, but was aware of very few software tools beyond a Fortran compiler. I was certainly not aware of the kind of tools which I shall be describing today, but I believe, from my present perspective, that they would have been very useful to me then and can have many uses in High Energy Physics today. The tools which I shall describe are:

(1) RAPPORT - a relational database management system which can have many uses in storing experimental data, or engineering and administrative data concerning experiments. The description of RAPPORT serves as a useful background for understanding the other tools, which use RAPPORT to store their data.

(2) The Fortran Program Analyser (FPA) which is an interactive tool for understanding and documenting large Fortran programs.

(3) The Test Coverage Monitor (TCM) which is a tool to measure how well a Fortran program has been tested either by special test cases or live running.

(4) The Static Analysis Package (SAP) which is a further development of the previous two tools, currently being implemented by Logica for the European Space Agency. As well as incorporating test coverage and program
analysis functions, it has a number of special functions in the area of
program quality.

(5) UNIX the well known operating system from Bell labs which also incorpor-
ates a highly esteemed set of software tools. I shall briefly describe
one or two of these which are useful in scientific applications.

2. RAPPORT

Why use a Database Management System (DBMS) for scientific application?
In fact a DBMS is such a general purpose tool that it is hard to give a spe-
cific answer to that question - it can be used on all sorts of different
ways and can give a lot of different benefits. To see whether you could use
a DBMS, ask a number of questions about the data in your computer application.
Is it (or are they):
- large in volume?
- complex in structure, with interrelationships between different pieces
  of data?
- interesting to different people from different viewpoints?
- likely to be used in a long-term, evolving application?
- to be used to answer unanticipated questions?
- of long-term value?
If the answer to one or more of these questions is yes, you could probably
use a DBMS. To see how, I shall use Logica's Relational DBMS, RAPPORT, as
an example.

RAPPORT uses the Relational model of data, which means that the user's
(or programmer's) view of the data is a collection of simple two-dimensional
tables. An example is shown in Figure 1 from a hypothetical database of two-
body reaction cross-section data. This database has three tables, one giv-
ing some basic particle properties, one giving information about 'experiments'
(where for simplicity I have assumed that each experiment has only a single
beam energy, beam and final state), and another giving the actual cross-sec-
tion data as a function of the momentum transfer, t.

The first advantage of using a DBMS is that the programmer need only
learn about the logical structure of the data (i.e. the meaning of the differ-
ent rows and columns in the tables), not about the underlying physical stor-
age methods. This contrasts with the direct approach of storing data on a
magnetic tape or whatever, where you have to know a lot of sordid facts
about tape format, blocking and data layout before you can use the tape. If
the data is in a DBMS, programs can be written knowing only the logical
structure of the database. With RAPPORT, typical statements in a Fortran
program would be

FETCH PARTICLES (NAME = 'Pi-')
to retrieve the record for the particle \( \bar{\Pi}^- \), or

SEARCH EXPERIMENTS (ENERGY.GE.60 AND ENERGY.LT.70)
to find all experiments with a beam energy between 60 and 70 GeV. These
statements are simple to write, simple to understand and make no reference
to the way the data are stored physically.

<table>
<thead>
<tr>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXPT</strong></td>
</tr>
<tr>
<td>BER74A</td>
</tr>
<tr>
<td>BER74A</td>
</tr>
<tr>
<td>BER74A</td>
</tr>
<tr>
<td>BER74A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EXPERIMENTS</th>
<th>Final State Particles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXPT</strong></td>
<td><strong>ENERGY</strong></td>
</tr>
<tr>
<td>BER74A</td>
<td>51.0</td>
</tr>
<tr>
<td>BER74B</td>
<td>51.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PARTICLES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NAME</strong></td>
</tr>
<tr>
<td>PI-</td>
</tr>
<tr>
<td>OMEGA</td>
</tr>
</tbody>
</table>

**Figure 1** A Simple Relational Database

The second advantage of a DBMS is that it gives tremendous flexibility
in retrieving data, either from within a program or interactively from a ter-

minal. RAPPORT has an interactive query language (IQL) which can be used
either to answer ad hoc questions from a terminal, or to build up a library
of predefined 'command sequences' to answer common questions. Typical IQL
sequences are:

```
ORDER ON DATA BY T DOWN
SEARCH DATA WHERE EXPT = 'BER74B'
WRITE DATA-T,DSIGDT,ERROR
LOOP
```
SEARCH PARTICLES WHERE CHARM = -1
SEARCH EXPERIMENTS WHERE (F1 = PARTICLES-NAME OR F2 = PARTICLES-NAME)
WRITE EXPERIMENTS-NAME,ENERGY,BEAM,TARGET,F1,F2
LOOP

The basic construct is the SEARCH.....LOOP pair, which retrieves (in succession) all records satisfying the conditions of the SEARCH. Every statement inside the SEARCH.....LOOP is performed once for each record found. Thus the first example gets all records for the experiment 'BER74B' (in order of momentum transfer, t) and for each record writes out t, the cross-section DSIGDT and its error - producing a small table of cross-sections versus t. In the second example, the outer loop finds any particle with Charm equal -1, and the inner loop finds any experiment with that particle in its final state (i.e. one of the particles F1 or F2) - so producing a list of all experiments with a Charm -1 particle in the final state.

A third advantage is that besides being very flexible in selecting records from the database, a DBMS can be very efficient when doing so. RAPPORT has an elaborate system of secondary indices (which are maintained and used automatically by the DBMS). Thus when you ask RAPPORT to do a selective retrieval, it will wherever possible use a secondary index rather than sweeping through all records. So retrievals cost typically less than two disc accesses per record - which, for a large database, can represent a major saving over 'brute force' searching.

Those are the main reasons why a modern DBMS can be of benefit - independence of physical storage structures, ease of understanding and programming, and flexibility of data retrieval. There are a host of other benefits, perhaps less obvious but equally important for many applications. For instance, RAPPORT has elaborate recovery features to protect data from all sorts of hardware and software failure, multiple access controls and data security controls to restrict accessing or updating some parts of the database on a password basis. Last and not least, it is portable - we have implemented it on most common minis and mainframes, including IBM, CDC, DEC, Prime and Data General - so that programs and data which use RAPPORT can be made portable across a very wide range of hardware.

To clear up one small potential misunderstanding about Relational databases - because they use a model of two-dimensional tables, that does not mean they are unsuitable for storing multidimensional (n>2) data. Quite the reverse. For instance in an experiment where events are binned according to some two-dimensional array of counters (dimensions x and y), with a variable beam energy providing a third dimension, the resulting relational table would look like Figure 2:
- 45 -

DATA

<table>
<thead>
<tr>
<th>BEAM ENERGY</th>
<th>x bin</th>
<th>y bin</th>
<th>No of events</th>
</tr>
</thead>
<tbody>
<tr>
<td>155.0</td>
<td>12</td>
<td>7</td>
<td>481</td>
</tr>
<tr>
<td>155.0</td>
<td>12</td>
<td>8</td>
<td>224</td>
</tr>
</tbody>
</table>

Figure 2 Use of a Relation to hold multi-dimensional data

Then by using RAPPORT Statements of the form

SEARCH DATA (XBIN = 12)

or

SEARCH DATA (ENERGY = 155)

or

SEARCH DATA (ENERGY = 178 AND XBIN = 1)

you can pick out any two-dimensional or one-dimensional 'slice' of the data very conveniently (secondary indices can be defined to make sure these retrievals are done efficiently). This generalises quite simply to four-dimensional or higher-dimensional data. Thus Relational databases offer one of the most convenient ways of storing multidimensioned data (a problem which as plagued compilers of two-body scattering data using the hierarchical Berkeley Database Management System).

3. THE FORTRAN PROGRAM ANALYSER

The Fortran Program Analyser (FPA) is a tool developed by Logica to help in the understanding, documentation and maintenance of large Fortran programs. This problem is particularly relevant to experimental physics, where very large suites of analysis programs grow up through the years; they are often not perfectly documented because their authors are more interested in the physics than in the programs; and authors move on, leaving others to modify and adapt the programs. So one is often faced with the problem of extending or debugging a large, poorly documented program after a short acquaintance.

The Fortran Program Analyser is designed to answer some of those questions about a large program which are not easily answered by looking at the source listings - questions such as 'Where else is this subroutine called from?' or 'Where else is this variable used?' or 'What does this subroutine do with its second argument?'. And it is designed to tell you what actually happens, not what the author once thought should happen.

Several other tools have been produced in the past with this aim in mind tools such as DAVE and FACES and compiler cross-reference listings. However, they have all in practice had one major failing. They tend to produce voluminous printed output, rather thicker than the original program listing,
which, being of an unfamiliar layout, sits on a shelf and gathers dust. Therefore these earlier tools have not had much practical impact where they were tried. So for the FPA we took a different approach; the result of the program analysis is not a fat printout, but a RAPPORT database. The user can then select from the database just the information which interests him, and not be overwhelmed by volume of output. He can choose both the scope of the reports (i.e. which parts of the analysed program they refer to) and the level of detail; he can display them directly on his VDU or spool them to a printer.

The first piece of 'global' information which is hard to extract from program listings is the subroutine and function calling structure - which routines call which others. This can be imagined as a tree-like structure (Fig.3) but the full structure is often much more than you actually need or want. The FPA allows you to answer directly the questions which are most often of interest, i.e.

(1) Which subprograms does subroutine X call (directly or indirectly)?
(2) Which subprograms is subroutine X called by (directly or indirectly)?

Both of these questions can be further constrained by restructuring the depth (in the calling tree structure) to which the search is taken.

![Diagram of subroutine calling tree]

**Figure 3** A subprogram calling tree

A sample of output in answer to question 2 is shown in Figure 4.

One is often faced with the problem of tracing the history and meaning of a local variable (say ILOCAL) in some subroutine and then coming across a statement of the form:

```
CALL OBSCURE1 (XXQYY),JSQUIRT,XZBB,ILocal,7)
```

What does OBSCURE1 do to its fourth argument? On going to look at OBSCURE1 you find:

```
JSQUIRT = OPAQUE(2*XZBB, ILOCAL)
```

and having established that OPAQUE is a function, not an array, you have nagging doubts that it might just modify its arguments.

The FPA has facilities to answer all these questions accurately and immediately, following subroutine arguments down through as many levels of calling as is necessary. For any subroutine or function argument, the FPA can determine whether it is used internally by the subroutine, modified by
the subroutine, both or neither. More detailed output can show precisely how and where the argument is used or modified.

<table>
<thead>
<tr>
<th>AT LEVEL</th>
<th>SUBPROGRAM</th>
<th>CHGET</th>
<th>CALLED BY</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>SKPSTR (LINE 15)</td>
<td>LEVEL -2</td>
<td>GETCH (LINE 17)</td>
</tr>
<tr>
<td></td>
<td>SKPSTR (LINE 19)</td>
<td>LEVEL -2</td>
<td>GETINT (LINE 20)</td>
</tr>
<tr>
<td></td>
<td>PPSTBF (LINE 24)</td>
<td>LEVEL -2</td>
<td>GETNM (LINE 24)</td>
</tr>
<tr>
<td></td>
<td>SKPSTR (LINE 24)</td>
<td>LEVEL -2</td>
<td>PPROLL (LINE 25)</td>
</tr>
<tr>
<td></td>
<td>GETINT (LINE 25)</td>
<td>LEVEL -2</td>
<td>GETNM (LINE 30)</td>
</tr>
<tr>
<td></td>
<td>VERS (LINE 35)</td>
<td>LEVEL -2</td>
<td>PPROLL (LINE 36)</td>
</tr>
<tr>
<td></td>
<td>PPSTBF (LINE 43)</td>
<td>LEVEL -2</td>
<td>PPSTBF (LINE 51)</td>
</tr>
<tr>
<td></td>
<td>GETNM (LINE 52)</td>
<td>LEVEL -2</td>
<td>SKIPTO (LINE 58)</td>
</tr>
<tr>
<td></td>
<td>FINDKW (LINE 59)</td>
<td>LEVEL -2</td>
<td>GEXPNM (LINE 67)</td>
</tr>
<tr>
<td></td>
<td>FINDKW (LINE 73)</td>
<td>LEVEL -2</td>
<td>GEXPNM (LINE 84)</td>
</tr>
<tr>
<td></td>
<td>NXTSTM (LINE 87)</td>
<td>LEVEL -2</td>
<td>GEXPNM (LINE 119)</td>
</tr>
<tr>
<td></td>
<td>DXFRS (LINE 119)</td>
<td>LEVEL -2</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4 A listing showing all calls of a subroutine

Finally one of the most difficult questions to resolve by examining program source is the usage of COMMON data. If COMMONs are used in even a slightly undisciplined manner, they can be a major barrier to understanding a large program, and a persistent source of bugs. Again, the FPA has features specifically designed to answer the most useful questions. You can ask for a report on the usage of a single COMMON variable, or a whole COMMON block or several - throughout the whole program, or in a particular subset of it. This subset of the program could be an explicit list of subprograms which you supply, or it could be defined by the calling structure - 'subroutine X and everything it calls'. So the FPA can be used to answer very specific questions such as 'Is COMMON variable A modified when I call subroutine Z?' or very general ones.

The user's interface to the FPA is through the RAPPORT IQL. So, even though the basic commands are highly flexible, you can use the macro-defining capabilities of the IQL to string together the combinations of commands which you use most frequently for extra convenience. Or, if you are familiar with the structure of the underlying database, you can use the power of the IQL to ask an unlimited range of questions about it. This gives a highly open-ended and flexible tool whose output can be tailored to very diverse needs, and which can provide a most useful 'assistant' in understanding, documenting and modifying large programs.

4. THE TEST COVERAGE MONITOR

The Test Coverage Monitor (TCM) is a tool to measure how well your programs have been tested. It can also provide performance-related data, showing how many times different sections of code are executed. Again, these
data are not presented as a fat lineprinter listing, but are stored in a RAPPORT database which you can interrogate interactively, selecting just the data which interest you.

Aside from work on formal 'program proving', which is too far from practical application to be of interest to practising scientists, work on program testing has concentrated on two main approaches - functional, or 'Black Box' testing and structural, or 'Glass Box' testing. 'Black Box' testing ignores the internal structure of the program under test and concentrates on its external behaviour; the idea is to create(somehow) a set of test cases which contain all the interestingly different sets of circumstances which the program has to deal with - special cases, boundary values and so on. (In practise any program has to deal with an essentially infinite set of circumstances which can never all be tested; but methods have been developed for defining the 'interesting' cases). 'Glass Box' testing concentrates on the internal structure of the program under test; here the intention is to generate(somehow) a set of test cases which exercises all the different interesting paths through the program. Again, the set of different paths through a large program is essentially infinite, and the trick is to find an 'interesting' finite subset. Various criteria have been proposed - notably statement coverage (that every statement in the program has been exercised at least once) and, more stringently, branch coverage - that every branch in the program (IF, GOTO or DO) has been exercised for every outcome of its decision. The TCM can be used to evaluate the level of statement coverage or branch coverage provided by a set of tests, and to highlight parts of a program not yet tested.

In practise, program testing is a very pragmatic art of the possible within limited resources. Where resources are available, some sort of combined approach (using both Black Box and Glass Box methods) seems to work best. More frequently, we know that program testing is often glossed over when challenged, the program user is forced back to the position 'Will I wrote it quite carefully and it seems to work' or 'It's been working for a long time'. Given the very large costs of High Energy experiments, and the dependance of their results on a few large anlaysis programs, this is not a happy situation. How much time do experimentalists spend in testing their analysis software, compared to the time they spend testing the apparatus?

The Test Coverage Monitor offers a way to improve the software testing process, however informally it is approached, by showing which parts of a program have not yet been tested. With this information, further tests can be designed to improve the coverage of testing. The operation of the TCM is shown in figure 5. First the source code of the program under test is passed through an instrumentation program, which inserts calls to data collection routines at every instrumentation point in the program. (The instrumentation program also makes some entries in a RAPPORT database which will be used to store test coverage data). The choice of level of instru-
mentation is left to the user, but typically there will be instrumentation points on entry and exit from subprograms, and at every branch of control (IF, GOTO, DO...) within a subprogram. Certain subprograms can be instrumented selectively. The instrumented program is then compiled and linked together with the data collection routines. As the program is run on test data, the instrumentation data collection routines communicate by inter-process message passing with a RAPPORT nucleus which updates the test coverage database. (Special Buffering mechanisms ensure that there is not a message sent every time an instrumentation point is executed. The run-time overhead of instrumentation is typically only 25% in execution speed).

After a test run, or after several test runs, the test coverage database can be examined using an enquiry and reporting program, either to answer specific questions interactively, or to produce reports offline. These reports can be very brief summaries (overall level of coverage achieved, as a percentage), or exception reports (highlighting only those parts of the program which were not exercised by the tests), or detailed listings showing the number of times each statement was executed. A typical sample of output, involving some rather perverse 'double IF' statements in the program under test, is shown in Figure 6. In the example shown, the tests achieved rather high coverage (95% overall). In more typical cases, coverage is much more patchy, and the monitor reveals whole areas of the program which have not been tested at all. So the TCM gives a very direct way of showing where more testing is needed, or, alternatively, showing which parts of a program are executed most frequently and need to be optimised.

![Figure 5 Operation of the Test Coverage Monitor](image-url)
5. THE STATIC ANALYSIS PACKAGE

Logica are currently performing a contract for the European Space Agency to develop a Static Analysis Package (SAP), which is essentially a refinement and further development of the Fortran Program Analyser and the Test Coverage Monitor. The SAP will combine the functions of these two in one package, and combine their results in one database, while incorporating a number of refinements in each function (based on experience in their use) and a number of new functions.

The new functions are largely in the area of checking program quality, by looking for dubious constructs such as jumps into DO loops, variables used before being set, unreachable code and infinite loops. It can also check that COMMON block declarations match in different subprograms, and calculate measures of the complexity of subprograms.

The SAP will be delivered to ESA early next year, and will be used as part of a wide-ranging Software Quality Assurance program, both for software developed by contractors and software developed in-house.

6. THE UNIX TOOLSET

Most people think of UNIX as an operating system: fewer know that it is in fact much more than that - it is an operating system with a very comprehensive set of software tools. These tools are very useful for developing any kind of software, including scientific software. Logica has considerable experience of these tools as we have not only been users of them on many different projects, but we are also the distributors and support centre in the UK for XENIX, which is Microsoft's version of UNIX on PDP-11s and a wide range of microcomputers. I shall not attempt to describe the whole range of UNIX tools, merely to give a brief description of two tools, MAKE and SCCS,
which are particularly useful for large scientific programs.

MAKE is a program which understands how software systems are construct-
ed and ensures that a system is kept up to date with respect to its compo-
nent parts. The basis for this is a file (a MAKEfile) set up by the user, 
which describes the dependencies amongst components of a software system. 
When one component is amended, MAKE ensures that the change is reflected 
throughout the dependent modules of the system. As an example, consider the 
case shown in Figure 7, where a program is constructed by compiling and link-
ing three separate modules. By looking at the creation dates of all the 
files in the system, MAKE can check that the target program is up-to-date 
and consistent - and if it is not, do just those things which are necessary 
to make it so. For instance, if module a has been edited and recompiled 
since 'prog' was built, MAKE will re-perform the link-loading. If a has only 
been edited, MAKE will recompile it and then do the link-loading. In this 
way a lot of tedious routine tasks in building large programs can be done 
automatically, efficiently and accurately. Furthermore:
- MAKE does not do the same job a command file would do (i.e. recompile 
  the log), that may take hours of machine time;
- any programmer can remember how to build a program out of three or four 
  source files - very few can remember how to build it from twenty or 
  thirty, particularly when a few programs may share several common mo-
  dules;
- the perennial problem of discovering that you are using an old version 
  of the software for testing because you remembered to compile the pro-
  gram with the newly fixed bug, but forgot to link load it - simply can-
  not occur with MAKE;
- consider how often you have had to try and figure out how your programm-
  er builds his software (because he is on holiday and you just have to do 
  it). XENIX programmers do not - all they do is type MAKE.

The Source Code Control System (SCCS) performs a function which I believe 
is similar to that of the PATCHY utility at CERN; it allows you to keep track 
of multiple successive versions and variants of the same piece of source 
code, to reconstitute any one version at will, and to update them all simulta-
naneously. It does this by storing the code as an initial version with a 
succession of changes, and re-computing all the changes any time you want to 
list, or compile, a particular version. An example of its use is shown in 
Figure 8.

![Figure 7. MAKE in action](image-url)
In this example, a product is taken from initial development to Phase 2 development, while the first production release must be retained for support to the production version. As the production version is modified (to correct bugs or alter its functions), separate versions are produced. SCCS can keep track of all these versions at one time, using essentially one source file.

7. THE FUTURE

I have talked about a number of different software tools, each useful in its own way, but I have discussed each one in isolation. No doubt these tools will continue to improve and other new tools will be developed; perhaps we will even start using other languages for scientific programming. However, I sincerely hope that those are not the only developments to take place over the next few years. Real progress in developing large suites of scientific programs, reliably and cost-effectively, can in the long-term only come from a much more radical step. This is the **Integrated Software Engineering Support System**, where a wide range of software tools are brought together to form a coherent whole. Essentially the different tools should cooperate (passing data easily from one to another using a common database having consistent user interfaces, and so on) to make the programmer's job much easier and perhaps to support a particular software development methodology.

UNIX points the way here; its strength lies not just in the individual tools, but in the very flexible ways in which they can be combined together (using Shell commands, pipes and so on) to give a system tailored to the individual project. However, this is only one part of the problem. Integrated Software development environments have been successfully demonstrated in commercial data processing, in systems such as Cullinane's integrated data dictionary or Cincom's Mantis. These can produce commercial DP systems for more rapidly than conventional COBOL methods. However, their success is due partly to the rather simple and stereotyped nature of the actual computation of most commercial applications (and their concentration on user interaction and data structure). For scientific or real-time computing, the computation is much less trivial, and the Integrated Software Engineering Environment is much further away. There are several research efforts currently planned or under way in this direction; let us hope they succeed.

* * *
MR. M. TURNILL BNOC GLASGOW

• Q ---› Our experience is that the JOIN command is an essential feature of a relational database system.

• A ---› I do not agree.

MR. M. TURNILL BNOC GLASGOW

• Q ---› Can RAPPORT compute across tables, since this is an important requirement for High Energy Physics?

• A ---› No.

MR. W. MITAROFF INST. F. HOCHENERGIEPHYSIK VIENNA

• Q ---› Concerning the FPA and TCM, they need necessarily all the source code, but that is normally not what you have. There can be calls to Assembler and/or library routines (hopefully with well defined and documented interfaces). Can you pass those informations to make FPA/TCM work correctly?

• A ---› There is a "library" facility which allows you to describe explicitly the effects of routines which are not analysed by the FPA, and to include them in the FPA database.

MR. W. MITAROFF INST. F. HOCHENERGIEPHYSIK VIENNA

• Q ---› I doubt whether FPA/TCM can handle such programming practices as passing the addresses of routines for error recovery, or handling dynamical link-structures like in Hydra?

• A ---› No, the FPA cannot in principle handle dynamic linkage, as it only analyses the static (non-running) program.

MR. R. PIKE - BELL TELEPHONE LAB. MURRAY HILL, N.J.

• COMMENT ---› I have three comments.
  - First, inside Bell Labs we have versions of very similar tools (beside MAKE and SCCS) which run under UNIX.
  - Second, MAKE can be used for many things other than maintaining programs. For example, we use MAKE to maintain data bases and CAD data.
  - Third, SCCS is too large and clumsy to bother using for programs which are continuously evolving. It is really only suitable for maintaining software with many versions which must be maintained in parallel.
MR. J. HUTTON RUTHERFORD APPLETON LAB.

- Q ——> What is the price for the various packages?

- A ——> Rapport costs about 30000 pounds for its full IBM implementation and a limited IBM implementation with academic discount can be obtained for $12000. FPA and TCM are at about 15000 pounds each. (***** these answers have not been checked by the speaker)
DATABASE APPLICATIONS IN HIGH ENERGY PHYSICS

Dr K G Jeffery

Rutherford Appleton Laboratory, Chilton, Didcot, Oxon, OX11 OQX

ABSTRACT

High Energy physicists were using computers to process and store their data early in the history of computing. They addressed problems of memory management, job control, job generation, data standards, file conventions, multiple simultaneous usage, tape file handling and data management earlier than, or at the same time as, the manufacturers of computing equipment. The HEP community have their own suites of programs for these functions, and are now turning their attention to the possibility of replacing some of the functional components of their 'homebrew' systems with more widely used software and/or hardware. High on the 'shopping list' for replacement is data management. ECFA Working Group 11 has been working on this problem. This paper reviews the characteristics of existing HEP systems and existing database systems and discusses the way forward.

1. INTRODUCTION

This paper attempts to survey the existing state of High Energy Physics programming related to handling data. It then surveys the existing data handling technologies and tries to draw some conclusions about a possible route forward for High Energy Physics. The basic argument is that there are advantages to the High Energy Physics community in adopting certain standard pieces of software (or hardware) where this will reduce the cost of development, or maintenance, of a given facility to the High Energy Physics community.

2. THE EXISTING SITUATION

High Energy physicists rapidly learnt to exploit computers for their data processing. In many ways the scientists were in advance of the computer manufacturers in providing facilities and way ahead of the introduction of computer science type concepts. Thus High Energy physicists addressed many of the problems which are now the subject of learned dissertations by computer scientists. To give some examples these included memory management, tape volume management, job generation, dataset management, file conventions, data standards and the simultaneous update problem. For example, BOS developed at DESY handles memory management (as do other HEP systems) while tape volume management is handled in the lower levels of systems such as TRIAB, and in the facilities of the TASSO run library. Most of the 'book-keeping' systems (such as TRIAB or the TASSO run library) provide job generation facilities. Dataset management, file conventions and data standards all tend to be experiment or collaboration specific although some conventions seem to run through from one major software system to another depending largely on the personalities involved. The simultaneous update problem has been addressed by the 'book-keeping' systems. A rather newer initiative in the use of pre-existing software has been taken by the team working on the Rutherford multi particle spectrometer (RMS). They
analysed their requirements and decided that their basic system reduced into the following components:

i) data reduction (RSG)
ii) analysis (ART)
iii) utilities.

The resulting system allows a user, by simple and easy commands, to control the processing of his HEP data. RSG uses the HYDRA memory management system and the job submission component of RSG relies heavily on EXECs (macros) to gather the information and provide power to the user. RSG relies on components of HYDRA, HBOOK, HPLOT, SMOG, TRIAB and SUMMARY. ART provides an easier interface to these types of facilities. To summarise, High Energy physicists have addressed in detail many problems now regarded as the province of computer science. These are being progressively disguised from the user by the use of higher level processing stream systems such as RMS.

Some components of the systems may be replaceable with other components supplied from outside the HEP community thus gaining advantages of external support and development, a wider utilisation (therefore reduced cost) and maybe even some new ideas and facilities. Some of the problems of memory management could be handled by virtual memory and paging systems. Most of the problems of tape volume management can be addressed with good cataloguing and tape management system of facilities and, as demonstrated in RMS, job generation can be improved considerably by the use of better JCL or macro facilities such as CMS EXEC. Dataset management, file conventions, data standards and simultaneous updates all fall within the province generally regarded as database systems.

3. DATABASE SYSTEMS

A database is an integrated collection of data and in general use of the term database implies computer based systems to handle the data. The initial reasons for applying database technology in data processing were to overcome problems of duplication of data with attendant update problems and control of access to the data. However, spin off advantages include better structuring of programs, better data standards and control, simplicity of system maintenance and ease of system development.

Terms commonly used include DBMS (Database Management System), DBS (Database System), and IS (Information System). An information system will almost certainly include one or more database systems and a database system will have as its core a database management system. In brief, a database system is a normal data processing system with a DBMS within it to manage the data. An information system is one or more database systems front-ended suitably to provide the relevant information at the right place at an appropriate time. Any of the systems will have its own architecture but a general pattern can be imagined consisting of a user who uses the system through a user interface and the system uses the data through a data interface. The form of the user interface and the data interface are critical in system design because they control the level of facilities offered.
The data interface may be characterised by three major criteria. The ability to handle data content, the ability to handle data structure and the ability to handle data independence. Data content is concerned with the data type, its length, its name, and any other pertinent characteristics such as precision, synonyms, range of validation and the like. Data structure concerns the inter-relationship of data items. The main structures are linear, hierarchic, network and plex, the latter being combined network and/or hierarchic structures. Data independence depends on the degree of change to be made to programs or to system procedures for a given change in the data content or structure. It is critically dependent upon the way in which the system binds data to the programs. Most systems require binding at pre-compile time, i.e. any change in the data needs re-precompilation and recompilation of all the software. Some systems allow binding at compile time so one only has to recompile the software and not run through pre-compilers. Very few systems, but those with maximum independence, have execute time binding i.e. the programs will work on any data and the binding between the data and the programs is achieved when the program is actually running. This implies the use of dynamic file assignment, dynamic connection of files to software and within the software dynamic assignment of data items to positions in the data buffers with full type independence.

The user interface may be considered in terms of the user it is serving. There are basically three types of user, the non programmer user, the programmer user and the manager of the system. The non-programmer user expects an interface so that he can use the facilities provided. It is likely that a query language, or in a system with more facilities a language to provide those facilities to the user, will be expected. A fully functional language at a higher level than existing conventional high level programming languages such as COBOL or FORTRAN is a great advantage in building a sensible user interface for the non-programmer user. The interface can be regarded as procedural or non-procedural. Procedural indicates that the statements that the user inputs to the system have a procedural flow such as that in a normal programming language. A non-procedural interface allows for the user to express his ideas which are then interpreted into a procedural language for actual computer processing.

The programmer user will wish to access across the user interface into the system facilities by use of procedure calls from his conventional programming language, particularly COBOL or FORTRAN. The programmer user will have to be aware of the data structure and content of the database unless it is fully independent with execute time binding because in all other cases he has to navigate through the database by use of set methods to achieve linkage from one data item or group of data items to the next, typical procedural calls of the form "get next record in this set" or "get any record related to this record on key value". The programmer user also has to take account of control mechanisms particularly with regard to avoiding simultaneous update by incorporating procedural calls to allow for locking and unlocking of the relevant area of the database.
The manager of the system requires facilities to set up databases to monitor their use and when necessary to control their use in terms of resources privacy and security. These will may be regarded as additions to a functional procedural or non-procedural language for a non-programmer user. However it is at this level that interfaces to operating systems are most obvious.

Database systems can thus be classified depending on the facilities offered at the data interface and at the user interface. The optimal system is one which handles the maximum range of data structures and the maximum range of user interfaces coincident with the user requirement. The well known IBM product IMS for example handles only hierarchic data structures and has a programmer level interface with procedure calls from existing computing languages. However more recent offerings from IBM allow pseudo-network data processing facilities so increasing the scope in terms of the data interface and also allow the use of other language interfaces for programmers and non-programmers including easy to use query languages such as SQL. TOTAL from Cincom handles a limited network structure or a hierarchic structure and has a programmer level interface to existing conventional high level languages. Recent product offerings from Cincom include user friendly interfaces for the non-programmer user. IDMS handles full network facilities and is written to the CODASYL specifications. It has a programmer level interface to allow use from existing high level languages and also more recently has its own user friendly query language. The relational systems handle (to a greater or lesser extent) any data structure. RAPPORT has a programmer level interface and a limited query language for use by non-programmers. INFO has an interface for non-programmer users, it is an easy to use command language, but also has its own programming language. It does not at the moment have any programmer-level interface in terms of an existing high level language although one is planned. G-EXEC and related systems developed in the Research Council community in the UK has interfaces at both these levels and also features execute time binding of data to program code. INGRES has a good implementation of SEQUEL, the rapidly becoming standard relational query language.

4. HOW TO PROCEED

In the rapidly evolving scene of database technology there are as many project management techniques as there are database teams (or even individual consultants). This situation is beneficial in encouraging research and development in this new technology, but leaves the customer (the user of the product from the database team) with a bewildering set of ideas which are changing as the project progresses. During the past two years the SERC RAL Database Section has evolved a style of project management which is both flexible enough to allow the use of the latest technological developments in the science, but is rigid enough that the customer can track the progress of the project and can be involved in the conception and development of the system he eventually uses in a production environment.
The philosophy behind this method for project management is as follows:

(a) changes made to a production system are very expensive and cause a great deal of disruption, therefore good design is essential;

(b) cost of development of software is high, therefore there should be the minimum number of changes during the development phase. Again, investment in the design phases is maximised;

(c) the overall structure of the phases is to ensure that the effort involved in the earlier phases, particularly 1, 2 and 3, is not wasted whatever happens in development and production, and the effort in development (phase 4) is not wasted whatever the target machine and machine systems proves to be.

The result of this philosophy is, hopefully, professionally managed projects producing systems which are machine independent, operating system independent and, to a large extent, independent of database management system or tp system.

The method can be tabulated simply as a set of phases and a set of techniques applied at each phase.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Techniques</th>
</tr>
</thead>
</table>
| 1. Preliminary (overview) analysis | User requirement interview  
Data Analysis  
Procedural Analysis  
Sizing  
Feasibility of certain techniques  
Report |
| 2. Detailed analysis        | User requirement interview  
Data Analysis  
Procedural Analysis  
Design (including privacy, security and audit requirements)  
Sizing  
Report |
| 3. Prototype                 | Sizing check  
Procedural check  
Data Analysis check  
Feasibility check |
4. Development

- Software engineering
- Defined interfaces
- Use of pre-existing software units
- Machine/system independence
- Choice of target production machine
- Interface software for machine-dependent environment
- Documentation
- Training
- Parallel running and handover

5. Production

- Maintenance
- Minor development (including training)
- Change control
- Monitoring

6. Evaluation

- Report

The objective of the preliminary analysis phase is to ascertain the basic parameters involved in the project. The preliminary analysis is done as a response to a request from a potential user. This analysis should normally take a relatively short time, and certainly no more than 5% of the total project effort. The objective is to discover whether Database techniques can be applied in a cost effective manner. At the end of this phase the report should lay out the conditions under which one can proceed. This includes a statement of the user's willingness to "pay" for the project.

The detailed analysis phase is where the full expertise of a database team is required. An in-depth appraisal is made of the user requirement and using modern techniques of data analysis and procedural analysis a design is constructed based upon those user requirements. At this stage detailed sizing can be done and the report is provided to act as the basic estimate for the amount of resources needed to complete the project.

The prototype phase is used to ensure that the user requirement has been met by the design. The prototype is run on any convenient system and is evaluated both by the members of the project team and the user or users. In particular, the prototype checks that the procedures that the user wishes to use are available, that all the linkages between the data items that are required are there, and that the sizing done in the previous phases is realistic. These things are combined to ensure that the target system is suitable for the running of a production system.
In the development phase conventional systems analysis and programming techniques are used. The development phase for a large system will be the longest phase in the overall project. The techniques used are those of modern software engineering. The aim is to develop a machine and system independent database or information system working from the information gained in the previous 3 phases and leading to eventual implementation in a production environment. In this way the man effort of the first 3 phases is protected: it is not committed to any one particular machine or system. In the development phase, by maintaining clear interfaces, it is possible that the choice of database management system, tp system, operating system and computer hardware can be left until the latest possible time. In parallel with the development phase training and documentation takes place and customer satisfaction is assured by a period of parallel running before handover.

The production phase is when capital invested in the project is repaid. The database team will be involved in maintenance and minor developments to meet changing user requirements; hopefully these developments will be minimised because of the amount of design effort in phases 1, 2 and 3. Introduction of new or changed software will be subjected to rigorous change control to ensure that the production system is maintained, and the overall operation of the system is continuously monitored for efficiency.

The techniques used within any database team through the different phases of projects are all more fully described elsewhere. They are included here in brief for completeness.

The purpose of the user requirement interview is to obtain information on (a) the data currently being used and additional data that the user would like to see included, (b) the procedures used, and any additional procedures the user would like to see, (c) the number of data records, the number of items that are recorded and the size of the items, (d) the number and frequency of the operation of the procedures, for example, the number of times that an item may be requested from a stores, and to integrate this information into a summary by user group within the project. The final result of all the user requirement interviews is an integrated summary of the user requirement, normally structured in 2 levels, the highest level being an overview of the requirement across all user groups and the lower level being the detail by individual user groups.

The data analysis consists of 2 techniques, namely entity analysis and normalisation. The purpose of data analysis is to ensure that the eventual system design encompasses all the data items required, and that the inter-relationships between them, both existing and possible, are preserved. Entity analysis is a top-down technique. The starting point is some thinking about the nature of the data to be handled, and by progressive decomposition of the entities one reaches the entity map which shows the inter-relationships between the different entities or objects manipulated by the procedures. Normalisation is a technique starting from the bottom and working up, and so complements entity analysis. The starting point is all the documents, both for output and for input, from any pre-existing manual or
computer system. By a series of formal procedures the unstructured string of item names
known as the unnormalised relation is progressively moved through first, second and third
normal forms to a structure which can be represented by a relations map. The reduction
from an unnormalised relation to first normal form involves the elimination of repeating
groups. The normalisation from first normal form to second normal form involves the
elimination of functional dependencies and the transition from second normal form to third
normal form involves the elimination of transitive dependencies.

Procedural analysis starts with the information obtained in the user interviews. The
major functions from the integrated summary of the user interviews are tabulated, and then
taking the detail of the user interviews events are noted. Between the events certain
processes happen; these are things normally triggered by the events. Each of these
processes belongs to one or more of the major functions, and so the combination of listing
the processes that depend on an event and the processes that can be decomposed from the
major functions provides a list of the necessary processes in a procedural analysis. By
adding in the entities derived from a data analysis the result is a data flow diagram.

The information for sizing is normally obtained from the user requirement interviews
and from the data analysis working from the documents. The information needed is the
number of bytes/field, the number of fields/record giving a record length, the number of
records, which normally (incidentally) relate to the entities, the rate of increase of
number of records, the number of changes made to items in the records and the frequency
of those changes and the number of procedures and their complexity which may give an
estimate of the number of lines of code to be written in the system.

If a particular design looks as if it will be necessary due to some particular
constraint obtained from the user requirement interviews, and if this particular design
requires some new technology, at this stage this new technology should be tested. An
example would be the linking together of two different database management systems or the
use of a new operating system. It is much better to test the feasibility of such a
facility at this stage in the project rather than to discover half way through the
development that there is a problem which may involve recoding.

The design process takes the procedural analysis down to functional units and
condenses them to modules from the different functions and procedures. The data analysis
from third normal form provides the entities. The two are combined together, the sizing
information is added in to avoid any bottle-necks or difficulties in the system and
finally any restrictions on the system are applied, for example, considerations of
security, privacy or audit. The result is a design specification normally produced in
diagrammic form.
Starting from the design and the prototype experience then a hardware specification considers the characteristics of the target production system to ensure that it can handle the requirement in terms of size, speed and telecommunication links, security etc.

The basic principles of software engineering involve the concept that one module is equivalent to one low-level function in the system. A hierarchy of modules is produced which models the structure of the functions and sub-functions as defined in the procedural analysis. A standard method of inter-module information transfer and control should be provided.

Defined interfaces should be provided to any database management system and, if necessary, tp system, and also between the modules.

Pre-existing software units should be used (a) where possible, and (b) where it is consistent with the overall design.

The target production machine is chosen based on the design and prototype experience. In the case of a new machine then there is a standard sequence of operational requirement, tenders and bench-marking that needs to be followed. Where it is not a new machine the target production machine should be chosen only after it has shown to have satisfied specification of the hardware system requirements.

The software is written to interface the database system as developed to meet the procedural and data analysis to the target hardware and software systems supplied, which may include the database management system and tp system.

Documentation should be provided for the users, operators and the systems support staff. Training should be provided for the users, the operators and the systems support staff. The developed system should be run in parallel against any pre-existing manual or computer system. During this time the necessary plans for conversion must be formalised. During the parallel running the output should be cross-checked and the contents of the datastore cross-checked to ensure consistency. Maintenance for a production system involves the repairing of any errors that have been discovered. This is the development of the system to meet any changing user requirements as defined and agreed between the Project Team and the users. Tuning developments are normally instigated by the Project Team when they notice some way of improving the performance of the system. Change control is a procedure for ensuring that there is not random updating of the software. It normally involves the concept of a Chief Programmer and a Program Librarian and all changes being monitored, tested in a test environment first, and then released to the production environment after thorough documentation. This is the process of watching the production system to see if any improvements can be made.
5. **Desirable Features of an HEP System Based on Database Techniques**

This is a guess at the desirable features without doing a full project following the methods outlined previously.

The system should be machine independent. High Energy Physicists are not restricted to one machine range necessarily and they may wish to split the components of a total High Energy Physics system over several different computers. The system must be modular and yet integrated, modular to allow it to be distributed across different computers and integrated to allow ease of passing of data from one process to the next. It must have interfaces for the non-programmer and the programmer so that people can use the system at different levels depending on their particular requirements. It must be easy to use and yet rich in facilities, this implies a high power ratio between what the user types in and what he can get out of the system. The system will need to run both interactively and in batch mode and will probably also require job generation facilities for long runs to process masses of data. The system will need maximum flexibility in data independence and will need to handle a wide range of different data structures because of the nature of different HEP experiments. The system must be supported yet must be able to be developed either by or in conjunction with the HEP user community. The system must be so organised that it can be run either in a centralised mode or as a distributed system; this has large implications on the system design. Finally the system must be capable of progressive replacement of software components by hardware components as hardware database engines of various types become progressively available.

* * *

**Questions**

**Mr. R. Zelazny Inst. of Nuclear Research, Otwock-Swierk, Poland**

- Q ---› Do you know about a real implementation of DBMS in a real experiment data acquisition and processing (not necessarily HEP)?

  - A ---› No.

**Mr. P. Van Binst Universite Libre De Bruxelles**

- Q ---› What are the drawbacks of relational data bases ?

  - A ---› The evaluation of many systems indicates that relational data bases are, on average, better or not worse than others, except in the case where data and data structures are little or not at all modified during the transactions.
MR. M. JACKSON SYSTEM LTD LONDON

- Q ---→ Should relational data bases not be regarded as standing to other data views much as binary arithmetic stands to other arithmetics?

- A ---→ Not in my view.

MR. R. WORDEN LOGICA LTD LONDON

- Q ---→ Was R. Bock’s bad experience one with a relational data base system? If not, would he be prepared to have a go with a relational system?

- A ---→ (Mr. R. Bock CERN) Yes, it was relational, but none of those mentioned today.
M.D. Canon  
IBM  
San Jose Research Lab.

Michael D. Canon researches in storage systems and  
technology at the IBM Research Laboratory, San Jose,  
California.

Magnetic recording technology is the basis of virtually all storage  
products currently used in data processing applications. This tutorial talk  
begins with a discussion of state-of-the-art rigid disk, floppy disk, and  
tape, all of which make use of magnetic recording techniques. We then comment  
on the technical challenges which must be met in order to extend the technol-
ogy during the 80's. The role of alternative storage technologies such as  
magnetic bubbles and optical storage will be discussed.

During the 80's changes will be required in computer system struc-
ture to accommodate the more central role to be played by future hardware and  
software storage products. As time permits, we will describe alternative  
storage system architecture which facilitate sharing of storage devices among  
multiple processors, physical space management, hardware redundancy for  
improved reliability and availability, simplified I/O programming, and better  
performance.

***

(Paper not received)
SCIENTIFIC COMPUTING IN METEOROLOGY

Dave M. Burr ridge
Head of Research
European Centre for Medium Range Weather Forecasts
Berkshire, England

David Burr ridge is trained as a mathematician and heads the Research Department of the European Centre for Medium Range Weather Forecasts. He is responsible for the development of operational forecasting models and data analysis procedures. The ECMWF computer centre owns a CRAY-1 and modern graphics facilities.

Numerical forecasts of the weather for 10 days ahead require a global analysis of the state of the earth's atmosphere and the solution of the complex time-dependent three dimensional partial differential equation that describes its evolution. The programming approach, taken at ECMWF, to the large-scale computational work required will be discussed. In our approach, important constraints have been the requirements for portability/transportability and the rapid turnover of scientific staff; scientists at ECMWF are usually employed on short-term secondments. This poses special problems, particularly in the areas of code management and documentation.

* * *

QUESTIONS

MR. R. BOCK CERN

- COMMENT --- > The example of variable naming conventions in DOCTOR seems to show that some of the freedom of naming conventions in programming languages is felt to become a nuisance in very large programs. HEP has clearly had the same experience although no solution was ever generally enforced.

MR. D. MYERS CERN

- Q --- > The ECMWF enforces a programming style. Is there available a formatting program for source code?

- A --- > No.

(Paper not received)
SOFTWARE IN THE OIL INDUSTRY

M C Turnill
Computer Services Department, Britoil, Glasgow, UK

Abstract

The paper reviews the challenges of today's oil industry which is dominated in Europe by offshore production. Some of the key computer applications are examined, discussing new software development methods which have been adopted in order to achieve significant reduction in development times. The range of modern software development tools is considered, with the decreasing impact of traditional programming languages such as COBOL and FORTRAN. The use and benefits of non-procedural languages are also discussed together with some views on their relevance to high energy physics. The paper concludes with a look into the not-too-distant future, stressing the need for new approaches to software development and improving the facilities for information handling.

1. Introduction

Computing technology now plays a vital role in the oil industry, after nearly a century of exploration and production using no more than slide rules.

The major impact has come in the last decade, following the necessity of developing offshore fields which require advanced engineering techniques and vast capital sums (over $2000m) for each platform of which nearly 50% is directly attributable to manpower costs.

The pressure of the large capital decisions, and the high daily revenue (often exceeding $5m) have led to the widespread use of computers in order to improve productivity and to maximise the revenue.

In addition to these pressures, the falling cost of hardware, and the high expectation levels of our young, computer trained engineers, means that the computing department within an Oil Company must respond by producing systems faster, make them more usable and more flexible than ever before.

This paper takes a look at the industry, the use of computer technology together with its challenges and the attempts being made to meet these challenges. The views set out below are those of the author and while they are inevitably flavoured by the activities in his own organisation, it is believed they also reflect the wider concerns of the industry.
THE OIL INDUSTRY AND COMPUTER TECHNOLOGY

Today computing technology affects the workings of nearly every department within an oil company and every stage in the life of a major off-shore project.

During the initial exploration stages, large quantities of data are collected during seismic surveys, and the analysis of each survey will require many hours of CPU time, often using array processing techniques. At the next stage more data is captured during test drilling in which measurements (logs) are taken from a variety of instruments lowered in the hole. The use of computer techniques to analyse these logs reduces decision times to a few days, and yields more accurate estimates of quality and quantity of oil in place.

During the planning and development of an off-shore platform computers are used to provide up-to-date network plans of the project and the inter-dependencies between the many phases. As the platform design progresses a whole variety of numerical simulations are carried out, for example, examining the stresses within the structure, and optimising the design of the processing plant. Finally during the production stage computers are used to provide a constant flow of management information on materials, costs and timescales to aid decision making.

Modelling the oil reservoirs' expected and actual performance is also a crucial application requiring large-scale computational tools which enables engineering managers to make decisions regarding the best development strategy, by optimum placement of each production or injection well. Each such decision will involve over $2 m. and ultimately affects the total recoverability possible from the field.

Thus the all pervasive effect of computer technology requires a strategic plan of the facilities which are needed to meet the users requirements, and these depend upon four vital ingredients -

- the people
- the environment in which they work
- the projects they undertake
- the technology available

which are examined in turn, looking at the challenges which each creates.

3 THE USERS

For the purposes of studying the types of computer systems we can divide users into three main categories:

- 'workers'
- technical decision makers (or 'professionals')
- strategic decision makers (senior management).
6.1 'Worker Systems'

6.1.1 Systems for Offshore Projects

The main problems encountered in the early platforms were often excess or shortage of material (through lack of timely information) or late material delivery. To attack these problems new systems have been developed which are based on interactive terminal systems to replace the earlier, ineffective, batch systems.

The systems adopted form a pattern of s:x feeder systems which exchange information with a central project co-ordination system (figure 1).

Each system is however self-contained and can operate in its own right. The terminals can be located anywhere with connections to the telephone system which allows siting not only in the company's HQ but also in construction sites, storage yards etc. Two of these systems are:

a. Steel Control System

A central feature is a 'material' database, from which regular, on-demand exception reporting can be obtained. Details of all orders are entered 'on-line' directly into the database so that the system operates in real time. Over 50,000 tons of steel are involved in a major platform. Each steel plate ordered is indexed by the tonne, grade and quality and in addition, has a unique serial number, with specification dimensions, with harth and cast numbers. This linking of plates to their related documentation is of vital importance to the systems success. The steel is shipped to a variety of storage yards in construction sites and the system is designed to:

- prevent delays in construction schedules due to material shortages on individual sites
- minimise surpluses and hence costs
- maximise the use of steel off-cuts
- audit deliveries

b. Control of Equipment and Material System (CEMS)

Purpose is to monitor all material (other than steel) as it passes through a complex manufacturing sequence, covering purchasing, expenditure, movement and audit trail. A crucial factor in this system is use of the Shell MESC
assignment codes which uniquely identifies over 50,000 different items of material. This enables coding procedures to be standardised, and allows enquiries which are simple to construct. The final result is ability to track material through the procurement cycle and to record its exact location at any point in time.

6.1.2 Platform Systems

Many platforms require nearly twice as many staff as the early estimates to operate successfully, because of:

- constant engineering improvements and maintenance
- complex operating set up

Serious efforts are now being made to reduce the operational staff by installing computerised monitoring equipment to allow centralised control of the platform. The techniques, ostensibly used in refineries for some years, have only recently become applicable to platforms since the advent of highly reliable micro-based instruments and very small but powerful computers. These have to operate in the cramped and complex environment of the platform top-side which is usually constructed off-shore in a series of box modules. Typically the computer systems (such as Ferranti Argus 700 or DEC PDP-11) will monitor many hundreds of flow meters, instruments, fire and gas alarms.

6.1.3 Use of Micro-Computers

Stand alone micros have also been used successfully offshore and in one example used to monitor the flow of oil through the 12 mile pipeline from the BEATRICE platform to the Nigg Terminal. The oil from Beatrice is very heavy and has a tendency to solidify at normal temperatures - it is therefore important to maintain sufficient pumping pressure at the platform to keep the oil moving. A micro was used to predict the pumping pressure during start-up of Beatrice and after any shutdowns. The advantage of the micro solution was its portability and independence from an unproven communications network at the time of platform commissioning.

6.2 Decision Support Systems

6.2.1 Reservoir Simulation

North Sea reservoirs have proved to be far more complex in their geological structure than the on-shore fields in the U.S. and the Middle East. This
has made their development (from a fixed location) extremely difficult and requires an accurate numerical model in order to attempt predictions of oil recovery, and the placement of each production or injection well. One of the more successful models is a three dimensional, three phase (gas, oil and water) fully implicit model called PORES, which uses finite difference methods to solve the partial differential equations describing the fluid flow in the reservoir.

The size of the model depends upon the number of active blocks in the reservoir, (inactive blocks being eliminated to save space). Typical models are 2-3,000 blocks, while large reservoirs such as Ninian require 8,000 blocks or more. (Such a block size still implies a mesh size of 1/2 km!). For the larger models it is not unusual to hit the virtual limit of 8 M/b in an IBM 3033, to have run time in excess of 4 hours for 5-10 year simulations, and generate very large output files of 20 Mb or more. The scale of these simulations has now reached the stage where traditional programming techniques are hampering the engineers ability to assimilate the results quickly and easily.

6.2.2 Financial Reporting

The success of any large company depends upon an accurate financial reports, showing current revenue and expenditure, both on project basis and against the Corporate Profit and Loss Account. Until recently these batch suites were written entirely in COBOL (eg. 100,000 lines) and were very unwieldy, timeconsuming and expensive to modify. This caused frustration among the finance managers who felt that the data processing department was unresponsive and problematical. To address these problems new methods are now being tried using non-procedural languages.

6.3 Strategic Information Systems

Systems for senior managers have in the past frequently met with failure because the technology posed problems for the manager, rather than solved his need for up-to-date information. The senior manager requires to have access to a considerable amount of summary data not only relating to to his own company but also to its partners and competitors, and hence seeks systems which provide the information as a normal business tool.

Three examples of typical management systems are set out below to which solutions are still being developed.
6.3.1 Corporate Information Library

Two questions posed by managers might be "show me a list of the North Sea licenses held by company X and their development record to date", or "what are the current corporate assumptions being made on inflation and price movement?"

To meet this need the relevant information must be extracted from the appropriate decision support systems and placed on a regular basis into a Corporate Information Library. Key Corporate assumptions also need to be defined in this library so that consistency is maintained. Simple business graphics facilities, preferably in colour, are a preferred means of presentation.

6.3.2 Corporate Financial Model

UK oil tax is extremely complex, now requiring four inter-depandan layers, resulting in taxation at the 80-90% level. These taxes are usually altered every year in some major way, and such changes can critically affect new investment on development decisions. To make such decisions requires a corporate financial model, which includes all current projects and all tax models both past and present. The answers are critically dependant on assumptions made about inflation, oil price and exchange rates which may also be varied as part of the simulation process. The need is for a terminal interface to pose the questions, although the calculations themselves do not need to be performed in real time.

6.3.3 Rig Scheduling

The growth of off-shore exploration has made drilling rig availability a critical factor in being able to start exploring new areas. A simple system is required to give a manager up-to-date availability information.

7 THE SOFTWARE DEVELOPMENT PROBLEMS

Over the last decade there has been a rapid and fundamental shift in computer technology which now makes software development costs the predominant factor in providing computer solutions. The rate of technological change seems to be accelerating rather than slowing down and these changes are now spreading the use of computers into all walks of life.

The computing industry, in the midst of worldwide recession, is still confidently predicting a compound annual growth rate of 25% and more. If this growth rate is applied to existing data processing staff the result in 10 years time would be economic
nonsense. This means that the software designers have no option but to respond to the challenge of reducing the manpower effort required for each application.

Examination of the existing applications in the oil industry show that there are (at least) six key problem areas to be tackled - which are:

i the need to provide an environment which suits the user both in functionality and ease of use.

ii the need to reduce (or eliminate) the high re-development costs, resulting from the changes in technology. Users will no longer tolerate the current three year re-development cycle.

iii ability to build or adapt applications very quickly to meet changing business needs. The aim should be days rather than weeks or months.

iv to design data structures which permit easy flow of data either upwards to senior management, or across functions.

v to allow use of micros or distributed computers in such a way that local development remains compatible with an overall strategy, and allows integration into the network if required.

vi to ensure that software becomes adaptable to distributed hardware functions.

8 TOWARDS A SOLUTION

Introducing new development methods can only take place effectively in the context of an integrated approach to user workstations, networking, software tools, and hardware.

8.1 The User Workstation

The workstation available to the user is usually a key factor in the success of any system. It provides his interface to the system and must encompass the facilities he needs.

The reduction in the cost of terminals against manpower costs has been dramatic in recent years and Workstation policy therefore has to be the cornerstone in DP strategy, the key issue being to provide each user with just one terminal on his desk at a reasonable cost, which performs the combination of functions which he requires (figure 2).
a. Worker Systems

Most registration systems require facilities for full screen operation in which data can be entered on a menu basis, often directly into the database.

These users may also need business graphics, with a hard copy facility while secretaries will need word processing, electronic mail and other office products.

b. Decision Support Users

Will require the greatest range of facilities possibly all of these shown in figure 2. This includes medium or high quality colour graphics for presentation purposes. Also requires browsing features on 132 column screens to scan large print volumes. May also carry out own program development, and include intelligent terminals for personal use.

c. Management Users

Require terminals which are easy to use, preferably with medium quality colour graphics.

8.2 Networking

It is essential that the computer facility should be seen as a utility service which can be freely accessed from any location. Until recently such utility networks have been created using special purpose hardware and software (eg. CERN, British Steel etc), which carry a high development and maintenance cost.

An alternative is to create networks based upon the new micro based statistical multiplexers (such as CASE) and utilising PTT leased lines. Britoil now has the largest such network in the UK (although small by comparison with many purpose built networks). Any terminal is able to access any of the local processors (HP3000) or the central IBM mainframe facilities. The latter connection is achieved through an IBM Series I running the Yale ASCII system which allows all HP terminals to access the IBM using 3270 emulation. The interHP links are based upon HP DS software which allows both file transfer and virtual terminal usage. The HP's can communicate to IBM either a batch (MRJE) link or the interactive (IML) link.

The main limitation has been line speeds - but wide band circuits are now anticipated within the next year. The strategy has been to use industry standard components, accepting the limitations imposed for the saving in manpower costs.
8.3 Software Development Tools

The normal systems development approach is to follow a set of standards covering

- feasibility study
- functional specification
- design reports
- construction
- testing and implementation
- system handover

The first three stages are agreed by the user and 'signed off' as a frozen specification. From this point the system is constructed and implemented by the data processing department and some time later (months?) is handed over to the user. Traditionally, the entire suite would be built in COBOL or FORTRAN with about 58% of the total development time spent in the construction phase.

It is now possible to reduce the amount of code produced in the construction phase by using the wide range of available software tools thus reducing the importance of the traditional languages.

These tools can be used in each of the four main system components i.e. data interfaces, input, computation and reporting. Taking each in turn, some examples of software available for HP & IBM equipment are given below.

Data Interfaces

The use of standard data base tools gives greater flexibility and make it easier to link systems together, and provide improved mechanisms for input and output. The choice is wide and selection should be made to meet the size of the problem.

Examples: IMAGE/3000, IDMS, FOCUS, STATUS

Input

Having chosen an appropriate database system input can be performed using full screen systems, usually mapping directly into the database.

Examples: VIEW/3000, IDMS/DC, FIDEL, SPF/DIALOG MANAGER

Reporting

Most database systems contain standard reporting tools which use English-like commands which give dramatic savings in coding, and are easy for non programmers to learn.

Examples: FOCUS, OLQ, QUIZ, RAPID, VIEWDATA
Computation

The traditional languages such as COBOL or FORTRAN are still excellent for carrying out numerical computation. However, special languages such as EPS-FCS are ideal for financial modelling and also yield substantial savings in effort.

8.4 FOCUS - A Non Procedural Language

FOCUS has been conceived (by Information Builders) as a complete COBOL replacement, and is based on a relational database which caters for both simple and complex interconnected files. FOCUS itself is interpretative and written in FORTRAN. There are eight principal modules (figure 3) which can be used on all aspects of the construction phase, covering screen input, data management, reporting, graphics, statistics etc.

The use of the term 'non-procedural' implies an English-like syntax, which allows considerable flexibility in ordering of VERBS and OBJECTS. Data objects are either defined in the file descriptions, or implicitly by their usage. The language is record orientated, and within the records the field values can be used freely in computational expressions that may contain normal FORTRAN intrinsic functions. However, it is much harder to perform computation across records, and the lack of array structures make FOCUS less attractive for some forms of technical work.

A simple example is shown below. This takes the file PROD, containing records with seven fields:

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROD-TYPE</td>
<td>Name of Product</td>
</tr>
<tr>
<td>AREA</td>
<td>Geographical Area</td>
</tr>
<tr>
<td>CUSTOMER</td>
<td>Name</td>
</tr>
<tr>
<td>MONTH</td>
<td>1-12</td>
</tr>
<tr>
<td>UNITS</td>
<td>No. Shipped</td>
</tr>
<tr>
<td>AMOUNT</td>
<td>Cost</td>
</tr>
<tr>
<td>FACTOR</td>
<td>Adjustment Factor</td>
</tr>
</tbody>
</table>

which are defined to FOCUS through the records maintenance language. The user can then input the following typical report request directly from the terminal:

```
TABLE FILE PROD
SUM AMOUNT OVER UNITS AND
COMPUTE RATIO = AMOUNT/UNITS;
ACROSS MONTH FROM 1 TO 3
BY AREA
    IF AREA IS EAST OR NORTH OR SOUTH
END
```
"OVER" ensures that 'UNITS' and 'RATIO' appear under 'AMOUNT'
"ACROSS" means tabulate months across the page
"BY" means sort by area
"IF" condition includes only three geographical areas
"FROM" takes only first three months
"COMPUTE" calculate new quantity for each area

This produces the following report:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>WEST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EAST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORTH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOUTH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMOUNT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNITS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RATIO</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The FOCUS graphics package allows simple histograms, pie-charts, bar-charts and scattergrams to be produced in a similar manner.

8.4.1 FOCUS Usage

FOCUS is extremely good for 'registration' or 'accounting' functions, such as machine accounting, personnel records, keeping track of experimental data, financial records etc. It is however currently less attractive for detailed numerical calculations, which will normally require FORTRAN or COBOL routines to be interfaced to the FOCUS code.

One of the most dramatic effects of introducing FOCUS has been to put some of the programming tasks back into the user department. Rather than have the user request the data processing dept. for a formal change, which then goes through a series of lengthy procedures the user can now produce new variations on reports quickly and easily. Such 'code' should be minimal and regarded as potentially 'throw away'. Although 'Throw-away' programming has not yet arrived, it could be within sight!

Information Builders are also currently implementing FOCUS on the IBM personal computer which could prove a very attractive facility.
A brief study has looked at possible use of FOCUS within high energy physics. While it should be possible to read data summary tapes, the event selection criteria could only be based on very simple tests such as V-zeros, inside target, and charge conservation. Calculations of momentum or effective masses could only be carried out by embedded FORTRAN. It therefore appears that its best uses are likely to be keeping track of experimental data, high level data summaries after main computation is complete, or in project reporting systems.

8.5 Graphics Facilities

Graphics is now seen as an essential facility by most departments within an oil company. While a few worker systems require graphics (eg. platform systems), the main requirement is for facilities which aid decision making.

Commercial and management users are normally content with medium quality business graphics, including piecharts, histograms and simple line graphics. Colour is often preferred although inessential, and hardcopy a must. This need can be met by a variety of standard software (eg. EASYCHART or DSG on HP3000,FOCUS on IBM). The ability to embed graphics into a text report, which can be output on a laser printer is seen as a vital facility. A substantial demand also exists for automated overhead slide production.

Technical users on the other hand often require high quality high resolution colour graphics as a day-to-day tool. Two new graphics standards are now emerging (GKS, CORE) and packages such as DI3000 can be successfully used as machine independent base library. Since each user department has its own jargon and requirements, a front end graphics language is needed to provide a user friendly interface, which will allow graphs to be merged, labelled, annotated or shaded as required.

Two dimensions is still adequate for most CAD work, which can be met by variety of standard packages (eg. DRAGON).

8.6 Office Systems

Are a vital part of the future. The many new products appearing (such as HPMAIL, PROPS etc.), together with possible use of local area networks is making it difficult to choose the best method of integration and most companies are still finding their way forward.

8.7 Micros

The uncontrolled spread of micros, acquired as 'office equipment' is one of today's key problems and may be tackled successfully by selecting multi-purpose workstations
which can be linked to the network, and setting guideline standards for micros in terms of:

- function
- connectability (eg. 3270 compatibility, RS232)
- local software (CP/M, UCSD, FORTRAN, BASIC VISICALC, Graphics etc).

9 DEVELOPMENT METHODS

9.1 Worker Systems

The successful integration of the Capital Project systems, has been largely due to extensive use (on HP3000) of V/3000 for screen input, IMAGE for database work and QUIZ for reporting, thus making major savings in COBOL code produced.

Initial result with FOCUS systems on the mainframe show that savings up to 5:1 can be made in the construction phase (implying an overall saving of 2:1). This has to be traded against increased machine development costs (up 1.7), and increased production running costs (up by 3). One of its attractive features is the ease of learning, for example the reporting features can be understood by both programming staff and non-programming staff in about two days.

9.2 Decision Support Systems

The methods used for constructing large numerical simulations have hardly changed in the last ten years. Development and maintenance costs are frighteningly high, and leadtimes very long. The FORTRAN code produced today is better structured and more modular but still contains serious inflexibilities because it yields large, unwieldy serial files, and large print volumes (eg. 250,000 lines). Thus, while major advances in physical simulations have been achieved in many disciplines, due to improved mathematical techniques increased computing power - the engineer or physicist is seriously hampered in his ability to analyse his results quickly.

To tackle this problem the basic building blocks are now being re-examined to determine whether some of the new software tools can be used, to give improved input methods (replacing card orientated input) and more flexible reporting systems. There is a clear need to separate the phases of input, computation and output more rigorously. This will make it easier to use special techniques in the computation phase if they are required (eg. array processing). Any further attempts to enhance FORTRAN should now (in the authors view) be restricted to improvements in this area since the languages's strength continues to be in its computational features.
The use of graphics as an alternative tool to scanning heaps of printed output is also vital. The development of sophisticated graphics workstations (at a reasonable cost), is now essential and possible contenders for this are APOLLO, PERQ, etc. Significant development effort will be needed to integrate these new workstations into existing applications.

9.3 Strategic Management Systems

VIEWDATA techniques for paged data bases are now becoming very attractive as a solution to the problem. Instead of building monolithic data base schemes it may be far better to extract data from the lower level systems into paged data base which can be called in a very simple manner using a key pad. This minimises computer jargon facing the manager and may well allow use of colour television monitors in the $2,000 range.

Considerable effort is now being invested in integrating VIEWDATA techniques with existing computer technology - terminals data-base and graphics. Ideally, the same (cheap) terminals might also be used for other office functions such as MAIL, etc.

The problems inherent in this type of system are often organisational, rather than technical both at management level and the support level. One approach to the support problem may be to create an 'Information Centre' whose sole purpose to maintain the data in the managerial database and to co-ordinate the linkage to the various systems.

9.4 Systems Analysis Productivity

New tools such as FOCUS successfully attack the construction phase, but still leave a substantial time in the rest of the systems analysis cycle. It should be possible to attack this in a variety of ways such as:

- avoiding over-specifying the application
- prototyping
- allowing end-users to produce own reports
- using data dictionaries
- automating the documentation (HELP features, word processing, use of laser printing)

Traditional methods suffer from freezing the specification too early. Building a prototype using systems such as FOCUS and then approaching final agreement with the user more slowly seems much more likely to lead to satisfaction than the system delivered six months after the specification was frozen.

New applications will rarely be stand-alone and the use of data dictionaries to ensure consistent data definition and assumptions appear to be a key tool for control.
10 HARDWARE STRATEGY

Most companies need to provide a set of facilities which will allow for natural systems growth - which will probably include:

- central number crunching and mass storage
- distributed power where it is required
- network which allow visibility of all systems
- network which permits attachment of micros etc
- single terminal policy

and to minimise support costs these should use industry standard components wherever possible. The key to forward planning is to choose manufacturers who have total commitment to maintain forward development of their operating systems so that the applications base can be carried forward at minimal cost.

11 THE FUTURE

Over the last decade there has been a dramatic fall in the cost of providing a professional user with a terminal. This cost, made-up of the terminal together with connection costs at both ends is already under 10% of the employee's cost to his organisation. The impact has been to extend the number of terminals very rapidly so that ratio of terminals to professional staff is now between 8:1 and 10:1. With falling costs it is anticipated that given equivalent productivity benefits, the ratio will probably reach 4:1 by 1987, and that within this decade, a composite terminal for voice, text, video etc. will finally replace the telephone.

To meet this challenge the data processing function has to create a hardware and software environment in which the three main functions of COMPUTING, COMMUNICATIONS and OFFICE SYSTEMS are fully integrated. While technical problems still remain to be solved, it is likely that many will also suffer from serious organisational problems because these functions have traditionally always been very independent of each other.

12 CONCLUSIONS

The biggest challenge facing the oil companies is the rapid technological changes which are taking place which will affect the business methods in use and hence the organisational structure itself.

The high redevelopment costs caused by these changes are becoming increasingly unacceptable and the driving force must be to improve productivity by reducing code produced, and improving system flexibility.

Much effort has in the past been put into language standards - and equal effort is now needed for new standards in communications, the new software tools, and graphics facilities.
Finally, the training of both new and existing staff in the new concepts is a vital task to ensure that old hands adapt to new ways of working and that next year's graduates arrive with an understanding of today's technology.

Acknowledgements

The author wishes to thank Britoil for permission to publish this paper, and would like to acknowledge the contributions of numerous people within Britoil, with particular thanks to NJ Lillie, MG Ribbins, NH Shelness and MD Vickers.

* * *

Fig. 1 Capital projects systems
Fig. 2 FOCUS

Fig. 3 User workstations
MR. R. BOCK CERN

- Q ---> Are the software tools mentioned by the speaker commercially available?

- A ---> The speaker answered in the affirmative and stressed that the company tends to use commercial software whenever possible.

MR. C. JONES CERN

- Q ---> How was it possible to convince the management to provide terminals for all the professionals that have need of them? The same attempt seems to have failed in certain areas at CERN.

- A ---> A terminal:person ratio of 1:8 is accepted now. The cost of reducing this ratio can be justified provided that the information flow works. Getting a system for senior managers to work is essential.

MR. J. HUTTON RUTHERFORD APPLETON LABORATORY

- Q ---> How good are the estimates of the time taken to develop a given software project?

- A ---> The estimates are good but the speaker cannot be quantitative.

MR. D.K. MARETIS UNIVERSITY OF OSNABRUECK

- Q ---> How can the development time needed for a system developed with Focus be compared to the time needed if an other language had been used?

- A ---> Some parts of the systems have been in parallel in Focus and in COBOL.

MR. D. WILLIAMS CERN

- Q ---> The total cost of our oil platform quoted by the speaker is 2000 M$ how much is spent on computing services?

- A ---> The amount is estimated to be 2M$ per year, over a period of 5-7 years.
SOFTWARE PROBLEMS IN MAGNETIC FUSION RESEARCH

R. Gruber
Centre de Recherches en Physique des Plasmas, Association Euratom-Confédération Suisse, Ecole Polytechnique Fédérale de Lausanne, 21, Av. des Bains, CH-1007 Lausanne, Switzerland

ABSTRACT

The main world effort in magnetic fusion research involves studying the plasma in a Tokamak device. Four large Tokamaks are under construction (TFTR in USA, JET in Europe, T15 in USSR and JT60 in Japan). To understand the physical phenomena that occur in these costly devices, it is generally necessary to carry out extensive numerical calculations. These computer simulations make use of sophisticated numerical methods and demand high power computers. As a consequence they represent a substantial investment. To reduce software costs, the computer codes are more and more often exchanged among scientists. Standardization (STANDARD FORTRAN, OLYMPUS system) and good documentation (CPC program library) are proposed to make codes exportable. Centralized computing centers would also help in the exchange of codes and ease communication between the staff at different laboratories.

1. FIRST EXPERIMENTAL OBJECTIVE: SUN ON EARTH

The main objective of two of the most promising experiments under construction, i.e. TFTR in the US and JET (Joint European Torus) in Europe, is to show the scientific feasibility of break-even (energy produced equal energy used in a magnetic fusion device). For both of these experiments the presently most advanced Tokamak concept was chosen. A future reactor based upon this concept consists of a toroidal container filled with very hot, totally ionized gas of a temperature of up to 100 Million degrees Kelvin (~ 10 keV) with a density of about $10^{20}$ ions and electrons per m$^3$. This gas has to be kept away from the walls of the toroidal container for at least one second. This is accomplished by means of strong, confining magnetic fields. In a Tokamak (JET, as an example, is shown in Fig. 1), a helical magnetic field is produced by toroidal field coils and by a poloidal field due to an axial current of up to 5 MA. The axial current is induced in the plasma by changing the current in the primary coils of an iron core transformer. To keep the plasma in equilibrium, additional poloidal fields are created by currents in external poloidal field coils. It is not possible to reach the high temperature of 10 keV, with only ohmic heating by the plasma current. Additional heating has to be provided by powerful neutral particle injection or by high frequency electromagnetic
FIGURE 1:

THE JET APPARATUS

1 Vacuum vessel (double walled)
2 Limiter defining the outer plasma edge
3 Poloidal protective shields to prevent the plasma touching the vessel
4 Toroidal field magnet of 32 D-shaped coils
5 Mechanical structure
6 Outer poloidal field coils
7 Inner poloidal field coils (primary or magnetising windings)
8 Iron magnetic circuit (core and eight return sections)
9 Water and electrical connections for the toroidal field coils
10 Vertical and radial ports in the vacuum vessel
resonance heating. One of the major problems in the future development of magnetic fusion power reactors will be the constant bombardment of the container wall by the 14 MeV neutrons produced in the D-T fusion reaction.

\[ _2^1H + _3^1H \rightarrow _4^1He + n(\sim 14 \text{ MeV}) \] (1)

Other major experimental problems still to be solved are the construction of shielded superconducting coils needed to run a reactor for an extended period of time, and the elimination of impurities in the discharge. These two problems will be investigated in the Tokamak T15 in the USSR and in the JT60 Tokamak in Japan which are the other two major Tokamak experiments under construction. Since major Tokamak devices are extremely costly, the world fusion research community is discussing the possibility of building INTOR (International Torus) as the only next stage experiment.

Such experiments, large in size and cost, need strong support in the form of computers for control and data interpretation.

2. COMPUTER EXPERIMENTS

Each computer code made to control the experiment, to interpret measured data, to simulate inaccessible data or to study a theoretical model can be compared with a major piece of an experiment. These codes should be modular, easy to use and understand (well documented) and readily adapted or modified.

We propose to subdivide the computer codes into three major categories, even though overlaps are possible.

2.1 Codes to analyse measured data

A Tokamak experiment, such as the Joint European Torus (JET), consists of different consecutively or simultaneously running steps:

- Produce an initial vacuum;
- Fill in the chamber with H, (hydrogen), D (deuterium) or D-T;
- Preionize the gas;
- Induce a plasma current of \( \sim 3 \text{MA} \);
- Turn on the toroidal field of \( \sim 3 \text{T} \);
- Control the equilibrium with external poloidal field coils;
- Inject a neutral particle beam of \( \sim 10 \text{ MW} \);
- Diagnose the plasma via a few tens of diagnostic subsystems.

In JET the pulse duration ranges up to \( \sim 20 \text{ seconds} \). During this time all the above-mentioned activities must be controlled and coordinated. Bet-
ween two consecutive pulses, there are 10 minutes, thus enough time to col-
lect, transfer, file and interpret the data of one pulse.

This is done by the integrated computerized Control and Data Acquisi-
tion System CODAS\(^2\). This system deals with \(~ 6000\) monitoring channels,
\(~ 4000\) control channels and \(~ 3000\) data acquisition channels. The total
data per pulse to be stored and treated is about \(250\) kbytes engineering
data and \(2.5\) Mbytes diagnostics data.

In general, there are different diagnostic systems for the same plasma
parameter. The electron temperature profile, for example, can be measured
through Thomson scattering, soft X-ray detection, microwave systems, far
infrared spectroscopy, spectral line intensity ratios or recombination ra-
diation. The raw data has to be interpreted and compared by means of diag-
nostic computer codes.

2.2 Simulation of inaccessible data

The most complete simulation model would be to follow all the individ-
ual particles in a 3D electromagnetic field. In JET, for example, there
are about \(10^{22}\) particles which can be electrons, fusion ions, impurity
ions, neutral particles or, hopefully, neutrons. It is evident that a com-
plete simulation cannot be carried out on a computer.

The description of an ionized gas can be simplified by considering
each species of the gas as a fluid governed by a set of equations of the
form

\[
\frac{\delta A_j}{\delta t} + \nabla F(A_j) = S_j(A_k)
\]  

(2)

where \(A_j\), for example, can be the density, momentum or the energy of the
particle \(A\). \(F\) denotes the flux and \(S\) contains source or sink terms. In
addition, all the charged particles are subject to the electromagnetic
field. In this reduced model, the most rapid timescale is given by the
electron cyclotron time which is about \(10^{-11}\) seconds and physicists would
like to have information on the microsecond to second timescales. Still
such a model is by far too general to be solved on current and future gene-
ration computers.

There are two ways for further reduction of the model:
Magnetohydrodynamics (MHD) models consider a plasma as a single average
fluid. Transport models consider each species individually in a time and
space averaged manner.
2.2.1 Magnetohydrodynamic equations

Denoting the mass density by $\rho$, the pressure by $p$, the velocity by $v$ and the magnetic field by $B$, the MHD equations are:

\[
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho v) = 0 \quad \text{(Continuity eq.)}
\]

\[
\frac{\partial \rho v}{\partial t} + \nabla \cdot (\rho v v) = -\nabla p + (\nabla \times B) \times B \quad \text{(Newton's eq.)}
\]

\[
\frac{\partial p}{\partial t} + \nabla \cdot (pv) = -(\gamma-1)p \nabla \cdot v \quad \text{(Eq. of state)}
\]

\[
\frac{\partial B}{\partial t} - \nabla \times (v \times B) = \nabla \times (\eta \nabla \times B) \quad \text{(Faraday + Ohm)}
\]

Since reactor plasmas are very hot, the resistivity $\eta$ is often neglected. The case $\eta = 0$ is called ideal MHD theory.

2.2.2 MHD equilibrium

Even though the plasma evolves slowly in time, confined plasmas are well described by the static equilibrium equation:

\[
\nabla p = (v \times B) \times B \quad \text{(4)}
\]

This is the remaining equation of the MHD model when all the quantities are chosen to be time-independent and $v = 0$. Note that eq. (4) is nonlinear. For the axisymmetric case, the equilibrium solution is homogeneous in the toroidal direction and the problem is reduced to a two-dimensional (2D) one.

Almost every laboratory has its own equilibrium code. It is used mainly for two purposes: In the design phase of an experiment, equilibrium calculations are performed to design the poloidal field coils which have to control the equilibrium. In the operation phase, equilibrium solutions are compared with experimental data providing some information about current and ion temperature profiles which are difficult to measure. This is accomplished by adjusting some free parameters such that the final solution matches experimentally available data.
At present, much effort is made to calculate fully 3-D equilibria for helically-distorted tori (Stellarators or distorted Tokamaks). However, it is not yet clear if such equilibria solutions exist in a strict mathematical sense. The idea is to minimize the plasma energy. If such a minimum is found, it is stable with respect to certain eigenmodes.

Such equilibrium calculations only hold if external heating does not significantly affect the configuration. Due to neutral particle injection, the static equilibrium can become stationary \((\nu \neq 0)\) and the equilibrium equation more complicated.

2.2.3 MHD stability

Experiments are subject to instabilities with time scales of \(10^2\) to \(10^6\) times faster than the confinement time but, \(10^3\) to \(10^6\) times slower than cyclotron times. These instabilities can be internal (sawtooth instabilities) leading to strong energy outflux, or external (kinks or disruptive instabilities) destroying the plasma in the order of a few microseconds. They have detrimental effects on the wall and the circuitry. Since these instabilities are related to global displacements they can be described by the MHD model. The most rapid modes with growthrates of the order of a few microseconds are well described by the ideal \((\eta = 0)\) linearized MHD equations (3). Using an \(e^{\omega t+i\psi}\)-ansatz (where \(\psi\) is the toroidal angle and \(n\) the toroidal wavenumber) and a spatial discretization with finite elements, one obtains an eigenvalue problem

\[
A\vec{x} = \omega^2 B\vec{x}.
\]  

(5)

The matrices \(A\) and \(B\) represent, respectively, the spatial discretization of the potential and the kinetic energies, \(\omega^2\) is the eigenvalue and \(\vec{x}\) the eigensolution. The case \(\omega^2 \ll 0\) gives a growing solution: The configuration is unstable. The validity of the ideal MHD model expires for modes with growthrates \(r^2 = -\omega^2 \ll 10^{-4}\). Modes with smaller growthrates are strongly affected by non-linearity, resistivity or other nonideal phenomena. The necessary fine spatial discretization needed to reach such a precision leads to hudge sparse matrix eigenvalue problems with up to 50000 components in the \(\vec{x}\) vector. The displacement pattern of a typical internal kink instability eigensolution of eq. (5) is shown in Fig. 2.

If non-ideal or non-linear effects are included, the whole set of partial differential equations (3) have to be solved by integrating ahead in time. The time step is governed by the most rapid oscillations in the system which are the fast magnetoacoustic compressional waves. Since one is interested in the study of unstable modes with very small growthrates, these stable highly oscillating fast modes are eliminated from the model. Even though the resistivity is very small, the existence of so-called
magnetic islands can be demonstrated. Another important result of these computations is the nonlinear stabilization of linearly weakly unstable modes.

2.2.4 Transport calculations

In the MHD model, the particles move on magnetic flux surfaces. In reality however, due to collisions and field fluctuations, there is always a drift of the particles across the magnetic field. Also there is an undesirable influx of partially-ionized impurities causing energy loss due to radiation. The transport of particle and energy is computed. Even though, many physical phenomena are taken into account, the transport calculations (eq. 2) are not able to explain the experimental observation of an anomalously high energy outflux. The observed electron thermal transport seems to be due to microinstabilities leading to field fluctuations. In current transport codes, measured data can be adjusted by the introduction of ano-
malous transport coefficients, which simulate these microfluctuations and by the variation of many parameters and profiles.

Transport computations are carried out in order to understand the penetration of highly energetic neutral particles into the plasma. Even though these particles are not affected by magnetic fields, Monte Carlo calculations are performed to simulate the ionization and the energy deposition.

In the burning phase of a reactor, additional external heating is stopped. The temperature of 10 keV is maintained by the thermalization of the energetic $\alpha$(He$^+$)-particles produced in the fusion reactions. The study of the deposition of the $\alpha$ particle energy in the plasma is also simulated by the transport calculations. In this case a comparison with the experiment is not yet possible.

Transport calculations are very time-consuming. Many time-steps have to be performed. Each time-step consists of a resolution of a system of non-linear equations. The power of the most powerful computers (CRAY1, for example) determines the dimensionality and the physics which can be treated in transport codes.

2.3 Theoretical experiments

Under this category we include computations which cannot be directly compared with experimental data. An example is the microinstabilities mentioned above, that lead to anomalous transport coefficients. Microinstabilities are studied by diluting the plasma locally so that it is possible to follow the individual particles one by one. Up to $2^20$ particles are taken in domains of $2^N$ (N ~ 5-10) Debye lengths. Generally one imposes periodic boundary conditions. To solve the long range Poisson equation, the domain is subdivided into a grid. To reduce the numerical noise, the particles which move in this grid obtain a finite size. To use very efficient Poisson solvers, the grid is equidistant and the number of intervals in all directions is a power of 2. Then FFT (Fast Fourier Transform) or Buneman's cyclic reduction algorithm can be used. In a 60 or 64 bit-word machine, the position and the velocity of one displacement component can be packed into one word and integrated ahead with one operation. The periodicity conditions are automatically introduced when an overflow or an underflow occurs. The number of particles and grid points which can be taken and the dimensionality are strictly bound by the power of the computer. For such "particle in cell" (PIC) simulations, the computers will never be powerful enough since in a real Tokamak plasma there are $10^{20}$ electrons and ions per $m^3$ and the plasma volume of JET is 100 $m^3$!
3. IS SOFTWARE STANDARIZATION POSSIBLE?

Software cost is comparable with the cost of major pieces of experimental apparatus. This is the reason why much effort is made towards production of exportable computer codes. Such programmes have to be written in a standard programming language, should be well documented, should be easy to understand, easy to use and easy to modify. Below, these requirements are discussed in more detail.

3.1 Choice of compiler

When writing a computer code, only perhaps 5 to 10% of the time counted from starting with the idea to the final test phase is spent in coding. This means that the choice of the compiler is of minor importance. More weight should be given to the organization, readability, documentation, ease of use and, at least in the plasma physics community, to portability. For our codes, it is very important that the compiler produces a highly-optimized binary code. It should include vector operations. In any event, codes which are planned now should make extensive use of vector operations since experience shows that such codes also run more rapidly on scalar machines.

The only compiler which fulfills today these requirements is STANDARD FORTRAN with additional vectorizable DO-loop subroutines. It will remain the compiler producing the most efficient binary code as long as the scientific benchmarks are mainly done with FORTRAN programmes. FORTRAN is evolving. It includes more and more the so-called advantages of other compilers. However, to fulfill the readability requirement it seems that one should only use a well-chosen subset of all possible commands.

3.2 Programme organization

In magnetic fusion research there are trends to standardize the programme organization. K.V. Roberts proposed the OLYMPUS system\textsuperscript{3}) which is a prestructured programme skeleton to which the user has to add the rest of the body. It consists of seven major blocks which are: control, prologue, calculation, output, epilogue, diagnostics and utilities. The data is organized in 5 groups of COMMON blocks: General OLYMPUS data, Physical data, Numerical data, Housekeeping and I/O and diagnostic quantities. Conventions on the use of variable names are introduced: Initial letters for variables in COMMONs are A-H, O or Q-Y for real variables, L, M or N for integers and LL, ML or NL for logical variables. Local variables should start with Z, I or IL and loop variables with J. Subprogramme arguments initiate with P, K or KL. The initially-chosen programme testing data is modified through NAMELIST.
An example of a code strongly influenced by the OLYMPUS concept is ERATO\textsuperscript{4). This code solves the linear, ideal MHD stability problem (see eq. 5). It has been structured in such a way that five main blocks (Blocks 1 to 5 in Fig. 3) preceded by an equilibrium solver (Block 0) are executed successively. Each of these blocks has a structure similar to that prescribed in OLYMPUS. All six parts are interlinked by a data base section. The blank common is different in each block giving an efficient usage of the memory space.

Each block is a well-defined major task. It is easy, even for external people, to replace such a block by another one. Even though ERATO originates at Lausanne, Block 1 was entirely redone by a Japanese group, Block 2, the vacuum part, was rewritten at the Max Planck Institute at Garching and the new mathematical eigenvalue solver package (Block 4) is due to a mathematician from Oak Ridge National Laboratory. Each block is highly structured and written in a modular way. Each subroutine fulfills a specific task and most of them can be tested independently. In Block 3, for
example, where the eigenvalue matrices \( A \) and \( B \) (eq. 5) are filled in, the variational formulation of equation (3) is defined in one subroutine, with the basis functions for the finite element approach, the integration scheme, the construction of the matrices and the introduction of the boundary conditions all in other separate subroutines. Changing boundary conditions only changes one subroutine!

3.3 Documentation

In an IAEA (International Atomic Energy Agency) consultants meeting on computer codes for fusion research\(^5\) strong recommendations were made to encourage exchange of fusion computing programmes. Also it was proposed that the codes should be published in the "Computer Physics Communications" International Physics Program Library at Queen's University, Belfast. This guarantees well-documented codes. The programmes published there are available to everyone. Well-documented means here that the published text explains the physics, the numerical methods used and the limits of applicability of the code. References to the computer code should be given. In the code, references to the published text should help to follow the coding. Computer codes, together with the published text, should be readable like textbooks.

3.4 Magnetic fusion computing centers

In the U.S., the magnetic fusion research computing requirements are fulfilled by the centralized National Magnetic Fusion Energy Computing Center (NMFECC) at Livermore. Presently, there are three powerful computers available (CDC 7600, CRAY 1 and CRAY 1S). Such centralized computing centers have advantages: researchers who often move can use their codes from different places in the U.S. No adaptation to new sites is necessary. Exchange of codes is trivial.

The tremendous amount of computing power of the NMFECC is mainly used by the Princeton Plasma Physics Laboratory, the Oak Ridge National Laboratory, Los Alamos and Lawrence Livermore. The data is sent via two distinct satellite channels!

In Europe, the Max Planck Institut für Plasmaphysik at Garching owns a CRAY 1 mainly used for magnetic fusion research. Most of the important 2D and 3D codes are available there.

ACKNOWLEDGEMENTS

We would like to thank Dr. F. Bombi who forwarded all documentation on CODAS. Many thanks to Dr. J. Vaclavik and Prof. A. Kritz who carefully read the manuscript and improved the English.
REFERENCES


* * *

QUESTIONS

MR. D. WILLIAMS CERN

- Q ---> What is the largest program that you maintains under OLYMPUS?

- A ---> Personally 3000 lines but the Meteorology people do more.

MR. M. METCALF CERN

- Q ---> Comment: 8x enhances readability by:
  array notation (better than DO-Loops)
  named control constructs
  CASE construct
  Long names

- A ---> Free form source is a facility which will require automatic re-formatting systems.

MR. A. BOGAERTS CERN

- Q ---> Why does vectorized code work better on "classical" computers?

- A ---> No answer supplied.
MR. R. BOCK CERN

- Q --> Is your communication between program modules standardized, i.e. do you leave externally similar data structures going to and from the database?

- A --> No, we use fairly trivial local conventions for this communication.

MR. J. HUTTON RUTHERFORD APPLETON LABORATORY

- Q --> Does the European fusion community have plans to use computer networks, for example based on the planned PTT networks?

- A --> This has been discussed but we are waiting for these to be fully available. Currently we use 300bps (dial-up, I assume).
DATA PROCESSING AT THE EUROPEAN SPACE AGENCY

Dr. C. Mazza
Data Processing Division, European Space Operations Centre, Darmstadt, Germany.

ABSTRACT

A short introduction explains ESA's activity and programmes. Satellites are seen as generators of space data. The aim of ESA missions is to deliver data to the end users. The end-to-end system satellite-end user is examined in order to understand where and how the data processing activity is performed. Centralized processing done by the Agency vs decentralized done by the end user is analysed. A concrete example of the data processing chain for an ESA Scientific Satellite is presented in order to understand the main characteristics of ESA Data Processing Systems. These characteristics require a rigorous software engineering approach, which is enforced at ESA through standard practices issued by a Board for Software Standardization. The main features of the standard practices are presented. Finally some ideas are presented concerning future standardized means for interchange of data.

1. Introduction

The activities of the European Space Agency undertaken within its five establishments require a broad spectrum of data processing tasks. The fundamental programmes of the Agency cover essentially three areas:

- the Space Transportation Systems, i.e. the European launcher Ariane and the Spacelab to be put in orbit by the American Shuttle

- the socalled Application Programmes, mainly all the telecommunication satellites and the earth observation satellites.

- the Scientific programmes, mainly orientated to the study of the celestial sphere (astronomy, astrophysics, solar physics etc.)

The phases of implementation of each project in the above areas extend from the preliminary studies of a mission to the final exploitation of the data delivered. In all the phases there is some software involved.

In terms of ground system complexity and sophistication needed in data handling, satellites of the scientific programme are most demanding. Here payload or scientific data generated from experiments on board the satellite has to be processed in addition to processing needed for
satellite control and monitoring. In order to adequately control such satellites an assessment of the performance of the experiments requires to be carried out on a regular basis and often in real time. Where satellites have more than one experiment on board (one can consider typically between 5 to 10) each with a multiplicity of operating modes the requirements for data handling software are demanding. With the advent of on board or satellite computer ground data handling required special development, i.e. facilities to load, modify and test the suite of programmes used by the satellite.

In order to illustrate the differing nature of scientific missions each requiring its specific data handling suite of programs, the following examples of past and future missions undertaken by ESA can be noted:

- **COS-B**, a satellite which concluded its 7 year very successful life few months ago. The mission purpose was to measure the gamma-ray emission in the energy range between 30 and 500 MeV.

- **GEOS**, a satellite for the investigation of all the phenomena related to the geomagnetic field.

- **IUE**, a satellite for the observation of the ultra-violet emission.

- **ISEE-B**, a satellite for the investigation of the interaction between the earth and the solar wind.

These three satellites are currently operational. In the next 4 to 5 years several other scientific satellites will be launched, i.e.

- **EXOSAT**, a satellite to be launched next November which will provide data on cosmic X-ray sources in the energy range from greater than 0.1 KeV to less than 50 KeV.

- **SPACE TELESCOPE**, a large high-quality optical telescope in the space, with a primary mirror of 2.4 m diameter.

- **GIOTTO**, a fly-by mission to the Halley's comet, which will be launched in 1985 and will encounter the comet in 1986.

- **ISPM**, a satellite for investigations on the solar corona, the solar wind, the sun-wind interface; the heliospheric magnetic field, the solar and non-solar cosmic rays, the interstellar and interplanetary gas and dust.
HIPPARCOS, a satellite for the measurement of the trigonometric parallax, proper motion and position of about 100,000 selected stars. The position will be measured with an average mean error of $2 \times 10^{-3}$ arc sec.

2. The data processing system

All these satellites, though completely different in their purposes, can be seen as devices which acquire data through various sensors, transmit these data to the earth, where by use of a processing system they are finally delivered to the end users.

Consequently we consider here the end-to-end data system (fig. 1) to comprise the satellite plus the related ground support system.

![End-to-end Data System Diagram](image)

Fig. 1 - End-to-end Data System

The overall process from the origin of the observation to the utilization of the data by the end users can best be analyzed in the following activity diagram (fig. 2) expressed with the Structured Analysis and Design Technique notation 1.
2.1 Functions undertaken

Basically, five activities are performed, i.e.
1 - the acquisition of the data on-board the spacecraft

2 - the acquisition of the data (telemetry) in form of radio-frequency signals on the ground and the transmission from the ground to the satellite of the commands (again in form of radio-frequency signals).

3 - the control of the satellite from the ground.
   This is based on the examination of telemetry data concerning the behaviour of the various spacecraft sub-systems which determines the commands to be sent to the spacecraft in order to maintain it in the planned status and to acquire the planned data.

4 - the processing of the data in order to present them to the final users in a form specified by them.

5 - the delivery or dissemination of the data from the processing site to the end-use site.

The measurements of the phenomenon under observation are acquired on-board the spacecraft by means of appropriate sensors.

The control of the spacecraft is performed through control devices on-board and through commands received from the ground. The combination of the two determine the orbit of the spacecraft, its orientation and the observation pattern.

The data acquired are transformed in radio signals which are transmitted in the selected frequency to a ground station. An appropriate antenna delivers the data the station equipment where the data are transformed in digital form to be transmitted, through a communication network to a control center. The control center can be colocated with the station or remote.

The function of the Control Center is two-fold:

a - one has to verify that all the sub-systems of which the spacecraft is constituted are performing correctly and take corrective measures whenever necessary. This implies orbit and attitude determination and prediction, and manoeuvres determination. The manoeuvres are then implemented by means of appropriate commands which are sent to
the spacecraft as well as the commands to the instruments which constitute the payload. Both the manoeuvre commands and the payload commands are sent in form of a telecommand bit stream to the ground station which provides their transformation into radio signals which are transmitted through the antenna to the spacecraft. This activity is monitored by spacecraft engineers which receive information from the computers on displays, graphics, printouts or any other suitable form and can activate various programs in order to perform the necessary actions. Whenever the mission is based on an observatory concept, some experimenters are permanently present at the control center, where they receive the so-called quick-look data, i.e. real-time data which allow them to establish an optimized observation pattern.

b - validate the payload data acquired, establish the necessary time correlations and put these data into a database for further processing.

The further processing activity consists then in taking the data from the database, in separating the streams of data generated by the various onboard instruments, in correcting the raw data to eliminate all the distortions introduced in the data by the system, to annotate the data with time and spacecraft position and attitude and to perform all other centralized processing agreed with the end users.

The level and the amount of centralized processing is normally established in advance, but can also be modified during the mission life according to the users' feedback after analysis of the data. Some instruments may not perform nominally, some interesting and unexpected phenomena may be discovered. These events may determine new and unforeseeable requirements. The data processing software need therefore to have a high degree of flexibility to accommodate quickly any such new requirement.

The pre-processed data are finally put onto an appropriate support (normally magnetic tapes) to be delivered to the end users or directly disseminated via digital links. The dissemination schedule is normally predetermined but, it can also be modified in near real-time according to the users' need.

The data processing system which we have examined is partly performed in real-time and partly as an off-line process.
Real-time functions which are performed for each spacecraft are:

- the acquisition of the telemetry stream, which is transmitted at various bit rates. Decommutation of the telemetry stream is based on the various formats existing for each spacecraft. Little standardization exists for such formats which complicates the related software.

- time checking. Correlations need to be established between on-board clocks, station clocks, control center and computer clocks. Propagation delays need to be taken into account.

- data quality checking. The quality of the data received need to be checked in order to eliminate corrupted data.

- limit and status checking. The various engineering parameters needed in order to monitor the spacecraft status of health are checked against limits outside which alarms are raised for the spacecraft engineers.

- data archiving. After the above checkings data are stored onto discs where they remain available for a prefixed amount of time. They can be accessed in real-time. At the end of the foreseen period the data are copied to magnetic tapes for further off-line processing.

- data display and printout. The spacecraft engineers sitting in the control room can receive normally various types of outputs. By means of appropriate commands sent to the computers through functional key-boards they can modify the information received with a high degree of flexibility.

- orbit, attitude and manoeuvre computations.

- telecommands processing. The telecommand processing software determines the sequence and the type of commands to be sent to the spacecraft based on the manual commands requested by the spacecraft engineers, the automatically scheduled commands and on sequences of experiments commands needed to achieve specified observations. Before being forwarded to the stations, commands are verified for validity. Once received on-board the commands are acknowledged. The values of appropriate parameters in the following telemetry confirm or not the correct execution of the commands on-board.
The validated data, available on the data-base or on magnetic tapes for
delayed processing, are accessed for pre-processing and for delivery to
the end-users.
The modalities vary according to each mission requirements. Possible type
of outputs/products are:
  o quick-look data displayed in real-time
  o observatory facilities (displays, hard copies, plots, plus
    software to allow analysis of the data)
  o high density tapes
  o computer compatible tapes (cct's)
  o printouts
  o stripcharts
  o plots
  o microfilm
  o pre-processed data via a data-link to experimenters' computers
    (on-site or remote)

The type of response for the above outputs can be in real-time, in
near real-time or off-line with delays from few hours to few weeks.

A traditional problem in all ESA mission is where to stop the
centralized part of the data processing, which is essentially an economic
problem. Tasks which are always done in a centralized fashion by ESA are
the sorting of data by experiment, the calibration of the data, the
annotation with the time reference and the position and attitude of the
spacecraft.

What else should be done by the Agency depends on the type of the
mission and on the users' wishes.

Whenever a standard product can be defined and the users are able to
organize themselves with an efficient interface body, then it is probably
more cost effective to do further centralized processing. Two typical
examples of this are the missions with observatory concept and our
meteorological satellite where rectified images, wld charts, sea surface
temperature charts, cloud top height charts etc. are prepared by the
Agency and disseminated to various national meteorological institutions.

In all other cases we agree with the experimenters the data format
and interface and we send them raw and/or preprocessed data.
2.2 Data processing example: the European X-ray observatory Satellite (EXOSAT)

Having described in general terms the ESA data processing activities, I should now like to take by way of example how the data processing for EXOSAT is organized.

EXOSAT, as already mentioned before, is a scientific satellite to study cosmic X-ray sources.
The satellite carries three types of instruments:

a. two imaging telescopes for the determination of the absolute position of low energy X-ray sources
b. a medium range experiment for measurement of X-rays in the range 1-50 KeV
c. a gas scintillator, for spectroscopic measurements in the range 1,5-80 KeV.

There is an on-board computer and a star tracker.

The satellite can operate in two modes, the so-called occultation mode (using the moon or the earth as occultation bodies) and the arbitrary pointing mode.

The satellite is 3-axis stabilized, the orbit has an inclination of 72° and is highly eccentric (perigee 500 km, apogee 200 000 km).

There are some processing requirements of general nature and some specific data products required.

To the first category belong requirements such as:

- availability: 90% of the scientific data acquired above the Van Allen Belts.
- time accuracy: 100 μs with respect to Universal time.
- observatory: facilities to schedule and monitor the observations in cooperation with resident scientists.

Data products were initially specified in the form shown in fig. 3
<table>
<thead>
<tr>
<th>Item</th>
<th>Data Type</th>
<th>Mission Phase</th>
<th>Data Processing Function</th>
<th>Availability</th>
<th>Data Form</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>GSPC Exp.</td>
<td>routine operations</td>
<td>Real time processing</td>
<td>Immediate</td>
<td>alpha-numeric graphical display + hardcopy when required</td>
<td>Direct telemetry of energy data</td>
</tr>
<tr>
<td>3.1</td>
<td>Basic mode HERI</td>
<td>&quot;</td>
<td>Display pulse height &amp; related spectrum data</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>3.2</td>
<td>Basic mode HTR1</td>
<td>&quot;</td>
<td>Display intensity profile (same as 1.2)</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>3.3</td>
<td>Basic mode HBL1</td>
<td>&quot;</td>
<td>Display burst length spectrum</td>
<td>&quot;</td>
<td>&quot;</td>
<td>Burst length processing</td>
</tr>
<tr>
<td>3.4</td>
<td>Basic mode HBL2 ( HBL3 )</td>
<td>&quot;</td>
<td>Integrate &amp; display pulse height and burst length histograms (similar to 1.5)</td>
<td>&quot;</td>
<td>&quot;</td>
<td>Energy/burst length histogram with/without filter</td>
</tr>
<tr>
<td>3.5</td>
<td>Basic mode HTR2</td>
<td>&quot;</td>
<td>Integrate &amp; display arrival time difference histograms (similar to 2.2)</td>
<td>&quot;</td>
<td>&quot;</td>
<td>High time resolution</td>
</tr>
<tr>
<td>3.6</td>
<td>Basic mode HTR3</td>
<td>&quot;</td>
<td>Display intensity profile with time resolution data</td>
<td>&quot;</td>
<td>&quot;</td>
<td>Intensity profile</td>
</tr>
</tbody>
</table>

Following issue of these requirements some four to five years before the launch our software engineers discussed with the experiments each of these requirements in order to determine exactly which software had to be produced and to draw up a functional software requirements specification.

Fig. 3
The software developed from these requirements runs on the European Space Operations Centre - Multisatellite Support System (ESOC/MSSS) which has some EXOSAT dedicated computers coupled to it (fig. 4).

The standard MSSS is a fully redundant configuration constituted by:

- 2 mainframes A, B which are SEL 32/77 computers
- 4 front-end machines OP1, OP2, DT1, and DT2 (all Siemens 330)
- 1 Message Router (MR) (Siemens 330)
- 1 Display Support (BD) and MR back-up (Siemens 330)

This configuration supports all the tasks which are common to all satellites.

There are then some mission dedicated computers, i.e.:

- EX1, (Siemens 330) which selects the raw-cata, checks them, archives them on magnetic tapes and sends them to the main configuration.
EX2, (Siemens 330) which supports the observatory in real-time or by retrieval up to 80 hours in the past. This computer interfaces also the Extended Data Handling Computer (HP 21 MX).

HP 21 MX -, Extended Data Handling Computer. The software running on this machine has been written directly by the experimenters to offer the necessary flexibility. The only constraint was at the interface between the two computers.

A package was written by our engineers to handle the interface between the experimenters' software and the link, according to the following specifications (fig. 5).

<table>
<thead>
<tr>
<th>ESCC DPD</th>
<th>EXOSAT SOFTWARE DESIGN AND SPECIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ISSUE 2</td>
</tr>
<tr>
<td></td>
<td>DATE 1.7.80</td>
</tr>
<tr>
<td></td>
<td>SECTION 4.4.5</td>
</tr>
<tr>
<td></td>
<td>PAGE 1</td>
</tr>
</tbody>
</table>

4.4.5 Observatory Computer to Experimenters' Computer Interface (EX2 - HP21MX)

The interface between EX2 and HP21MX is a hardware link carrying data from the Short History File (SHF) on EX2 to the HP21MX machine. Also carried are the associated control messages.

The hardware interface consists of a 'DTRSA' (output) card and a 'DTRSE' (input) card in EX2 connected to two '12930' universal interface cards in the HP21MX (one dedicated to input, the other to output).

Existing software as used on the link to MR will be used on EX2, the link having logical names (KRKA (2,0) (output) and KRBE (1,0) (input).

A subroutine package will be available such that the interface between the applications software and the link is as follows:

1. subroutine call to read one retrieval request and wait
2. subroutine call to write one whole format and wait
3. subroutine call to send a status reply (e.g. data not available on file) and wait
4. Indication that EX2 is not ready to accept requests is achieved by 'end'ing the HP Retrieval Process.

Fig. 5
3. Overall Characteristics of the ESA data processing systems

Our software is characterized by the following aspects which determine its nature. There is often a high amount of data to be processed. A mission such as GEOS has generated so far over 2 million Mbytes of data to be processed.

There is a variety of highly complex experiments to handle, with various bit rates, formats and commanding sequences. Most of these experiments are based on some sort of on-board intelligence which needs to be controllable from the ground. There is a variety of output supports, products and response time. The user requirements are subject to many changes, which need to be introduced when the software development is already very advanced.

There is a need to communicate with people having very different background and experience. For these reasons our software engineers need to have a solid multidisciplinary background in physics, celestial mechanics, electronics and telecommunications, and data processing.

The above characteristics require a rigorous system engineering approach. In particular, in the data processing area, this has generated a strong need for software engineering practices to be able to manage the variety of requirements and of people involved.


Few years ago the Agency established a Board for Software Standardization and Control (BSSC). This Board issues Software Engineering Standard Practices which are followed by the various groups which are developing software.

These Standard Practices are subdivided in:

- Mandatory Standards. These have to be followed always by everybody.

- Recommended Practices. A justification to the appropriate management level is needed if they are not followed.

- Guidelines. These are useful practices. No justification is needed if they are not followed.

The basis of our Software Engineering Practices is the Software Life-cycle Management Scheme (fig. 6).
<table>
<thead>
<tr>
<th>Phases</th>
<th>UR Items</th>
<th>SR Items</th>
<th>SR/R Items</th>
<th>AD Items</th>
<th>AD/R Items</th>
<th>DD Items</th>
<th>DD/R Items</th>
<th>TR Items</th>
<th>M&amp;O Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Users Requirements Definition</td>
<td>Software Requirements Definition</td>
<td><em>Identification of users requirements</em></td>
<td><em>Identification of system requirements</em></td>
<td><em>Definition of software architecture</em></td>
<td><em>Detailed design</em></td>
<td><em>Performance of provisional acceptance testing</em></td>
<td><em>Final acceptance</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>Feasibility assessment</em></td>
<td><em>Development plan and cost estimates (±30%)</em></td>
<td><em>Detailed development plan &amp; cost estimates (±10%)</em></td>
<td><em>Coding</em></td>
<td><em>Unit Testing</em></td>
<td><em>Operation of the sw system</em></td>
<td><em>Operation of the sw system</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>PA</em></td>
<td><em>PA</em></td>
<td><em>PA</em></td>
<td><em>PA</em></td>
<td><em>PA</em></td>
<td><em>PA</em></td>
<td><em>Maintenance of code &amp; documentation</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><em>Production of the sw life-cycle historical report</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><em>Final version</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Deliverable Items**
- **URD**: Users Requirements Document
- **SRD**: Software Requirements Document
- **ADD**: Architectural Design Document
- **DDD**: Detailed Design Document
- **STD**: Software Transfer Document
- **PHD**: Project History Document

**Under Change Control**
- Intermediate Reviews

**Reviews**
- Development Inception: Users request accepted
- SRD App’d
- ADD App’d
- Preliminary Acceptance: DDEV/ CODE Acc’d
- Final Acceptance

**Acceptance**
- Development Inception
- SRD App’d
- ADD App’d
- Preliminary Acceptance: DDEV/ CODE Acc’d
- Final Acceptance

**Major Milestones**
- End of Life-Cycle

*Fig. 6*
The scheme is valid for any type of software developed at ESA. It emphasizes the need to involve strongly the users of the software. It is their responsibility to deliver a User Requirement Document, they have to participate actively in the process of determining the software requirements. They are formally involved in the requirements and design review, in the specification of the test plans. Their feedback is sought in the compilation of the user's and operator's manuals and in the execution of experimental runs of the system prior to its final acceptance.

The standard practices are based on the 3 classic principles of the top-down construction, of concurrent design, programming and documentation and of structured programming.

As concerns the requirements definition phase and the architectural design phase, where the need to communicate with people which are not software engineers is intense, we have found the SADT diagrams\(^1\) a particularly useful tool for describing easily processes and procedures with complicate hardware/software/human interfaces.

We have not found so far a fully satisfactory formal language for description of software requirements.

For detailed specifications at module level we normally use high level pseudo-languages such as PDL\(^2\) or pre-compilers allowing structured constructions such as FLECS\(^3\).

All these tools are aiming to ease communication between people. The need for communication between people is in fact the source of most software costs.

5. How can the data processing experts offer a better service to the scientific community?

The current level of progress in the computer and software technologies should allow soon substantial improvements, if we are able to take advantage of it.

Costs of computing and storage power are still continuously decreasing and do not constitute any longer a problem. Huge amounts of data can be already transmitted via satellite links and packet switching networks. Database technologies should allow to archive and retrieve them in simple manners.

The combination of all these technologies should "provide means for interchange of data in open systems environment in a similar manner to worldwide telephone and telex services".
This recommendation is contained in an interesting paper of ISO, the International Organisation for Standardization.

The day we can achieve this electronic library service, the scientific community will have improved by orders of magnitude its potential.

But a substantial organizational effort is required to take advantage of the technological progress, which can only come out from a strict collaboration between the scientists and the dataprocessing experts.

ESA is performing a big effort in cooperation with NASA in order to standardize way of exchanging large quantity and variety of space borne data - (see fig. 7 - Standard Format of Data Unit of Information). Simple ideas and few conventions are needed, but big effort goes into making them accepted by all parties concerned.

**Standard Format of a Data Unit of Information**

*Fig. 7*
REFERENCES

1 - An Introduction to SADT, Structured Analysis and Design Technique, Report 9022-78R, Softech Inc., 460 Totten Pond Road, Waltham, MA 02154, USA.


3 - Beyer T., FLECS User's Manual, Department of Computer Science, University of Oregon, Eugene, Oregon 97403, USA.


* * *

QUESTIONS

MR. PH. GAVILLET CERN

• Q --> In the case of satellites acting just as a receiver/transmitter of data blocks, as in the STELLA project, does one have to follow the full control process as well as the apparently heavy "software cycle procedure"?

  • A --> Yes!

MR. J.T. CARROLL CERN

• Q --> Are you working towards a significant increase in the proportion of data reduction done on board the satellite?

  • A --> Data reduction on-board vs. on-ground is a trade-off. Availability of on-board intelligence allows more on-board data reduction. No matter where data reduction takes place, experimenters always want to be able to reconstruct the originally observed data.

MR. E. LILLESTOL - BERGEN UNIVERSITY

• Q --> What is the peak data rate from the satellite?

  • A --> It is about 100 - 150 kilobits per second.
MR. R. ZELAZNY - INST. NUCL. RES., OTWOCK-SWIERK, POLAND

• Q --> To which data is your standardization applied? Are they the raw experimental data or the compiled results? Furthermore, how many people are working on data processing, and do you use some computerized tools such as those used by SREM, for example, a software engineering data base?

• A --> Standardization of data units of information is supposed to be applied to any type of data: raw, pre-processed, or fully processed. Roughly 300 people are involved in data processing at ESA, and we have plans for the implementation of a highly automated software engineering environment. At the moment, we have various tools to help in the various phases. The main problem is that each has a different interface. We would like to avoid this in the future.

MR. M. TURNILL BNOC GLASGOW

• Q --> Are the formal software development standards accepted by the scientific community at ESA, and how do you ensure that the standards are followed?

• A --> Yes, the mandatory standards are accepted, but some experimental groups do not follow them! The standards are imposed by the individual managers at ESA.
HIGHLIGHTS OF A NEW SYSTEM FOR DATA ANALYSIS

Per Svensson

Swedish National Defense Research Institute, Stockholm, Sweden

ABSTRACT

A transportable system for the analysis of large sets of data, forming complex information structures, is being developed at the Swedish National Defense Research Institute, with financial support also from some civilian Swedish government agencies. The system is based on a relational data base handler of new design, permitting efficient data storage and fast evaluation of complex, spontaneous queries. A query language, based on set algebra and oriented towards scientist users, was developed for the system. To this kernel may be added subsystems for interactive sublanguage definition and user communication, data loading, correction, and archiving, tabular and graphic data presentation, statistical analyses, and data base backup and recovery. The paper will describe the project objectives, show how they influenced the language and system design, and present an overview of the preliminary system version now completed.

1. INTRODUCTION

Techniques and tools for the analysis of data have developed rapidly during the last decade. Much of this development effort has been directed towards developing large libraries of statistical analysis procedures and integrating them into user-controlled statistical systems. In recent years, several of these systems (e.g., SAS) have been supplied with data management and data presentation capabilities, turning them into quite powerful general-purpose data analysis tools. Users of statistical systems seem to be quite satisfied with the facilities provided. So why bother to develop yet another system, which in this light seems to stand a small chance of becoming commercially accepted?

Existing statistical systems serve their purpose well, as long as the data bases to be processed have a simple, static logical structure and are not too large, and moreover, stay that way during the course of the analysis.

In many scientific fields, however, the steps in an analysis involve the creation of complex data structures and large data sets, even when the basic data sets are of moderate size and complexity. Typically, such situations arise when looking for interactions between several separately
monitored classes of phenomena, for example the incidence rate of a
disease and various environmental factors. If the kind of interaction is
unknown a priori, the data space in which to look rapidly becomes unma-
ngageable.

Data collections with such characteristics are rapidly being created
in many fields. The build-up of large administrative data bases, the con-
tinuous monitoring of complex industrial processes and scientific exper-
iments, of physical processes within the earth, the sea, and the atmo-
sphere, the collection of biological or chemical data from our environ-
ment, and the execution of abstract simulation experiments are types of
activities that continuously create such data collections.

The question whether such data should be analyzed in depth and not
only collected and maybe checked for anomalies has to be answered in each
case. A good answer is by no means always easy to get. If you want to
prove something, you should design your experiment carefully, then run it
and collect only the pertinent data. Hopefully, most statistical program
systems are used to analyze such data sets.

But if useable data are already there, why not use them and save the
cost of experimentation? If the observed phenomenon is not amenable to
experimentation? Then, chances are that no reliable answer to any interest-
ing question will be obtained, no matter how sophisticated techniques are
used in the analysis. Large sets of data have to be studied in order to
find a single interesting "signal". A system by which such studies are to
be performed must be powerful and at the same time easy and safe to use.
Otherwise, the cost for the analysis would usually be too high, consider-
ing the high probability of "don't know" answers.

The last decade's developments in the theory and technology of data
base management systems has led to new possibilities for the design of
general-purpose software useful for the detection and analysis of unknown
kinds of interaction between loosely related classes of data. Recently,
the field of statistical data base system design can be said to have
gained scientific recognition. The first workshop on statistical database
management was held last December in Berkeley, California.

In a recent overview, Shoshani\(^1\) shows five example data bases,
demonstrating the range of practical considerations that have to be taken
into account when designing generally useful statistical data base sys-
tems. An experiment in atmospheric physics is described, in which photo-
metric measurements of the night sky are being made every 15 minutes at 5
wavelength and 360 positions. Running for five years at 10 stations, the
experiment will eventually generate about $10^9$ measurements. "Parameter data", i.e., descriptions of the measurement setup and execution, have to be added to this number. Shoshani's other examples are taken from socio-economic statistics, business structure of the states of USA, geographically based data, world trade time series, and U.S. energy statistics.

In these examples, Shoshani points to the following characteristic features:

i) the distinction between parameter data and measured data;

ii) the sparseness of many statistical data bases;

iii) the frequent need to create and work with "summary sets" in very large statistical data bases;

iv) the stability of many (most) statistical data bases - they represent data collected for future reference and analysis;

v) the proliferation of terms in large data bases - codes and abbreviations not only occur in large numbers but may also change their meaning over time.

Shoshani uses the term "summary sets" to describe the general notion of derived data. The system to be described below was designed with the goal of being able to analyze in depth large and complex data bases, without recourse to problem-specific programming in a general-purpose programming language. We judged that a user language based on an algebra of relations would be a suitable tool for a data analyst, if properly designed and implemented. In such a language, data transformations of almost any kind could be expressed and stored as data extensions, or "views", in the data base itself. The evaluation of expressions in this language is carried out within the framework of a relational data base management system, designed for efficient storage and access of data in a user environment where "queries", or data transformations, are statistical, i.e., whose results depend on large subsets of the data. The few commercially available data base systems which have enough functional power for this task, are optimized for queries, whose answers depend on small subsets of the data base.

Example. According to Sundgren, the Swedish statistics on traffic accidents is a collection on data on involved vehicles, drivers, passengers, and pedestrians, together with information about time and place for the accident, and the injuries inflicted on each person involved. This information may be studied together with information on road characteristics, taken from the road data bank. Examples of questions that might be put are:
- find an average accident rate by road standard
- find road segments with unusually high accident rates
- is there a correlation between the accident rate for a road segment and age of drivers involved?

2. PROJECT OBJECTIVES AND STATUS.

In 1975, a project with the following objectives was initiated at the Swedish Defense Research Institute:

i) to study general and open-ended support systems for the handling of large sets of data, resulting from experiments or simulations

ii) based on such studies, present a proposal for an advanced support system for technical and scientific data processing

iii) produce specifications for a practically useful system, running on the Dec System 10 computer available at the Institute but as far as possible, transportable to other computers as well.

It was decided from the outset to base subsequent work on the concept of a relational data base management system (rdbms). Also, it soon became clear that the application area motivated an approach to the design of such a system which was in many ways different from that of then existing research rdbms prototypes (notably PTHA\textsuperscript{6}, System R\textsuperscript{7}, and Ingres\textsuperscript{8}). Design goals were formulated\textsuperscript{9} after an analysis of the requirement on a data analysis support system in a scientific, large data base environment.

These goals were:

i) To allow the scientist users to work directly with their data, the system must be highly automatic. It must not require or even allow any tuning or extensive maintenance during its use.

ii) The system had to contain a very high level user language (query language) suitable for the succinct expression and gradual accumulation of complex data transformations (called views). On the other hand, it was considered reasonable to expect from the scientist user an ability and willingness to express these transformations in a formal language.

iii) High performance for the evaluation of complex transformations on large sets of data was required. Such transformations easily lead to unacceptable execution times unless efficient methods of query optimization are found, not least when expressed in a very high level, "non-procedural" language. Certain classes of queries are however, inherently impossible to evaluate efficiently.

iv) The typical user was envisaged as member of a small group of analysts, working either with a dedicated, comparatively small com-
puter system, or with a large, time-shared central computer. The
system should allow for easy sharing of results within the group.
To access data not originating within the group, conventional copying
and loading was considered adequate. Requirements for simultaneous
reading and updating of data therefore existed but did not impose
stringent restrictions on the system design.

v) Within the user community many different types of computer
would exist. To allow the system to be used on different
computer types without unacceptable conversion costs, it
had to be transportable, i.e., written in a high level lan-
guage subset acceptable by most compilers for the chosen
language.

vi) Economy of data base storage space was found important both
directly, to reduce storage space costs for large data sets,
and indirectly, as a means of reducing the processing time
of a query. The more compactly data can be stored, the less
data transportation between primary and secondary storage will
obviously be required.

vii) A set of general-purpose application functions should be in-
cluded in the basic system. Requirements for subsystems for
descriptive statistics, basic statistical analysis, data pre-
sentation, and bulk data test, input, and transfer were formu-
lated.

viii) A programming user must be able to add new application func-
tions, written in some commonly used programming language,
to the system. Such additions always require additions to the user
language as well. To support a controlled language growth, a language
definition subsystem allowing the incremental addition of new gramma-
tical rules was found highly desirable. Language and function exten-
sibility were thus required.

ix) To enable adequate documentation and retrieval of data and views,
a meta-data subsystem must contain both system-created and user-
provided information about the elements of the data base.

To design an rdbms adequate for our purpose both theoretical studies
and experimentation were found necessary.

Theoretical work was concentrated on the problem of evaluating com-
plex queries.

A general class of queries in a relational dbms are the "multi-
variable queries", so named because they may be expressed in the
form
\[ \{ F(x,y,...) \mid x \in X, y \in Y, ... \Rightarrow f(x,y,...) \} \]
where \( X, Y, ... \) are sets of objects belonging to the data base
(relations as a special case), $f(x,y,...)$ is a predicate, a logical function, of the variables $x,y,...$, and $F(x,y,...)$ is some set- or object-valued function of $x,y,...$

Fundamental query evaluation problems in a rdms are:

1) to define general subclasses of multivariable queries that are of practical importance and can be evaluated without searching through the entire cartesian product space $X \times Y \times ...$

2) to characterize general subclasses of multivariable queries as inherently difficult, i.e., not resolvable in polynomial time when the number of variables is allowed to increase without bound.

Although such studies were made in some depth, their practical value consisted mostly in making the designers more familiar with the nature of the problems. It now stands clear that more work should be done in this area. A recent paper by Kim\(^{12}\) contains some ideas of practical importance, and generalizations of his rules will probably be applied in our system.

Two other primary design problems were the determination of a suitable storage structure and development of associated access algorithms, and the selection or development of a query language.

The storage structure design problem was approached by way of experimentation. A prototype system was built according to principles aimed at satisfying the special performance requirements of statistical applications. Data access patterns typical for statistical work are quite different from those of typical commercial data base applications. Since almost all dbms performance studies were at the time motivated by non-statistical applications, new storage and search techniques had to be designed, implemented, and evaluated\(^{13}\).

The reason why rdms systems are usually "record oriented" seems to be the fact that most business data base transactions deal with data about a single object (e.g., a product shipment). In a typical system, many such transactions need to be processed each second. In statistical applications, one deals with the entire record mainly when collecting the data. As soon as data are ready for analysis, typically a few selected fields in a large number of consecutive records are accessed. This suggests a field-wise arrangement of data rather than a record-wise one.

The fact that statistical data and in particular, parameter values often have a small range of variation, contain runs of equal values, and are sparse indicated that data compression techniques might be profitably
used. By ordering data according to the parameter values, a higher degree of compression may be obtained.

It is generally accepted that the practical value of data compression lies at least as much in the reduced need for data transportation as in the reduction of storage cost. Perhaps more surprising at first, since it is generally thought that compression and decompression are cpu-intensive operations, is that certain data compression techniques may dramatically reduce the amount of cpu operations necessary for the evaluation of some kinds of data transformations.

We found that a general-purpose data analysis system must provide facilities for interactive update, insertion, and deletion of records, although many statistical data bases are so stable that they do not seem to require these facilities. This requirement is felt most strongly during the tedious and time-consuming validation and correction phase of an analysis.

In summary, the storage and access subsystem had to be based on a storage structure that allowed efficient sequential and direct read access to compressed data. Also, the storage structure must be dynamic in the sense that arbitrary subsets of the data could be deleted or inserted, without causing significant reductions in access performance or space utilization efficiency.

3. SYSTEM OVERVIEW FROM A USER'S VIEWPOINT
3.1. System status and environment

The system, previously called Datalab but for trade-mark reasons provisionally renamed to Cantor in this paper, is in its present form a single-user, interactive relational dbms with an unusually powerful query language and storage and access performance characteristics designed to fit the intended application area, as discussed in the previous section. A few facilities remain to be implemented or revised, in order to bring the data base system to the intended level of usefulness. Subsystems for descriptive statistics, statistical analysis, data presentation, bulk data format test, input, and transfer, and backup-recovery also remain to be designed and implemented. I will describe here mainly those functions which are already more or less in their final form, but I will on occasion mention facilities which have not yet been firmly decided.

Cantor is designed to run under a time-sharing operating system (or, obviously, on a single-user computer). All details of terminal interfacing, transaction queueing, and communication network management are assumed to be handled by the host operating system, as are the allocation to different users of common resources, such as primary memory and processor time.
It is intended to eventually allow several simultaneous users to share data in its data bases. A user who wants to update a copy of a data (relation) table can do so, as long as he is authorized by the operating system to access the data base which contains it. An update of a non-private data table must wait, however, until the user has been granted exclusive access to it by the Cantor system. In this way, a crude but in our opinion sufficient multiple user facility is planned for later versions of the system.

Cantor is written mainly in a subset of the Pascal high-level programming language. The system version which is at present being tested, consists of almost 70 000 lines of Pascal code and 3 500 lines of Assembly language code, comments included.

At present, the system runs only on the Dec-10 computer under the operating system Tops-10. Our preliminary plan is to transport the code to Vax/VMS as part of a (not yet decided) second phase of the project.

3.2 Data bases and data objects

Data in Cantor reside in two kinds of areas: one group data base (GDB) contains quantities of longer duration used by a designated group of users, and one user data base (UDB) for each group member contains data private for that user. All data in UDB is automatically visible to the user, but data in GDB is visible only on request.

Objects recognised by the system have unique names and a well-defined type. They are either flat objects viz. scalars or tuples (cf. records of a file) or set objects. Sets of tuples of a common type are called relations. Sets of scalars of a common type are called basic sets.

Each kind of data object can have either of two modes: base and view. A base quantity consists of explicitly stored data and may be updated. A view is an expression defining a value which can be computed, ultimately in terms of base quantities. Views may have parameters.

3.3 Types

Scalar types are predefined: integer, float (optionally with restricted precision or constant exponent), logical, literal (used for names of objects), and text. In comparison with a conventional programming language, such as Pascal, these types are generalized in the sense of being...
extended by the special value UNDEFINED. An unordered set of pairs (attribute name: scalar type) determine a tuple type. A tuple value is an instance of its type, i.e., a set of pairs (attribute name: scalar value). In the same way, a set type is defined as the set of sets of values of a given flat type. A relation is a special case of a set, namely a set formed on a tuple type (even if this tuple type has only one attribute).

3.4 Meta database

Associated with each data base in the system is a meta database, which contains all information required by the system to keep track of stored data and its properties. The meta database is organised as three relations, maintained automatically by the system. Views may be defined whose value depend on the meta database relations.

3.5 Basic commands and terminal user language

The terminal user language is planned to consist of a standard part used for predefined operations and a variable set of private extensions to this language. In the existing version of Cantor, only the standard language is recognized.

The most important commands available in the existing system are:

Define a base relation               - BASERELATION
Load a base relation from an external file - LOAD
Print a base relation on an external file   - PRINT
Define a view                         - VIEW
Evaluate a view and store its value as temporary data - STORE
Evaluate a view and store its value as base data - LET
Insert tuples into a relation         - INSERT
Remove tuples from a relation         - REMOVE
Update tuples of a relation           - UPDATE
Add attributes to a relation          - ADDATTRIBUTES
Delete an attribute from a relation   - DELATTRIBUTE
Delete a data object                  - DELETE
Exit from system                      - CFF

The commands to define base relations and views will be discussed in more detail below.
3.6 The command BASERELATION

The name and type of a relation value must be predefined, "declared", using the BASERELATION command. Additionally, the concept of key of a relation is recognized by the BASERELATION command. A key is a subset of the attributes of a relation, which uniquely identifies its tuples. It is a consequence of the definition of a relation value that a key always exists. By defining a key as a proper subset of the set of attributes of relation, a user may force the system to check this identification property. In a statistical data base, the keys are subsets of the parameter data of a data collection.

It is convenient to view the concept of a relation value as a rectangular table, satisfying the following rules:

i) the rows (tuples) of the table must be unique;
ii) no operation on the rows of a table may depend on their ordering;
iii) the columns (attributes) of the table have unique names.

3.7 An example data base

In his description of the Swedish statistics on traffic accidents, Sundgren suggests a data base structure, which in Cantor would be described as follows. Key attributes are enclosed in an extra pair of parentheses:

BASERELATION Accidents ((accno: INTEGER),
roadsegment: INTEGER, date: INTEGER);
BASERELATION Vehicles (accno: INTEGER, vno: INTEGER);
BASERELATION Drivers ((accno: INTEGER, pno: INTEGER),
vno: INTEGER);
BASERELATION Pedestrians (accno: INTEGER, pno: INTEGER);
BASERELATION Passengers ((accno: INTEGER, pno: INTEGER),
vno: INTEGER);
BASERELATION Roads ((roadsegment: INTEGER),
traffic: FLOAT, length: FLOAT, roadstandard: INTEGER);
BASERELATION Injuries (accno: INTEGER, pno: INTEGER, injury: LITERAL);

We see how the various relations are interconnected through common attributes, whose names have been chosen here so as to support the human user's intuition and memory. It is possible and probably desirable to define a stronger and therefore more secure type concept by introducing
named and typed "domains", unique over the data base, as sets in which attribute values are members.

3.8 The VIEW command and the query language SAL

One of the commands listed above has a much more complex syntax than the others. This is the VIEW command, used to express arbitrary transformations of the contents of a data base. The view concept in Cantor is a natural generalization of the concept of a function procedure (subroutine) in conventional programming languages. Like functions in Pascal, say, a view may have a number of parameters of arbitrary type.

The query language of Cantor, here provisionally called SAL, is the language in which the view definitions are expressed. SAL was developed under explicit assumptions about the users' educational background. A user with a basic mathematical education at university level should be able to use the language with fairly little training, because all its important concepts are already in his repertoire. In the requirements definition phase of our project, a number of existing query languages were studied, but for various reasons, no one was considered suitable.

The SAL language was designed according to the following basic objectives:

1) It is only used to compute new data from existing data. For example, no input-output statements are part of the language.

2) The query language is not intended as a general-purpose computing facility. Many kinds of computation will have to be done by special programs interfaced to the dbms. A user should not be misled to use the query language for purposes where conventional programming is the only adequate technique. Control and data structures proper to algorithmic languages were therefore excluded.

3) A simple and formal structure was desired rather than similarity with natural language, with its many subtle ambiguities.

4) The intended users should be familiar with the formalisms of elementary algebra and set theory. The language should not introduce concepts outside of this domain unless necessary.

5) Data transformations are usually derived using the same stepwise abstraction process found indispensable when solving non-trivial problems in other domains, such as programming or elementary calculus. The view definition mechanism is suitable for this purpose and was therefore made central to the language.
3.9 Operators and expressions in SAL

3.9.1 Operators on scalars and sets

Operators are functions from operand values to result values, and the result type is uniquely determined by the operator and the operand type(s). If the result cannot be computed, the result is UNDEFINED of the appropriate type.

Unary operators are NOT, IS_UNDEFINED, ROUND, TRUNC, CARDINAL, and the arithmetic functions ABS, SQR, SIN, COS, EXP, LN, SQRT, ARCTAN. They are written in functional notation, e.g., NOT (a). ROUND or TRUNC must be used when converting from float to integer.

The binary operators +, -, *, /, DIV, MOD, =, <>, <=, >=, OR, AND have their usual meaning. They are valid for the appropriate scalar type, except = and <>, which are valid for all flat types.

Binary operators acting on sets are EQUALS, CONTAINS, CONTAINEDIN, UNION, INTERSECTION, DIFFERENCE, and MEMBER with the conventional set algebra interpretation. Binary operators have fixed priorities and association rules. To override these rules, or for arbitrary purposes, parentheses may be used.

3.9.2 Operators on tuples

To operate on tuples, the identification (:), concatenation (.,.,.), and extraction (.) operators exist, as well as the relational operators (=, <>). The purpose of the first three operators is to provide adequate facilities for naming attributes which occur as intermediate or final results of a sequence of operations on relations.

The identification operator, which is binary, creates a tuple with one attribute when applied to a scalar right argument, and adds a prefix to the attribute names when applied to a tuple right argument. Thus, a:1 is a tuple with attribute name a and attribute value 1. Identification may also be used with a set argument, with the effect of applying it to each member of the set. E.g., assuming that Accidents has been defined as in Sec. 3.7, a:Accidents creates a relation value which is the same as Accidents, except that its attribute names are a.acno, a.roadsegment, a.date instead of acno, roadsegment, date, respectively.

We say that two attribute names are disjoint if they do not have a common prefix. Thus, a.b and b are disjoint, whereas a.b and a are not.
The catenation operator, which can take an arbitrary number of arguments, may operate only on disjoint tuples. If one wants to catenate two tuples with coincident attribute names, the identification operation must be applied first.

The binary extraction operator is the inverse of identification; it extracts a subtuple with a given name prefix. Thus, \((a:\text{Accidents}).a\) represents the same relation value as \text{Accidents}.

Analogously, \((a:1).a\) is the scalar value 1. The parentheses are required, since extraction has higher priority than identification.

Example: \(.(a:10, b:20).a = 10\) is TRUE. In this example, all three operators catenation, identification, and extraction were used.

3.9.3 Operators on relations

There are three kinds of operators acting on relations, namely functional form operators, the cartesian product operator, and the partitioning operator.

Functional form operators are binary operators with a relation expression as left argument and a flat expression, whose type depends on the operator, as right argument. The right argument must be enclosed in brackets \([\ ]\). They are:

- restriction
- generalized projection
- selection
- aggregation

- WHERE
- no keyword
- SELECT, SELECTMAX, SELECTMIN
- COMPUTE, SUM, PRODUCT, MAX, MIN, AVERAGE, EXISTS, ALL, COUNT, CARDINAL.

Let \(R \text{ ffop} [e]\) be a functional form expression. Then, the flat expression \(e\) may reference attribute names of \(R\). When \(e\) is evaluated for a tuple \(t\) of \(R\), such a reference, \(id\) say, evaluates to \(t.id\). In this way the meaning of nested \(\text{ffop}'s\) is defined.

The following table shows how the functional form \(R \text{ ffop}[e]\) is defined for different operators:
<table>
<thead>
<tr>
<th>Operator</th>
<th>Type of flat expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE</td>
<td>LOGICAL</td>
<td>The set of tuples of ( R ), for which ( e ) evaluates to TRUE.</td>
</tr>
<tr>
<td>SELECT</td>
<td>LOGICAL</td>
<td>The unique tuple of ( R ), for which ( e ) is TRUE, otherwise UNDEFINED of appropriate tuple type.</td>
</tr>
<tr>
<td>SELECTMIN</td>
<td>Numeric</td>
<td>Analogous to select, but selects the unique tuple for which ( e ) is extreme.</td>
</tr>
<tr>
<td>SELECTMAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>Numeric</td>
<td>The sum, product, etc, of the values obtained when evaluating ( e ) for each tuple of ( R ). UNDEFINED if not computable.</td>
</tr>
<tr>
<td>PRODUCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVERAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXISTS</td>
<td>LOGICAL</td>
<td>TRUE if some, respectively all, values of ( e ) on the tuples of ( R ), are TRUE, otherwise FALSE.</td>
</tr>
<tr>
<td>ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNT</td>
<td>LOGICAL</td>
<td>The integer number of tuples of ( R ), for which ( e ) evaluates to TRUE.</td>
</tr>
<tr>
<td>CARDINAL</td>
<td></td>
<td>The integer number of tuples of ( R ).</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>Flat</td>
<td>Used to evaluate several aggregates &quot;in parallel&quot;, ( e ) is any flat expression depending on ( R ) only via aggregates SUM, PRODUCT, etc.</td>
</tr>
</tbody>
</table>

The generalized projection operation, which has no keyword, evaluates any flat expression \( e \) over the tuples of \( R \) and forms the set of the resulting flat values.

The cartesian product operator \( \times \estimated{\ldots} \times \) takes one or more relations ("factors") as argument and produces a result relation, whose attributes are the union of the sets of attributes of the factors, provided that these sets are disjoint. Otherwise the expression is syntactically incorrect (if the attribute sets are not disjoint, they can always be made so by applying the identification operator).
The set of tuples of the cartesian product is the set of all catenations of tuples of the factors. Thus, the cardinality of the cartesian product is the product of the cardinalities of its factors.

**Example:** *(a:Accidents, Drivers)*

is a relation with the six attributes a.acenno, a.roadsegment, a.date, aconno, pno, vno. Each tuple in Accidents is combined with each tuple in Drivers, giving the resulting cartesian product.

The partitioning operation BY has a special form: R BY [e] agop[s]

where e is a tuple expression, s a scalar expression and agop stands for any aggregation operator. R BY [e] may be viewed as a functional form operator with a set of relations as result, which is however not an allowed object type. Aggregation over each of the relations in the set is needed to obtain a result of set type.

**Example:** Vehicles BY [a:acenno]

    COMPUTE [(acenno:a, nv:CARDINAL)]

computes a relation, giving the number of vehicles (nv) involved in each accident (acenno).

3.10 Example queries on the Traffic Accidents data base

**Query 1:** "Find an average accident rate by road standard."

The rate is defined as the number of accidents divided by the number of vehicle kilometers for the segment, which in turn is the product of traffic and segment length. We start by appending traffic volume and accident count to the Roads table:

```
VIEW Roads1 <-
    r:Roads.[r, trvol:r.traffic * r.length, nacc:CARDINAL
     (Accidents WHERE [roadsegment = r.roadsegment])];
```

Here we have used the abbreviated form .[ for [(. Now we can compute accident rate by road standard:

```
VIEW Arat_by_std <-
    Roads1 BY [s:roadsegment]
    COMPUTE .[roadstandard:s, rate: SUM(nacc)/SUM(trvol)];
```

**Query 2:** "Find all roadsegments with more than twice the accident rate for their standard, and with more than five accidents."
This query is answered by adding the average rate for the standard as a new column in the Roads table, restricting to those road segments with a high accident rate, and projecting on the road segment number.

VIEW Roads2 <- *(r:Roads1, Arat_by_std)
    WHERE [r.roadstandard = roadstandard]
    .[r, acrate: r.nacc/r.trvol, avrate:rate];

VIEW Badroads <- Roads2 WHERE [acrate > 2 * avrate
    AND nacc > 5] [roadsegment];

4. SYSTEM DESIGN ASPECTS

4.1 System structure

The system structure of Cantor is sketched in Figure 1, below.

4.2 Subsystems

The current system consists essentially of the five main subsystems shown in Fig. 1, and an additional basic command subsystem.

4.2.1. Language analysis subsystem

This subsystem performs syntactic and semantic analysis of command language and query language syntax. It produces either error messages, or a syntax tree where all references to stored data, to attributes of relations in enclosing expressions (similar to the referencing of non-local identifiers in a block structured language), and to views have been resolved. In the case of a view reference, its syntax tree is generated and connected to the result tree. The types of each partial result are calculated, checked for consistency, and stored in the nodes of the tree.

4.2.2. Evaluator subsystem

This subsystem performs optimization, "code generation" and "code interpretation".

4.2.2.1 The optimizer

It works by transforming the syntax tree into a logically equivalent one, corresponding to a different (more efficient) query formulation, and
Fig. 1. System structure of Cantor illustrated by execution of a STORE command.
with special-case information added to certain nodes. The storage structure allows fast evaluation of certain important special cases of expression (see below, Sec. 4.2.3), notably box search and equijoin subexpressions. Such cases are detected. The optimizer also - efficiently - detects such common subtrees of the syntax tree, which need to be evaluated only once.

We are at present working on extending the optimizer with several rewriting rules, which enable much faster evaluation of important query classes. These rules are basically similar to, but more general than, those proposed by Kim for use with IBM’s query language SQL12).

4.2.2.1 The stream net generator

When evaluating a query in an rdbms, it is generally necessary to repeat the same basic operations on many items, a kind of vector processing. Cantor takes advantage of this fact to process queries efficiently. The method used was inspired by the concept of a dataflow computer14). A similar method has been proposed by Yao15) to manage the evaluation of a subset of relational algebra queries.

Basically queries are evaluated by an interpretation process, i.e. a graph describing the relation between operands and operators in the query is traversed by a program which calls a procedure for each operator in turn, producing an intermediate result from the operator's input data.

This graph, which we call the stream net, contains nodes from all stream operators called. Adjacent operators are separated by buffer nodes. In the example shown in Fig 2, the stream net has been obtained directly from the syntax tree that results from semantic analysis, i.e. bypassing the optimizer. Buffer nodes are drawn in thin lines, operator nodes in fat lines.

There are three different kinds of buffers in the evaluator: stream buffers, run buffers, and B-list buffers. Each kind of buffer has variants for the different scalar data types recognized by the system. Stream buffers are used for temporary data buffering only. An operator takes what data there is in its input stream buffers and produces a result in its output stream buffer. When executed, a binary operator (say) produces an output of the same length as the shortest input buffer. Run buffers are short variants of stream buffers used for storing constants and streams with long runs. B-list buffers contain the entities (data blocks) read from or written into the transposed files of the data base, called B-lists (see Sec. 4.2.3).
Fig 2 Unoptimized stream net for the relation valued expression

Roads WHERE [roadstandard <= 2] [roadsegment]

- buffer node
- operator node
Phases: The operators of a stream net are grouped together in phases, which are in turn arranged in a hierarchical structure showing the order in which they have to be executed. Phases always communicate via B-lists. In this small example Phase 1 is to be executed until complete, before Phase 2 may be started. In general, execution of a query requires looping over subexpressions, and in these cases the phase tree describes the hierarchic loop structure.

4.2.2.3 The execution program

This program has two main tasks. When initializing a phase, it assigns space to its buffers. Then, it calls the stream operators of the phase in a suitable order until all input to the phase has been consumed. The order of execution of operators is not rigidly determined. The structure could in principle be interpreted by several processors working in parallel.

4.2.2.4 Stream operators

The system contains a large number (about 80) of stream operators, classified into constant, scalar, tuple, set, relation, aggregation, and transfer operators. Transfer operators are Interval read and Append which moves data between streams and B-lists. The other classes of stream operators should be self-explanatory.

Many set and and relation operators call procedures in the search and sort subsystem (see Fig. 1). For example, the optimizer may detect that a restriction, or a part of it, has the special property of a so-called box search, or that a restriction of a cartesian product has components of so-called equi-join type. In these cases, special stream operators perform the required function in a much more efficient way than would be possible without optimization. The algorithms used in these cases are outlined in Sec. 4.2.4, below.

Execution of operators in the example of Fig. 2: The Interval read and Append operators were introduced above. The $<=$ operator takes two compatible streams and produces an output stream of logical values True, False, or Undefined according to the outcome of the comparison. This stream is fed into the Restrict operator, in parallel with a stream of "roadsegment" values. Restrict acts as a filter which lets through only those roadsegment values which correspond to True in the other operand stream.
The result of the query is the set of roadsegments which correspond to roadstandards below 3. In this example, it is a consequence of the roadsegment attribute being designated as key of the Roads relation, that no value replications will occur in the roadsegment stream. The net generator does not recognize this, so it inserts a projection operator node. The projection operator in general sorts its input and removes non-unique tuples. However, analysis of relation key information is done in the projection operator, which thus detects that the sort step is not necessary.

4.2.3 The storage and access subsystem

The design of Cantor is in many respects centered around its unusual storage structure. Basically, it is a development of the concept of a fully transposed file\(^2\). A fully transposed file is simply a file stored by attribute rather than by tuple (record). Such structures have been applied in earlier statistical data base systems, for example the Rapid System developed by Statistics Canada. In Cantor, additional principles have been employed: ordering, dynamic data compression, and B-list structure.

The tuples of a base relation are ordered according to a sort order implied by the key information given by the user in the BASERELATION command. Thus, not only the set of constituents of the key is significant, but also their order in the key, although the latter only affects performance and not function.

While storing a relation, the system applies a data compression algorithm to fixed-length segments of each attribute. This algorithm reduces the number of bits used for each object, and suppresses the storage of repeating values within an attribute subfile ("run length compression"). The combined effect of sorting and run-length compression provides for compact storage of relation tables with multidimensional keys.

Compact storage is only one desirable property of the storage structure, however. Other requirements that must be satisfied are fast sequential and direct read access and ability to update, insert, and delete values with good efficiency and without the accumulation of garbage (self-organization).

The "ubiquitous B-tree"\(^6\) provided the framework of a satisfactory storage structure. Efficient read and write access and self-organization are well-known properties of B-trees, which were originally conceived for use as a one-dimensional ordered index structure for large direct-access files. Knuth\(^7\) describes in an exercise how the B-tree principle may be adapted for storing a linear list (an ordered sequence of values) in a
self-organizing structure in which items may be directly accessed. To combine this technique with efficient data compression requires that varying size "blocks" of data can be handled, since fixed cardinality segments are compressed into bit strings whose length is data-dependent (but of course never greater than the original segment's).

We designed a structure, which we called a B-list. All nodes of the tree have the same physical size, large enough to contain at least three blocks.

An internal node contains an ordered sequence of (key, pointer) pairs, according to the B-tree principle (when a B-tree is used for storing a linear list, the key value associated with a given pointer contains a value equal to the number of items stored in all subtrees to the left of that denoted by the pointer).

A leaf contains data values instead of (key, pointer) pairs. Since in Cantor textual data (literals and free texts) are represented in base relations by automatically generated integer codes, all attribute values may be represented as either integer or float data, which significantly simplifies the algorithms for data compression and B-list access.

Since the block size is dynamic, it is necessary to maintain a table of block addresses in each node.

One detail of the sequential read access procedure has to be pointed out to enable the reader to understand the discussion of search algorithms in Sec. 4.2.4.

When reading data expected to contain a high proportion of compressed runs, we use a variant of the read procedure which does not decompress runs, although other kinds of compression are transparent to the program calling the read procedure. Data with this property is accessed very fast since a run of data is represented by its value and length in all underlying layers of the system.

The storage and access subsystem is a development of a prototype system. The performance of this prototype was measured and compared with a commercially available data base system of good quality. The results showed that very significant performance gains could be achieved by combining transposed file storage and data compression techniques. For example, sequential reading of attributes could be performed 30-100 times faster in our system. About a factor of 10 can probably be attributed to the use of transposed files and the rest to data compression and careful interface design.
The database of our prototype system was only one fourth as large as that of the comparison system, partly because inverted files were not required with our approach. The data compression employed in the comparison system required advance specification of the number of bits allotted to each attribute for storing each of its values. When storing each attribute value in a full word, the comparison data base required 10 times as much space as that of our prototype, which sets no other restriction on the value range of its data than the word length of the computer.

When designing Cantor, we simplified the elaborate data compression algorithm of the prototype system, which is based on the dynamic programming principle. The resulting heuristic method resulted in less than 10% space increase for a test data base, while working at least 5 times faster.

4.2.4 The search and sort subsystem

Batory\textsuperscript{17) showed that search algorithms designed for use with transposed files could outperform commonly used techniques such as the use of inverted files (indexes) in a large proportion of cases. In\textsuperscript{18), this author presented theoretical and empirical results showing that a certain class of associative queries, called conjunctive queries, may be evaluated much more efficiently when the transposed file structure is combined with sorting and run-length data compression, provided of course that the data is such that sorting gives rise to value runs.

These results, together with Batory's study and our prototype system evaluation, led to the decision to forgo the use of indexes in Cantor, although an efficient implementation of inverted files existed in the prototype system.

The search and sort subsystem of Cantor was designed to take advantage of these observations as far as possible. It contains algorithms for internal and external sorting, duplicate tuple removal, conjunctive query search ("box search"), key lookup, equi-join, and union.

These algorithms are all designed to work one attribute at a time, which is not the conventional solution. As yet, the only empirical support for this design decision is the studies of conjunctive query search discussed above.

The search phase of the equi-join operation, important in a relation dbms, is so similar to conjunctive query search that it is to be expected that analogous results hold. Consider the following simple example:
Compute: *(R, S) WHERE [(r1 = s1) AND (r2 = s2) AND (r3 = s3)]

given R and S as follows:

<table>
<thead>
<tr>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>row number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>row number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Represented in run-length compressed form,

\[
\begin{array}{c|c|c|c}
\text{R:} & \text{S:} \\
\hline
r^1 & r^2 & r^3 & s^1 & s^2 & s^3 \\
\hline
(3)1 & (2)1 & 1 & (2)1 & (1)1 & 1 \\
(4)2 & (1)4 & 2 & (3)2 & (1)3 & 1 \\
(2)3 & (2)2 & 1 & (1)3 & (1)1 & 2 \\
(2)5 & (1)3 & 2 & (2)4 & (2)2 & 1 \\
 & (1)5 & 5 & (1)1 & 2 \\
(2)2 & 3 & (2)2 & 1 & 1 \\
(1)2 & 2 & & 3 \\
(1)3 & 2 & & 4 \\
 & 5 & & 2 \\
\end{array}
\]
Step 1: Output those row numbers of R and S for which $r_1 = s_1$; using $r_1^*$ and $r_2^*$ as input.

\[
\begin{array}{c|c}
1_1(R) & 1_1(S) \\
(1,3) & (1,2) \\
(4,7) & (3,5) \\
(8,9) & (6,6) \\
\end{array}
\]

Step 2: Output those row numbers of R and S for which $r_2 = s_2$ and $r_1 = s_1$ as represented by $1_1(R)$, $1_2(S)$:

\[
\begin{array}{c|c}
1_2(R) & 1_2(S) \\
(1,2) & (1,1) \\
(4,5) & (4,5) \\
\end{array}
\]

Step 3:

\[
\begin{array}{c|c}
1_3(R) & 1_3(S) \\
(1,1) & (1,1) \\
(4,4) & (5,5) \\
\end{array}
\]

Each of these steps consists of a linear scan of row intervals. Only in step 1 are the entire attributes scanned. Since the most significant sort key attributes are involved in this step, these attributes are also the most highly compressed ones.

Of course, when a join search is to be done on non-key attributes, one or both factors have to be resorted first. In such cases, sorting will dominate in the equi-join process, unless the resulting cartesian product has much greater cardinality than its factors.

5. FUTURE DEVELOPMENT

As mentioned in Sec. 3.1, we have now completed most of the rdbms part of the system. The independent evaluation being done by the Swedish Bureau of Statistics will soon be completed. The outcome of this evaluation will be an important factor in the decision to be made about future development. If the evaluation result is favorable, it seems probable to me that development will be continued, beginning with completing the rdbms part. The specifications also call for the development of data presentation, descriptive statistics, statistical analysis, bulk data format test, and backup subsystems. It would be premature to discuss here whether, or to what extent, the original plans will be followed.
6. SUMMARY AND CONCLUSIONS

A relational data base management system, designed for the analysis of complex statistical data was presented. The system shows several unusual design features, motivated by the intended application area which in many respects poses different problems from more conventional dbms applications.

The system has a powerful, formal query language whose concepts closely follow those of elementary set algebra. Its design is strongly oriented towards fast evaluation of complex queries. Basic design decisions of the storage, search, and query evaluation subsystems were made to this end. The use of ordered transposed files and data compression techniques provide both economic utilization of available storage and fast data access through mechanisms discussed in this paper. Query optimization is performed in several levels of the system.

7. ACKNOWLEDGMENTS

Many individuals and organizations have contributed to this project in different ways, and it would be unreasonable to list them all here. I am directly indebted for parts of this paper to my former colleague, Professor Stefan Arnborg of the Royal Institute of Technology in Stockholm and to my present colleague, Dr. Ilkka Karasalo of the Swedish Defence Research Institute.

***

REFERENCES


8) Stonebraker, M. et al: The design and implementation of INGRES. ACM Trans. on Data Base Systems 1, 3 (Sept. 1976), 189-222.


* * *
MR. J. GAMBLE CERN

Q ---› As an example of a query language aimed at statistical analyses, there appear to be some obvious omissions. For example, do functions for correlation coefficient calculations such as standard deviation and statistical significance, exist in the language?

A ---› No, but they could be added, and in particular, statistical significance would be best added as a FORTRAN program.

MR. J. GAMBLE CERN

Q ---› How do view definitions help? Will a user library of views be set up which perform such functions?

A ---› Yes. There will be a standard library of such functions set up.

MR. R. ZELAZNY - INST. NUCL. RES., OTWOCK-SWIERK, POLAND

Q ---› Could you elaborate on how a FORTRAN procedure can be incorporated in your system?

A ---› Programs must obey the rules imposed by the relational model of data, as it is implemented in our system. For example, instead of ordinary files, system-provided interfaces for reading and writing relations must be used. This is the database interface. There also has to be a user interface. The mechanism for defining this is not yet implemented in the intended way, but only specified. It would be described as a formal grammar, possibly using already defined "sub-grammars" for certain parts, for example when defining the syntax for parameters given by the user. The user interface program would eventually call one or more subroutines to effect the function(s) required with user-provided parameters. The actual FORTRAN code would be compiled and link-edited together with the rest of the system.

Example: S <- REGRESSION <R.y> ON <R.x1,R.x2, ..> ;

might be the user syntax for some command. When such a command is issued, a standard subsystem does the decoding, and passes the internal identifications of S,R,y,x1,x2,.. as parameters to the FORTRAN subroutine. Using the data base interface, the subroutine reads y,x1,x2,.. performs the required calculations, and again outputs them to S, which might be a result of "tuple" type, giving the regression coefficients. If more than one output object is required, some other command syntax should be used. Also, interactive sublanguages would be permitted, for example, the REGRESSION routine might ask the user for extra information.
THE GRAPHICS FUTURE IN SCIENTIFIC APPLICATIONS
- Trends and developments in computer graphics -

G. Enderle
Kernforschungszentrum Karlsruhe
Karlsruhe, West Germany

ABSTRACT
Computer graphics methods and tools are being used to a great extent in scientific research. The future development in this area will be influenced both by new hardware developments and by software advances. On the hardware sector, the development of the raster technology will lead to the increased use of colour workstations with more local processing power. Colour hardcopy devices for creating plots, slides, or movies will be available at a lower price than today. The first real 3D-workstations appear on the marketplace. One of the main activities on the software sector is the standardization of computer graphics systems, graphical files, and device interfaces. This will lead to more portable graphical application programs and to a common base for computer graphics education.

1. INTRODUCTION

In a scientific environment, computer graphics are used for a broad spectrum of applications, on a large number of different graphical devices, and using a variety of graphics software. The main application areas are:

- Data presentation: Data generated by experiments as well as by computer simulations have to be presented in a way suitable for interpretation by researchers. The presentations also serve for communications between different persons.\(^1\)

- Modelling: The models used as input data for simulation programs are set up under visual control by the operator. The visual representation is an aid to control the validity of the models. Computer graphics also provide the basic tools for the communication with the interactive operator and for his guidance.

- Process control: In some areas of scientific work, complex processes have to be supervised and operated. Process control graphics helps in human perception of the state of the process, especially if the process is in an abnormal state.

- Picture processing and pattern recognition: Data obtained from observations of the objects of the scientific research are digitized and made available to numeric processing. At various stages of this process, a visual representation is generated for perception and judgment by the human user.
The large amount of data to be handled, and the high complexity of the data are main problems when computer graphics are used in scientific applications. The result of a single experiment can easily amount to several hundreds of thousands of data. The data are often structured in a way that an easily comprehensible graphical representation is difficult to find. For instance, the display of different three-dimensional time-varying vector fields in a complex geometry is a difficult problem.

Another problem is a more general one: Nowadays's computer graphics systems neither have a standard user interface nor a standard interface to graphical devices. Thus application programs using computer graphics are not easily convertible or portable between graphics systems, between different installations, or between different graphical devices.

2. MAIN AREAS OF DEVELOPMENT

The developments going on in computer graphics will have a major impact on the use of computer graphics in a scientific environment. Main areas of the current developments in computer graphics are:

- Hardware developments: The main drive will come from colour raster graphics devices, colour hardcopy devices, the increased local processing power available in graphical devices, and devices capable of generating real 3D images.

- Software developments: Here, the progress is governed by the development of standardized interfaces to computer graphics systems. Not only the graphical systems themselves, but also application systems based on them will become more portable.

- Both the increased procession power of the hardware and the development of standardized graphical systems will increase the use of interactive techniques in scientific applications of computer graphics.

2.1 Hardware developments

2.1.1 Colour raster devices

The capabilities of colour raster graphics devices are steadily increasing, while at the same time the cost/benefit ratio is decreasing. In the years to come, this type of displays will in many cases replace the storage tube type displays. The resolution of the display monitors has reached about 1000 x 1000 addressable and displayable points (picture elements, pixels) in commonly available devices. These pixels are refreshed at a rate of more
than 30 times per second. Devices are equipped with fast parallel interfaces to the memory of the host computer, increasing the speed of picture generation considerably. The number of colours available for displaying pictures has increased at the same time. Devices that cost some ten thousand dollars can display 256 colours at the same time on the screen; these colours can be selected out of a range of more than 16 million possible colours. Continuously shaded objects can be displayed by such devices. Along with the increased resolution of the monitors and the increased memory capacity of the pixel store (leading to more displayable pixels in more colours), the local processing power of the devices is increasing. Microprocessors and random access memory, together with read only memory, allow for realization of complex functions in the devices themselves. Tasks like area filling, patternning or shading areas, generation of high quality text fonts, local segment store may be delegated to those more intelligent graphics devices. The local segment store for local transformation, change or deletion of segments, i.e., parts of pictures. Performing these functions locally reduces the amount of data that has to be processed and transmitted for picture generation and picture manipulation, and thus reduces the system response time for interactive applications.

2.1.2 Colour hardcopy devices

Along with the displays, hardcopy devices that can create colour hardcopies on paper, overhead transparencies, slides and motion pictures, will be available at a much lower price than today. Up to few years ago, the only colour hardcopy devices were expensive microfilm recorders. They are still used for high quality picture presentations on film or microfilm. For creating computer generated movies or slides, they offer the highest quality possible. However, their high price allows their use only in large computing centers. Lately, hardcopy devices have been developed for usage together with raster colour hardcopy devices, that copy the screen image onto paper or film. The range of such devices begins with the colour printers; they have few (four to six) colours and low resolution. The technology used is either needle-printing with a multi-colour ribbon, or multi-colour inkjet printing. An inkjet plotter is available that creates poster-size shaded colour-drawings of high quality. The other type of hardcopy equipment uses the method of taking a photographic image off a black-and-white screen. With (at least three) different colour filters, three colour images are superimposed on the film, thus creating a colour image of high resolution. The black-and-white screen of the hardcopy device is driven by the video signal of a colour raster display device. A number of different films and cameras can be used. The major film types are: 35 mm film used with a normal reflex camera (both for prints and transparencies), 16 mm film used with a movie camera, 3" and 8x10" polaroid film, and 8x10" transparent polaroid film. Polaroid film allows for instantaneous inspection of the generated pictures. More than one image can be placed side by side on one 8x10"
print. Fast generation of high quality pictures for presentation and documentation purposes, and the possibility to create animated movies at reasonable costs, is the main advantage of raster colour hardcopy systems.

2.1.3 Real 3D devices

In 1981, for the first time a real three-dimensional (3D) graphical display system that is commercially available was introduced. In Europe, the first presentation took place in September, 1983. For several years, display systems are on the marketplace that perform the reduction from 3D models (mostly wireframe models) onto a two-dimensional display surface in the device itself, by hardware. The perspective projection is under complete operator control. Experiments have been conducted with pseudo-3D images where two images are produced from two viewpoints in two colours (red and green). The observer has to wear red and green glasses for the perception of the 3D image. With the new 3D system, no glasses are needed. Although black-and-white at present, it is only a matter of technology to present coloured 3D images. When the observer moves his head, he really looks at the images from a different viewpoint.

![Diagram of principle of vibrating mirror 3D device]

**Figure 1**: Principle of vibrating mirror 3D device

Because of the novelty of the display technique, the basic method is explained in the following. The user looks into a mirror that reflects the image of a plane 2D screen. The distance at which the observer sees the virtual image of the screen is determined by his distance from the mirror and the distance of the mirror from the screen (see Figure 1). If, while the screen and the observer stay at their place, the mirror is moved, both the distance observer-mirror and the distance mirror-screen will change, and hence the virtual image of the screen will vary in depth. The principle is simple: the mirror is moved back and forth 30 times per second, and for every deflection of the mirror, an image has to be created on the screen that shows those parts of the 3D model that correspond to the actual depth in 3D space. Since for every cycle of the mirror, i.e., in 1/30 second, a
large number of 2D images have to be presented on the screen, an advanced technology is needed for the realization of the simple principle. Figure 2 shows the actual arrangement of screen and mirror. Instead of a plane mirror, a flexible mirror is used that is excited by a normal HiFi loudspeaker driven at 30 Hertz.

Figure 2: Configuration of 3D workstation (taken from 2)

2.2 Software developments

Here, the development is governed by the development of standardized interfaces to CG systems. Standardization has the following merits:

- The functional capabilities of a basic multi-purpose computer graphics system are described. This leads to the portability and device-independence of application programs based on the standard.

- Both the process of designing the standard and using it stimulated the development of a computer graphics methodology. The underlying model of a standardized system and the concepts applied designing it can only be defined and described on the base of such a methodology.

- Both the functional description of a computer graphics system and a computer graphics methodology needs a terminology accepted by all. The establishment of such a terminology facilitates the communication between researchers designing standards, implementers realizing such standardized systems, and users applying them for their applications.

- A standard puts computer graphics education on a sound common base. The number of concepts and systems that programmers have to learn will be reduced.
Once a standard in an area is established, new developments and improvements can be compared to the standard, and the progress they make can be evaluated in the light of a present standard.

2.2.1 Graphical Kernel System

The Graphical Kernel System (GKS) is the first international standard for computer graphics. It serves as a base for programming computer graphics applications. GKS covers the most significant parts of the area of generative computer graphics. It also lends itself for use with applications out of the areas of picture analysis and picture processing. GKS offers functions for picture generation, picture presentation, segmentation, transformations and input.

The main concepts of a graphics system are closely related to the tasks of such a system. Among these tasks are:

- generation and representation of pictures;
- routing parts of the pictures created in different user coordinate systems to different workstations and transforming them into the respective device coordinate systems;
- controlling the workstations attached to the system;
- handling input from workstations;
- allowing the structuring of pictures into parts that can be manipulated (displayed, transformed, copied, deleted) separately;
- long time storage of pictures.

An important aspect of a graphics system is the dimensionality of the graphical objects it processes. The current GKS standard defines a purely two-dimensional (2D) system. However, efforts are under way to define a consistent 3D extension. The major GKS concepts are outlined in the following sections:

Output
One of the basic tasks of a graphics system is to generate pictures. The concept corresponding to this task is graphical output. The objects from which a picture is built up are output primitives, given by their geometrical aspects and by the way how they appear on the display surface of a workstation. The way to present objects is controlled by a set of attributes that belong to a primitive (e.g., colour, linewidth). Certain attributes may vary from one workstation to the other. E.g., a line may appear on one workstation black and dashed, on the other one red and solid. These aspects of a primitive are called workstation-dependent attributes. In GKS, functions are present for the creation of primitives and for the setting of attributes (including workstation-dependent attributes).
GKS has output primitives that allow the convenient addressing of line graphics devices as well as special output primitives for addressing raster device capabilities. However, raster primitives will be displayed on line graphics devices as well; and line primitives will be displayed on raster devices. Line drawing primitives are: POLYLINE and POLYMARKER, the text primitive is TEXT, raster primitives are PIXEL ARRAY and FILL AREA, a special escape-primitive function is provided for addressing device capabilities, the GENERALIZED DRAWING PRIMITIVE (GDP). Figure 3 gives an overview over GKS primitives.

![GKS output primitives](image)

**Figure 3: GKS output primitives**

**Coordinate systems and transformations**

The application program can use one or several user coordinate systems that are related to the application for the creation of graphical elements. Output devices that are used for presenting the visual image of the elements, however, normally require the use of a device specific coordinate system. The routing and the transformation of output primitives along this output pipeline is performed by GKS. By using appropriate functions, the output transformations can be controlled by the application program.

**Workstations**

The output devices and several input devices are assembled into groups called graphical workstations. They usually are operated by a single operator.

A workstation is, e.g., a plotter or a display with a keyboard or a tablet connected to it. The workstation concept is one of the original contributions of GKS to the methodology of graphics system design. The graphical workstations of GKS are an abstraction of physical devices. An abstract graphical workstation can have one display surface and a number of input devices. Output can be sent selectively or in parallel to one or several workstations. Also, input can be obtained from different workstations.
Input
With input, the new dimension of interactivity is added to GKS. The actions of pointing, selecting, sketching, placing or erasing in a direct manner and the instantaneous system response to these actions are truly adapted to the human way of dealing with his environment.

Besides input that is specific for graphical applications (coordinate data or the identification of a part of the picture), GKS also handles alphanumeric input, choice devices like function keys, and value-delivering devices like potentiometer dials. GKS handles input in a device-independent way by defining logical input devices. Each logical input device can be operated in one of three different operating modes (REQUEST, SAMPLE and EVENT). Depending on the mode, input values can be entered by the operator and passed to the application program in different ways: one value at a time, requested by the application program and supplied by an operator action (REQUEST); sampling an input device irrespective of an operator action (SAMPLE); and input values collected in a queue by operator actions (EVENT).

Segmentation
The task of manipulating parts of the pictures leads to the concept of segmentation. A picture is composed of parts called segments that can be displayed, transformed, copied, or deleted independently of each other. Segments can be identified by an operator and their identification passed to the application program. GKS contains a very powerful segment facility, primarily by providing a device-independent segment storage, together with functions for copying segments to workstations or into other segments.

Metafile
The metafile concept results from the need to store pictures for archiving purposes or for transfer to a different location or different system. GKS addresses a metafile called GKS metafile (GKSM) that allows for long-term storage and retrieval of pictures. The metafile interface of GKS adds considerably to the flexibility of the system.

As part of the standard, the GKS document contains a definition of the interface to and from the GKSM. The contents and the format of the GKSM are described in an appendix that is not part of the standard. This separation was done in order to allow for a development of standardized graphics metafile independently of specific systems or devices.

Error handling
GKS contains an error handling facility. All errors expected during system operation are listed. A standard error handling procedure is provided. However, the user can replace it by his own error handling.
GKS levels

The GKS standard defines a powerful computer graphics system that includes output, input and segmentation. In many cases, not all of these facilities are needed within an application area. For this reason, a number of hierarchically ordered subsets, called GKS levels, have been defined. The lowest level merely offers graphical output, very much on the level of the widely used plotter-packages.

2.2.2 3D graphical system

For the majority of computer graphics applications a 2D-system, as GKS in its present form, will be sufficient. All output and input coordinates are two-dimensional. Some applications, however, require 3D-output primitives, like lines or areas in 3D-space, or even 3D-coordinate input.

Now, as the 2D GKS is a draft international standard, it seems that a straightforward extension of GKS to three dimensions (3D) is possible and desirable. Therefore, the ISO working group TC97/SC5/WG2 "Computer Graphics" decided on its Spring '82 meeting that:

"ISO TC97/SC5/WG2 recommends that a 3D subgroup be established with the following terms of reference:

a) to prepare a document setting out the scope, purpose, goals and underlying model of a 3D graphics standard that is an extension to GKS, by October 1982.

b) to start the process of obtaining a work item and a sponsoring body in November 1982.

c) to prepare a document setting out an outline of the functionality of such a standard, by February 1983".

Meanwhile the subgroup has started its work, first models for a 3D GKS-extension have been designed discussed. It can be expected that a 3D computer graphics standard will be available in 1984.

2.2.3 Graphics metafiles

2.2.3.1 Introduction

Graphics metafiles have been used since considerable time for storing and transmitting pictures. During the last years, efforts have been started to standardize graphics metafiles. Main reasons using graphics metafiles are:

- The graphics data must be displayable on a number of different display devices. The user wants to be able to choose among different plotters, output on microfilm or display screens for the representation of his pictures.
- Graphics data must be retainable for later use. They must be stored in a device independent way, so that the output device can be chosen after the generation of the data.

- Graphics data must be transportable, both by transmitting them over lines and by transporting a storage medium, e.g. a magnetic tape.

- Several sources of graphics data exist in most computing environments. Pictures produced as the result of picture processing techniques, of simulation computations or experimental records, using a number of different graphics packages, have to be merged into a uniform representation.

- Finally, some way must be provided for editing graphics data that have previously been produced and stored. Editing means: changing, deleting or adding parts of pictures, modifying the visualisation of parts of pictures, and merging of pictures.

The main impact of graphics metafiles results from the fact that they are able to interconnect various graphical devices and graphics systems in a standardized and straight-forward way. They allow cost-efficient use and sharing of expensive graphical equipment.

2.2.3.2 Proposals of graphics metafiles

In many application areas a variety of graphics metafile formats is being used, together with different graphics systems, and on different levels of functionality. The spectrum reaches from very simple, but very general designs containing few graphics primitives and attributes, to very sophisticated data formats for specific application areas. Examples for recent graphics metafile proposals are:

- GKS metafile, developed together with GKS by the DIN-subcommittee "Computer Graphics" since 1977;

- GSPC metafile, developed by the "Graphics Standards Planning Committee" of ACM-SIGGRAPH, 1979 4);

- Telidon, a graphics metafile format for routing graphical data to television sets connected to the VIDEOTEX-network, developed by the Canadian Department of Communication, 1980 5);

- AGF-plotfile, a graphics data exchange format developed by German research centers (and used as the base for the GKS metafile), 1976 6).
The GSPC metafile, the Videotex metafile, the AGF plotfile as well as lower levels of the GKS metafile are very basic formats for the description of pictures. Another metafile standard that is not a graphics metafile, but a product definition data file for the CAD/CAM field is IGES (Initial Graphics Exchange Specification), developed under the supervision of the US-National Bureau of Standards, ANSI-Standard 1981.

It provides a very complex, application oriented schema for describing CAD/CAM design objects together with their attributes and properties. Although IGES contains graphics entities, the scope of IGES was considered to be sufficiently distinct from the scope of graphics metafiles that up to now both developments were independent of each other. However, CAD-files will sure have an influence on the development of graphics metafiles in the future.

Within the International Standardization Organisation (ISO), working group TC97/SC5/WG2 "Computer Graphics", a metafile subgroup started work on standardizing a graphics metafile. The goal is to create a system-independent graphics metafile that can be used with a wide range of systems and devices. The subgroup can base its work on experiences with various metafiles and with GKS and its metafile. However, the most impact will come from the metafile group of the American National Standards Institute, ANSI X3H3. This group is currently developing a national US standard for a metafile called "Virtual Device Metafile, VDM". X3H3 is cooperating with the WG2 metafile subgroup; a joint effort has been started to reach an international metafile standard.

The current situation:

- ANSI X3H3 was encouraged to cooperate with WG2, to consider proposed changes to the current VDM draft, and to submit its work to WG2 by the end of 1982;

- agreement was reached to concentrate on a "Basic Metafile" that contains the basic functionality. The minimal set of required functions for a metafile was identified. Later, more complex metafile levels or additional modules can be defined;

- a formal grammar for graphics metafile was sketched, that gives a formal definition of the metafile structure, allows for generation of metafile parsers, and offers a framework for extensions;

- it was agreed that a metafile standard may have different bindings, i.e. coding formats and physical file formats, but that at least one binding should be specified as part of the standard. This could serve as a general communication format.
Based on the latest VDM-proposal and on the suggestions of the WG2-subgroup on metafiles, X3H33 will submit a proposal for a basic device- and system-independent metafile to WG2. I see no major obstacles for processing this proposal into an international standard. However, for higher levels of metafiles, the areas of intersection with other standards (e.g., IGES, OSI) will become larger, and thus agreements will be more difficult to achieve.

Metafiles are the means for transmitting pictures to different locations and devices within a network. With the evolution of both local area networks (LANs) and globe-spanning international networks, a metafile standard becomes an essential tool.

2.2.4 Device interfaces

Whereas GKS defines the application interface of a computer graphics system, the interface between the graphics system and different graphical devices is not yet standardized. This interface is called "Device-independent/device-dependent interface" (DI/DD) or "Virtual device interface" (VDI). GKS and its interfaces are shown in Figure 4.

![Figure 4: GKS interfaces](image)

The DI/DD interface separates all device-independent parts of a GKS implementation and those parts that are specific for a particular device type. The adaption of the information flowing over the DI/DD interface (in both directions) to the requirements of the hardware, firmware, and software on a
device (or workstation), is performed by the device drivers, or workstation drivers. If such an interface could be standardized, computer graphics applications were not only portable on the application interface level, but also plug-compatible at the DI/DD interface. The design of drivers could be standardized and thus greatly facilitated. Also, when graphics systems (presently: GKS) are to be validated and certified, automatic tests could be performed at a standardized DI/DD interface. Addressing a large number of different graphical workstations in a network would be easier if all these devices could use the same DI/DD protocol. Two groups are working on a standard at the DI/DD-interface: A certification working group sponsored by the EEC that bases its work on a GKS DI/DD interface designed by DIN, and ANSI committee X3H33, working on a VDI proposal that is to be used with a range of different graphics systems. In this field, too, international cooperation should lead to one standard supported by all. The expected time frame extends into the year 1984.

3. INFLUENCE ON SCIENTIFIC APPLICATIONS

All the developments, both in the hardware and the software field, will have an influence on the application of computer graphics in a scientific environment. The main trends that can be expected are summarized in the following sections.

3.1 Influence of the hardware developments

The hardware development will result in more colour, mainly in data representation graphics. Using colour opens a new dimension for displaying large amounts of complex data. Representations built up from lines will be augmented by representations built up from coloured areas. Main benefits from colour will be in the enhancement of human perception of data representations and of model images. For the display of transient processes, computer generated movies will be generated easier and cheaper and thus their use will increase. For some application areas, the use of real 3D devices for visualisation of 3D objects will start.

3.2 Influence of the software developments

The new graphical standards will place CG programming on a sound, uniform base. The GKS standard describes the user interface of a CG system that is capable of interfacing to the whole range of graphical devices, from simple plotters to highly interactive workstations.

Using a standardized CG system that addresses different devices in a uniform way, and using a standardized programming language, will make application programs using CG more easily portable and adaptable to different devices.
Since the programmers are trained using the standardized system (just as they are used to a programming language), programming and changing programs will be easier and faster.

Using a standardized data format for the storage and the transportation or transmittal of pictures will ease communication of graphical data between different institutions. The integration of standardized picture exchange procedures into the forthcoming local, national and international computer networks is under way.

3.3 Interactive computer graphics

Both the hardware advances, offering more local processing power and faster interactions at a lower price, and the GKS standardization, allowing to control user interactions in a standardized and device-independent way, will increase the use of interactive graphics. Interactive graphics need resources such as: readily available graphical workstations having bright, flicker-free, high-resolution displays and versatile input devices, reasonable (better: fast) response times and picture generation times, and high quality hardcopy devices. These resources are the base for the development of user-friendly application programs available on different computers and different workstations. The tools are available, and this will stimulate the spreading of interactive techniques.

4. COMPUTER GRAPHICS IS A COMMUNICATION MEANS

Computer graphics is a communication means. The future of computer graphics in scientific applications will, hopefully, ease and increase communication. Of course, the first thought will be the communication between computer and man. Computer-generated graphical output enhances the perception of computer generated data by the human user. With interactive techniques, man-computer communication goes in two ways, and with this kind of communication improved, the tool "computer" can be applied in a more efficient and pleasant way.

Computer graphics can, and will, also improve communications between men. A scientist can present the data he is dealing with by computer graphics methods in a better comprehensible way. Presentation and discussion of scientific results between different researchers working in one field is greatly facilitated by computer graphics presentations. Even more important is the potential benefit from computer graphics for overcoming the communication gap between the scientists and the "normal" citizen. The more complex and extensive the scientific and technological developments get, the more important becomes the dialogue across the borders of the scientific environment. Computer graphics are providing one tool that can help to reach this goal.
REFERENCES


* * *
QUESTIONS

DR. U. WAMBACH GSI/6100 DARMSTADT

• Q --> What are the present day chances, that the American manufacturers will support GKS as a standard?
  Comment added: There are press releases indicating the formation of a working association (INTEL, DEC, TEKTRONIX) which does not support GKS.

• A --> They are present in GKS presentations; but only the future will tell. As soon as ANSI sets the standard they might follow.

MR. PH. GAVILLET CERN

• Q --> Can you tell where the GKS software package is likely to sit. In the intelligent graphics environment or in a Main-Frame?

• A --> Can be on either one, depending on the local situation and user needs.

MR. R. BOCK CERN

• Q --> Does GKS emphasize the area of simple user interface in the standard? It seems that the powerful digital - to - analogue device, graphics, suffers seriously from the fact that special skills are necessary for its use. Skills which casual users are not ready to acquire repeatedly.

• A --> The standard foresees user interfaces, but some learning will be necessary to make use of its facilities. Real ease of use could be achieved only by restricting the freedom of representing internally the relevant user data.

MR. M. TURNILL BNOC GLASGOW

• Q --> Can the GKS standard cope with hardware designers who have more ideas for implementation

• A --> Yes, for about 5 years from now!

MR. W. MITAROFF INST. F. HOCHENERGIEPHYSIK VIENNA

• Q --> GKS is a breakthrough by defining for the first time a world-wide standard for a device-independent graphics package. But such packages have been existing before, e.g. the GD3 system for more than 10 years at CERN. That package has a
rather limited applications program interface (reflecting the capabilities of graphics devices from a decade ago), and achieves device independence by a metafile at the lowest level.

1) How does GKS achieve device independence when being able to support both "intelligent" and "dumb" graphics devices?

2) How does GKS, with its sophisticated applications program interface, avoid blowing up the memory space of the program it is supporting?

- A --- 1) GKS has a very flexible device interface - the workstation interface. Conceptually, every workstation looks to the application program like a very intelligent workstation. If the real workstations are intelligent, GKS will just pass functions down to them, e.g. "TRANSFORM SEGMENT NO i". For dumb workstations, GKS will have to simulate the workstation above the workstation interface, e.g. "clear screen, redisplay all segments (including the transformed one)".

2) First, GKS has properly defined subsets that are hierarchically ordered. So, for simple applications, the lowest GKS level with just output and output attributes can be used.

Secondly, the implementer has to design the system in such a way, that an application program using only a few functions is not burdened by system functions that are not used.
OLYMP -
EVALUATION OF MULTI-DIMENSIONAL DATA ON VAX 11/780

P. Egloff

1. Abstract

The software system OLYMP was designed for highly interactive evaluation of approx. 100 GByte/year of multi-dimensional data. It provides a wide range of data manipulation tools and is highly gaining by interactive graphic abilities. The system functions may be dynamically extended by user written routines.

2. The OLYMP System

The rise in complexity in experiments as in heavy ion physics forces a drastic increase of the amount of measured data. Up to 100 million bytes per hour have to be recorded in an average experiment. Data acquisition is done with a fixed time base and all related data in a time slot is collected in a so called event. Each event may consist of up to 200 single data. The software system OLYMP allows a controlled reduction of this high dimensional space to a meaningful subspace that holds the information of the actual experiment.

Data evaluation is done separated from the actual experiment. Furthermore, evaluation is done interactively on an OLYMP workstation that consists of an alphanumeric terminal used for dialogue and two graphic terminals, one storage and one colour raster terminal.

OLYMP is implemented on two VAX 11/780 processors from Digital Equipment; it is written in VAX-11 PASCAL and runs under the VAX/VMS operating system. The entire graphic software is written in FORTRAN IV. Both processors have 4 MByte main memory and a mass storage capacity of 500 - 1000 MByte.
3. **Purposes and Features**

The purpose of OLYMP is the evaluation of high dimensional data called analyte data and the accumulation of spectra. Both data structures can be displayed graphically.

OLYMP offers the following variety of general features:

- OLYMP is interactively controlled by about 100 commands.
- Command sequences may be concatenated to procedures.
- Command procedures may be edited under OLYMP's control.
- A subset of commands may be executed as a batch stream. This subset covers all significant commands when executed in batch.
- The user may extend the OLYMP function set dynamically with his own subroutines.
- All commands that take a reasonable amount of execution time may be interrupted. If more than one command is executed at a time, the user can specify the command that has to be interrupted.
- An OLYMP session can be suspended and later continued. This mechanism furthermore enables a warm restart in case of a crash either in OLYMP or in VAX/VMS.
- All resources such as disks, magnetic tapes and main memory are granted by OLYMP. Thus access conflicts and system overload is disregarded.
- The actual documentation is extracted from the source code. This also covers help information and user manuals.

4. **Systemarchitecture**

The software system OLYMP is divided into five processes in the means of VAX/VMS. This allows a clear software structure as well as concurrent activities. Because of semantics this concurrency is restricted to a subset of all OLYMP commands.

The administration of all hardware resources of VAX is done by a unique process to avoid access conflicts and system overload. The multiple process concept of OLYMP forces the use of inter process communication mechanisms. OLYMP uses the entire range of such mechanisms supported by VAX/VMS. Inter process communication is especially needed for both internal OLYMP administration data and external data, i.e., data that have to be evaluated. Run time aspects have to be regarded especially for this last type of data.

Figure 4.1 shows a scheme of the OLYMP system architecture.
Fig. 4.1 OLYMP system architecture
The purpose of the ZEUS process is general administration in OLYMP. ZEUS is unique within VAX/VMS. All hardware resources of VAX 11/780 are controlled by ZEUS. Every allocation of hardware requested by a user must be granted by ZEUS. The actual usage of hardware is kept in a ZEUS controlled table that is updated with each request. To ensure integrity of this table in case of fatal errors it is furthermore updated according to system state as defined within VAX/VMS. This update is done due to a fix clock rate.

To avoid system overload by too many concurrent OLYMP sessions every OLYMP session must be granted by ZEUS before startup of the individual session occurs. For this purpose ZEUS separates between normal and privileged users. The maximum value of users per class may be set during system generation. Whenever attempts to start a OLYMP session that would exceed the maximum value in its class the user request is rejected. The current implementation allows 4 normal users and 2 privileged users.

The purpose of the HERA process is to communicate with the OLYMP user. HERA owns the alphanumeric terminal of the OLYMP workstation.

Whenever a command is received it is checked for correct syntax. Faulty syntax is, as far as possible corrected in a user dialogue. Next the command is checked whether it is executable or not. If it is executable it is passed to the highest level execution routine. Actual execution may occur either in HERA or in one of the other OLYMP processes.
All user specific administration is done by HERA. Furthermore, HERA offers abilities for general user - OLYMP interactions and information about the current OLYMP state, the current session and external and internal data.
Each OLYMP user owns his own HERA process under VAX/VMS.

The purpose of the APOLLON process is to execute all commands related to evaluation of spectra. This covers mathematical evaluation such as computation of coefficients for spectrum approximation as well as general spectrum handling.
To the means of VAX/VMS APOLLON is a subprocess of HERA. Each OLYMP user owns his own APOLLON process.

The purpose of the THALIA process is to execute all graphic related commands such as display of spectra on analyse data. The administration of all graphic resources is done by THALIA. The two graphic terminals of the OLYMP workstation were owned by THALIA.

The showing of spectra can be done in several modes. One and two dimensional spectra may be shown. The showing of analyse data is done as a dot display of two components of an analyse data event. The screen of the graphic terminals can be split into up to 12 subpictures that may be used for separate displays.
Beside the static display of existing data THALIA is able to show data that are concurrently evaluated, i.e. created. This enables a highly efficient control of the correctness of data
evaluation. Furthermore, THALIA offers abilities for interactive
definition of reduction conditions with the help of the graphic
cursor.
To the means of VAX/VMS THALIA is a subprocess of HERA. Each
OLYMP user owns his own THALIA process.

The purpose of the URANIA process is to execute all commands
related to the evaluation of analyse data. This evaluation
consists of a user specified sequence of the following actions:

- Transformation : Correction of existing and computation of new
  components of analyse data
- Reduction : Selection of a subset of analyse data that
  fulfill specific conditions
- Accumulation : Creation of spectra by weighted counting of
  events of analyse data

A detailed description of the semantics of these actions is
given below. The execution of the user specified command is done
separately for each event of the analyse data set by a special
interpreter.
To the means of VAX/VMS URANIA is a subprocess of HERA. Each
OLYMP user owns his own URANIA process.

The communication mechanism used by OLYMP must be able to cover
the following purposes:

- Demand of actions that have to be executed in another OLYMP
  process
- Multiple process read and write access to internal
  administration data
- Multiple process read and write access to all data that have
  to be evaluated

The action demand in another process is done by using VAX/VMS
mailboxes. A pair of mailboxes exists for each possible inter-
process communication, one for each direction. Therefore, six
mailboxes exist in a OLYMP session for communication between HERA
and its subprocesses. Two mailboxes are used for communication
purpose between HERA and ZEUS. Those two mailboxes are
associated to ZEUS, therefore unique within VAX/VMS. To avoid
communication conflicts between ZEUS and multiple HERA processes
the VAX/VMS lock management services are used.
Both internal data, i.e. administrative data, and external data,
i.e. data to be evaluated, are located in VAX/VMS global
sections to enable easy multiple process access. The name of
these global sections is extended by a user specific code based
on the VAX/VMS UIC. Thus the global sections are qualified not
only by the group number as under VAX/VMS but by the member
number. However, this restricts the number of concurrent OLYMP to
one per UIC.
For some activities additional synchronisation is required. This
is done by VAX/VMS common event flag services.
To keep the OLYMP system independent from the actual hardware
configuration of the VAX processor VAX/VMS logical names are used
when referring to devices and files.
5. External Data Structures

External data may be given either as spectrum files or as analyse data files. Each file is a file by the means of VAX/VMS.

An analyse data set is a sequence of points in a n-dimensional space with n arbitrary but fixed for the whole analyse data set. Each individual point may be of an arbitrary subspace of that n-dimensional space.

Beside the file structure given by VAX/VMS each analyse data file has an internal structure. Therefore, additional administration data is part of an analyse data file.

A spectrum is a frequency distribution. It is characterised by its dimension, the range per dimension and the numeric value of its components (channels). A spectrum is represented as an array. The current implementation supports one and two dimensional spectra.

A spectrum file has an additional internal structure that allows to keep several spectra in a single VAX/VMS file. Therefore, additional administration data is contained in a spectrum file.

The internal structure of an analyse data file should first be described by using a Backus-Naur notation. Following this the components will be described in detail.

The symbols of the Backus-Naur notation were used as follows:

< > Brackets of meta symbols
 ::= Definition of meta symbols
 { } Repetition (incl. Zero)
 # # Brackets of comments
   (termination symbols by the means of the Backus-Naur notation)

<analyse data file> ::= <file header>
   {<workspace descriptor>}
   {<data block>>}
   <end of file header>

<file header> ::= $ general information about the entire file $

<workspace descriptor> ::= $ description of an analyse data set $

<data block> ::= <data header>
   <data list>

<data header> ::= $ description of a subset of an analyse data set $

<data list> ::= {<event>
Each analyse data file (\texttt{<analyse data file>}) by means of VAX/VMS is a Files-11 Structure Level 2 file. The additional internal structure enables the keeping of several independent analyse data sets in a single VAX/VMS file. Furthermore, it offers direct access to complete analyse data sets as well as to subsets of an analyse data set.

At the physical beginning (\texttt{<file header>}) and ending (\texttt{<end of file header>}) general information about the entire file is kept. User access to this information is gained by specific OLYMP commands.

A complete analyse data set is called a workspace. It starts with a description (\texttt{<workspace descriptor>}) of the above mentioned n-dimensional space.

The complete sequence of points in a workspace is divided into sufficient small blocks (\texttt{<data block>}). This allows direct access to a physical subset of the workspace. Each block consists of a description (\texttt{<data header>}) and the actual data (\texttt{<data list>}).

A single point (\texttt{<event>}) starts with a description (\texttt{<event descriptor>}) of the actual subspace followed by the actual data (\texttt{<event data>}). This descriptor is represented as a bit mask.

A single event data, also called analyse data parameter is represented as an unsigned integer. The number of bytes used for this representation may vary between one and four. This number is arbitrary but fixed for each individual analyse data parameter throughout the entire workspace.

To enable quick, direct access to components of an analyse data file most components are linked. Figure 5.1 shows the internal linkage of an analyse data file. The linkage is done using the VAX/VMS virtual block numbers of the Files-11 structure.

The internal structure of a spectrum file should first be described by using a Backus-Naur notation. Following this the components will be described in detail.
Each spectrum file (<multi spectrum file>) to the means of VAX/VMS is a Files-11 Structure Level 2 file. The additional internal structure enables the keeping of several independent spectra in a single VAX/VMS file.

The access to a single spectrum (<spectrum>) is done by the Files-11 virtual block number. This virtual block number is kept in a library (<library header>) that may hold up to 25 spectrum addresses. All library headers again are linked.

The actual spectrum data may be represented as INTEGER*2, INTEGER*4, REAL*4 or REAL*8 values. Two dimensional spectra are represented according to the conventions of FORTRAN IV.

Each group of data of the same data type in a spectrum or analyse data file is preceded by a description that holds information about the data type and the number of elements in this group. This ensures a complete self description of the entire file, thus enables data transfer in a heterogenous computer network including automatic data type conversion according to the different representation of data types on different computers.

All data that have to be evaluated by OLYMP and do not have one of the above mentioned data structures must be converted. This conversion is not part of OLYMP. It must be checked for each case individually if data conversion is possible. Conversion should be possible for a wide range of potential OLYMP users.

At the Hahn-Heitner-Institute data acquisition for OLYMP is done on a PDP 11/70 under RSX. Data storage is done on a RP06 disk (176 MByte). Data conversion is done on VAX with a special program.

6. Evaluation

Evaluation with OLYMP is controlled by commands. Each command is immediately executed as far as syntax and semantic checks do not signal any errors.
A OLYMP session is logically divided into the following three phases:

- **Preparation**: Allocation of all necessary internal and external hard- and software resources
- **Definition**: Description of evaluation semantics
- **Production**: Execution of a complete evaluation with a large amount of input data

All three phases are meant only logically and are transparent to the OLYMP user.

In the **Preparation** phase the user allocates required internal resources as well as external resources. Internal resources are called OLYMP-variables. Variables have to be declared before usage. Variables are identified by a unique name within an OLYMP session. The name is defined during declaration. A detailed description of the available variable data types is given below. The identification of external data, i.e., analyse data or spectra is also part of the **Preparation** phase. It is done with special OLYMP commands that assign a link between internal and external data.

The **Preparation** phase also includes commands for information about the current OLYMP state, internal and external data.

In the **Definition** phase the OLYMP user describes the semantics of the desired evaluation. All necessary transformation functions for computation of analyse data parameters are defined as well as required reduction conditions.

In the **Definition** phase a number of test evaluation is executed to demonstrate correctness of the selected transformation functions and reduction conditions.

The graphic abilities of OLYMP are mainly used within the **Definition** phase. They allow interactive definition of reduction conditions and an easy testing of transformation function correctness.

The actual evaluation is done in the **Production** phase using all data of the source analyse set. Evaluation control is done with a special OLYMP command that describes all necessary single actions (transformation functions, reduction conditions, accumulation prescriptions) in a binary tree. The complete evaluation is performed for every event of the input analyse data set separately.

The execution of this command is done by a special interpreter. Before actual execution starts the user specified command is converted into a pseudocode that contains a number of additional checks to minimize the number of not necessary computations for events that will be rejected within the flow of the user specified command.

The **Production** phase offers three basic actions for evaluation of analyse data: transformation, reduction and accumulation. The transformation of analysed data parameters covers all numerical operations with analyse data parameters. These
operations are divided into corrections and computations. Correction of an analyse data parameter means the assignment of a new numerical value to this parameter without changes of the dimensional structure of the event. Correction is used for calibration purposes.

Computation of analyse data parameters means creation of a new component in an event. Thus it implies a change in the dimensional structure of this event.

Transformation functions are specified either by arithmetic expression or by application of an available function. A transformation function may only refer to existing parameters in an event at the time of actual execution of this function.

Reduction means the selection of a subset of an analyse data set. Reduction conditions may be defined by both structural and numerical conditions. Within the production phase different actions, i.e. different branches of the binary tree describing the evaluation may be selected according to the result of the used reduction condition. Definition of reduction conditions is done as a part of the definition phase. It may be done either explicitly by specifying numerical or structural conditions or implicitly by using the graphic cursor. All reduction conditions may be logical concatenated.

Accumulation means the conversion of an analyse data set to a spectrum. The values of arbitrary analyse data parameters were used as the index of a spectrum channel. The content of the identified spectrum channel is incremented by the result of a weight function. If no weight function is specified, the spectrum channel is incremented by one.

Besides evaluation of analyse data, the evaluation of spectra is also part of the production phase. This evaluation is not done by an interpreter but by specific routines. OLYMP offers abilities for mathematical evaluation of spectra as well as abilities for general spectrum handling. The mathematical features allow the computation of coefficients for approximation functions. Three basic approximation functions are supported by OLYMP: exponential and Gauss-approximation and approximation using an explicit given function, defined by an spectrum. Currently only approximation of one-dimensional spectra is implemented. Furthermore, OLYMP offers all basic arithmetic operations (+, -, *, /) on spectra.

The general spectrum handling covers actions like shifts, compressions, projections and evaluation of reduction conditions on spectra.

7. Internal Data Structures

The conception of the OLYMP user interface is based on the OLYMP internal data types. These data types enable equal handling of
similar activities with different data structures. Thus OLYMP commands are easy to learn which is important due to the large number of commands.

Internal data are called OLYMP-variables. They have to be declared before used. During declaration the name of a variable is defined. Access to internal data is done by specification of the corresponding variable name. All names must be unique within an OLYMP session.

OLYMP supports a total number of seven different data types. Some of those data types have an additional substructure. The data type of an OLYMP variable can not be changed. Values are assigned to variables by the use of commands. The assigned value must match the data type of the variable.

8. Command structure

Communication with OLYMP is done through commands. The command syntax is similar to the syntax of VAX/VMS command language (DCL), thus an experienced VAX users should find OLYMP commands easy to learn.

There are three basic kinds of commands:

- Assignments: The assignment is the most simple form of an OLYMP command. The syntax is similar to assignments within high level languages or the DIGITAL Command Language (DCL).

- Commands: A normal OLYMP command consists of a command name, the actual data source and a list of command parameters. A command parameter consists of a keyword followed by an equation sign and the value assigned to this command parameter. Predefined abbreviations may be used for command names and keywords as well as full qualified names.

- ANALYSE: For complex evaluation of analyse data OLYMP offers a block-structured command similar to high level languages of the ALGOL-type.
9. Graphic Tools

Due to the complexity of evaluation of analyse data, a graphic aid is required. The display of two arbitrary analyse data parameters is done by using their values as either X- or Y-coordinates. This type of display is called a SCATTER-DISPLAY. It shows correlations between analyse data parameters, areas of accumulated data and enables interactive definition of reduction conditions. Due to the large number of possible parameter combinations, concurrent display of several pictures is necessary. Furthermore, graphic interaction is recommended. The graphic cursor may be used for:

- Definition of an interval on a single analyse data parameter
- Definition of a rectangular window on a pair of analyse data parameters
- Definition of a closed polygon on a pair of analyse data parameters

Reductions are used for evaluation control, i.e., different actions are executed whether or not an actual event is inside of the defined area.

In addition, a SCATTER-DISPLAY allows effective control of correct parameter adjustment for transformation functions. Transformations may be extremely complex, therefore graphic control is possible the best way of controlling the final adjustment of these functions. Live displays of concurrent active evaluation is an additional aid in evaluation control. Both analyse-data evaluation and spectrum accumulation may be observed, thus allowing on-line control of an eventual shift in statistics.

The display of finally evaluated data allow a competition of several experiments against each other as well as competition of experiments against theoretical computations. Furthermore, control of spectrum approximation is possible.

10. Special Features

Evaluation of an analyse data set may take from a few hours to a few days. Thus OLYMP offers the ability to suspend a session at most any time and to continue later on. The VAX/VMS mechanism used for this feature also allows a warm restart in case of a crash of either OLYMP or VAX/VMS. If a warm restart is necessary only the information of the last executed OLYMP command may be lost. The multiple process concept of OLYMP allows concurrent
activities. All administrative actions are executed completely independent from actual evaluation. An active command may be aborted on user request.

A great deal of attention during implementation was spent on a sophisticated command parsing ability. All necessary but absent command parameters were queried from the user. Most command parameters have an optional value that is assigned by OLYMP if the parameter is omitted by the user. In case of a syntax error only the faulty part of the command have to be reentered. Error messages were designed as in VAX/VMS, i.e., beside abbreviated code they contain clear text messages.

Repeating command sequences may be concatenated to command procedures. They can be edited in OLYMP. Furthermore, procedures may be submitted into an OLYMP specific batch queue.

The user may extend the wide range of built in functions of OLYMP by his own routines. Routines may be written in any native VAX/VMS language and will be linked dynamically against OLYMP.

11. Software Technologies Tools and Methods

The development of OLYMP was done in three major phases. In a functional specication the OLYMP system architecture and general requirements were specified. The command syntax was described using a Backus-Naur notation. The software company ACTIS implemented the processes ZEUS, HERA and URANIA. A formal specication for this process was done using the ISAC method. The processes implemented by the HMI, i.e., APOLLON and THALIA were specified in as PASCAL-like form after a detailed description of the OLYMP internal data structures. Implementation was done using all features of VAX/VMS and VAX-11 PASCAL. The entire documentation such as internal documentation, user manuals and help information is an integral part of the source code. Each routine is preceded by a documentation header. The extracted information is processed by a special word processing system to provide text layout.

12. VAX/VMS Environment

The hardware environment of OLYMP consists of the following components:
- 2 VAX 11/780 processors
- 4 MByte main memory per processor
- 6 diskdrives each with 176 MByte capacity (RP06); one diskdrive with dual port option
- 1 diskdrive with 516 MByte capacity (RP07) with dual port option
- 2 tape drives with 1600 bpi (TU16)
- 1 tape drive with 6250 bpi (TU78)
- approx. 30 alphanumeric terminals (VT100)
- 8 graphic storage terminals (TEKTRONIX 4014)
- 2 hardcopy units (TEKTRONIX 4631)
- 1 graphic colour raster terminal (AED 512)
- 1 graphic plotter (BENSON)

The implementation of OLYMP was done using the following software:

- VAX/VMS Version 2.5
- VAX-11 MACRO
- VAX-11 FORTRAN 77
- VAX-11 PASCAL

Furthermore, OLYMP was implemented using parts of software available at the Hahn-Meitner-Institut such as CESSNA and GRAFIX.

13. Future Aspects

The original concept of OLYMP contains some commands and features not being part of the current implementation. The user interface will be enhanced allowing different interaction between OLYMP and the user according to the users experience. Furthermore, the graphic abilities for spectrum display will be extended and high dimensional spectra will be supported.

14. References

/1/ J.C. Brun; R. Jory
Nouvelle Méthode de Répresentation sur Disque des tres grands Spectres multidimensionnels
Institut de Physique Nucléaire, Orsay / France, 1976

/2/ K. Keim, Ch. Labas, M. Richter, H.P. Rother, K. Schoeffel, B. Strand
Datenaufnahme/Analyse mit dem Online-Rechner am Muenchner Tandem-Beschleuniger Beschleunigungslaboratorium der UNI und TU Muenchen/Garching 1975

/3/ C.C. Maples

A Specialised, Multi-User Computer Facility for the High-Speed, Interactive Processing of Experimental Data Lawrence Berkley Laboratory, University of California Berkeley, California, USA, 1979

/4/ K.-P. Eckert, P. Esloff, S. Gipp

Spezifikation des Programmsystems OLYMP zur Auswertung mehrdimensionaler Messdaten HMI-Bericht B 341, 1980

/5/ N. Azak, A. Guenther, F. Jochum, F. Thoma

Formale Spezifikation der von ACTIS zu realisierenden Teile des Programmsystems OLYMP zur Auswertung mehrdimensionaler Messdaten Hahn-Meitner-Institut, 1981

/6/ W. L.-Bauerfeld

Beschreibung eines rechnerunabhaengigen Messdatenkopfes Hahn-Meitner-Institut, 1977

* * *
MR. J. GAMBLE CERN

• Q ——— How many simultaneous users do you support on one VAX?
  
  • A ——— Three to four users are allowed as a maximum.

MR. F. GAGLIARDI CERN

• Q ——— Would you please comment on why you have written 80% of the code in PASCAL and 20% in FORTRAN 77, rather than wholly in PASCAL?
  
  • A ——— The only part left in FORTRAN 77 is the graphics package because this was already available at the time of the implementation.

MR. W. MITAROFF - INST. F. HOCHENERGIEPHYSIK, VIENNA

• Q ——— Apart from the fact that "OLYMPUS" is highly computer dependent and not portable to other mainframes, I would like to make a general comment on its basic philosophy: It is a "Master" system (doing everything hidden from the user), steered by a syntax of its own, and leaving more complicated calculations to linked user routines. CERN already had such a system in the early 60's, called "SUMX" (of course, with more primitive syntax, no interactive use, no graphics), but we had good reasons to give it up and replace it in the mid-70's by a "Slave" system called "HBOOK", working from within a user-written FORTRAN program (which may be very small), and providing all functions via user-callable library routines. It was easy to add graphics ("H PLOT") and interactive usage ("HTV"). It will be transparent to the users if the graphics package is changed from the present GD3-based system to another one. It is easily implemented on a wide variety of mainframes. Why did you choose the "Master" approach, and force users to learn another high-level syntax language?
  
  • A ——— The problem was discussed with the users before the design of "OLYMPUS", and they preferred such an approach.

MR. D.R. MYERS CERN

• Q ——— Is any use made of sparse matrix techniques to analyse "events" with very many dimensions? For example, 100,000 events with 100,000 dimensions are possible according to an IBM research worker.
  
  • A ——— No.
COMPUTER AIDED DESIGN AROUND THE A 300 AIRBUS

Jean José AUDY
E D P and CAD/CAM manager, AEROSPATIALE, Aircraft Division, PARIS

ABSTRACT

More than thirteen years after the launching of the first AIRBUS development, a survey of CAD/CAM introduction and operation at AEROSPATIALE, Aircraft Division, is conducted.

The following points are reviewed:

- short description of AEROSPATIALE Aircraft Division
- technical level and limits reached with the Concorde program
- first CAD experience and the decision to proceed and generalize CAD/CAM techniques towards computer aided engineering (CAE)
- CAE basic concepts.

Then is shown the implementation of CAE in the structural, aerodynamic and electrical fields in response to the growing success of the various Airbus programs and in front of an everyday tougher competition. After a description of the to-day situation and a first estimate of CAE benefits, special attention is devoted to the difficulties encountered with both hardware and software used in the systems involved.

To conclude an attempt is made to define the respective roles of the system buyer and vendor in the CAD/CAM field.

1 - AEROSPATIALE

European aerospace manufacturer, Societe Nationale Industrielle Aerospatiale—more familiarly known as Aerospatiale—is a company which designs and builds a comprehensive range of products.

You may be familiar with the commercial success enjoyed by the Airbus widebody transport, the company's helicopters serving offshore operators (Aerospatiale is the second-ranking manufacturer in the world) and Aerospatiale's contribution to the defence effort with its tactical missiles (530,000 sold) and with its strategic missiles for the French nuclear deterrent force.

Aerospatiale also plays a leading role in such programmes as the Ariane launch vehicle and the Intelsat V telecommunications, TDF1 television and Meteosat weather satellites.

Turn around is 2.5 billion dollars and more than 35,000 people work in its 14 facilities.

The aircraft division, used as example in the following presentation has an approximate 33% weight in Aerospatiale activity (5.6 billion dollars turn around, 14,500 people in 5 different plants). Main productions are aiming at the civil air transport market (A 300, A 310, ATR 42 and in the past CARAVELLE, N. 262, CORVETTE, CONCORDE) but military transport (TRANSALL), trainers (EPSILON, FOUGA 90) and general aviation (RALLYE, TOBAGO, etc.) keep a permanent and significant share of the global activity.
2 - FIRST EXPERIENCE IN THE CAD/CAM FIELD

At the end of the sixties difficulties were encountered during the development phase of the CONCORDE program: the limits of classical methods were reached, typical cases being the design of the external shape of aircraft, and the cutting of real size templates needed as references for parts production. Improvements needed in accuracy, reproducibility, homogeneity and response time combined with the non-availability of more drafting and lofting specialists resulted in the first attempt to use computer oriented techniques: the well known APT language for geometry description and processing and a large Ferranti Cramic plotting table. The investment was small and the results were in accordance: lack of true design capability, reduced flexibility in handling complex shapes were more than compensating for the proven gains in accuracy and reproducibility. But feasibility was established and, in the absence of any adequate system on the market place, the decision was taken to specify and develop the SIGMA package (in French SIGMA stands for "integrated system for computer aided geometry"). Existing computers (CDC 6600) had to be used but a modern plotting table was ordered (GERBER, 48 x 6 meters). Operation of the system was planned for draftmen without data processing experience so debugging and tests were of particular importance as well as the existence of a complete documentation previous to any operational use. The system entered operation in November 1976, a CDC 777 interactive graphic terminal being used as working station for design work.

**Fig. 1** shows the general organization of the system, the most important points being the existence of a database and the use of a command language for man/system communication thus allowing the same dialogue to be used for interactive sessions or in batch mode.

**Fig. 2** shows the hardware configuration in the first years of operation.

Another important feature was the use of Bézier polynomial functions for curves and patches representation, a choice following developments made by Renault for car design and confirmed later on when it became a de facto standard in the 3D surfaces field.

3 - SIGMA RESULTS AND CONSEQUENCES

Several major points were made in less than two years of operation:
- computer aided design was recognised as a solution for some of our problems
- its use was not limited to above average data processing specialists
- storage and management of data so created was a critical point
- graphic representation of entities was possible and economically justified on existing computers
- there was a clear improvement in products quality.

Most important of all, our management, by seeing actual work on interactive displays, plotting tables, etc... was able to check by himself the veracity of these conclusions and a decision was taken to define a general policy in the CAD/CAM field and to use this technique as often as possible in order to improve productivity and remain competitive in front of an everyday tougher competition. It is fair to add that this move was without doubt facilitated by the sudden start of the A 300 AIRBUS sales and the resulting launch of the A 310 program!
4 - THE COMPUTER AIDED ENGINEERING (CAE) CONCEPT

This concept has progressively emerged from CAD/CAM use and studies of its impact on organization. It can be summarized as follows:

- CAD/CAM techniques (typically interactive work on alphanumérique and graphic data) provide a maximum benefit when applied throughout the complete industrial process, from marketing to operation support (continuous automated data flow - see fig.3)

- data storage and transfer are of paramount importance: data banks, managed by people in charge of creating the data must be able to deliver without delay any information required to any body implied in the industrial process

- each particular sector of activity (engineering department, production plant, after sales..etc..) can use its own CAD/CAM tool (e.g. a drafting system, a numerical programming system ..etc.) provided communication is possible - in both ways - with the Data Bank.

In terms of data processing tools three main functions are required:

- **Storage and management of large quantities of data (billions of characters of many different types)**:
  - market study results
  - preliminary project definition (geometry, characteristics, and performances)
  - detailed definition of the aircraft (geometry, parts list, wiring diagrams, performances, noise footprints, airworthiness requirements, etc.)
  - production data (N.C. programs, inspection results, etc.)
  - tests results
  - technical documentation (maintenance manuals)
  - etc.

- **Application programs and systems adapted to each particular function of the company branches**:
  - solid modelling
  - complex shapes design
  - mechanical parts drawing
  - printed circuits boards drawing
  - N.C. programming
  - text and image processing
  - etc.
- Communication facilities between application systems and data bank(s)

When achieved, the implementation of the function must allow a continuous flow of information covering the complete industrial process.

5 - CAD/CAM EVOLUTION WITH THE A 300 FAMILY

Since the first A 300 AIRBUS flight was made in 1973 the AIRBUS program development provides a good reference for CAD/CAM start and progressive integration in the aircraft division. (see fig. 4).

Designed at the end of the sixties and the beginning of the seventies the first A 300 has obviously not benefited of much CAD/CAM techniques although computers were largely used in the aerodynamic and structural fields as well as in the production phase (N.C. machines were generalized early in the sixties for the CONCORDE program and this technique was fully operational when AIRBUS production started).

In the engineering department most of computer work was batch processing and drawing boards had not yet suffered from cathode ray tube competition.

The following A 300 version, launched in 1971 (A 300 B4) was treated in the same way but in 1975 (launch of the A 300 Cargo version) the use of SIGMA was decided provided the system, which was been developed at that time, proved successful. At the same moment the engineering department was evaluating the use of CAD techniques in the electrical field.

In 1978, when the A 310 was given a green light, the situation was completely different: CAD/CAM was taken into account as much as possible, the only limitation being the availability of enough systems and trained people in the division. As a consequence:

- the external shapes of the aircraft were completely treated with SIGMA,
- all wiring diagrams and associated data (lists of cables, connectors, parameters...etc...) were produced using computer aided design
- about 10% of the mechanical parts drawings were produced on CAD systems
- the technical documentation is being prepared (A 310 entry into operational service: early 83) with CAD systems (see fig. 5 and 6).

The last major version - the A 300/600, enlarged cabin and A 310 systems technology - was launched in 1981. All modified components of the original A 300 were designed on CAD systems and the opportunity was taken to enter the existing A 300 wiring diagrams in the general CAD/CAM data bank.
It was the first occasion for a completely integrated CAE system to be tested in an operational environment in our company (design on CAD systems, storage and management in the data bank, transfer to CAM systems and word and graphics processing systems without manual handling of data).

Apart from the A 300 AIRBUS family it must be noted that another major program was launched in 1981: the A.T.R. 42, a commuter for regional airlines and a joint venture with AERITALIA. Taking into account the implementation of CAD/CAM it was decided to set an objective of 100% drawings and wiring diagrams produced by CAD systems for the AEROSPATIALE share (50% of the aircraft).

6 - TODAY SITUATION AT AEROSPATIALE AIRCRAFT DIVISION

6.1 - Technical fields covered

- Preliminary design

All preliminary projects are designed on an in-house developed system integrating most of the know-how of the division and continuously upgraded.

- Complex shapes

All external shapes of aircraft (being produced by AEROSPATIALE or coming from partners) are stored, handled and processed with the SIGMA system. It must be noted that the SIGMA format has been adopted as a standard of exchange for 3D surfaces by the AIRBUS INDUSTRIE partners (BRITISH AEROSPACE, MESSERSCHMITT-BÖlkOW-BLOHM, CASA, AEROSPATIALE). Fig 7 to 10 show some examples of AIRBUS components in SIGMA form.

- Parts design

COMPUTERVISION turnkey systems have been acquired since 1978 and about 50 working stations are used.

- Structural analysis

Most of the finite elements method programs can only be used in batch processing, but pre and post processing (mesh generation, loading definition, results analysis, etc.) are performed in interactive mode with the UNISTRUC package.
- Electrical systems design

List of cables, connectors and parameters are defined and managed with GICE\* which deals with the alphanumeric side of the problem while 40 COMPUTERVERSION workstations are used for wiring diagrams production.

- N.C. programming

COMPUTERVERSION systems and the AD 2000 package are jointly used for this task and another in-house development, "Panoplies", provide the nesting function needed for sheet metal parts with automatic optimization.

- Technical documentation

COMPUTERVERSION systems are used for modifications of the wiring diagrams and structural drawings needed in the maintenance manuals.

6.2 - Level of integration

A general data bank has been developed in the seventies, dealing first with SIGMA products and later on with wiring diagrams, mechanical parts drawings, parts lists, etc.. Implemented on the engineering department Cyber computers the data bank can be fed by all CAD/DAM systems used in the aircraft division and transfer data to them. (see fig. 11 and 12).

6.3 - Communication capabilities

Most CAD/CAM systems have a direct link with the data bank host. Meanwhile the obligation of using telephone lines provided by the Post-Office Administration is often a limiting factor for transfer rates and magnetic tapes are used when the amount of data to be transferred result in too long transfer sequences.

\* In-house development - C.I.C.E. stands for integrated management system for electrical circuits.
DIFFICULTIES ENCOUNTERED DURING CAD/CAM IMPLEMENTATION

Various problems had to be solved, some of them being still waiting for a better solution.

Implementation of a general data bank

The total lack of adapted tools on the market resulted in the necessity of important and expensive in-house developments. In particular, all data base management systems proved inadapted to this task.

Moreover the volume of data to be kept on-line (several billions of characters), the need for a strict access control system as well as the safety aspects (no more paper files) proved to be difficult challenges with standard hardware. One of our particular aeronautical constraints - keeping all data related to the design, production and support of an aircraft during its whole operational life - has not yet been given a satisfactory answer (safe storage of data over more than 20 years).

Non- compatibility of CAD/CAM systems

No system on the market can cover the whole range of applications. Interfacing several systems is a difficult task when the corresponding lists of entities and capabilities have only a rather small common set.

Communication problems

To the already mentioned non compatibility of systems one must add:

- distances: from a few to a thousand miles between data bank and systems

- data transfer rates: a typical drawing needs about 15 minutes to be transferred on a 9600 bauds telephone line.

CAE BENEFITS

Since an aircraft development takes approximately 5 years it is difficult today to compare the global performance of our division on a purely manual performance and with full use of CAE. Meanwhile some conclusions have already been widely agreed:

- better quality product design, the reduction in phases duration been used to study and compare more solutions
- easier co-ordination of the division branches: they all get the same data from the data bank, a particularly important point during the early development phase of a program when project features and characteristics are changing very quickly.

- suppression of many controls generally used to detect manual transfer errors

- cost reduction, mainly in the production phase and in operational support (up-dating drawings, diagrams, texts for 20 years or more!)

- more interesting tasks.

9 - CONCLUSION

Computer Aided Engineering is now a well defined concept, more and more industrial branches being committed to it everyday. But implementing it in a particular company is still a challenge:

- which data bank?

- how many different CAD/DAM systems?

- what kind of communication lines?

- etc.

After nearly ten years of activity in this field some answers are now becoming clearer:

- the particular C A E concept adapted to the needs of one company can only be defined by the company itself. In particular the data bank and the communication network organization are of vital importance.

- vendors are still far from offering a complete set of basic tools: communication facilities between different systems, data management and access control tools, better price/performance ratio in the graphics field.

But success will as always be depending mostly of man reaction in front of this new concept: educating and preparing all people involved is probably the most important point if one wants to maintain or improve the firm competitiveness by significant investments in computer aided engineering. Born with the A 300 in our company CAD/CAM is now one of the best guarantees of AIRBUS family development over the next decade.
Fig. 1 SIGMA organization

Fig. 2 SIGMA hardware configuration

**PRODUCT LIFE**

Fig. 3 Industrial process data flow
Fig. 4 AIRBUS family
Fig. 8 A300 project: cockpit and forward fuselage

Fig. 9 AIRBUS nacelle and pylon
Fig. 10 AIRBUS air conditioning duct

Fig. 11 Structural field organization
Fig. 12 Electrical systems organization

* * *

QUESTION

MR. J.V. ALLABY CERN

- Q --- The personnel who operate the CAD-CAM system interest me. Were they conventional designers who were re-trained, or did you have to hire a lot of new staff?

- A --- The question is important and re-training is a major problem when one introduces a CAD-CAM system. Typically about 90 hours over a period of 3 months were needed to train a person with no computer experience before he could work without close supervision.
PERSONAL WORKSTATIONS

I.M. Willers
CERN, Geneva, Switzerland

ABSTRACT

Personal workstations may be used to construct a distributed computing system in which the computing power of a processing unit (cpu) is dedicated to individual users and shared resources are accessed via a local area network. This talk will examine the requirements of some applications which exist in a large physics laboratory such as CERN and will compare such a computing system of personal workstations with a traditional computer installation.

The specifications of personal workstations will be discussed and special reference will be made to the Apollo Domain and the PERQ.

1 INTRODUCTION

Physics laboratories, such as CERN, traditionally use large number crunching computers for data analysis, maxi computers for graphical analysis of data and mini computers for accelerator control and experiments. In recent years the micro computer and bit slice technology have been used to simplify control equipment and to replace some of the functions of the mini computer.

As the cost of hardware has decreased, more attention has been given to simplifying the labour intensive job of programming. Programming has also become the most expensive, most time consuming and least reliable part of creating a computer system. CERN has a large IBM computing centre which is a great success because it provides extensive program preparation facilities.

Now, there is a new type of computing system appearing on the market which uses the micro computer and bit slice technology and has the sole aim of increasing a programmer's productivity. This new system is based on personal workstations and local area networks. The personal workstations contain their own central processing unit (cpu), local file storage and the ability to attach local devices. Shared resources such as printers, magnetic tape units and central file storage are accessed using the local area network.

This talk will endeavour to demonstrate in what respects the distributed personal workstation system can be used in the Physics environment. The PERQ and Apollo Domain systems will be taken as examples of state of the art personal workstations.

2 WHAT DO PERSONAL WORKSTATIONS OFFER

When the user of a PERQ or Apollo Domain sits at his personal workstation he has a moveable keyboard, a high quality bit mapped display that is capable of animation, a graphic input device, access to a Winchester disc with at least 25 Mbytes of storage and a floppy disc drive. He is also connected to a local area network which gives him access to other personal workstations, printers, tape drives, a common file base and any other expensive peripherals required by his user community.
Personal workstations have recently entered the commercial market and for many applications their price remains relatively high. Various models offer more or less the same facilities for the same sum of money. This ranges from ten thousand dollars for something slightly better than an intelligent terminal to sixty-five thousand dollars for the equivalent of a full mini computer configuration. The pricing structure includes reductions for educational or quantity orders.

The PERQ and Apollo Domain represent the top of the range of scientific personal workstations and will more than satisfy requirements for some applications. It is possible to buy a system consisting of five PERQs or Apollos and a network for the equivalent price of a Vax computer. In terms of processing power this is equivalent to two to three VAX computers. In order to increase the power of a personal workstation system, the user buys more personal workstations and adds them to the network. Increasing the power of a Vax computer system is more difficult.

2.1 Single user operating system

The PERQ and Apollo Domain personal workstations each offer the Unix operating system and one other. Apollo provide Aegis which is based on Software Tools (Ref. 1) and PERQ provide POS (Perq Operating System).

The personal workstation's software is constructed to enhance the productivity of the programmer. The software is specifically designed to give good source program maintenance. The latest in full screen editors, compilers, cross compilers and symbolic debuggers are available or being developed. New facilities include menu driven utilities and multi-tasking through a windowing mechanism. For example, a user can edit his source in one window, run his program in a second window and control the debugger through a third window. The programs take advantage of good interaction and the presence of a pointing device for selection from a menu. The presence of menus saves the user remembering how to access the facilities available in each program.

2.2 High Level Programming Language

The operating systems for the Apollo Domain and the PERQ are written in high level languages. Unix is written in C, Aegis in Fortran and POS in Pascal. On the PERQ the machine instructions are microcoded for good high level language performance. A machine code assembler for the PERQ is not available.

2.3 Local Area Network

A personal workstation is built around a local area network connection. The system is constructed to accept connections to other workstations, devices, computers or networks. The network connection is an integral part of the personal work station. It may be used for file transfer, remote job submittal and retrieval, electronic mail or to provide terminal access.

Apollo Domain connects to other Apollo personal workstations via a twelve megabit token ring and has external connections to Ethernet. A PERQ can be attached to a Cambridge Ring or an Ethernet.

The reliability of the system as a whole is improved by the amount of redundancy that has been introduced. If a personal workstation is not working the rest of the system will continue as before. If the local area network is not functioning each personal workstation can work alone. Maintainability is reduced due to the
geographical distribution of the system. This may be compared to the maintenance of terminals with additional devices such as central processors, Winchester discs and graphical input/output devices.

Personal workstations may be configured on a local area network or as a stand-alone system so that a user can develop his programs in a rich environment and then run on a single stand-alone personal workstation.

2.4 Raster Scan Bit Map Display

The PERO and Apollo Domain are equipped with advanced bit mapped displays. The setting of bits in a part of memory is directly mapped to the display. Special machine instructions which operate on the memory are used for very fast graphics including the possibility of animation. For good graphical working it is essential to have excellent interaction. This can only be provided by a dedicated processor.

The bit mapped display of the PERO and Apollo Domain is the size of an A4 sheet of paper and is capable of displaying many character sets. This type of display may be compared to printing dots on the page of a newspaper giving one the power to display graphics and illustrations. Updating the display screen is done by the dedicated processor operating on the bit map in memory. The PERO uses microcoded rasterop instructions to copy rectangular sections of memory and the Apollo uses a microcoded bit mover instruction in a programmed loop in order to do the same thing.

Minicomputers can only support a small number of people doing interactive graphics and the resulting response times are unpredictable.

2.5 Central Processing Unit

The Apollo Domain contains two Motorola 68000 chips. This is available with a four kilobyte cache memory and a separate floating point processor. The PERO uses the AMD2901 bit slice with four kilobytes of writeable control store. The AMD2901 is microcoded for efficient high level language implementation.

2.6 Memory

Half to one megabyte of physical memory is now considered standard. Access to this memory is via virtual addressing. The Apollo Domain can page its memory over the network so that a local disc is not necessary.

2.7 File Storage

Personal workstations normally include at least twenty-five megabytes of Winchester disc. The Apollo Domain allows easy access to all files on the local area network.

2.8 Interfaces

The PERO and Apollo personal workstations are equipped with RS232C connections. The Apollo Domain has a multibus option and the PERO provides a GRIP interface. A user may connect special devices to his system without interfering with others or he may make special devices become generally available to the user community on the local area network.
3 APPLICATIONS

This is an attempt at giving requirements for computer systems that are used by physicists. There will be an attempt during the talk to solicit requirements from the audience. Whatever type of system one tries to construct there are some requirements which are general. The main ones of these are cost effectiveness, maintainability or reliability, how easily the system can be tailored to the user’s needs etc.

In the text which follows the attention will be on specific needs associated with types of applications. Each section could be extended into a complete paper.

It will be seen that personal workstations can satisfy some of these requirements better than the traditional computer systems.

3.1 Control Systems

Control systems for accelerators are very special. They are constructed out of special fast local area networks and commercially available microprocessors.

3.1.1 Program Development for Micro Processors

In order to develop programs for a microprocessor the user needs to prepare program source, compile or assemble that source, link, load the resulting object module into a microprocessor and debug.

The problem of source maintenance may be solved by the provision of a reliable filing system, an editor, version management and access to a central library of routines. Compilation or assembly, for efficiency reasons, must take place on a computer containing a reasonably fast processor and fast access to file storage.

The microprocessor used in the development stage should be configured so that it is as near the finally required system as possible. Therefore both the source maintenance and compilation or assembly are best done on a machine other than the target microprocessor. It should be easy for a user to connect the microprocessor to such a system in a way which does not interfere with this aim. His connection at the host computer end should be simple and should not affect other users of the system.

A good debugger should debug using the symbolism developed in the user's program source. This is important for compilers and assemblers that produce relocatable code. The interactive symbolic debugger needs access to the symbol table produced by the linkage editor, compiler or assembler. The interactive symbolic debugger is best run on the same computer that linked, compiled or assembled the program. It is necessary to have a means of communication between the debugger on the host system and the target microprocessor.

A dedicated processor is needed if one is to keep track of selected locations, or take performance statistics.

The personal workstation can provide the processing power and the interfaces.

3.1.2 In Situ Micro Processor Fault Diagnosis

Debugging microprocessors is difficult due to the real time nature of their programs and their geographical distribution. In order to keep costs low it is not possible to have extensive debugging facilities in each microprocessor. So, as for the development stage,
it is desirable to have the same minimum software and hardware communication to another computer. The other computer must be able to connect easily to any of the microprocessors in the system. This implies portability and the ability to connect via a simple mechanism to the microprocessor.

A stand-alone personal workstation can be taken to the microprocessor or a local area network connection can be established.

3.1.3 Control System Display Console

The control system of an accelerator is difficult to control. There is maximum emphasis on the interface between the people who control and the control system. On the hardware side this includes special input devices, high quality colour displays, lots of computer graphics, a dedicated processor and a network connection. The display system must be fast when interacting with the operators. On the software side the handling of the graphics must be fast and easy. Special languages such as Nodal enable the operators to interact with the control system at a high level. In order to guarantee consistent response times it is necessary to dedicate a processor to each device.

This set of requirements describe a personal workstation.

3.2 Experimental Data Collection

The speed and the large amount of data which is generated during experiments require special techniques to read the data and to select that data which is to be kept for later processing. The processing of that data is complex so that large number crunching computers are required. The data storage must be accomplished by a computer with a means of accepting and storing large amounts of data. In order to check the validity of the data, some of that data must be selected as a sample to be sent immediately to a large number crunching computer where the parameters for the experiment are set up and checked. The results of sampling must be returned to the experimenter and displayed in a meaningful way.

The raw data sent back from the number cruncher may be further processed and displayed by the personal workstation.

3.3 Program Preparation for Number Cruncher

As for all program developments the main problem is one of source maintenance, compilation and debugging. While number crunching computers excel at fast real number calculation they are often lacking in other support functions. Front end computers are often required in order to control the stream of jobs entering the computer. It is therefore better thought of as a batch job device. This precludes any direct interaction which is normally necessary for editing source text and debugging. Such work should be carried out on a computer which is good at source maintenance and is connected to the main computer. The connected should enable the user to submit a job, examine its progress and obtain the results. The personal workstation should be capable of displaying the results in the most meaningful manner.

The problem of source maintenance may be solved by the provision of a reliable filing system, an editor and version management. The user would like as much checking of his program as possible before submitting his job to the number crunching computer. Local syntax analysis of the program source and computer generated job control language can be used to ensure that the program will at least get as far as linkage edition and probably execution.
The personal workstation is well equipped to provide a good user interface to an otherwise unfriendly system.

3.4 Analysis and Examination of Experimental Results

The analysis of experimental results needs a large number crunching computer as has already been described. However, once the results have been generated there are many techniques for examining them. Graphical representations of large data is necessary. This can be the simple generation of histograms to two dimensional representation of three dimensional data on particular tracks. The latter requires three dimensional rotation or true three dimensional techniques. Again this implies communication between a special purpose graphics processor and the number crunching computer.

A personal workstation with a special attached graphic's processor or detached array processor (e.g. PERQ in future) would replace today's expensive partially interactive, three dimensional graphical devices.

3.5 Documentation and Papers

The aim of a text processing system is to give the user high quality layout of documents with the minimum attention to such details as text input and alteration. Hence, text will automatically be aligned. The user will need to work in terms of the completed document rather than some cryptic equivalent. Any changes must appear immediately in the final form. The system must be capable of displaying all parts of a document including mathematical formulae, graphs and illustrations. All of this should be done interactively so that the user may experiment with changes to the layout etc.

Personal workstations solve exactly this type of highly interactive computing intensive task. The provision of menus removes the problem of remembering the awkward commands that exist in present day systems.

4 CONCLUSION

The personal workstation has the following advantages -

- Improved reliability against system failure due to greater redundancy in hardware and software.

- Improved flexibility in tailoring the system to the users needs.

- Fast and consistent interaction with the system.

- High quality bit mapped displays for good graphics and text processing.

- Easy connection of special devices from microprocessors to graphical input devices.

The minicomputer has the following advantage -

- Large numbers of terminals may be attached.

- The maintenance of the computer is centralised.

When your way of working is not as good as using an Apple home computer there is definitely something wrong. Professional personal workstations can increase your productivity!
REFERENCE


* * *

QUESTION

MR. M. TURNILL BNOC GLASGOW

* Q — To us in the oil industry, personal workstations look very attractive. However, we certainly need a much higher level of interfacing for the user than FORTRAN can provide. For each range of users, e.g. structural engineers, reservoir engineers, secretaries, a user-orientated station must be made available.

* A — (Mr. U.M. Wambach GSI/6100 Darmstadt) I see the workstations as a useful development tool in the development of physics instruments, if they satisfy these criteria:
  - Bring the software tools as close to the equipment as possible for debugging and system extension,
  - Provide a range of I/O capabilities to make interfacing easy,
  - Support a high-level system implementation language.
HMINET 2 - A LOCAL X.25 NETWORK CONNECTED TO THE GERMAN PUBLIC DATA NETWORK

Berthold Butscher
Hahn-Meitner-Institut für Kernforschung (HMI)
Department of Computer Science
Berlin, Fed. Rep. of Germany

HMINET 2 is a local network in the environment of a nuclear research institute. It is based on the X.25 technology and connects about 24 hosts. Services available for the inhouse users are: dialogue service, remote data access and file transfer based on a virtual file concept and a network administration and message service. This local network is connected as a 'private automatic data exchange' (PADX) to the German public data network. The paper describes the structure of the local and public network and the technology of the interconnection to the German public data network. As a pilot project HMINET 2 is funded by the ministry of research and technology.

1. Introduction

At the end of 1974 the Hahn-Meitner-Institut für Nuclear Research Berlin (HMI) and the Siemens AG started a joint effort to develop a star-shaped local computer network. The project was sponsored by the German ministry of research and technology (BMFT) and was planned for a term of four years. In 1976 the first version of the computer network, called HMINET 1/1/ was installed and at the end of 1978 this network offered a broad variety of services. Seventeen PDP 11, four Siemens 300 as minicomputers and three Siemens 7.7XX as mainframes were connected to a central switching node, performing packet switching in datagram technique. The system architecture of HMINET 1 is characterized by four distinct layers called link, transport, subsystem and application level.

HMINET 1 provides a set of services to the inhouse user:

- Timesharing access /2/
  Any terminal connected to host has access to the timesharing facilities on the mainframes by the DIALOGUE service.

- File transfer and record access /3,4/
  The file transfer (FT) and remote data access (RDA) service permits exchange of files and access to components
of files between all host computers including file data and structure conversion. This service also supports a remote job entry (RJE) and remote spool (RSP) facility.

- Interprocess communication /5/
  Processes running on different host computers can communicate and exchange data using the IPC service.

- Network administration /6/
  Network monitoring and host system status reports as well as user and operator message service are provided by the MAD facility.

New technologies and extended user requirements at HMI demanded a great deal on further expansion of the network services and performance criteria:

- the need of a factor ten of computing power in the next 3 years imposes the change of the mainframes.

- the requirement for online communication with other similar institutes in Germany and Europe by accessing programs or data using public networks.

- manufacturer independence by adopting network standards.

- new impacts and know-how in network technology.

- new network and realtime applications which have to be integrated into an overall data processing system in the institute.

Two strategies adopt to the above sketched demands:

- maintain the reliable network architecture, implement an improved soft- and hardware in the new mainframe, and provide the network with some gateway for the interconnection with the public network.

- change the network completely using a reference model which ensures a coherent network design and promises a functional compatibility between networks.

Analyzing the costs of the two solutions and considering the fact that by adopting a reference model for the software architecture a better integration would result the decision was to start the design of HMINET 2.

2. HMINET 2

2.1 Design Principles

HMINET 2 must be considered in the context of the following design principles:
- a network architecture adapted to a system architecture reference model accepted of national and international bodies. This guarantees a consensus in higher level protocol development.

- the usage of communication protocols recommended or standardized by international bodies. Computer manufacturers are forced to accept those recommendations and standards and have to integrate them into their product lines. Interdependencies between manufacturer and user can turn into the user's independence.

- a smooth change from HMINET 1 to HMINET 2. The accepted network services and their user interfaces should be preserved as far as possible to minimize user training expense and application program updates.

- new network applications are integrated in the second network. Depending on the limited data transmission bandwidth of a X.25 network the new applications are non time-critical services - e.g. file transfer, inhouse message service, access to data bases connected to public networks etc., rather than real time oriented applications in laboratory/industrial instrumentation and control environments.

Again, packet switching technology is used for HMINET 2, but now virtual circuits are supported as defined by the CCITT-X.25 recommendation /7/. An essential postulate for the project was to get all X.25 host interfaces and a X.25 switching facility from the different computer manufacturers. Therefore the R&D tasks are focused towards the definition of higher level protocols including new network services and the interconnection to the public data network. The installation of a 'private automatic data exchange' (PADX) and the contributions to the national and international standardization bodies on higher level protocols led to a grant provided by the German ministry of research and technology.

2.2 The computational and communication environment

In HMINET 2 the computational environment consists in 3 types of computing systems:

1) Minicomputers, performing process control and data acquisition in a real time environment of physical and chemical experiments. These minis are equipped with medium sized storage devices for short term data storage, and by alphanumeric and graphical terminals for program development and control as well as for graphical result analysis. The minis should have access to the mainframe hosts, to time sharing services, to very large data storage devices and to a message service for all participants of the network.
2) Mainframes as specialized servers, providing powerful time sharing and batch facilities, graphical evaluation systems, remote access to shared resources such as large file systems an data bases, magnetic tapes and high quality printers.

3) All kind of computer systems connected to the German public data network, which will be able to cooperate in the sense of coordinated or standardized applications. In this subject the hosts included in BERNET (Berlin Network for Research Institutes and University Computer Centers) are of special interest to the HMI users.

Table 1 gives a classification of the hosts which will be connected to the HMINET 2. The number of involved terminals, the operating systems and the main tasks are listed additionally.

<table>
<thead>
<tr>
<th>Host type</th>
<th>Number of host terminal</th>
<th>Operating system</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siemens S 7880</td>
<td>1</td>
<td>40</td>
<td>BS 3000</td>
</tr>
<tr>
<td>DEC VAX 11/780</td>
<td>1</td>
<td>12</td>
<td>VMS 780</td>
</tr>
<tr>
<td>DEC PDP 11</td>
<td>15</td>
<td>60</td>
<td>RSX 11-M</td>
</tr>
<tr>
<td>Siemens 300 R30</td>
<td>2</td>
<td>8</td>
<td>ORG-PV</td>
</tr>
<tr>
<td>DEC LSI-11</td>
<td>2</td>
<td>12</td>
<td>COM.EXEC</td>
</tr>
</tbody>
</table>

Table 1: Host classification

Summarized the inhouse computational environment of the HMINET 2 is as follows:

- Number of hosts: 23
- Number of terminals: 130
- Degree of heterogeneity: 4

The specification of the communication links and the packet switching facilities require a careful estimation of the expected network traffic. An evaluation was based on the traffic and utilization measurements done in HMINET 1, which has similar computational environment, on the experienced network user behaviour and on an user inquiry about expected future network usage and services. An average packet throughput and the corresponding necessary line speeds were calculated. An overview is given in table 2. Maximum packet length was adjusted to the recommended 128 octets.
<table>
<thead>
<tr>
<th>No. of lines</th>
<th>average packet throughput p/s for each line</th>
<th>peak values p/sec</th>
<th>line speed kbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>18</td>
<td>19.2</td>
</tr>
<tr>
<td>12</td>
<td>25</td>
<td>45</td>
<td>48</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>56</td>
<td>64</td>
</tr>
</tbody>
</table>

Total average packet throughput: 480 p/sec
Total peak packet throughput: 832 p/sec.

Table 2: Traffic characteristics of HMINET 2

2.3 Network structure

A networking structure of the star type serves best the identified needs for a more distributed computing environment. This structure recognizes the level of autonomy existing in a research institute community. Figure 1 outlines the topology of the HMINET 2.

![HMINET 2 Diagram](image)

Figure 1: HMINET 2 and the DATEX-P interconnection

Switching nodes for packet switching in virtual circuit technology are available for public networks since 1976. In a final selection only two of them fulfilled the set of HMI requirements, especially the packet throughput. With some additional features for private local usage and public network interconnection a Siemens packet switching processor of type EDX-P/8 as a 'private automatic data exchange' (PADX) is selected and installed. The EDX-P node is equipped with two minicomputers, one is responsible for switching (NN), the other one acts as the network control center (NCC). The lines and the link procedure are controlled by micro-processors.
2.4 Layered architecture

The purpose of the ISO reference model for open systems architecture /9/ is to provide a common base for the coordination of standards development and to enable the adoption of existing standards in perspective within the overall architecture. The reference model seems to be accepted internationally at present. It offers already a frame to develop and design networks. Considering these aspects, we proposed the following layering for HMINET 2.

Level 7: Application
- Remote terminal access, Remote data access, File Transfer, Remote job entry, remote spool,
- Virtual file, Message system, Network Management

Level 6: Presentation
- Message Link Protocol

Level 5: Session

Level 4: Transport
- X.25 packet level

Level 3: Network
- X.25 link level (HDLC LAPB)

Level 2: Link
- X.21 bis

Level 1: Physical

On the first three levels we adopted the recommendation X.25 provided by CCITT and proposed in the ISO reference model. We developed a "Specification of a Transport and Session Layer Protocol" /10/ based on the Message Link Protocol /11/ which fulfills the requirements of level 4 and 5 in the ISO reference model. On the presentation and application level we defined a Remote terminal access- (DIALOGUE), a File transfer- (FT), a Remote data access- (RDA), a Remote job entry and spool- (RJE,RSP), a Virtual File- (VF), a Network administration- and a Message System (MAD).

3. Local to public network interconnection

One of the reasons to fund the HMINET 2 project was to support the interconnection of local networks to the German public data network using a PADX. The following sections describe the features of the German public data network DATEX-P and the technique and problems of the interconnection.

3.1 DATEX-P: The German public data network

In 1978 the German PTT decided to install a public packet switching network based on the CCITT-recommendations X.25.
An international call for tender by the PTT led to the choice of the Canadian Northern Telecom processor SL 10 as an appropriate switching node. In August 1980 the DATEX-P network came into operation. For one year, test operation is offered free of charge. The initial installation of the network consists of 18 switching nodes for 2000 access lines. One additional node is for test and training purposes, another one serves the international connections to TRANSPAC, TELENET, DATAPAC, IPSS, starting in mid 81.

The DATEX-P network provides following features:

DATEX-P10: X.25 direct access with line speeds ranging between 2400 - 48000 bps, SVC and PVC, multi channels, reverse charging, closed user groups, subaddressing.

DATEX-P20: PAD-facility for asynchronous character-ori-ented terminals (X.28,X.3,X.29), direct access or access via telephone or DATEX-L network with the line speeds 300 and 1200 bps, SVC and PVC.

DATEX-P32: PAD facility for terminals compatible to IBM 3270 (in preparation)

DATEX-P33: PAD facility for terminals compatible to SIE-MENS 8160 (in preparation)

The charges for the use of the DATEX-P network are made up as follows (The distance has no influence on the amount of the charges!):

- monthly basic charge (depending on line speed and additional facilities)
- charge for the duration of the call
- charge for the volume of data transmitted.

3.2 Interconnection HMINET 2 - DATEX-P

Considering the situation at HMI, where a local packet switching network based on X.25 virtual circuits will be interconnected to a public packet switching network with X.25 technology, interconnection has to be provided by the network layer. This requires a more general network addressing scheme. Some possible solutions for this problem are discussed briefly in the following.

1) Usage of internodal protocols.
The internodal protocols in the public network (SL 10 switches) and for the local switch (EDX-P) are of different nature. Thus the expense for adaption is very high and
the existing internodal protocols, provided by the manufacturer do not cover a general standard.

2) Usage of the X.75 internetworking protocol. /13/
X.75 is designed to define 'terminal and transit call control procedures and data transfer system on international circuits between packet switched data networks'. The X.75 uses the recommendation X.121 'data network identification code' (DNIC) with 4 digits for network addressing. Only the last digit is usable to identify a specific network in one country. Therefore, X.75 seems not to be sufficient for extensive local network addressing.

3) Usage of the X.25 recommendation. /7/
Both the public and the local network are based on X.25 technique. Therefore, connecting the local network to the public by a X.25 addressing scheme is possible in two different ways:

a) An address structure simply derived from a hierarchical addressing scheme like in a telephone network is shown in Figure 2. The local network as seen from the public network is a single subscriber. The local network subscribers are identified by address extensions, e.g. using the call user data field. This can cause conflicts with present standardization. Moreover, the PADX processor has to fulfil additional non-X.25 address and extension mapping.

![Figure 2: Single DTE addressing](image)

b) The second possibility is to assign addresses to all of the local network subscribers according to the numbering scheme of the public X.25 network and to define the interconnection line as a multi-DTE address of the public network. This principle is defined as DTE-subaddressing. (Figure 3) Now, all local network subscribers do belong virtually to the total address space of the public X.25 network and the switching facility (PADX) can be considered as an integral part of the public network. No special
interconnection protocol has to be provided. A great advantage of this solution is that on one hand the tariff autonomy of the public network ends with the multi-DTE access line, on the other hand the local network can be seen in-house as an autonomous X.25 network. This addressing scheme even fits into network topologies, in which many local X.25 networks are connected to the public network. Calling one local network from another one is automatically supported.

Figure 3: Multi-DTE addressing

In accordance with the German PTT an address scheme and numbering plan including multi-DTE access by DTE-subaddressing is supported in the HMINET 2 and the DATEX-P network. Figure 4 describes the DATEX-P numbering plan and the variable subaddress structure. /12/

<table>
<thead>
<tr>
<th>national DTE address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
<tr>
<td>x y service area code</td>
</tr>
</tbody>
</table>

local subscriber incl. subaddress

x=4 DATEX-P
y=4 PAD service
y=5 X.25 service

A address
S: subaddress

Example: 45 3000 21707 subscriber address
in a local network, service: DATEX-P X.25, location: Berlin, address in the local net: 07 (subaddress), multi DTE address for the public network access: 217

Figure 4: DATEX-P address scheme
To support this kind of local/public X.25 network interconnection two changes has to be made in both the public and the local X.25 switching nodes. In the public nodes, all calls directed to the subscribers in the local network are to be mapped to the logical channels of the multi-DTE access line. This results in additional call routing software. It is necessary to provide enough logical channel capacity for multi-DTE lines to avoid a bottleneck when internetwork traffic increases. The local PADX switch has to convert the DTE behaviour of the multi-DTE line to the DCE behaviour as a local X.25 switch (see fig. 5). Addressing of internal subscribers in the local network can be done either by full DTE addresses or only by the subaddresses or by symbolic host identifiers. We preferably use symbolic addresses for inhouse communication and full DTE addresses for public network subscriber access.

Figure 5: HMINET 2 services and layer architecture

4. HMINET 2 services

HMINET 2 provides user services on application level both locally (intra-networking) and in access to public network (inter-networking). They can be classified in:

- local services: 'classical' applications like file transfer (FT) and remote job entry and spool (RJE,RSP), as proved in HMINET 1 and timesharing access (DIALOOGUE) based on recent standards, as well as new applications like network administration and message system;
public services: applications covered by international recommendations like terminal access (PAD service by DIALOGUE) or such which will be directed to new standards like virtual file service (VFS).

Figure 5 gives an architectural overview of the provided services and their layer relationship.

In the following these services are briefly described.

DIALOGUE-service: terminal access to host services /14/

The DIALOGUE service enables access to the operating system features of any host (timesharing, DE access, etc.) for all terminals. The terminals (see table 2) can be either in local host mode or in network mode. The specification of the DIALOGUE service is based on the CCITT recommendation X.28, X.3 and X.29 /15/. A host integrated PAD facility (X.3) serves the terminal interaction and a Host-X.29 facility serves the X.29 protocol and links from network layer (X.25) to the operating system. The PAD (packet-assembler-disassembler) can be adjusted to the terminal and operating system environment by PAD parameters and by a PAD command set. We added private parameters for editing functions.

The PAD commands supported are: Call, Read parameter, Set (and Read) parameter, Reset, Interrupt (and Discard Output), Clear and End of PAD session.

FT/RDA service: file transfer and remote data access /16/

To get access to distributed files and records in the heterogeneous network we specified a Remote Data Access protocol (RDA). The RDA system covers the layers 5, 6 and 7 of the ISO reference model. The RDA protocol is characterized by a remote connection phase and a data access and transfer phase. The RDA user interface supports both common characteristics and some special attributes of the involved data management systems (DMS) and provides following service primitives:

- RINIT, RDISC - remote connection management,
- ROPEP, RCLOSE, RDEL, RCNTRL, RFSTAT - remote file management,
- RGETF, RPUTF - file access and transfer,
- RGCTR, RPUTR - record access and transfer.

The incompatibility between the different data management systems, data types and file structures demands conversion which is additionally provided as far as possible.

A special FT-utility is available on top of the RDA system with a netwide common command language for convenient user interactions. This utility also supports remote job entry (RJE) and remote spool (RSP).
VF system: virtual file system /17/

The virtual file system is designed for a network with a high degree of heterogeneity or interconnected inhomogeneous networks. The VFS provides a consistent netwide virtual file structure with corresponding operations and a VF protocol in two sublayers. The MAP sublayer performs the mapping mechanisms between the local DMS and the VFS. The TRAN sublayer is responsible for the communication and coordination between the VF system entities in different hosts.

MAD service: network message and administration /18/

The MAD service with distributed components resides both in hosts and in the extended network control center (ENCC) of the switching node. This service is available only in the local network and represents two application oriented functions:

- Administration function for network, host and access line information, status report and control.
- Message service for text communication among network users, network- and host operators.

Actions provided in the administration function are:
- network oriented: get general network configuration, host location, service description and actual network status, enter, delete or update entries in the host table.
- host oriented: get information about host operating systems (OS) features, network services of dedicated hosts, actual OS status, and online devices.
- line oriented: activate and deactivate network access lines.

Following features are provided by the message function:
- send text items to a host (operator console) or to an identified user at a terminal
- send text items to a user's mailbox, ask for messages in user mailboxes and maintain mailboxes
- broadcast a text item and store it in ENCC for deactivated network subscribers
- ask for last broadcast messages stored in ENCC.

The message service of MAD uses the extended NCC to store user address mappings and text messages. With the chosen addressing scheme the message service user can reach hosts, terminals at hosts and identified users of the network.

All described network services provide additional 'user assistance function' (HELP) for user convenience.
5. Conclusion

The interconnection of local and public networks points out the absolute necessity of coordination and standardization of higher level protocols to support internetwork applications. These higher level protocols may also be applied if interconnected local networks differ from HMINET 2, like bus or ring networks. Then a gateway function may support the interconnection up to the network layer, but the upper layers still remain the same.

6. References


/2/ Eberle S, "Documentation of the subsystem DIALOG", TR 22 Project HMINET 1, HMI


/6/ Baum D, B Lausch, E Elsholz, "Specification of the MAD service", TR 24 Project HMINET 1, HMI

/7/ CCITT Recommendation X.25, "Interface between DTE and DCE for terminals operating in the packet mode on public data networks", Geneva 1980

/8/ Huber J F, "Siemens packetswitching system EDX-P" telcom report Heft 5, 1/78


/13/ CCITT Recommendation X.75, "Terminal and transit call control procedures and data transfer system on international circuits between packet switched data networks", Geneva 1980


/15/ CCITT Recommendations X.3 "PAD facility in a public data network"; X.28 "DTE/DCE interface for start-stop mode data terminal equipment accessing the PAD in a public data network situated in the same country", and X.29 "Procedures for exchange of control information and user data between a packet mode DTE and a PAD facility", Geneva 1980


/17/ P.-Zeletin R, B Butscher, "The Virtual File System" HMINET 2 TR IV/1

/18/ Ossefort M, B Butscher, "Design Guidelines for a Network Administration and Message Service", HMINET 2 TR VII/1 (in preparation)

* * *

QUESTION

MR. C. MAZZA - ESA, DARMSTADT

- Q --- Do you see any way of standardizing at the application layer (level 7 of the ISO-OSI model)?

- A --- ISO will present, presumably by the end of 1982, an application protocol for virtual file transfer which should give guidelines for standardization at the application layer level.
LOCAL AREA NETWORKS

P. Zafiropulo
IBM, Zurich

Pietro Zafiropulo is a staff member at the IBM Research Laboratories at Zurich. He joined IBM in 1968 and has worked on protocol validation, synthesis, integrated networks, network reliability, PABX's and speech recognition.

Local Area Networks are intended to provide improved communication capabilities such as high data throughput rates, low error rates and ease of connection among terminal stations and computers. These new types of networks operate within a limited geographical range like an establishment, campus or building and are owned by a single organization.

The presentation introduces these networks and the main techniques are described. It then proceeds to evaluate the main switching techniques as they apply to LAN ring and bus configurations. The preferred technique of token-ring distributed switching is identified.

(Paper not received)

The contents of Mr. Zafiropoulos talk was somewhat similar to the paper of Mr. Kummerle, which is therefore included in these Proceedings.

* * *

LOCAL-AREA COMMUNICATION NETWORKS — AN OVERVIEW

K. Kümmerle

IBM Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract: Local-area communication networks represent a new field of activity. In this paper we first describe three scenarios for the use of these networks, and then discuss various technical approaches. Particular emphasis is put on bus and ring systems with various media access control mechanisms. Specifically, we compare the delay-throughput characteristic of two access methods, carrier-sense multiple access with collision detection and token passing, and discuss some significant differences of bus and ring systems concerning wiring, media, transmission, and reliability.
1. Introduction

In the seventies, major activities in networking were focused on private and public data networks using either circuit and packet-switching technology or integrating both switching methods into a single network. In this time frame, also most manufacturers of main frames defined and implemented network architectures, e.g., [1-3], ISO started work on the reference model of the Open-System Interconnect Architecture [4], and CCITT recommended the X-series of interfaces [5].

Local-area communication networks represent a new field of activity which can be viewed as an extension to data networks for making packet-switching technology widely available to the in-house domain. Currently, much research and development work is being pursued in this field, both at universities and in industry. The term local refers to communication on the users premises, i.e., within a building or among a cluster of buildings.

A typical example of a system widely used today for local data communication is depicted in Fig. 1: sets of terminals are attached to control units and these are tightly coupled to a processor via I/O channels. The terminals are not intelligent and the necessary control functions are provided in the control unit and shared among a set of terminals. This leads to low attachment costs per station. There are, however, three shortcomings/problems with systems of this category: i) since the control units are channel-attached, they have to be physically located in the computer center for transmission reasons; ii) the star-type system structure requires an individual coaxial cable for each terminal. This and the previous fact may lead to the well-known problems of crowded cable ducts and difficult
system expansion, particularly for large installations, and iii) terminal switching is usually not available. This implies that if a user wants to work with software systems residing in different host machines, he has to be connected to more than one controller where each one is associated with a different host.

Switching would have to take place manually at the station. The undesirable features of the approach are the number of control units required and the cabling problem.

Another possibility to provide the flexibility referred to in iii) above is the remote attachment of stations. This means to make use of system architectures and network technologies as developed for TP applications. Fig. 2 is an example of such a network.

Long-haul or TP networks can be characterized by the following technical facts which, as we shall see later, give them different characteristics compared to local-area networks. First, communication lines represent an expensive commodity and therefore we have the design objective to optimize utilization of the transmission capacity. Second, the data rate of TP lines is in the range of 2400 bps to 50 Kbps. This means that the speed of processors either attached to the network or used in switching nodes is high compared to the data rate on transmission lines; the consequence being that processors in network switching nodes have time to perform functions which ensure error-free transmission and message integrity. These functions are performed in each switching node in the path between end points, i.e., hop-by-hop.
From the point of view of a system, irrespective of whether we have a network topology according to Figs. 1 or 2, current systems have a structure as shown in Fig. 3. Communication between terminal and host computer is provided by the front-end network, whereas communication between the host computer and its associated mass storage devices takes place through the backend network which in current installations degenerates to an I/O channel.

Transmission lines in the local domain, on the other hand, do not represent an expensive commodity and data rates in the range of 1 - 10 Mbps are readily available. The implication is that processors are no longer fast compared to rates on transmission lines. A potential consequence is that functions executed hop-by-hop in teleprocessing networks should be moved to the end points and executed end-to-end.

The availability of low-cost LSI/VLSI components is another driving force behind local-area networks from various points of view:

i) Network adapters can be provided at reasonably low cost, i.e., the cost advantage of using shared-logic controllers diminishes.

ii) Workstations/personal computers will have substantial amounts of processing capacity due to powerful microprocessors. However, it is unreasonable to expect the availability of sufficient mass storage, files, or high quality and powerful printers, e.g., laser printers, for each workstation for cost reasons. In the case of cost-effective hard disks
which could be afforded for each workstation, one might like to have them geographically separated from the workstation since they are noisy.

In Section 2, three scenarios will be discussed of how local-area networks can be used. Section 3 shows various possibilities of technical approaches. It also contains a description of several access methods for systems with either bus or ring topology. Finally, in Section 4, we provide some arguments for a system comparison.

2. Local-area network usage scenarios

The intent of this section is to illustrate how local-area networks are already being used or might be used in the future prior to discussing technical approaches. In the subsequent considerations, we deliberately do not address the question of whether these scenarios physically use the same or separate networks. It should be noted, however, that all scenarios represent a departure from the systems outlined in the previous section.

2.1. Terminal-to-Host Communication — Frontend Network

A first possibility to use a local-area network is for terminal-to-host communication. In this case, Fig. 4, the most important function provided by the network is allowing terminals to select a host machine, i.e., a capability generally not available in systems according to Fig. 1. Furthermore, the local network solves the problems of additional control units and of additional cabling between control units and stations. It should be pointed out that in this scenario, emphasis is on system flexibility rather than on exploitation of the high-speed communication facility. In this sense, the scenario of Fig. 4 can also be viewed as the migration of current equipment and applications to a
new network whose full functions can only be exploited by new equipment and new applications.

There is a problem which needs careful consideration: Should a terminal be attached direct to the network or through a control unit which it shares with other terminals? Apart from cost considerations, the answer will also be determined by whether one has the concept of a general wall plug in mind associated with the capability to readily move stations from office to office, i.e., to dynamically change the configuration during operation of the system.

2.2. Backend Network

Direct-access storage devices (DASD) are unintelligent in current systems, Fig. 3, and are tightly coupled to the I/O channels of their respective processors. Given a local-area network of adequate bandwidth, processors can be attached through serial I/O channels and communicate with DASD's through the network and also share them, Fig. 5. This implies some intelligence at the DASD for network-access purposes and for the execution of protection mechanisms. Besides the communication between processors and DASD's, a backend network can also carry channel-to-channel traffic. The system described in [6] is an example of a local-area network supporting the latter.

![Diagram](Fig. 5. Local-area network as backend network)

2.3. Client-Server Network

The two preceding scenarios are powerful generalizations of today's system structures. The client-server network [7] shown in Fig. 6 represents the scenario currently being most discussed, particularly in the context of
office communication. The stations attached to the network are assumed to be intelligent workstations. Compared with the configuration described in Fig. 1, the attachment costs per station — as already mentioned — will only be slightly higher and be fully justified by the additional functions made available through the network.

These workstations, called clients, communicate through the network with functions, called servers, which can be implemented in a centralized or distributed way. Some examples of servers follow. For many workstations it will not be cost-justified to have a private data base. In this case, they will have to share the file system with other workstations. Key element of the file system is the file server which on behalf of the workstations stores and retrieves information. The transport of information between workstation and file server is achieved through a file-transfer protocol. Compared with the structure of current systems, Fig. 3, this scenario has remarkable, new features. Previously, the station had its application resident in the host computer where it was executed in a time-sharing mode, and the host computer communicated with the unintelligent direct-access storage devices. Now,
these functions are split. Application processing is performed in the in-
telligent workstation, and the intelligence in the host computer, required
for controlling and communicating with the DASD's, got moved to the file
server. The scenario shown in Fig. 6, in this sense, represents a step away
from the world of time-sharing, potentially just as significant as the step
from batch processing to time-sharing. We should like to point out, however,
that in our opinion, this new scenario will not decrease the importance of
the big data-processing machines.

Similarly, workstations will share high-quality printers, e.g., laser prin-
ters, which are attached to the local-area network via a printer server.
Communication takes place only between workstation and printer server, most
likely using the same file-transfer protocol. The printer server then has
files printed on behalf of workstations. Other examples of servers are an
electronic-mail server which allows workstations to exchange mail, and a
name server with whom all stations have to register when they join the net-
work. The name server allocates addresses to stations which register, and
resolves names into addresses upon request.

We conclude the discussion of the client-server network with another func-
tional capability it can support, Fig. 6. We assume that there are work-
stations with a common architecture but for cost reasons be optionally
equipped with different amounts of RAM, diskettes and/or hard disks. Based
on their actual configuration, they will depend on a central processing fa-
cility which will: i) download the appropriate software packages into the
workstation through the network, or in other words customize the workstation,
and ii) perform functions on behalf of the workstation.

Networks for process-control applications [8] or for use in the production-
floor environment also fall into the category of local-area networks. Their
special requirements and characteristics and how they relate to the scenar-
ios discussed above will not be addressed here.
3. Approaches and classification

3.1. Technical Approaches

In this section, we shall briefly discuss possible approaches to local-area communication without making an attempt to predict which one will prevail in the future.

The best-known local communications system is the private automatic branch exchange (PBX). Most of the PBX's installed today are optimized for real-time voice, and use analog technology. The advent of computer-controlled private branch exchanges (CBX), implemented in digital technology and providing channels with the standardized speed of 64 kbps, paves the way for one approach to local-area networks.

CBX's have the potential to handle data, text, and videotex in addition to real-time voice. In this sense, it is an interesting idea to view them as an extension of the Integrated Services Digital Networks (ISDN) [9], currently being defined and studied for the public domain, into the local area. On the other hand, CBX's cannot be expected to cover the whole range of local-area communication since they are not suited to handle bursty computer traffic.

CBX's employ centralized switching with central control; the switching technology is circuit switching and from a topological point of view, they represent star systems. System proposals not based on CBX's but also using centralized switches are discussed in [10] and [11]. These switches use packet-switching technology instead of circuit switching, and therefore have the capability to handle bursty traffic.

Conceptually, the disadvantage of any centralized approach is the entry cost for small installations due to the use of centralized switching and control facilities. The alternative to centralized switching/control is to use a distributed control structure to regulate access to the transmission system. Topologies which inherently provide broadcasting, such as busses and rings, lend themselves readily to implement distributed access control,
see Fig. 7 and [12-14]. For both topologies, the functions provided in the set of network adapters A over which stations are attached, represent this distributed access-control system. It should be noted that all adapters are peer partners and that there does not exist a master adapter or a master station controlling access of the others.

This approach has important architectural consequences which become clear when we compare a multipoint system currently used for TP applications with data-link control procedures such as HDLC or SDLC [15,16] and a bus or ring system for local-area networks as being standardized by the IEEE Committee 802 [17]. These differences are represented in Fig. 8. In a multipoint system, the primary station has to poll the secondary stations before they can transmit a frame to the primary. From an architectural point of view, this clearly means that access to the transmission system is controlled with commands and responses of the elements of procedure of HDLC or SDLC. In local-area networks, on the other hand, the current data-link layer, level 2 in the ISO reference model [4], is split into two independent sublayers:
media-access control and data-link control. Media-access control is part of the distributed control structure mentioned above and determines, as the name suggests, when a station can transmit.

![Diagram of primary-to-secondary versus peer-to-peer communication](image)

Fig. 8. Primary-to-secondary versus peer-to-peer communication

It also allows peer-to-peer communication among all stations attached to the network and not only between primary and secondaries as before. The data-link control layer can contain any data-link control procedure and is no longer responsible for controlling access to the transmission medium.

The IEEE 802 standard activities on local-area networks in relationship to the complete ISO reference model is shown in Fig. 9.

![Diagram of ISO reference model and IEEE 802 project](image)

Fig. 9. IEEE 802 project and OSI
A single bus or ring system will not be able to serve establishments with a large number of attachments. Therefore, the capability to interconnect bus or ring systems, called subsystems in this context, is an important requirement, Fig. 10. Subsystems are connected to the high-level network via elements called bridges. The high-level network can degenerate to a few bridges interconnecting systems direct in case only a few subsystems have to be linked. Otherwise, the high-level network may be again of the bus or ring variety or it may consist of a star network. An important feature of a bridge is that it only performs a simple routing function and buffering, but does not execute higher-level protocols for performance reasons. Access to other local-area networks or to the public domain is provided via gateways.

In the remaining part of this paper, we shall confine the discussion to bus and ring systems with distributed access control.

3.2. Classification of Local-Area Networks

This section provides a classification of subsystems and a brief discussion of several access-control mechanisms. An obvious differentiation is according to topology where we distinguish between bus and ring systems, Fig. 7. Within these categories, we can distinguish according to access methods, see Table I.

Basically, the access can be either controlled, in which case no collisions will occur, or random, which implies that collisions of transmission attempts may happen. As a consequence, mechanisms are required to recover from collisions.
Table I. Classification according to access method

<table>
<thead>
<tr>
<th>BUS</th>
<th>Controlled Access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Token</td>
</tr>
<tr>
<td></td>
<td>Multilevel multiple access (MLMA)</td>
</tr>
<tr>
<td>Random Access</td>
<td>Carrier-sense multiple access with</td>
</tr>
<tr>
<td></td>
<td>collision detection (CSMA/CD)</td>
</tr>
<tr>
<td>RING</td>
<td>Controlled Access</td>
</tr>
<tr>
<td></td>
<td>Token</td>
</tr>
<tr>
<td></td>
<td>Slotted</td>
</tr>
<tr>
<td></td>
<td>Buffer/register insertion</td>
</tr>
</tbody>
</table>

3.2.1. Token ring

In a token ring [13, 14, 18], access to the transmission channel is controlled by passing a permission token around the ring. When the system is initialized, a designated station generates the first free token which is passed around the ring until a station ready to transmit changes it to busy and puts its information onto the ring. In principle, the information block can be of arbitrary length. The sending station is responsible for removing its own frame from the ring. At the end of its transmission, it passes the access permission to the next station by generating a new free token.

3.2.2. Slotted ring

In a slotted ring [18-20], a constant number of fixed-length slots circulates continuously around the ring. A full/empty indicator within the slot header is used to signal the state of a slot. Any station ready to transmit occupies the first empty slot by setting the full/empty indicator to "full", and places its information in the slot. When the sender receives back the busy slot, it changes the full/empty indicator to "free". This prevents hogging of the ring and guarantees fair sharing of the bandwidth among all stations.
3.2.3. Buffer insertion ring

In buffer insertion rings, the contention between the traffic to be transmitted by a station and the data stream already flowing on the ring is resolved by dynamically inserting sufficient buffer space into the ring at each ring adapter [21,22]. In contrast to the token ring where the sender is responsible for removal of the frame it transmitted, this function is performed by the receiver.

With regard to the point in time when a station is allowed to transmit one of its pending messages, we can distinguish between two operational modes: station priority and ring priority.

In the case of station priority, a station having a transmit request pending is allowed to transmit its information blocks immediately, if there is no block in transit at that moment. Otherwise, it must defer its transmission until the end of the transit block being currently transmitted. With ring priority, a station cannot begin transmission of one of its information blocks prior to the state when the insertion buffer is empty, i.e., all traffic already on the ring passed the station.

3.2.4. CSMA collision detection bus

The best-known random-access scheme for bus systems is carrier-sense multiple access with collision detection (CSMA/CD) as described in [12,18,23]. Under a CSMA protocol, every station ready to send must listen before transmitting an information frame in order to detect transmissions already in progress. If another transmission is already in progress, the station defers its sending until the end of the current transmission. Due to the non-zero propagation delay on the bus, carrier sensing cannot completely avoid the collision of information frames. A sending station can detect a collision by comparing transmitted with received data. In case of a collision, the transmission is aborted and the station reschedules its packet by determining a random re-transmission interval.
3.2.5. MLMA ordered-access bus

One possibility for ordered access control on a bus is the multilevel multiple access (MLMA) introduced in [24]. In its simplest version, the method works as follows. Information transmission occurs in variable-length frames with a structure as shown in Fig. 11. A controller generates start flags at appropriate time intervals indicating the beginning of a frame. A frame is divided into two parts: a request slot and an arbitrary number of information blocks. Every station attached to the bus owns one bit within the request slot. By setting its bit, a station indicates a request for transmission within this frame. At the end of the request cycle, all stations know which other stations are going to transmit within this frame. The actual transmission sequence is determined by a priority assignment known to all stations.

3.2.6. Token bus

Another controlled-access scheme on a bus uses a permission token in a similar way to that described for a ring system. This scheme is currently being studied by the IEEE 802 group [17]. Token access on a bus means that the station ready to transmit and which received the free token can send an information frame. At the end of the transmission, the station frees the token and passes an addressed token to the next station which should have an opportunity to transmit. It is important to note that in contrast to a token-ring system, the token has to be addressed since a bus system does not provide sequential ordering of the stations attached.

In addition to topology and access method, we can classify subsystems according to the transmission technique. We can differentiate between baseband and broadband systems. The term baseband is self-explanatory. In a broadband system, the available frequency spectrum is subdivided into different frequency bands where frequency multiplexing techniques are required to separ-
ate the bands. Ref. [25] provides an example of such a system. Fig. 12 schematically shows the principle. The bands can be used independently of one another, e.g., one band for CSMA/CD or token access, a band for voice, and other bands for video. The economic viability of these systems is determined by the modem cost incurred on top of the cost for implementing the access scheme.

Fig. 12. Principle of broadband system

4. Subsystem comparison

The important and difficult question which arises here is to compare subsystems. Obviously, it cannot be expected that a particular subsystem with a particular access method can be proven inferior for all conditions and applications. Comparing, rather means to understand the relative merits of various systems for a broad spectrum of parameters. The following parameters must be considered in such a comparison: Performance, transmission, wiring, reliability, availability and serviceability. For the sake of conciseness, we subsequently use the two most widely discussed subsystems, i.e., token access on a ring and CSMA/CD on a bus.

4.1. Performance

Two performance aspects are of primary interest: The delay-throughput characteristic of the media-access control schemes discussed in Section 3.2 and system behavior when the load approaches the saturation point. A fairly com-
Prehensive performance comparison was recently published [18]. In the following, we use some of the results reported in the paper referenced.

Figs. 13a, b, and c show the performance of token ring and CSMA/CD bus for two data rates: 1 Mbps and 10 Mbps. The general conclusions we can draw from these results are: i) at a data rate of 1 Mbps, both systems perform equally well; ii) if the data rate is increased to 10 Mbps, the token ring clearly has the better performance characteristic over a wide range of parameters. In Fig. 13a, the frame-length distribution is negative exponential with an average value of 1000 bits. A frame represents the entity transmitted by a station if it has access to the medium. The critical parameter which determines the performance of the CSMA/CD bus is the ratio of propagation delay and mean frame transmission time. Since the propagation delay is independent of the data rate, this ratio increases with the data rate. Theory shows [18] that a CSMA/CD bus behaves ideally as long as this ratio is sufficiently low. If for reasonable traffic loads it exceeds 2-5 percent, the increasing collision frequency causes significant performance degradation.

Apart from the early saturation point, the CSMA/CD bus has another undesirable property. In case of a collision, transmission is aborted and the station reschedules its frame by selecting a random retransmission interval the length of which is dynamically adjusted to the actual traffic load in order to avoid an accumulation of retransmissions. The high collision frequency at high load levels together with the retransmission policy causes the variation of the transfer delay to grow.

The practical consequence is the danger of stations becoming locked-out for an unpredictable period of time.
The general validity of the conclusions drawn above is supported by Figs. 13b and c. In Fig. 13b, all parameters are the same as before except for the length of the cable which is now 10 km instead of 2 km. The curve for the CSMA/CD bus at 10 Mbps illustrates the impact of the propagation delay, and confirms the importance of the ratio propagation delay and average frame transmission time. As a practical consequence, all CSMA/CD systems being discussed specify a maximum distance which is less than 10 km. Finally, Fig. 13c further demonstrates the robustness of the results. There, the frame-length distribution has a coefficient of variation of 2.

4.2. Other Considerations

The other parameters previously mentioned, transmission, wiring, reliability, availability, and serviceability are equally important but cannot be discussed in detail in this paper. Subsequently, we shall concentrate on several general properties and again limit the discussion to token ring and CSMA/CD bus.

In the design of a ring system, basically three problems have to be solved which are sometimes claimed to be weak points when compared against a bus: i) Each ring adapter contains a repeater and, therefore, the entire ring consists of a string of repeaters. The basic problem here is that a failure in any one repeater can disrupt the whole ring. ii) Clock coordination. Here the problem is that the set of repeaters has to agree on a common clock rate, and that this rate must result in an integral number of bit times of delay when going around the ring. iii) Initialization of the ring and recovery from errors without resorting to centralized control.

These three problems have solutions which work well in practice. Therefore, they are no longer considered as exposures of ring systems. Some key aspects follow. Problem i): Instead of trying to build very reliable repeaters, the basic idea is to bypass malfunctioning repeaters or inactive stations through bypass relays. These relays can be located either directly behind the wall plug or, more favorably, in distribution panels. In the latter case, the ring consists of several interconnected distribution panels, Fig. 14, which
can be placed at strategic and protected places in the building. Wiring from each panel to its stations is radial and the powering of the relays from the station is through the local lobe, see [13,26]. The solution with distribution panels not only solves the problem, but buys some advantages rings have over busses. Distribution panels represent several centralized locations for maintenance and reconfiguration and provide the potential to automate these functions. Trouble isolation and repair in a bus system, on the other hand, cannot be easily centralized. Also, they allow cutting out of the local lobes of inactive stations, a property which helps to gain distance. Furthermore, the concept of distribution panels with bypass relays facilitates the systematic prewiring of a building including the installation of wall plugs in all offices. CSMA/CD busses, on the other hand, require transceivers for each attachment which have to be installed very close to the bus for transmission reasons. Since transceivers do not represent low-cost elements, they make the prewiring of buildings and the installation of wall plugs in all offices fairly expensive.

Problem ii): A robust solution to this problem is described in [26]. Each repeater has a phase-locked loop which tracks the next active repeater upstream. The PLL filters are designed such that operation of the whole cascade of PLL's is stable and also that the accumulated clocking jitter is minimized to allow for the number of stations desired. In [27], further solutions are indicated.
Problem iii): As shown in [14], ring initialization and recovery from errors can be solved by introducing a monitor function which is available in each ring adapter. At any given point in time, only one monitor function is active and protects the circulating token against loss, permanent busy condition, or duplication. The monitor functions in other adapters are passive and supervise the health of the active monitor function. In case of a monitor failure, the passive monitor functions activate themselves, compete for the role of the active monitor, and after the contention has been resolved, one monitor function is again the active one.

In ring systems, transmission is unidirectional and point-to-point. This has several consequences, two of which will be explained briefly. First, rings allow media to be mixed easily, i.e., to use different media in different sections of the ring, e.g., twisted pair, coaxial cable, low-cap cable, see Fig. 14. This also means that rings can easily migrate to new transmission technologies, e.g., optical fibers. Bus systems do not have such flexibility. Moreover, the use of higher-speed transmission media, such as optical fibers, would create difficult problems. The first one is the performance problem CSMA/CD has at high data rates, the other the lack of a satisfactory technique for tapping an optical fiber and detecting a signal without diverting too much optical energy. Secondly, the analog engineering component in a ring system is small. The solutions to the ring engineering problems mentioned above can all be implemented in digital technology, and therefore can benefit from technological improvements in VLSI. The major problems for a CSMA/CD bus, on the other hand, are in the analog engineering domain. Some examples follow. i) A transmitter's signal must be receivable by all receivers on the cable. Similarly, each receiver must be able to hear every transmitter. The design must be such that under worst-case echo buildup and attenuation, any transmitter-receiver pair can communicate. Also, collision detection requires that a transceiver must be capable of detecting the weakest other transmitter during its own transmissions and of distinguishing the signals from the other transmitter from its own transmitter's echoes. To meet these requirements, transceivers have to be carefully designed. They have a significant analog engineering component. ii) Other
engineering problems are concerned with ground reference and power supply, the use of an unbalanced transmission medium, and the need of bidirectional repeaters to obtain the required physical distance. Details on these items can be found in [27].

5. Summary

The following key ideas have been highlighted in this paper:

i) Computer-controlled private branch exchanges and ring and bus systems with various access methods represent feasible approaches to solve the problem of local-area communication.

ii) The most likely application scenarios will encompass terminal-to-host communication, channel-to-channel and CPU-to-DASD communication, and the case where intelligent workstations operate in a distributed processing environment and access-shared resources through the network via servers.

iii) Ring and bus systems can employ a broad spectrum of access methods; the particular choice will have a significant impact on performance. The comparison of token ring and CSMA/CD bus shows that for data rates of 1 Mbps, they have the same delay-throughput characteristic, whereas for data rates of 10 Mbps, the token ring clearly shows the better performance over a wide range of parameters.

iv) Significant differences of ring and bus systems are in the areas of wiring and of providing centralized maintenance and reconfiguration facilities. The ring has a considerable amount of flexibility to mix media and to migrate to newly emerging transmission technologies, due to the point-to-point nature of its transmission system. Bus systems do not have this flexibility. Also, since rings do not require transceivers located adjacent to the transmission medium, prewiring of buildings and the installation of wall plugs in all offices can be cost justified. Furthermore, the use of distribution panels in ring
systems provides points for centralized maintenance and reconfigura-
tion. To have the same capability in CSMA/CD bus systems seems to be fairly difficult.

v) In CSMA/CD bus systems, difficult analog engineering problems have to be overcome. In ring systems, on the other hand, the difficult problems have solutions which can be implemented in digital technology and thus have the potential to benefit more from advances in VLSI technologies.

vi) Based on technical arguments, one cannot rule out either one approach. Therefore, it might well be that issues such as ease of installation, maintenance, automatic fault isolation and system reconfiguration, capability of migrating to new transmission technologies, and the way these systems are going to be used will dominate the technical arguments.

Acknowledgment

The author gratefully acknowledges useful discussions with W. Bux and P. Janson during the preparation of the paper, and their help in reviewing the manuscript.

References


* * *

QUESTIONS

MR. R. PIKE - BELL TELEPHONE LAB. MURRAY HILL, N.J.

- Q --> With hardware doing some of the protocol features, are you able to write software protocols that can use the hardware features? Protocol specification is difficult, and it would be nice not to duplicate features in protocols which will just be connected back-to-back.

- A --> The hardware protocol can do collision detection and error detection but not error recovery. A software protocol is required for that. I do not see any good solution yet to avoid duplication of protocol design and implementation effort until there is a protocol standard.

MR. D.R. MYERS CERN

- Q --> How do you see the market developing between BM & Texas Instruments versus Intel, Zerox & DEC?

- A --> I cannot comment on this question.
MR. J. AUDI - AEROSPATIALE, AVIONS, PARIS

- Q ---&gt; Considering the continuously changing industrial environment, is it not a practical problem to add new workstations in places where, unfortunately, distribution panels have not been placed when the ring was installed?

- A ---&gt; There are several ways to deal with this problem, and it has been anticipated as shown in these figures (transparencies re-displayed).

MR. P. VANN BINST - UNIVERSITE LIBRE DE BRUXELLES

- Q ---&gt; Can you explain the significance of the horizontal axis on your "systems evaluation" graphs of ring versus bus systems?

- A ---&gt; The scale shown is a properly normalized scale for this comparison. The detailed analysis was published in the Proceedings of the IEEE towards the end of last year.

MR. D. WILLIAMS CERN

- COMMENT ---&gt; Users want to connect terminals, minicomputers, and microcomputers together soon, and independently of the company which supplies the equipment. IBM has a duty to make its position clear on this question quickly (i.e. by the end of 1982).

MR. C. CURRAN CERN

- COMMENT ---&gt; I am delighted to see that IBM is moving towards a simpler physical interconnection system. We have enough difficulty at present connecting even IBM devices to IBM devices, but even CERN might manage to plug in a coaxial connector correctly! More strength to your elbow....!

MR. J. GAMBLE CERN

- Q ---&gt; Could you comment on the use of the token ring technology in high-noise environments? One point that is disturbing is that if the token is "HIT", then the whole ring is disabled for the timeout period, whereas, for a bus, it is only the passing message. This may be important for applications to Large Area Networks in experiments or control systems.

- A ---&gt; The only solution for ring systems is to incorporate redundancy or to use shielded cable.
THE PROVISION OF WIDE-AREA NETWORKING FACILITIES

Dr Barrie J Charles

Joint Network Team of the Computer Board and Research Councils, c/o
Rutherford Appleton Laboratory, CHILTON, Didcot, Oxon OX11 0QX, UK

ABSTRACT

The academic community in the United Kingdom is currently engaged in an extensive programme to provide comprehensive networking facilities within and among the 47 universities and numerous research institutions in the British Isles. The programme is described and the possible provision of similar facilities on an international scale discussed.

1. INTRODUCTION

Since the 1960s, funds for large scale computers in universities and academic research institutes in the United Kingdom have been provided centrally through organisations reporting to the Department of Education and Science. This central funding made possible the establishment of national or regional facilities designed to handle the needs of users which could not be met by an institution's local computing equipment. The distance between these users and the central machines encouraged the early establishment of data communication facilities.

Today the primary requirements for wide-area networks are seen as:

- access to large national or regional centres;
- facility or resource sharing between sites;
- database access;
- electronic mail;
- software distribution;
- a tool for multi-site scientific collaborations;
- a means for the itinerant worker to access his home base.

Many of these activities are also now becoming practical internationally and local networks are seen as satisfying similar requirements on a smaller geographical scale.
2. HISTORY

During the early to mid 1970s many separate networks were set up in the UK academic community. These were based either on the packet-switching techniques made popular by the advent of the ARPA network in the United States or on proprietary RJE protocols such as IBM HASP, CDC 200UT or ICL 7020. Growth during the 1970s led to a large increase in the number of point-to-point private lines and a number of arrangements on each site to access each distinct network.

The Joint Network Team was formed in 1979 as the result of the recommendation of a former body set up by the various organisations responsible to the Department of Education and Science. Its objective is to coordinate the evolution of networking so as to arrive at a situation where uniform arrangements are provided to allow any user to access any facility.

3. THE NETWORK MODEL

The scheme whereby useful network connections can be established without prior development work by the parties involved is now known as Open System Interconnection. To achieve a useful connection there must be an interconnection path between the systems and the systems must agree how they are going to interwork. The interconnection path will be made through one or more networks and the interworking will be achieved by agreeing on the high level protocols (or rules and procedures) to be used for eg, terminal access, file transfer etc.

The model for the provision of networks in the academic community to achieve interconnection between systems is illustrated in figure 1. It is a hierarchical scheme where one or more wide-area networks are used to provide a national system for linking together sites and each site is equipped with one or more local networks. In the short-term, systems on a site may be directly connected to the national network.

![Figure 1](image-url)
To achieve interworking, an agreed set of protocols is required. Protocols are usually built up in layers each forming a separate level of the communications hierarchy. There is now an ISO draft standard which divides this hierarchy into seven layers. These range from the physical layer which describes the electrical connection to the network up to the applications layer which will handle, for example, file transfer procedures. Unfortunately, although there is an international standard on how the hierarchy should be built up there is no accompanying set of protocols to be used for each layer. Because of the timescales associated with such complex international agreements, it is likely to be many years before there is such a complete set.

Using the expertise gained in the early 1970s, a number of protocols have been defined in the UK to fill the gaps left by the lack of international standards. The current complete set of protocols adopted by the academic community for Open System Interconnection is as follows:¹⁻⁶:

- CCITT X25 for packet-switched network access;
- CCITT X3/X28/X29 for terminal access;
- TRANSPORT SERVICE ("Yellow Book");
- FILE TRANSFER PROTOCOL ("Blue Book");
- JOB TRANSFER & MANIPULATION PROTOCOL ("Red Book");
- ARPA MAIL PROTOCOL ("Grey Book").

The last four protocols are regarded as interim and will be replaced once international standards are firmly established.

4. REALISING THE MODEL

The Joint Network Team has adopted the following approaches to ensure the model is turned into a practical reality:

1. The placement of development contracts for machine protocol packages and network components: the resultant products are then installed widely in the community using centrally-provided funds;

2. The protocols are included as mandatory items in Operational Requirements for all new major computing systems to encourage the provision of manufacturer support;

3. Detailed technical advice is given to the providers of services and the funding bodies in the evolution of communications arrangements;
4. Collaboration within the community is ensured by way of workshops, meetings, committees etc.

5. THE CURRENT STATE

National communications are currently provided by two separate wide-area networks: the public packet-switched network (Switchstream 1 or PSS) and the private X25 network set up by the Science and Engineering Research Council (SERCnet). These two networks are interlinked by a gateway which can ensure communication between a system on one network and a system on the other. There are currently 25 X25 connections to PSS from 25 community sites and around 100 connections to SERCnet from about 55 sites. Approximately 20 different types of system are connected.

The provision of local area networks is at a somewhat earlier stage of development. Local X25-based communications are generally being supplied using GEC 4000 based Campus Packet-Switching Exchanges and terminal concentrators (PADs) supplied by Camtec Electronics. Service networks of this sort are now installed on 7 campuses and planned for 16 further sites. Cambridge Ring ("Slotted Ring") networks are currently installed at 3 institutes and planned for 8 further sites.

The following systems have been connected to these networks and in general support most of the protocols listed above:

- Honeywell/Multics
- Honeywell/GCOS
- IBM/VM
- IBM/MVT
- IBM/MVS
- CYBER/NOS
- ICL 1900/GEORGE 3
- ICL 2900/VME
- PRIME/PRIMOS
- GEC 4000/OS4000
- DEC10/TOPS10
- DEC20/TOPS20
- VAX/VMS
- VAX/UNIX
- PDP-11/UNIX
- PDP-11/RSX11M
- PDP-11/RT-11
- LSI-11/RT-11

Most of the packages are in a form suitable for installation at other sites and are supported either by manufacturers or by universities under contract to the Joint Network Team.

Figure 2 shows an example campus, the University of Exeter, with X25-based local communications. Terminals connected to PADs can access either local hosts connected to the X25 switch or (via the gateway) any host connected to SERCnet or PSS. International access is also possible
through the connections between PSS and the public networks in other countries. File transfers can be initiated between local machines and remotely, and jobs prepared on one machine (the ICL System 4 or the PDP-11 UNIX) are routinely submitted for execution elsewhere (eg on the ICL 2900 or SERCnet hosts respectively).

6. CONCLUSIONS FROM UK EXPERIENCE

The UK academic community is implementing an open networking system which, although there is still much work to be done, already works. The following benefits are starting to accrue:

- flexibility in choice of new systems (because customers are not tied to one manufacturer);
- flexibility in assigning users to systems;
- reduced requirements for applications software by moving users, not software;
- effecting economies and bringing a versatile terminal closer to the user by divorcing terminals from systems;
- removal of media translation problems (file transfers over the network replace costly extra peripherals);
- improved human communications through electronic mail.

There are of course several problems associated with the approach. The lack of firm international standards has forced the community to develop much of its own software with the corresponding costs, long lead times and heavy support requirements. The evolution of protocol standards means that this commitment will continue for some time. How to ensure protocol conformance is also a problem which seems a long way from solution.
The fluidity in the Local Area Network scene has also been a cause for concern with user expectations far in advance of standards and products. The separation of the user's terminal from the computer by the network is also causing difficulties, particularly with some Digital systems where single character input with echoing by the host is essential.

However, despite these problems, we feel that essentially the right decisions have been made and that the UK academic community is now moving into the new era of "teleinformatique" with much of the ground work done.

7. INTERNATIONAL COMMUNICATIONS

Public X25 networks are now becoming available in many countries: an indication of those accessible from the UK is given in table 1. Many of these networks are interconnected to provide an international packet-switched service. This means that it is now practicable for computers in different countries to communicate through the medium of an X25 call. The general acceptance of the X3/X28/X29 protocol means that many hosts will also offer a useful end-user service for terminal access. Slight incompatibilities between the interpretations of X3/X28/X29 may however degrade the quality of the user interface offered.

Internationally approved protocols offering file and job facilities are still many years away, however. ISO does not expect to come up with draft proposals before 1984. This means that implementations are unlikely to be available before 1986 even if these timescales are kept to (ISO's track record is not good). The CCITT recommendations for Teletex can be used for electronic mail although they are not well suited for message exchange between computer-based message systems.

Table 1

<table>
<thead>
<tr>
<th>International Connections From UK</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUSTRIA</td>
</tr>
<tr>
<td>BELGIUM</td>
</tr>
<tr>
<td>CANADA</td>
</tr>
<tr>
<td>DENMARK</td>
</tr>
<tr>
<td>EUREUNET</td>
</tr>
</tbody>
</table>

COUNTRIES TO BE ACCESSIBLE BY END 1982

The UK is not alone in adopting interim high level protocols and useful international facilities may be provided through the use of protocol convertors. The European Committee for Future Accelerator's working group on data processing standards, sub-group 5, has been
looking into the possibilities for harmonising communications among high-energy physicists. This could involve the development of protocol conversion services and the promulgation of additional HEP protocol recommendations.

ACKNOWLEDGEMENTS

The author played only a small part in the work described in this paper which was accomplished through the efforts of many people in the academic community, some manufacturers, his colleagues in the Joint Network Team and through the support of the funding bodies and other organisations.

REFERENCES


2) Character Terminal Protocols on PSS (Revision 1), SG3/CP(81)6, Study Group 3 of British Telecom PSS User Forum (Feb 1981).


6) JNT Mail Protocol, C J Bennett, Department of Computer Science, University College, London (Jan 1982).

References 1-3 are available through PSS Marketing, 5th floor, Seal House, 1 Swan Lane, London EC4R 3TH.

Reference 3-6 are available from the Joint Network Team, c/o Rutherford Appleton Laboratory, Chilton, Didcot, Oxon OX11 0QX.

* * *

QUESTIONS

MR. R. DOBINSON CERN

- Q ---> What speed do present campus-wide networks run at ?

- A ---> The present maximum is 48 Kbits/s. This allows a surprisingly high activity.

MS. P. RIMMER CERN

- Q ---> Do you foresee proprietary difficulties when offering network access to facilities like software packages which have been purchased as single copies?
A --> So far the problem has not come up seriously. The problem might however, arise in the future. For example, the UNIX licensing charges are proportional to the number of terminals with potential access; that might be difficult to estimate and monitor.

MR. W. MITAROFF INST. F. HOCHENERGIEPHYSIK VIENNA

Q --> As long as file transfer protocols do not exist, do you think it reasonable to use an existing serial standard, adding some tricks. The host might well consider your local mini as a terminal, but locally information could be assembled on disk, using e.g., an X-11 interface.

A --> This is certainly possible, but many limitations would arise that either degrade the user service or have to be overcome by more and more 'tricky' software:
- Interspersed random messages to the terminal,
- Flow control problems at the terminal end or with an overloaded host,
- Restrictions of character sets for transmission, impossibility of binary files,
- Necessity to synchronise transfers with terminal use, i.e., no batch mode,
- Poor user interface.

MR. R. MARTY INST. F. INFORMATICS, UNIVERSITY OF ZURICH

Q --> Have you done any work on standardising network-wide name servers?

A --> Only in LAN-s, like the Cambridge Ring. Country-wide name servers will first be implemented as manual 'registration service'.

MR. J.M. GERARD CERN

Q --> Is it really your intention to connect terminals via networks and Triple-X rather than by circuit switching?

A --> Yes, in principle.

MR. M. TURNILL BNO GLASGOW

Q --> The attachment of all graphics and full-screen facilities directly to X-25 type networks might lead to overload. Would it not be safer to provide these facilities locally, using full networks only for long-distance communications.

A --> Yes, that is indeed an alternative.
THE DESY COMPUTER CENTRE: OBJECTIVES TODAY AND TOMORROW

P.E. Kuhlmann
DESY

Paul Egon Kuhlmann is trained as a physicist and is Head of the Computer Centre at the Deutsches Elektronen-synchrotron (DESY) in Hamburg. The central services at DESY include full data acquisition for all experiments.

The talk tries to give an overview on the services the computing centre at DESY gives to the HEP community today and which development can be foreseen in the near future.

The first part deals with the tools that are available in the computing centre, e.g. for Full Screen Editing, Interactive Graphics, Foreground Calculation, etc. The second part describes how the finite resources that a computing centre can provide are distributed among users with infinite wishes. Special methods developed to solve special problems (e.g. online support of experiments) are shown in the third part and the last part tries to define the role that DP-people can have in handing over ideas to the users.

(Paper not received)
MR. P.H. GAVILLET CERN EF

- Q ---» Comment on Distributed Systems. They will appear more and more necessary, if not needed in future HEP environment. Data sampling is a good example among others: Large Fortran programs have to be run on large data sample during data taking to provide fast responses. The only reasonable place for such a task is the data taking place.

- A ---» A solution avoiding the dispersion of many distributed different systems can be built using the concept of emulators which allow to run standard Fortran programs developed on Main Frames in any remote place with very high efficiency.

MR. C. JONES CERN

- Q ---» If I have understood correctly, you at DESY have exactly the same size of Mass Storage System and the same space for user disks as at CERN. In view of the difference in the number of users between the two sites, would you like the comment whether you are spoilt or we are badly treated.

- A ---» The 100GB MSS has been filled-up in 9 months we have asked for an extension which has just been signed.

MR. VAN BINST UNIVERSITE LIBRE DE BRUXELLES

- Q ---» Could you comment on the present possibilities and future concerning communication between the DESY Computer Centre and other laboratories in Germany and Europe?

- A ---» Networks surely coming and more important in the future. Some RJE connections existing now. Experimenting with link with MPI Munich. Easier to envisage links between IBM computers. Others may come later - more difficult - watch DFN project.
PANEL DISCUSSION

"Where do we go from here?"

A Panel Discussion chaired by Ch. Jones and I. Willers.

Panelists: H. Grote, M. Jackson (who had to leave early, but left a written statement), K. Jeffery, R. Marty, M. Metcalf, W. Mitaroff and R. Pike.

The following is a free transcription of the discussion for which the participants assume no responsibility. Contributions are not ordered chronologically but by a loose classification by subject.

WHICH PROGRAMMING LANGUAGE?

There are three stages of program development:

1. Making an explicit model of the 'real world', ie of the subject matter of the computation.

2. Specifying, on the basis of this model, the function that the program is to perform.

3. Implementing the specification resulting from the first two stages. All programming languages are unhappy compromises between the needs of the first two stages and those of the third: no language can serve both needs well. Programmers must therefore use at least two languages: a specification language and an implementation language. FORTRAN is so naive and so strongly biased towards implementation needs that it is useless for specification. So the question arises: what language should physicists use for specification? The answer, of course, is that physicists should use mathematics - that is, appropriate formal systems which allow description of the subject matter and reasoning about it. In part, this mathematics can be the mathematics used now to describe particle physics. But different mathematics is needed to handle experimental apparatus and experimental results, and even to handle particle physics from a programming point of view. Some sound mathematics has been developed by computer scientists, as Rob. Pike said. The value of this mathematics lies no: in the theorems that can be proved, but rather in the mental apparatus it provides for abstraction in the domains. The fundamental requirement for good programming is a clear distinction between specification and implementation. For example, if the subject matter of the computation is strongly time-ordered, a good specification would be one written in terms of multiple concurrent sequential processes: the fact that they must be somehow implemented as a single sequential process in FORTRAN is an implementation requirement that should be kept separate from the specification considerations. If specification work is explicit and sound, the programming language is relegated to its properly limited role, in implementation. FORTRAN has many advantages, and its users may rightly demand compelling reasons to justify a change. The danger of FORTRAN lies in regarding it as more than an implementation language, in allowing its very limited view of programming and its impossibly model of reality to contaminate the thinking in the problem domain, where richer and more powerful concepts are essential.

M. Jackson
A very important aspect of inducing the physics community to use a language different from FORTRAN is the enormous investment existing in this language, an investment both in expertise and library routines or analysis programs. It seems to me that languages like PASCAL may well be a much better tool for developing and implementing operating systems, but that they stand little chance in the environment of High Energy Physics Data Analysis.

J. Allaby

FORTRAN may be the most portable language today, but examples exist that a much used and presumably 'correct' code produces wrong results or even does not compile when brought to a different machine. More important than the language of implementation is the correct way of conceiving the program.

D. Myers

Software will stay with us for many years. Is it enough to use inertia as an argument for keeping FORTRAN forever? What will be the language a century from now?

R. Pike

The very nature of the physicist's work, very close to data which he generates in his detector bit by bit, makes for the necessity of using a machine oriented language for his work. FORTRAN seems to be far ahead of all other languages in this respect.

H. Renshall

What is necessary is not so much this or that facility in the language; clearly someone generating bit patterns in a detector must have access to them in his programming language. However, it is extremely relevant to have the best tools available in the design of the program, in modeling his solution. It is there that FORTRAN does not provide enough guidance.

R. Marty

The more important issue is the availability of good compilers. Programmers learn a new language fast, and I see no reason why all problems have to be solved by the same language. There may be some interesting developments, however, bringing totally new concepts, like data flow languages. One should follow the possibilities for such fundamental changes.

C. Mazza

Languages other than FORTRAN really do not solve the physicists' problems. PASCAL as defined by Wirth is unusable. One needs interfaces like personal workstations to make use of the new languages, and we will need the new languages to utilise efficiently the future computer architectures and the factor 10 in capacity due to VLSI.

U. Wambach
I think it is important to think of programs as objects not only for communication between man and machine, but also for communicating amongst people. In any language can you achieve this readability.

O. Hell

Person-to-person communication is indeed an important objective for programs; This and other problems arising in very large programs have not received enough attention in Computer Science. Compared to dangerous concepts like COMMON or EQUIVALENCE statements in FORTRAN, the much advertised harmfulness of the GO TO statement looks to me like peanuts. Likewise, the information carried in variable names must be a very different one in large programs. Best example how basic some of these problems are, is the existence of conventions as drawn up in the OLYMPUS system.

R. Bock

In our set of standards for highly reliable software, COMMON and EQUIVALENCE statements are rigorously banned.

C. Mazza

Both ADA and FORTRAN 8X will introduce solutions to the COMMON variables problem, indeed a serious one.

R. Marty

The issue of programming language, in the present thinking of Computer Science, does include the programming environment issue and the structuring of problem solving, by the definition of the primitives.

P. Svensson

THE EFFICIENCY ISSUE

Physicists have nothing to learn from anyone to make their big programs run faster. They have explored the possibilities more than anyone else. However, that may not be an essential question. Writing big programs in FORTRAN is perhaps very close to writing them in machine language, which nobody seriously proposes although programs would certainly come out more efficient that way. Maybe using FORTRAN is too small a step, and resembles someone who would walk a few steps southwest from Geneva, because he wants to go to Madrid instead of going to a travel agency to book a flight. What matters more than how fast a program runs is how difficult it is to write it, and how it is written. The concepts that will lead physicists in that direction, cannot be learnt in the short time of a workshop, but Computer Science has had, for many years, quite a few relevant ideas in store for physicists.

R. Pike
When I hear that we should discard our way of thinking about machine efficiency as an out-of-date obsession, I should remind you that many High Energy Physics Laboratories do not have access to enough computing capacity, and can certainly not take a factor 2 or 3 of execution time, even if the program writing process is improved. I think FORTRAN 77 is just all that we can take.

W. Mitaroff

Modern languages are not inefficient because of their new concepts. Block structuring your program or using data structures will not cost you anything. The same is true for using pointers to build dynamic data structures, or for recursion, where you need them. Abstract data typing may be different, but their existence may be invaluable during testing. I don’t believe that efficiency arguments alone can seriously keep FORTRAN alive.

R. Marty

I am afraid that variable precision or dynamic local storage (both proposed for FORTRAN 8X) will indeed seriously cost efficiency.

H. Renshall

Physicists will write in any language as long as it is called FORTRAN. That may well settle the future issue FORTRAN 8X versus ADA. More important, however, is that we will one day end up with special systems and tools written for us, and that we will forget the efficiency issue, given the decreasing price of hardware.

D. Williams

TOOLS IN SOFTWARE

I would hold that the present environment of development of software (in as much as it exists) will prove to be inadequate in the context of LEP already. Particular tools for High Energy Physics tasks will have to be created and distributed a few years from now.

R. Zelazny

Modern Programming languages perpetuate a few serious shortcomings which we were faced with in FORTRAN, and which have cost us in the past the writing of complementary tools, e.g. for I/O of data structures in a machine independent way. In the disussion about ADA it came out already that this and similar problems do not get solved soon. Our outlook seems to be that we will not get away from creating our own tools in areas which are problematic not only for High Energy Physics.

H. Grote

Physicists have not been educated enough to break up their problems into parts such that existing solutions from outside the High Energy Physics community can be used.

K. Jeffery
Physicists use FORTRAN only because it allows them to move from one machine to another with minimal effort. Beyond that, they have developed their higher level concepts, and they have oriented them along the thinking of modern languages. If they come as packages, it is only because no satisfactory language supports them.

J. Zoll

To me it seems very clear that High Energy Physics really must make a definite effort to use generally available tools, or else will shut themselves out from future technological advances.

M. Turnill

SPECIAL PROBLEMS OF SOFTWARE IN HIGH ENERGY PHYSICS

One impression that I retain from this workshop is that of the inadequacy of resources which CERN invests in computer-related problems. With all the exciting advances like new languages or personal workstations, High Energy Physics is clearly far behind other organisations (e.g. the talks of Mazza and Turnill) in organising its staff tightly and in costing its projects correctly. There are no indications that this kind of necessary streamlining is likely to happen, and the future of software in our field may well turn out to be disastrous.

M. Metcalf

How can physicists ever become good at writing programs with the right concepts, if they are daily confronted with the worst piece of software ever written, the IBM operating system? (General Applause)

Ch. Curran

This one can, of course, be solved by removing the operating system’s direct interface from the user, e.g. by personal workstations, which are setting out to offer a much more human interface.

K. Jeffery

One of the problems of the present environment is that physicists are faced with too many systems, particularly as most physicists are casual users. This will become even more serious in the future, unless we are very careful. An important step towards giving the physicist more time to do his physics will be to introduce easier interfaces, and preferably in an interactive environment.

H. Grote

I believe that the interactive environments of personal workstations at present seem so unstandardized, that they are likely to require even more knowledge. Not even keyboards are standardized. Things are likely to get much worse.

D. Williams
A point that has not been brought out sufficiently is the misconceptions that exist in our thinking when talking about or writing software; I would call them structural clashes of thought in analogy to M. Jackson's structure clashes of data. I consider them much more serious than questions of language. As an example, consider the generally accepted requirement of clear specification of software: Much of what exists in High Energy Physics is created in an environment that evolves steadily. There is, therefore, a constant (but not conscious) switching between 'engineering phases' of proper implementation, and 'research phases' of generating feedback i.e. new ideas on physics, on detectors, on analysis methods. All of this keeps modifying the specifications. It is relevant to keep this in mind when making global statements about the approach of the physics community to software.

R. Bock

Should one not consider to split the interfaces in writing software into a high-level interface for the scientist, and an expert one for professional programmers?

K. Jeffery

Where would the experts come from?

M. Metcalf

That seems to generate the situation covered by this slogan: The physicists don't know how to program, and the professionals don't know what to program.

J. Zoll

How can we seriously discuss about the efficiency of introducing workstations or new languages, if software is not even managed to the extent that proper costing figures exist for software projects?

R. Cailliau
LIST OF PARTICIPANTS

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Allaby</td>
<td>CERN EP Division</td>
</tr>
<tr>
<td>J.P. Albanese</td>
<td>Université de Marseille</td>
</tr>
<tr>
<td>S.R. Amendolia</td>
<td>INFN, Pisa</td>
</tr>
<tr>
<td>E. Arık</td>
<td>Bogazici University, Istanbul</td>
</tr>
<tr>
<td>J. Audy</td>
<td>Aérospatiale, Avions, Paris</td>
</tr>
<tr>
<td>J. Barlow</td>
<td>Rutherford Appleton Laboratory</td>
</tr>
<tr>
<td>R. Bazzolli</td>
<td>Université de Marseille</td>
</tr>
<tr>
<td>Y. Belokopytov</td>
<td>Serpukhov, Protvino</td>
</tr>
<tr>
<td>U. Berthon</td>
<td>Ecole Polytechnique, Palaiseau</td>
</tr>
<tr>
<td>J. Bettels</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>R. Bilhaut</td>
<td>LAL, Université de Paris-Sud</td>
</tr>
<tr>
<td>J. Blake</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>V. Blobel</td>
<td>Experimental Physics Institute, Hamburg</td>
</tr>
<tr>
<td>R. Blokzeyl</td>
<td>NIKHEF, Amsterdam</td>
</tr>
<tr>
<td>R. Böck</td>
<td>CERN EP Division</td>
</tr>
<tr>
<td>A. Bogaerts</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>B. Bouquet</td>
<td>LAL, Université de Paris-Sud</td>
</tr>
<tr>
<td>H. Brück</td>
<td>University of Wuppertal</td>
</tr>
<tr>
<td>R. Brun</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>F. Bruyant</td>
<td>CERN EP Division</td>
</tr>
<tr>
<td>M. Budinich</td>
<td>INFN, Trieste</td>
</tr>
<tr>
<td>D. Burridge</td>
<td>ECN, Berkshire</td>
</tr>
<tr>
<td>N. Butscher</td>
<td>Hahn-Weitner-Institut, Berlin</td>
</tr>
<tr>
<td>R. Campbell</td>
<td>Imperial College, London</td>
</tr>
<tr>
<td>M. Canon</td>
<td>IBM Research Laboratory, San Jose</td>
</tr>
<tr>
<td>J.T. Carroll</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>B.J. Charles</td>
<td>Rutherford Appleton Laboratory</td>
</tr>
<tr>
<td>F. Chevrier</td>
<td>CERN EP Division</td>
</tr>
<tr>
<td>C. Coutures</td>
<td>CEN, Saclay</td>
</tr>
<tr>
<td>F.J. Crijns</td>
<td>Physics Laboratory, Nijmegen</td>
</tr>
<tr>
<td>M.G. Crowley-Milling</td>
<td>CERN DD-LEP</td>
</tr>
<tr>
<td>C. Curran</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>I. Dahl-Jensen</td>
<td>Niels Bohr Institute, Copenhagen</td>
</tr>
<tr>
<td>E. Dahl-Jensen</td>
<td>Niels Bohr Institute, Copenhagen</td>
</tr>
<tr>
<td>G.H. Diercksen</td>
<td>Max-Planck-Institut, Munich</td>
</tr>
<tr>
<td>R. Dobinson</td>
<td>CERN EP Division</td>
</tr>
<tr>
<td>J.P. Dufey</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>P. Dufour</td>
<td>Université de l’Etat à Mons</td>
</tr>
<tr>
<td>P. Egloff</td>
<td>Hahn-Weitner-Institut, Berlin</td>
</tr>
<tr>
<td>G. Enderle</td>
<td>Kernforschungszentrum, Karlsruhe</td>
</tr>
<tr>
<td>J.-M. Ferte</td>
<td>Institut de Physique, Louvain-la-Neuve</td>
</tr>
<tr>
<td>F. Fidecaro</td>
<td>INFN, Pisa</td>
</tr>
<tr>
<td>S.M. Fisher Rl</td>
<td>Rutherford Appleton Laboratory</td>
</tr>
<tr>
<td>A. Flavell</td>
<td>University of Glasgow</td>
</tr>
<tr>
<td>D.G. Fong</td>
<td>DESY, Hamburg</td>
</tr>
<tr>
<td>D. Foster</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>M. Freytag</td>
<td>DESY, Hamburg</td>
</tr>
<tr>
<td>P. Frenkkiel</td>
<td>Collège de France, Paris</td>
</tr>
<tr>
<td>V. Frigo</td>
<td>CERN DC</td>
</tr>
<tr>
<td>R. Frühwirth</td>
<td>CERN EP Division</td>
</tr>
<tr>
<td>E. Gabathuler</td>
<td>CERN DG</td>
</tr>
<tr>
<td>F. Gagliardi</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>J. Gamble</td>
<td>CERN ISR Division</td>
</tr>
<tr>
<td>Ph. Gavillet</td>
<td>CERN EP Division</td>
</tr>
<tr>
<td>F.X. Gentit</td>
<td>CEN, Saclay</td>
</tr>
</tbody>
</table>
S. Gill  
J.C. Gordon  
L. Griffiths  
H. Grote  
R. Gruber  
N. Grüner  
R. Hagelberg  
M. Hansroul  
F. Harris  
J.C. Hart  
I. Hughes  
M. Heiden  
G. Heiman  
C. Helft  
O. Hell  
M. Huber  
J. Hutton  
M. Jackson  
D. Jacobs  
H.P. Jakob  
K. Jeffery  
P. Jeffries  
T. Johansson  
P. Johnson  
H. Johnstad  
C. Jones  
G. Kellner  
H.J. Klein  
P.E. Kuhlmann  
P.F. Kunz  
G. Lafferty  
J.P. Laugier  
F. Liello  
E. Lillestøl  
D. Lord  
B. Lörrstad  
M. Maire  
D.K. Mareis  
J.C. Marin  
R. Marty  
M. Masetti  
R. Matthews  
R. Maybury  
C. Mazza  
N. McRubin  
R. Meinke  
M. Metcalf  
W. Mitaroff  
R. Morano  
A. Moreton  
J.V. Morris  
D.J. Munday  
R. Myers  
B.S. Nielsen  
B.S. Nilsson  
A. Nobile  
S.W. O’Neale  
J.M. Olsen  
A.M. Osborne  
P. Palazzi  
L. Pape  
G.N. Patrick  
A. Petit  
R. Pike  
A. Poppleton  
J. Prevost  
A. Putzer  
CERN EP Division  
Rutherford Appleton Laboratory  
CERN DD Division  
Ecole Polytechnique Fédérale, Lausanne  
Max-Planck-Institut, Munich  
CERN DD Division  
CERN DD Division  
University of Oxford  
Rutherford Appleton Laboratory  
University of Glasgow  
CERN EP Division  
INFN, Bologna  
DESY, Hamburg  
Max-Planck-Institut, Munich  
Rutherford Appleton Laboratory  
Michael Jackson Systems Ltd., London  
CERN DD Division  
Rheinische Friedrich-Wilhelms Universität, Bonn  
Rutherford Appleton Laboratory  
Rutherford Appleton Laboratory  
University of Uppsala  
University of Liverpool  
Peri National Accelerator Laboratory  
CERN DD Division  
CERN EP Division  
CERN EP Division  
DESY, Hamburg  
SLAC, California  
University of Manchester  
CEN, Saclay  
INFN, Trieste  
University of Bergen  
CERN DD Division  
University of Lund  
LAPP, Annecy  
Universität Osnabrück  
CERN EP Division  
Inst. f. Informatics, University of Zürich  
INFN, Bologna  
CERN DD Division  
Rutherford Appleton Laboratory  
ESA, Darmstadt  
CERN EP Division  
University of Bonn  
CERN DD Division  
Öst. Akademie d. Wissenschaften, Vienna  
Ecole Polytechnique, Palaiseau  
University of Liverpool  
Rutherford Appleton Laboratory  
Cavendish Laboratory, Cambridge  
CERN DD Division  
CERN EP Division  
Niels Bohr Institute, Copenhagen  
Int. School for Advanced Studies, Trieste  
University of Birmingham  
University of Bergen  
CERN DD Division  
CERN DD Division  
CERN EP Division  
CERN DD Division  
LAL, Université de Paris-Sud  
Bell Telephone Lab. Murray Hill, N.J.  
CERN DD Division  
CERN, Saclay  
Institut für Hochenergiephysik, Heidelberg
<table>
<thead>
<tr>
<th>Name</th>
<th>Institution/Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>H. Renshall</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>J. Richards</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>P. Rimmer</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>L. Robertson</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>A. Romayya</td>
<td>University of Oxford</td>
</tr>
<tr>
<td>M. Rumpf</td>
<td>Ecole Polytechnique, Palaiseau</td>
</tr>
<tr>
<td>D. Samyn</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>H. Schiller</td>
<td>Institut für Höchnergphysik, Zeuthen</td>
</tr>
<tr>
<td>M. Schoorel</td>
<td>NIKHEF, Amsterdam</td>
</tr>
<tr>
<td>W. Schoett</td>
<td>DESY, Hamburg</td>
</tr>
<tr>
<td>H. Seebrunner</td>
<td>Max-Planck-Institut, Munich</td>
</tr>
<tr>
<td>I. Seis</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>S. Squarcia</td>
<td>CERN EP Division</td>
</tr>
<tr>
<td>C. Story</td>
<td>University of Oxford</td>
</tr>
<tr>
<td>P. Svensson</td>
<td>Swedish Nat. Defense Research Inst., Stockholm</td>
</tr>
<tr>
<td>A.S. Thompson</td>
<td>University of Glasgow</td>
</tr>
<tr>
<td>M. Turnill</td>
<td>BROC, Glasgow</td>
</tr>
<tr>
<td>R. Urbani-THS</td>
<td>ESTEC, Noordwijk</td>
</tr>
<tr>
<td>P. Van Binst</td>
<td>Université Libre de Bruxelles</td>
</tr>
<tr>
<td>W.M. van Leeuwen</td>
<td>NIKHEF, Amsterdam</td>
</tr>
<tr>
<td>J. Vermeulen</td>
<td>NIKHEF, Amsterdam</td>
</tr>
<tr>
<td>I. Willers</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>A.S. Williams</td>
<td>Rutherford Appleton Laboratory</td>
</tr>
<tr>
<td>D.O. Williams</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>G. Wilquet</td>
<td>Université Libre de Bruxelles</td>
</tr>
<tr>
<td>M. Windmolders</td>
<td>Université de l'Etat à Mons</td>
</tr>
<tr>
<td>R.W. Witty</td>
<td>Rutherford Appleton Laboratory</td>
</tr>
<tr>
<td>W. Wojcik</td>
<td>CERN, Saclay, Palaiseau</td>
</tr>
<tr>
<td>R. Worden</td>
<td>Logica Ltd., London</td>
</tr>
<tr>
<td>P. Zafiropulo</td>
<td>IBM Research Laboratory, Zurich</td>
</tr>
<tr>
<td>P. Zamella</td>
<td>CERN DD Division</td>
</tr>
<tr>
<td>S. Zaminotti</td>
<td>CERN, Saclay, Palaiseau</td>
</tr>
<tr>
<td>R. Zelazny</td>
<td>Inst. of Nuclear Research, Otwock-Swierk, Poland</td>
</tr>
<tr>
<td>J. Zoll</td>
<td>CERN EP Division</td>
</tr>
</tbody>
</table>